

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

КАФЕДРА КОНСТРУЮВАННЯ ЕОА

ЗВІТ

з лабораторної роботи №4
по курсу «Основи мікропроцесорної техніки - 1»

Виконав:

студент гр. ДК-82

Сопіра Р. Я.

Перевірив:

доц. Корнєв В. П.

ст. в. Бондаренко Н. О.

Київ – 2020

Завдання

Створити програму, що виконує віднімання двох довільних чисел заданої розрядності у ASCII форматі:

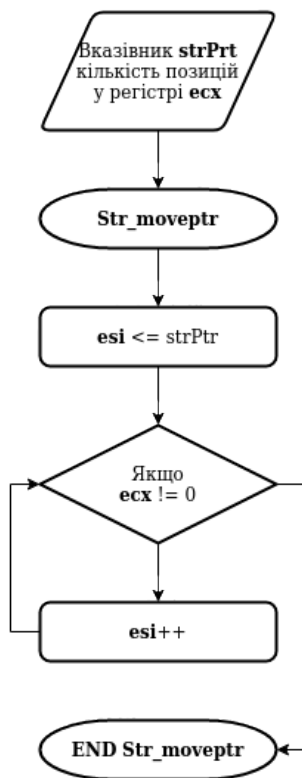
- 1) Числа вводяться по черзі з клавіатури;
- 2) Відняти введені числа;
- 3) Результат операції вивести у консольне вікно;

Блок-схема алгоритму програми

Алгоритм даної програми є непростим і трохи громіздким, тому його варто розбити на кілька окремих взаємопов'язаних блоків, які будуть розглядатися по черзі разом із їх реалізацією на мові асемблеру.

Str_moveptr:

Блок-схема та вихідний код



```
Str_moveptr PROC USES ecx,  
    strPtr:PTR BYTE  
  
    mov esi, strPtr  
  
move_place:  
    inc esi  
    loop move_place  
  
    ret  
  
Str_moveptr ENDP
```

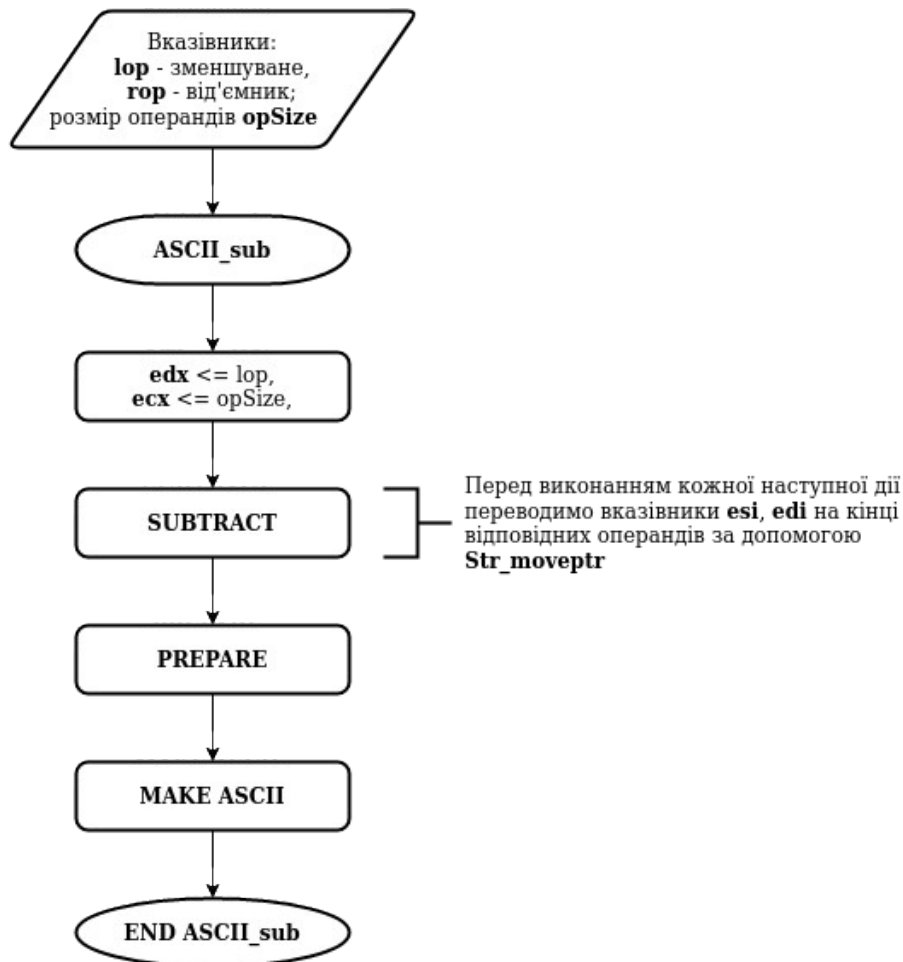
Дана процедура призначена для пересування вказівника вперед на задану кількість позицій у рядку. Кількість позицій на які потрібно пересунути вказівник береться з регістру **ecx**, а вказівник передається як аргумент **strPtr**, що записується у регістр **esi**, який потім повертається як результат.

Далі відбуватиметься інкрементація регістру **esi** і декрементація **ecx** до тих пір, поки останній не буде рівний нулю.

Директива *USES* використовується щоб зберегти значення регістру **ecx** до стеку й повернути його після виконання процедури.

ASCII_sub:

Блок-схема



Ця процедура реалізує алгоритм віднімання двох, наприклад, введених з клавіатури чисел у вигляді рядків однакового розміру (результат у першому операнді), корегування (за необхідності) отриманого результату віднімання і формування уже правильного вихідного ASCII рядка.

Дана процедура виконується у три основні етапи:

- 1) **SUBTRACT** — етап віднімання рядків;
- 2) **PREPARE** — корегування результату за необхідності;
- 3) **MAKE ASCII** — формування вихідного рядка для виводу в консольне вікно;

Початок процедури

```
ASCII_sub PROC USES eax ebx edx esi edi,  
    lop:PTR BYTE,  
    rop:PTR BYTE,  
    opSize: BYTE  
  
    mov edx, lop                ; mov lop start address for checking pointers  
  
    mov ecx, DWORD PTR opSize   ; string size to move string pointers to the end  
    invoke Str_moveptr, rop      ; move esi to the end of rop  
    mov edi, esi                ; copy esi into edi to point into rop  
    invoke Str_moveptr, lop      ; move esi to the end of lop
```

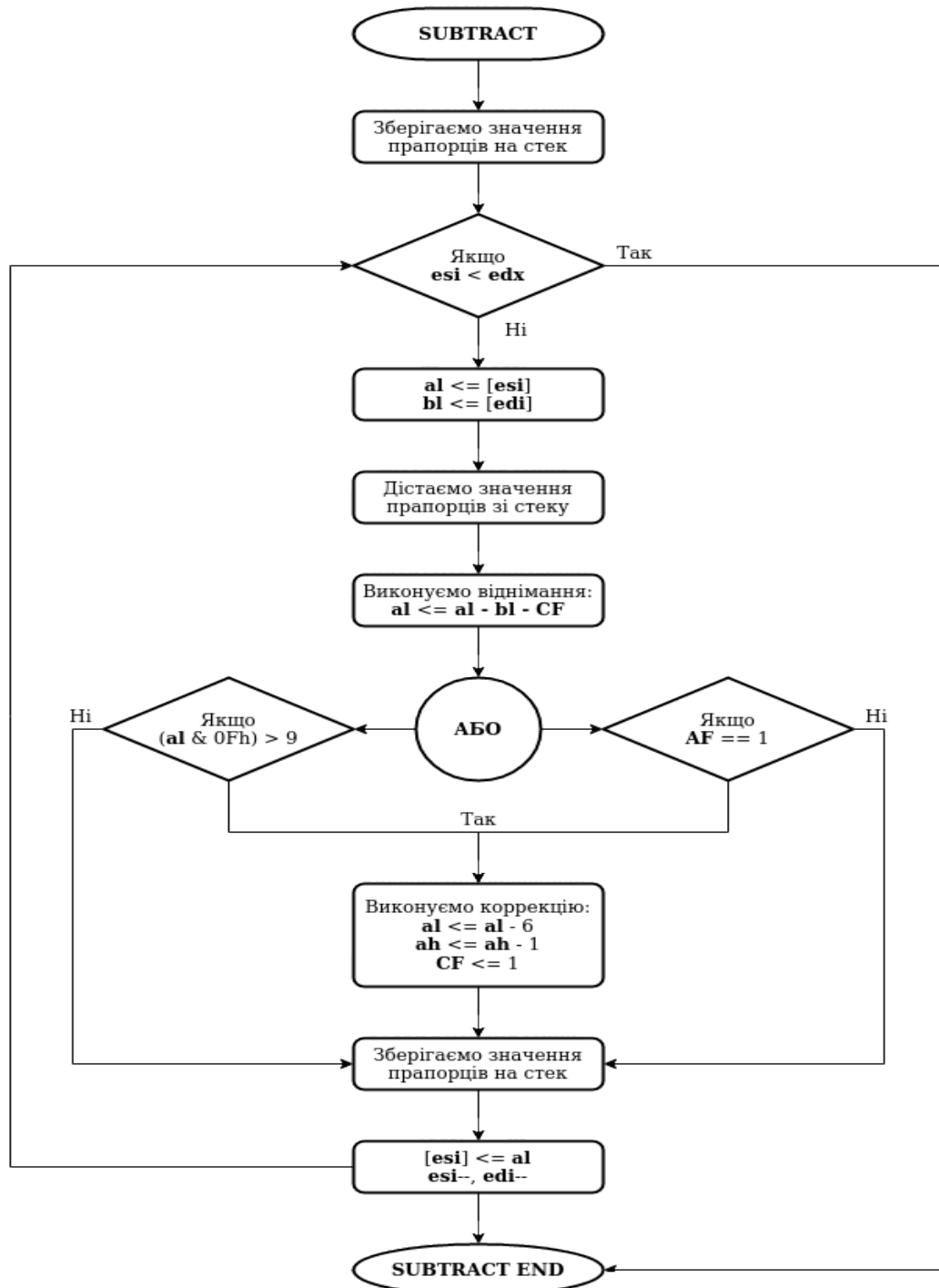
Спочатку у **edx** завантажується адреса початку лівого операнда (він же результат), далі серія з викликів процедур **Str_moveptr** і команди **mov** переводить **esi** і **edi** (використовуються як вказівники на лівий і правий операнди відповідно) на певну кількість позицій вперед, а саме на кінці рядків операндів.

Першим, для зручності, виконується зсув вказівника на правий операнд на кількість позицій завантажену у регістр **ecx**, значення із **esi** копіюється у **edi** — тепер **edi** вказує на наймолодший розряд у правому операнді — далі, **esi** зсувається на ту ж кількість позицій і ми готові виконувати віднімання.

Тепер, розглядатимемо кожен із етапів послідовно:

1) Віднімання операндів (SUBTRACT):

Блок-схема



Вихідний код

```
xor eax, eax                ; not really required, but we make sure
pushfd                      ;
sub_loop:
    cmp esi, edx             ; check if end is reached
    jnb prepare              ; if yes -> start preparing the result

    mov al, BYTE PTR [esi]   ; move left-hand ascii-char into al
    mov bl, BYTE PTR [edi]   ; move right-hand ascii-char into bl

    popfd                    ; get flags back
    sbb al, bl                ; subtract
    aas                      ; adjust
    pushfd                   ; save flag states onto stack

next_char:                   ; get next character
    mov [esi], al             ; write the result back

    dec esi                  ; decrement pointers
    dec edi
    jmp sub_loop              ; loop back
```

Спершу, виконується очищення регістру **eax** і тим самим скидання прапору **CF**, необхідне нам для правильного віднімання наймолодших розрядів. Потім **efl** (регістр прапорців) завантажується на стек командою **pushfd**, виконується перевірка адрес вказівників за схожим принципом, що і в минулій лабораторній і, нарешті, розряди операндів завантажуються у відповідні регістри (лівий у **al**, правий у **bl**), повертаються значення регістру **efl** зі стеку командою **popfd**, виконується команда **sbb** — віднімання, що враховує прапорець переносу **CF**, корекція результату, якщо вона необхідна, і збереження прапорців на стеку для наступних розрядів знову командою **pushfd**.

Результат такого порозрядного віднімання двох розрядів деяких чисел наведений нижче.

$$31_{16} - 31_{16} = 00_{16}, \quad \text{корекція не потрібна}$$

```
Disassembly: test.asm, 8 X
51 mov ecx, DWORD PTR opSize ; mov ecx=00000004 for checking pointers
52 invoke Str_moveptr, rop ; string size to move string pointers to the end
53 mov edi, esi ; move esi to the end of rop
54 invoke Str_moveptr, lop ; copy esi into edi to point into rop
55 ; move esi to the end of lop
56 xor eax, eax ; not really required, but we make sure
57 pushfd
58
59 sub_loop:
60 popfd ; get flags back
61
62 pushfd ; push/pop required to save flags from being changed by cmp
63 cmp esi, edx ; check if end is reached
64 jb prepare ; if yes -> start preparing the result
65 popfd
66
67 mov al, BYTE PTR [esi] ; move left-hand ascii-char into al
68 mov bl, BYTE PTR [edi] ; move right-hand ascii-char into ab
69
70 sbb al, bl ; subtract < 1ms elapsed
71 aas ; adjust
72 pushfd ; save flag states onto stack
73
74 next_char:
75 mov [esi], al ; get next character
76 ; write the result back
77 dec esi ; decrement pointers
78 dec edi
79 jmp sub_loop ; loop back
80
```

Registers:

- EAX = 00000031
- EBX = 7EFD0031
- ECX = 00000004
- EDX = 00A66000
- ESI = 00A66004
- EDI = 00A6600A
- EIP = 00A636A0
- ESP = 003AFDD4
- EBP = 003AFDE8
- EFL = 00000246

OV = 0 UP = 0 EI = 1 PL = 0
ZR = 1 AC = 0 PE = 1

```
Disassembly: test.asm, 8 X
51 mov ecx, DWORD PTR opSize ; mov ecx=00000004 for checking pointers
52 invoke Str_moveptr, rop ; string size to move string pointers to the end
53 mov edi, esi ; move esi to the end of rop
54 invoke Str_moveptr, lop ; copy esi into edi to point into rop
55 ; move esi to the end of lop
56 xor eax, eax ; not really required, but we make sure
57 pushfd
58
59 sub_loop:
60 popfd ; get flags back
61
62 pushfd ; push/pop required to save flags from being changed by cmp
63 cmp esi, edx ; check if end is reached
64 jb prepare ; if yes -> start preparing the result
65 popfd
66
67 mov al, BYTE PTR [esi] ; move left-hand ascii-char into al
68 mov bl, BYTE PTR [edi] ; move right-hand ascii-char into ab
69
70 sbb al, bl ; subtract < 1ms elapsed
71 aas ; adjust
72 pushfd ; save flag states onto stack
73
74 next_char:
75 mov [esi], al ; get next character
76 ; write the result back
77 dec esi ; decrement pointers
78 dec edi
79 jmp sub_loop ; loop back
80
```

Registers:

- EAX = 00000000
- EBX = 7EFD0031
- ECX = 00000004
- EDX = 00A66000
- ESI = 00A66004
- EDI = 00A6600A
- EIP = 00A636B0
- ESP = 003AFDD4
- EBP = 003AFDE8
- EFL = 00000246

OV = 0 UP = 0 EI = 1 PL = 0
ZR = 1 AC = 0 PE = 1

$38_{16} - 39_{16} = FF_{16}$, необхідна корекція результату

```

51  mov ecx, DWORD PTR opSize      ; string size to move string pointers to the end
52  invoke Str_moveptr, rop        ; move esi to the end of rop
53  mov edi, esi                   ; copy esi into edi to point into rop
54  invoke Str_moveptr, lop        ; move esi to the end of lop
55
56  xor eax, eax                  ; not really required, but we make sure
57  pushfd
58
59  sub_loop:
60  popfd                          ; get flags back
61
62  pushfd                         ; push/pop required to save flags from being changed by cmp
63  cmp esi, edx                  ; check if end is reached
64  jb prepare                     ; if yes -> start preparing the result
65  popfd
66
67  mov al, BYTE PTR [esi]         ; move left-hand ascii-char into al
68  mov bl, BYTE PTR [edi]         ; move right-hand ascii-char into ab
69
70  sbb al, bl                     ; subtract < 1ms elapsed
71  aas                           ; adjust
72  pushfd                         ; save flag states onto stack
73
74  next_char:                     ; get next character
75  mov [esi], al                 ; write the result back
76
77  dec esi                        ; decrement pointers
78  dec edi
79  jmp sub_loop                  ; loop back
80

```

Registers:

- EAX = 00000038
- EBX = 7EFD0339
- ECX = 00000004
- EDX = 00A66000
- ESI = 00A66003
- EDI = 00A66009
- EIP = 00A636AD
- ESP = 003AFDD4
- EBP = 003AFDE8
- EFL = 00000246

OV = 0 UP = 0 EI = 1 PL = 0
ZR = 1 AC = 0 PE = 1

```

51  mov ecx, DWORD PTR opSize      ; string size to move string pointers to the end
52  invoke Str_moveptr, rop        ; move esi to the end of rop
53  mov edi, esi                   ; copy esi into edi to point into rop
54  invoke Str_moveptr, lop        ; move esi to the end of lop
55
56  xor eax, eax                  ; not really required, but we make sure
57  pushfd
58
59  sub_loop:
60  popfd                          ; get flags back
61
62  pushfd                         ; push/pop required to save flags from being changed by cmp
63  cmp esi, edx                  ; check if end is reached
64  jb prepare                     ; if yes -> start preparing the result
65  popfd
66
67  mov al, BYTE PTR [esi]         ; move left-hand ascii-char into al
68  mov bl, BYTE PTR [edi]         ; move right-hand ascii-char into ab
69
70  sbb al, bl                     ; subtract < 1ms elapsed
71  aas                           ; adjust < 1ms elapsed
72  pushfd                         ; save flag states onto stack
73
74  next_char:                     ; get next character
75  mov [esi], al                 ; write the result back
76
77  dec esi                        ; decrement pointers
78  dec edi
79  jmp sub_loop                  ; loop back
80

```

Registers:

- EAX = 000000FF
- EBX = 7EFD0339
- ECX = 00000004
- EDX = 00A66000
- ESI = 00A66003
- EDI = 00A66009
- EIP = 00A636AF
- ESP = 003AFDD4
- EBP = 003AFDE8
- EFL = 00000297

OV = 0 UP = 0 EI = 1 PL = 1
ZR = 0 AC = 1 PE = 1

$FF_{16} \& 0F_{16} = 0F_{16}$

$0F_{16} - 06_{16} = 09_{16}$

```

51  mov ecx, DWORD PTR opSize      ; string size to move string pointers to the end
52  invoke Str_moveptr, rop        ; move esi to the end of rop
53  mov edi, esi                   ; copy esi into edi to point into rop
54  invoke Str_moveptr, lop        ; move esi to the end of lop
55
56  xor eax, eax                  ; not really required, but we make sure
57  pushfd
58
59  sub_loop:
60  popfd                          ; get flags back
61
62  pushfd                         ; push/pop required to save flags from being changed by cmp
63  cmp esi, edx                  ; check if end is reached
64  jb prepare                     ; if yes -> start preparing the result
65  popfd
66
67  mov al, BYTE PTR [esi]         ; move left-hand ascii-char into al
68  mov bl, BYTE PTR [edi]         ; move right-hand ascii-char into ab
69
70  sbb al, bl                     ; subtract < 1ms elapsed
71  aas                           ; adjust
72  pushfd                         ; save flag states onto stack < 1ms elapsed
73
74  next_char:                     ; get next character
75  mov [esi], al                 ; write the result back
76
77  dec esi                        ; decrement pointers
78  dec edi
79  jmp sub_loop                  ; loop back
80

```

Registers:

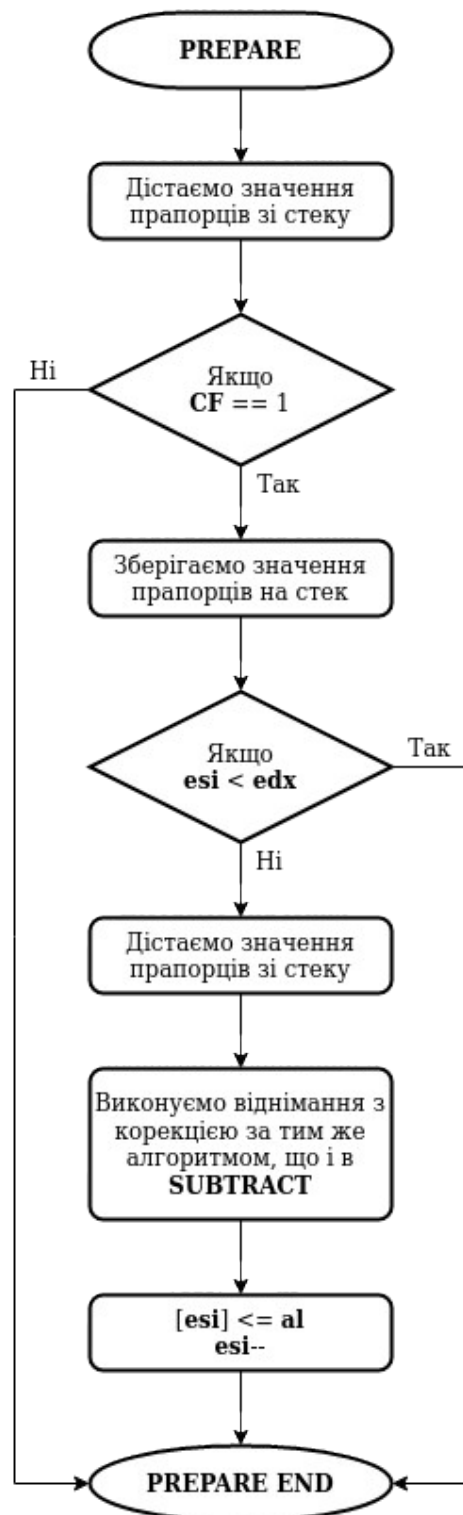
- EAX = 0000FF09
- EBX = 7EFD0339
- ECX = 00000004
- EDX = 00A66000
- ESI = 00A66003
- EDI = 00A66009
- EIP = 00A636B0
- ESP = 003AFDD4
- EBP = 003AFDE8
- EFL = 00000297

OV = 0 UP = 0 EI = 1 PL = 1
ZR = 0 AC = 1 PE = 1

Отримуємо поки що правильний результат.

2) Корекція результату (PREPARE):

Блок-схема



Виконуємо перетворення:
 $result = 10^N - result[N]$

Наприклад маємо:
00015 - ;(N = 5)
07819 =
92196

Щоб отримати правильний результат:
100000 - ;(10^N)
X92196 =
X07804

Вихідний код

```
prepare:
    popfd                ; if carry -> the result is negative
    jnc resp_out         ; check whether positive, if yes skip conversion

    mov al, '-'          ; if not, place '-' before the result
    call WriteChar
    invoke Str_moveptr, lop    ; move pointer to the end

    popfd                ; <- this can be a problem

convert:                ; if result is negative then do (10^N - result[N-1, 0]) to get true value
    pushfd              ; again -> done to save the flag states
    cmp esi, edx        ; check if end is reached
    jb resp_out         ; if yes -> string is ready to become ascii again
    popfd

    mov al, 00h         ; do the (10[i] - result[i])

    sbb al, [esi]       ; subtract
    aas                ; adjust
    mov [esi], al       ; write back

    dec esi
    jmp convert

resp_out:
    invoke Str_moveptr, lop    ; move pointer to the end
```

На початку даного етапу ми повертаємо зі стеку значення прапорців, що були отримані після віднімання найстарших розрядів, для того щоб перевірити чи було отримано від’ємне число (**CF** == 1), чи додатне (**CF** == 0). Якщо було отримано від’ємне число, то результат необхідно відкорегувати за наступною формулою:

$$10^N - \tilde{X}_N = X_N$$

Це реалізовано за майже таким самим алгоритмом, що був описаний вище. У даному випадку збереження перенесення було виконано через своєрідне “екранування” частини коду, що виконує перевірку умови перетинання адрес вказівників - а значить змінює стан прапорців - командами **pushfd, popfd**.

Якщо ж отримане число додатне, то цей етап фактично пропускається і ми переходимо до фінального етапу.

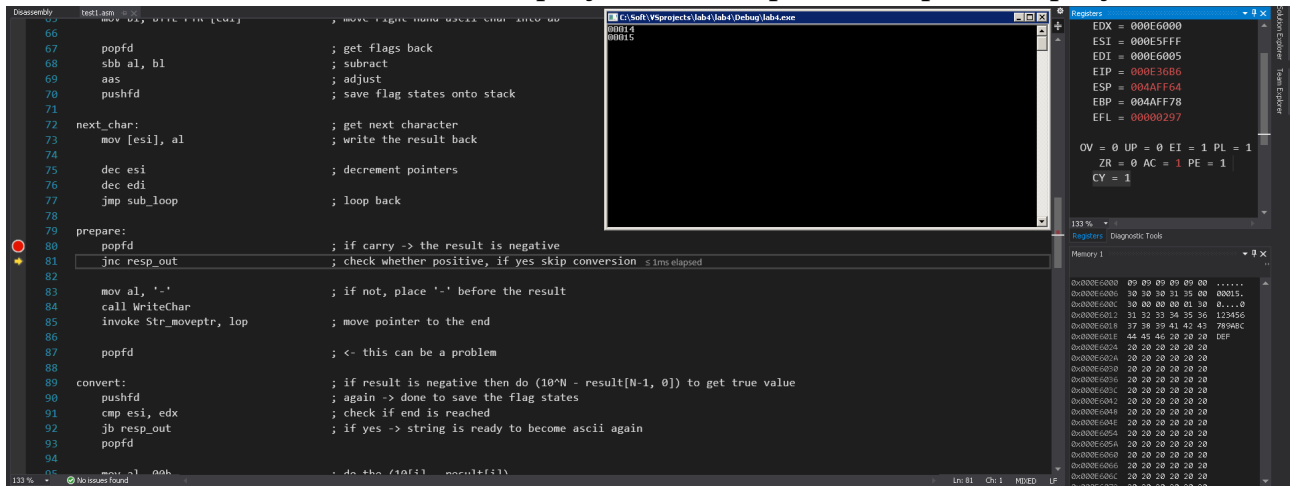
Робота даного етапу проілюстрована нижче:

$00015_{10} - 00014_{10} = 00001_{10}$, результат додатний корегування не потрібне

```
64      mov al, BYTE PTR [esi]      ; move left-hand ascii-char into al
65      mov bl, BYTE PTR [edi]      ; move right-hand ascii-char into ab
66
67      popfd                       ; get flags back
68      sbb al, bl                  ; subtract
69      aas                        ; adjust
70      pushfd                     ; save flag states onto stack
71
72      next_char:                  ; get next character
73      mov [esi], al               ; write the result back
74
75      dec esi                     ; decrement pointers
76      dec edi
77      jmp sub_loop               ; loop back
78
79      prepare:                    ; if carry -> the result is negative
80      popfd                       ; check whether positive, if yes skip
81      jnc resp_out
82
83      mov al, '-'                 ; if not, place '-' before the result
84      call WriteChar
85      invoke Str_moveptr, lop      ; move pointer to the end
86
87      popfd                       ; <- this can be a problem
88
89      convert:                    ; if result is negative then do (10*N - result[N-1, 0]) to get true value
90      pushfd
91      cmp esi, edx               ; again -> done to save the flag states
92      jnb resp_out               ; check if end is reached
93      popfd                       ; if yes -> string is ready to become ascii again
```

```
91      cmp esi, edx               ; again -> done to save the flag states
92      jnb resp_out               ; if yes -> string is ready to become ascii again
93      popfd
94
95      mov al, 00h                ; do the (10[i] - result[i])
96
97      sbb al, [esi]              ; subtract
98      aas                        ; adjust
99      mov [esi], al              ; write back
100
101      dec esi
102      jmp convert
103
104      resp_out:
105      invoke Str_moveptr, lop      ; move pointer to the end 51ms elapsed
106
107      make_ascii:
108      cmp esi, edx               ; check if end is reached
109      jnb finish                 ; saving flag states is pointless here, finally
110
111      mov al, BYTE PTR [esi]      ; get value
112
113      or al, ascii_num           ; make ascii char
114      mov [esi], al              ; write back
115      dec esi
116      jmp make_ascii
117
118      finish:
119      ret
120      ASCII_sub_CARRY
```

$00014_{10} - 00015_{10} = 99999_{10}$, результат не вірний, потрібне корегування



The screenshot shows a debugger window with the following assembly code:

```
66: mov edi, 00000000 ; move flags into edi and into edi
67: popfd ; get flags back
68: sbb al, bl ; subtract
69: aas ; adjust
70: pushfd ; save flag states onto stack
71:
72: next_char: ; get next character
73: mov [esi], al ; write the result back
74:
75: dec esi ; decrement pointers
76: dec edi
77: jmp sub_loop ; loop back
78:
79: prepare: ; if carry -> the result is negative
80: popfd ; check whether positive, if yes skip conversion < 1ms elapsed
81: jnc resp_out
82:
83: mov al, '-' ; if not, place '-' before the result
84: call WriteChar
85: invoke Str_moveptr, lop ; move pointer to the end
86:
87: popfd ; <- this can be a problem
88:
89: convert: ; if result is negative then do (10*M - result[N-1, 0]) to get true value
90: pushfd ; again -> done to save the flag states
91: cmp esi, edx ; check if end is reached
92: jb resp_out ; if yes -> string is ready to become ascii again
93: popfd
94:
95: mov al, 00h ; do the (10*i) - result[i]
```

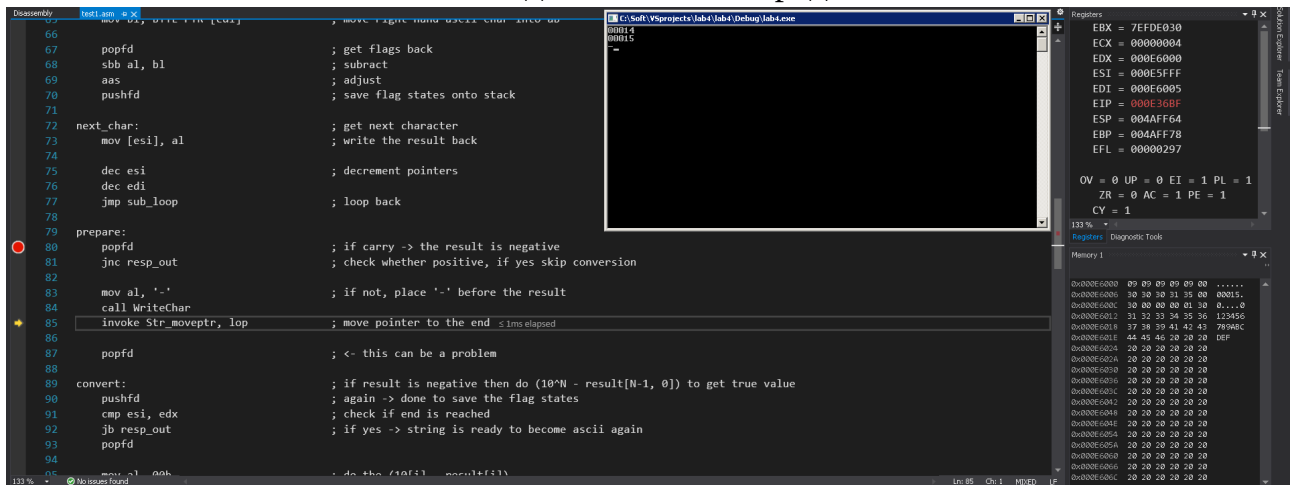
The registers window shows the following values:

Register	Value
EDX	000E6000
ESI	000E5FFF
EDI	000E6005
EIP	000E36B6
ESP	004AFF64
EBP	004AFF78
EFL	0000297

The memory window shows the following values:

Address	Value
00000000	00 00 00 00 00 00 00 00
00000001	00 00 00 00 00 00 00 00
00000002	00 00 00 00 00 00 00 00
00000003	00 00 00 00 00 00 00 00
00000004	00 00 00 00 00 00 00 00
00000005	00 00 00 00 00 00 00 00
00000006	00 00 00 00 00 00 00 00
00000007	00 00 00 00 00 00 00 00
00000008	00 00 00 00 00 00 00 00
00000009	00 00 00 00 00 00 00 00
0000000A	00 00 00 00 00 00 00 00
0000000B	00 00 00 00 00 00 00 00
0000000C	00 00 00 00 00 00 00 00
0000000D	00 00 00 00 00 00 00 00
0000000E	00 00 00 00 00 00 00 00
0000000F	00 00 00 00 00 00 00 00
00000010	00 00 00 00 00 00 00 00
00000011	00 00 00 00 00 00 00 00
00000012	00 00 00 00 00 00 00 00
00000013	00 00 00 00 00 00 00 00
00000014	00 00 00 00 00 00 00 00
00000015	00 00 00 00 00 00 00 00
00000016	00 00 00 00 00 00 00 00
00000017	00 00 00 00 00 00 00 00
00000018	00 00 00 00 00 00 00 00
00000019	00 00 00 00 00 00 00 00
0000001A	00 00 00 00 00 00 00 00
0000001B	00 00 00 00 00 00 00 00
0000001C	00 00 00 00 00 00 00 00
0000001D	00 00 00 00 00 00 00 00
0000001E	00 00 00 00 00 00 00 00
0000001F	00 00 00 00 00 00 00 00

Також виводимо знак '-' перед числом



The screenshot shows a debugger window with the following assembly code:

```
66: mov edi, 00000000 ; move flags into edi and into edi
67: popfd ; get flags back
68: sbb al, bl ; subtract
69: aas ; adjust
70: pushfd ; save flag states onto stack
71:
72: next_char: ; get next character
73: mov [esi], al ; write the result back
74:
75: dec esi ; decrement pointers
76: dec edi
77: jmp sub_loop ; loop back
78:
79: prepare: ; if carry -> the result is negative
80: popfd ; check whether positive, if yes skip conversion
81: jnc resp_out
82:
83: mov al, '-' ; if not, place '-' before the result
84: call WriteChar
85: invoke Str_moveptr, lop ; move pointer to the end < 1ms elapsed
86:
87: popfd ; <- this can be a problem
88:
89: convert: ; if result is negative then do (10*M - result[N-1, 0]) to get true value
90: pushfd ; again -> done to save the flag states
91: cmp esi, edx ; check if end is reached
92: jb resp_out ; if yes -> string is ready to become ascii again
93: popfd
94:
95: mov al, 00h ; do the (10*i) - result[i]
```

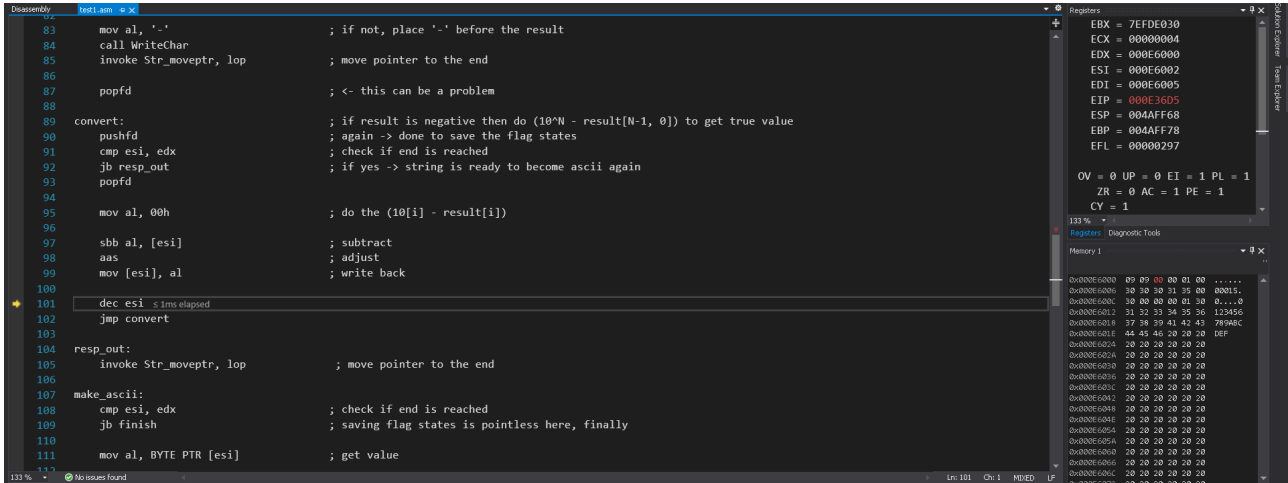
The registers window shows the following values:

Register	Value
EBX	7EFDE030
ECX	00000004
EDX	000E6000
ESI	000E5FFF
EDI	000E6005
EIP	000E36B6
ESP	004AFF64
EBP	004AFF78
EFL	0000297

The memory window shows the following values:

Address	Value
00000000	00 00 00 00 00 00 00 00
00000001	00 00 00 00 00 00 00 00
00000002	00 00 00 00 00 00 00 00
00000003	00 00 00 00 00 00 00 00
00000004	00 00 00 00 00 00 00 00
00000005	00 00 00 00 00 00 00 00
00000006	00 00 00 00 00 00 00 00
00000007	00 00 00 00 00 00 00 00
00000008	00 00 00 00 00 00 00 00
00000009	00 00 00 00 00 00 00 00
0000000A	00 00 00 00 00 00 00 00
0000000B	00 00 00 00 00 00 00 00
0000000C	00 00 00 00 00 00 00 00
0000000D	00 00 00 00 00 00 00 00
0000000E	00 00 00 00 00 00 00 00
0000000F	00 00 00 00 00 00 00 00
00000010	00 00 00 00 00 00 00 00
00000011	00 00 00 00 00 00 00 00
00000012	00 00 00 00 00 00 00 00
00000013	00 00 00 00 00 00 00 00
00000014	00 00 00 00 00 00 00 00
00000015	00 00 00 00 00 00 00 00
00000016	00 00 00 00 00 00 00 00
00000017	00 00 00 00 00 00 00 00
00000018	00 00 00 00 00 00 00 00
00000019	00 00 00 00 00 00 00 00
0000001A	00 00 00 00 00 00 00 00
0000001B	00 00 00 00 00 00 00 00
0000001C	00 00 00 00 00 00 00 00
0000001D	00 00 00 00 00 00 00 00
0000001E	00 00 00 00 00 00 00 00
0000001F	00 00 00 00 00 00 00 00

Виконуємо порозрядне віднімання



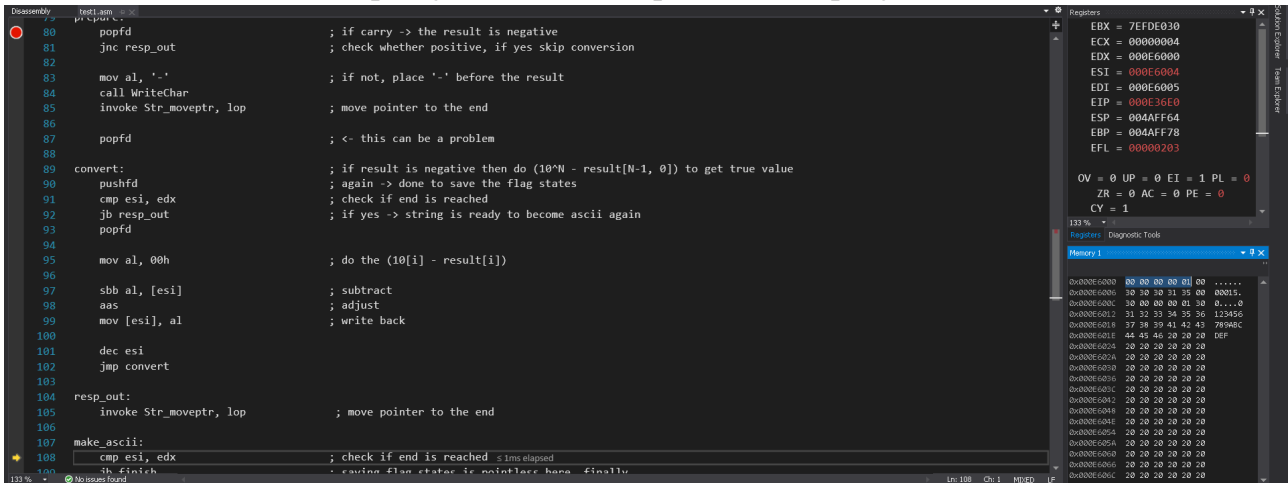
```
Disassembly: test1.asm: 4 X
83 mov al, '-' ; if not, place '-' before the result
84 call WriteChar
85 invoke Str_moveptr, lop ; move pointer to the end
86
87 popfd ; <- this can be a problem
88
89 convert: ; if result is negative then do (10*N - result[N-1, 0]) to get true value
90 pushfd ; again -> done to save the flag states
91 cmp esi, edx ; check if end is reached
92 jb resp_out ; if yes -> string is ready to become ascii again
93 popfd
94
95 mov al, 00h ; do the (10[i] - result[i])
96
97 sbb al, [esi] ; subtract
98 aas ; adjust
99 mov [esi], al ; write back
100
101 dec esi ; 1ms elapsed
102 jmp convert
103
104 resp_out:
105 invoke Str_moveptr, lop ; move pointer to the end
106
107 make_ascii:
108 cmp esi, edx ; check if end is reached
109 jb finish ; saving flag states is pointless here, finally
110
111 mov al, BYTE PTR [esi] ; get value
112
113 % No issues found
```

Registers

EBX	=	7EFD030
ECX	=	00000004
EDX	=	000E6000
ESI	=	000E6002
EDI	=	000E6005
EIP	=	000E3605
ESP	=	004AFF68
EBP	=	004AFF78
EFL	=	00000297

OV = 0 UP = 0 EI = 1 PL = 1
ZR = 0 AC = 1 PE = 1
CY = 1

І отримуємо дійсно правильний результат



```
Disassembly: test1.asm: 4 X
80 popfd ; if carry -> the result is negative
81 jnc resp_out ; check whether positive, if yes skip conversion
82
83 mov al, '-' ; if not, place '-' before the result
84 call WriteChar
85 invoke Str_moveptr, lop ; move pointer to the end
86
87 popfd ; <- this can be a problem
88
89 convert: ; if result is negative then do (10*N - result[N-1, 0]) to get true value
90 pushfd ; again -> done to save the flag states
91 cmp esi, edx ; check if end is reached
92 jb resp_out ; if yes -> string is ready to become ascii again
93 popfd
94
95 mov al, 00h ; do the (10[i] - result[i])
96
97 sbb al, [esi] ; subtract
98 aas ; adjust
99 mov [esi], al ; write back
100
101 dec esi
102 jmp convert
103
104 resp_out:
105 invoke Str_moveptr, lop ; move pointer to the end
106
107 make_ascii:
108 cmp esi, edx ; check if end is reached ; 1ms elapsed
109 jb finish ; saving flag states is pointless here, finally
110
111 % No issues found
```

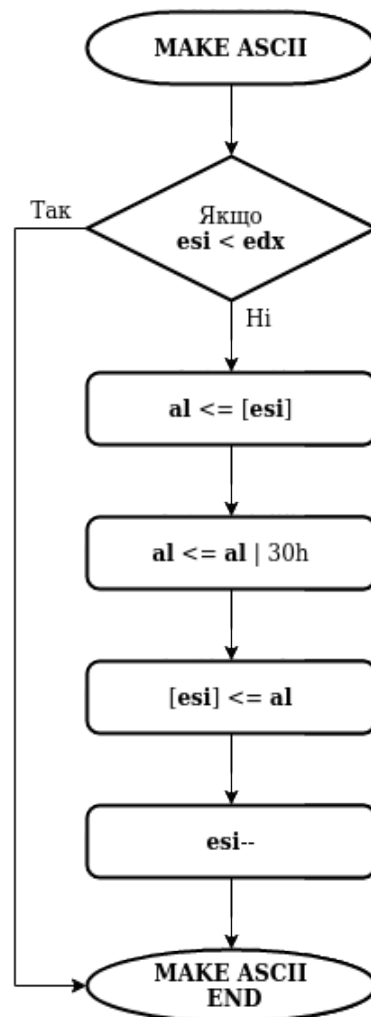
Registers

EBX	=	7EFD030
ECX	=	00000004
EDX	=	000E6000
ESI	=	000E6004
EDI	=	000E6005
EIP	=	000E3606
ESP	=	004AFF64
EBP	=	004AFF78
EFL	=	00000203

OV = 0 UP = 0 EI = 1 PL = 0
ZR = 0 AC = 0 PE = 0
CY = 1

3) Підготовка результату (MAKE ASCII):

Блок-схема і вихідний код



```
make_ascii:
    cmp esi, edx                ; check if end is reached
    jb finish                  ; saving flag states is pointless here, finally

    mov al, BYTE PTR [esi]     ; get value

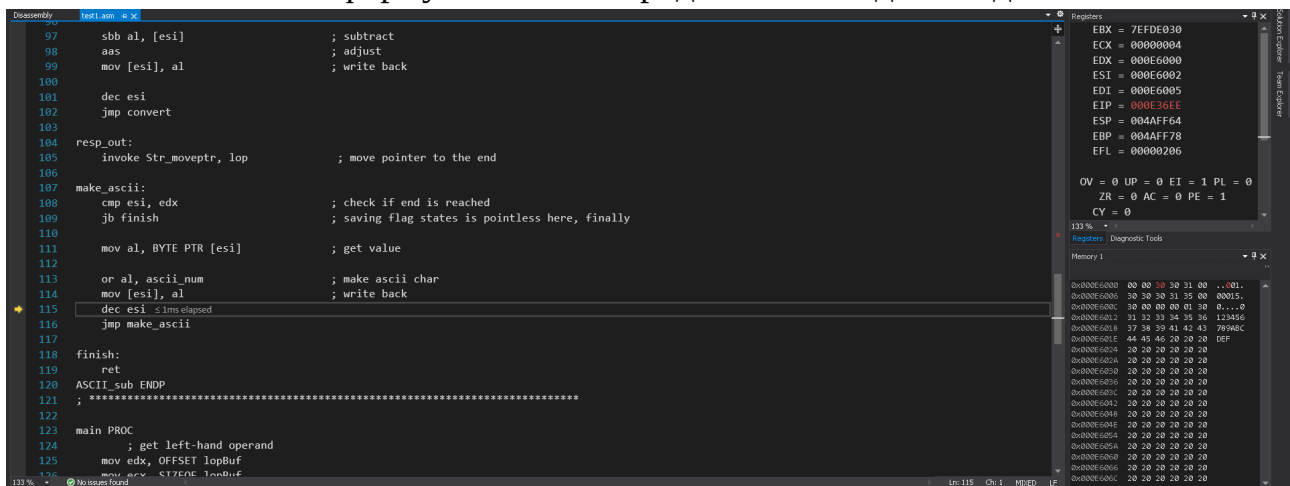
    or al, ascii_num           ; make ascii char
    mov [esi], al              ; write back
    dec esi
    jmp make_ascii

finish:
    ret
ASCII sub ENDP
```

На цьому етапі вважається, що у результаті дій виконаних раніше був отриманий коректний результат.

Останній етап досить прямолінійний: перевіряємо чи не перетнулися адреси вказівників, завантажуюмо значення за адресою, що збережена в **esi** у **al**, виконуємо операцію АБО між завантаженим значенням і константою *ascii_num*, що рівна 30₁₆ і завантажуюмо результат назад за його адресою.

Таким чином формується ASCII рядок готовий до виведення в консоль.



The screenshot shows a debugger window with assembly code on the left and registers/memory on the right. The assembly code is for a function named `ASCII_sub`. It starts with `sbb al, [esi]` (subtract with borrow), followed by `aas` (ASCII adjust after subtract), and `mov [esi], al` (write back). It then decrements `esi` and jumps to `convert`. The `convert` label points to `resp_out`, which calls `invoke Str_moveptr, lop` to move the pointer to the end. The `make_ascii` label follows, where it compares `esi` and `edx` to check if the end is reached. If not below (`jb finish`), it gets the value at `[esi]` into `al`, performs a bitwise OR with `ascii_num` (making it an ASCII char), and writes it back to `[esi]`. It then decrements `esi` and jumps back to `make_ascii`. The `finish` label contains `ret`. The `ASCII_sub` procedure ends with `ENDP`. The `main` procedure starts with a comment to get the left-hand operand, moves `edx` to `OFFSET lopBuf`, and then moves `eax` to `CTEONE lopBuf`. The registers window on the right shows the current state: `EBX = 7EFD0030`, `ECX = 00000004`, `EDX = 000E6000`, `ESI = 000E6002`, `EDI = 000E6005`, `EIP = 000E36EE`, `ESP = 004AFF64`, `EBP = 004AFF78`, `EFL = 00000206`. The memory window shows the contents of memory starting at `000E6000`.

```
Disassembly: test.asm - x
97 sbb al, [esi] ; subtract
98 aas ; adjust
99 mov [esi], al ; write back
100
101 dec esi
102 jmp convert
103
104 resp_out:
105 invoke Str_moveptr, lop ; move pointer to the end
106
107 make_ascii:
108 cmp esi, edx ; check if end is reached
109 jnb finish ; saving flag states is pointless here, finally
110
111 mov al, BYTE PTR [esi] ; get value
112
113 or al, ascii_num ; make ascii char
114 mov [esi], al ; write back
115
116 dec esi ; 1ms elapsed
117 jmp make_ascii
118
119 finish:
120 ret
121 ASCII_sub ENDP
122 ; *****
123
124 main PROC
125 ; get left-hand operand
126 mov edx, OFFSET lopBuf
127 mov eax, CTEONE lopBuf
128
```

Registers:

- EBX = 7EFD0030
- ECX = 00000004
- EDX = 000E6000
- ESI = 000E6002
- EDI = 000E6005
- EIP = 000E36EE
- ESP = 004AFF64
- EBP = 004AFF78
- EFL = 00000206

OV = 0 UP = 0 EI = 1 PL = 0
ZR = 0 AC = 0 PE = 1
CY = 0

Memory:

Address	Value
000E6000	00 00 00 00 ...001
000E6001	30 30 30 31 35 00 00015
000E6002	30 00 00 00 01 30 0...0
000E6003	11 12 13 14 15 16 17346
000E6004	37 38 39 41 42 43 789ABC
000E6005	44 45 46 20 20 20 DEF
000E6006	20 20 20 20 20 20
000E6007	20 20 20 20 20 20
000E6008	20 20 20 20 20 20
000E6009	20 20 20 20 20 20
000E600A	20 20 20 20 20 20
000E600B	20 20 20 20 20 20
000E600C	20 20 20 20 20 20
000E600D	20 20 20 20 20 20
000E600E	20 20 20 20 20 20
000E600F	20 20 20 20 20 20
000E6010	20 20 20 20 20 20
000E6011	20 20 20 20 20 20
000E6012	20 20 20 20 20 20
000E6013	20 20 20 20 20 20
000E6014	20 20 20 20 20 20
000E6015	20 20 20 20 20 20
000E6016	20 20 20 20 20 20
000E6017	20 20 20 20 20 20
000E6018	20 20 20 20 20 20
000E6019	20 20 20 20 20 20
000E601A	20 20 20 20 20 20
000E601B	20 20 20 20 20 20
000E601C	20 20 20 20 20 20
000E601D	20 20 20 20 20 20
000E601E	20 20 20 20 20 20
000E601F	20 20 20 20 20 20

Висновок

Результатом виконання даної лабораторної роботи стала програма, що може виконувати віднімання двох п'ятирозрядних чисел (за необхідності розрядність може бути збільшена) і виводити результат у консольне вікно для відображення. Для написання програми були використані процедури описані в бібліотеці *Irvine32* та написані власноруч для можливості їх застосування при написанні майбутніх програм.

Вдалося познайомитися більш детально з алгоритмами виклику процедур, директивами та командами, що дозволяють часто значно полегшити написання і подальшу оптимізацію програм, ознайомилися з механізмами роботи стеку, командами, що виконують завантаження на або в стек,

Також ознайомилися з різними форматами представлення, збереження і обробки числових даних такими як, наприклад, BCD (binary-coded decimal).

Вихідний код програм доступний за посиланням:

[\[===\]](#)

Повний вихідний код програми: