Ⓟ1 Vlad Bogdan-Tudor,
917

First, we will write the procedure which replaces the C-th of N by F. We will consider that we have N in EAX. In EDX we will first put the value OFh. We will shift EDX to the left C times. Then, we will simply OR EAX with EDX. This will set all 4 bits from the C-th nibble to 1, thus making the C-th nibble F. We will make this procedure accesible to the main program by making it global.

⊛ This is code from module.asm:

```
bits 32
global swap_to_f
segment data use32


segment code use32
  swap_to_f:
      mov eax, [esp+4]
      ; Moved the dword that needs to be modified in EAX
      mov ecx, [esp+8]
      ; ECX now has the value C
```

```
mov edx, 0Fh
; EDX will be used to change the c-th byte to F
; Now we have to shift EDX to the left c times,
; where c is now in ECX                  (by 4 bits)
; If ECX is 0, we don't start the loop; it
; means that the 0-th byte has to be set
; to F

cmp ecx, 0
je .no_shifting
cmp ecx, 8   ; if ECX is 8, we don't shift
je .return
.shift_left:
    shl edx, 4
loop .shift_left

.no_shifting
; All that is left is to apply the OR operator
; on EAX with EDX
or eax, edx
.return:
ret
```

P3

Vlad Bogdan-Tudor,
917

Now we write the code in the main module. We will take each dword in S, find the sum of the high word with the low word, and then count the longest sequence of bits of one by consecutive shifts. After this, we have our C and we can apply the procedure swap-to-f. We will make this procedure usable from main.asm by telling the assembler that it is a procedure from another file. For this, we will use <global>. Also, we will pass parameters to this procedure by pushing them on the stack.

```
bits 32
global start
extern exit, swap_to_f, printf
import exit msvcrt.dll
import printf msvcrt.dll
segment data use 32
        s dd 65534, -4473007, 15, -1, 2004322440
        len equ ($ - s)/4
        r resd len
        max_ones resd 1
        hexa_output db "%x ",0
```

```
segment code use32
    ; We'll use ESi to iterate through S
    mov ESi, 0
    .for-every-dword-in-s:
        mov eax, [s+esi*4]
        mov ecx, eax
        shr ecx, 16
        add ax, cx
        ; With the above code we've found the sum
        ; of the low word with the high word
        ; Now we need the longest sequence of ones.
        mov dword [max-ones], 0
        ; max-ones will have the number of ones in the
        ; longest seq. of consecutive ones
    mov ebx, 0
    ; EBX will keep track of the current seq.
    ; of consecutive ones
    mov ecx, 16 ; We shift AX 16 times

    .shift-ax:
        mov edx, 0
        shl ax, 1
        adc edx, 0
        cmp edx, 0
        je .reset
            ; If we get here => we continue to count
            ; ones
            add ebx, edx
            jmp .next
       .reset:
            cmp ebx, dword [max_ones]
            jb .ebx_to_zero
                ; We get here => new longest sequence
                mov dword [max_ones], ebx
          .ebx_to_zero:
               mov ebx, 0
    .next:
    loop .shift_ax

; With the above code, we've found in max_ones
; the number of ones in the maximum seq of
; cons. ones
; Now we need to divide max_ones by 2
; This can be done easily by shifting the
; value to the right by one bit.
mov ebx, dword [max_ones]
shr ebx, 1
; Now we have our "C" in EBX
mov eax, [s+esi*4]
; Now we have our current "N" in EAX
; We just have to call our written procedure.
; BUT! only if ebx is NOT 8 (not necessary since
; the procedure already takes care of this)
cmp ebx, 8
je .dont_call
    push ebx
    push eax
    call swap_to_f
    add esp, 2*4
   ; With the above code we've called our procedure
   ; and now have the desired value in EAX
.dont_call:
    mov [r+esi], eax
   ; We saved the value in R
```

```
    ; Now we can move on to the next dword
    ; in S

    inc esi
    cmp esi, len
jb .for-every-dword-in-s

; At this point, we have our string R and
; we just need to print it

 move esi, 0
 .for-every-dword-in-r:
      ; We push each dword in R on the stack and
      ; call printf to print it

      push dword [r+ esi*4]
      push hexa_output
      call [printf]
      add esp, 2*4

      inc esi
      cmp esi; len
jb .for_every_dword_in_r
push dword 0
call [exit]
```