Vlad Bogdan-Tudor,
917

This is the code for the module b.asm. It will have the 2 necessary procedures: procedura_1 and procedura_2.

In procedura_1, to get the mirrored hexadecimal value of a dword we will take each nibble from right to left from the dword, shift the dword to the right by 4 and save that nible in another dword. We will repeat this 8 times. In procedura_2 we have a simple print statement.

We will make these procedures accessible to a.asm by making them global. Then we will be able to import them. Also, the dword that needs to be passed to the 2 procedures will be passed by pushing it on the stack.

ii. asm code:

```
bits 32
extern printf
global procedura_1, procedura_2
import printf msvcrt.dll
segment data use32
        hexa_output db "%x, ", 0



segment code use32
    procedura_1:

        mov edx, [esp+4]
        mov eax, 0 ; EAX will have the mirrored dword
        mov ecx, 8 ; We repeat 8 times

      .for_every_nibble:
            shl eax, 4
            add al, dl
            ; With the above we keep adding nibbles
            ; to the end of the mirrored dword,
            ; multiplying it by 16 each time
           shr edx, 4 ; We get rid of the recently added
                                                  nibble
     loop .for_every_nibble
    ; now EAX has the mirrored dword
    ret
```

Vlad Bogdan-Tudor,
917

This is still code from b.asm:

```
procedura_2:
        mov eax, [esp + 4]
        ; We need to print the dword from eax now
        push eax
        push dword hexa_output
        call [printf]
        add esp, 2*4
        ret
```

Now we will write the code from a.asm.
It will have to call the 2 procedures from b.asm,
so we will import them using 'extern'!

```
bits 32
extern procedura_1, procedure_2
global start


segment data use32
        s1 dd 0FFFFFFF0h, 0AABBCCDDh, 12343210h, 3h,
0FFFFE000h, 17h, 0AFFEFFE0h
        len equ ($-s1)/4
        s2 resd len ; The resulting sequence
        s2_len resd 1 ; The length of <s2>



segment code use32
        mov esi,0 ; We'll use ESi to iterate <s1>
        mov edi,0 ; EDi - index for<s2>
        .for_every_dword_in_s1:
            mov eax, [s1+4*ESi]
            ; Now we have the current dword in EAX
            ; We need to check that it is negative and
            ; divisible by 16
```

This is the second part of the code from a.asm :

```
    cmp eax, 0
    jge .dont_save
        ; If we're here => we check if it's a multiple of 16
      mov bl, al
      and bl, 0Fh
      ; If the last digit (in hexa) of a number is 0,
      ; then that number is a multiple of 16
     cmp bl, 0
     jne .dont_save
         ; We got here => we need to save the
         ; mirrored value of the dword in EAX
         push eax
         call procedure_1
         add esp, 1*4

         ; Now procedure_1 returned the mirrored value
         ; in EAX. We save it in <s2>
        mov [s2+edi*4], eax
        inc edi
.dont_save:
   inc esi       ; We go to the next dword and if we
   cmp esi, len  ; reached the end we stop
jb .for_every_dword_in_s1
```

This is the 3rd part of the code from a.asm:

```
; Now we have in <s2> the mirrored values (in hexa)
; of the negative numbers divisible by 16 from <s1>
; We need to sort it and print it. Also in esi we
; have the length of <s2>
 mov [s2-len], edi

 ; We will use the selection sort algorithm. For this
 ; we will use ESI and EDI as the 2 indices
 mov esi, 0
mov edi, 1
.for-every-dword-in-s2:
    .for-every-dword-to-the-right:
        mov eax, [s2+esi*4]
       mov ebx, [s2+edi*4]
        ; According to the given example, we need to
       ; sort the elements unsigned
       cmp eax, ebx
       jna .dont-swap
            ; If we get here => eax > ebx, so we need
            ; to swap them
            mov ecx, eax
            mov eax, ebx
            mov ebx, ecx
```

This is the 4th part of the code from a.asm:

```
    .dont_swap:
        ;If the 2 dwords don't need to be swapped,
        ;don't do anything
    inc edi
    cmp edi, [s2_len]
jb .for_every_dword_to_the_right
    ;With the above 3 lines we keep ESI fixed and we
    ;go over all elements left in the array<s2>. Again,
    ;it's just a basic Selection Sort algorithm.
    ;This is the inner loop from Selection Sort
    inc esi
    cmp esi, [s2_len]
jb .for_every_dword_in_s2
    ;We move ESI so that every element in<s2>is
    ;checked. This is the outer loop from Selection Sort

;Now we have <s2> sorted. We just need to print it.
mov esi, 0
 .final_loop:
     mov eax, [s2+esi*4]
     push eax
     call procedure_2
     add esp, 1*4
```

This is the 5th (and final) part of the code from a.asm:

```
        ; We increase the index and if we reached the end we exit the loop
        inc esi
        cmp esi, [s2_len]
jb .final_loop

ret
```