# Cheatsheet IOCLA 2023

Autor: *Vlad Chira*

### **Macro-urile se pot pune intr-un fisier .mac si se includ prin directiva `%include "utils.mac"`**

## 1. Afisari

### Afisari tipuri de date de baza (atentie la format specifier!!)

Observatie: merge si pe string uri cu format specifier `%s\n\x0`.
Pentru short trebuie folosit %hi sau %hu

```
section .data
    myString: db "Hello, World!", 0
    X db 5
    Y dd -80
section .text
    extern printf
main:
    PRINTF32 `%s\n\x0`, myString ;  afisare string
    PRINTF32 `%hhu\n\x0`, [X]    ;  afisare unsigned byte
    PRINTF32 `%d\n\x0`, [Y]      ;  afisare int
```

```
%macro PRINTF32 1-*
    pushf
    pushad
    jmp     %%endstr
%%str:       db      %1
%%endstr:
%rep  %0 - 1
%rotate -1
    push    dword %1
%endrep
    push %%str
    call printf
    add esp, 4*%0
    popad
    popf
%endmacro
```

### Afisare vector de orice dimensiune (atentie la format specifier!!)

Macro ul este pe baza lui PRINTF32 definit prin lab uri si teme. Este generic, merge pe vectori de orice dimensiune si tip. A nu se folosi pe string uri decat in cazuri exceptionale (ex: doar primele n caractere).

Utilizare:

```
section .data
    array dd 143, 210, 722, -40, 31, 59

PRINT_ARRAY array, 6, 4, `%d \x0`    ; pointer vector, lungime vector,
dimensiune element, format specifier element
```

```
; print array at address %1 with length %2
; each element has size and %3 is printed with format specifier %4
%macro PRINT_ARRAY 4
    pusha
    mov eax, %1 ; load the array
    mov ebx, %3 ; load the size of each element
    mov edx, %2 ; load the length of the array

    mov ecx, 0
%%start_iteration:
    cmp ecx, edx
    jge %%end_loop

    mov esi, [eax]
    PRINTF32 %4, esi

    add eax, ebx
    inc ecx
    jmp %%start_iteration
%%end_loop:
    PRINTF32 `%c\n\x0`, 0
    popa
```

# 2. Citiri de la tastatura

## Citirea tipurilor de date de baza

Foloseste scanf, trebuie pus ca extern in fisierul .asm.
Observatie: merge si pe string uri dar citeste DOAR PANA LA PRIMUL SPATIU. Pentru short trebuie folosit %hi sau %hu.
!! Atentie. NU se poate citi **direct** intr-un registru. Se foloseste o variabila declarata in .data
Utilizare:

```
SCANF32 `%hhu\x0`, x    ; citirea unui byte
SCANF32 `%d%d\x0`, x, y ; citirea a doi intregi
```

```
; reads numbers from stdin using scanf
; %1 - format specifier for ALL numbers (ex `%d%d%d\x0`)
; %2 - address to place the first number into
; %3 - address to place the second number into
; ...
; WARNING: scanf CANNOT read directly into a register!!
; use a variable declared in .data to store the number
%macro SCANF32 1-*
    pushf
    pushad
    jmp     %%endstr
%%str:       db      %1
%%endstr:
%rep  %0 - 1
%rotate -1
    push    %1
%endrep
    push %%str
    call scanf
    add esp, 4*%0
    popad
    popf
%endmacro
```

## Citirea sirurilor de caractere

Foloseste functia gets, trebuie declarata ca extern in fisierul .asm.
Functia este considerata 'periculoasa' dar este usor de folosit.
Utilizare:

```
section .data
    str: db 101 dup(0) ; cream un buffer suficient de mare

section .text
    extern gets
main:
    READ_STRING str
```

```
; reads a string using gets
; your compiler might complain
; just ignore it
%macro READ_STRING 1
    push eax
    mov eax, str
    push eax
    call gets
    add esp, 4
```

```
    pop eax
%endmacro
```

## 3. Interschimbare a doua variabile / swap 32biti

Face swap corect doar pentru variabile de 32 de biti.
Daca acestea se regasesc direct in registre, atunci `xchg eax, ebx` este suficient. Daca registrele sunt pointeri catre variabile, atunci:

```
; swaps two variables X and Y such that
; if at first: %1 -> X and %2 -> Y
; after:  %1 -> Y and %2 -> X
%macro SWAP_VARS 2
    ; Swap the values using the stack
    push dword [%1]
    push dword [%2]
    pop dword [%1]
    pop dword [%2]
%endmacro
```

Utilizare:

```
mov eax, X ; pointer la Y
mov ebx, Y ; pointer la X
SWAP_VARS eax, ebx
```

## 4. Conversia string -> numar intr o anume baza

Se bazeaza pe functia strtoul, se declara ca extern in fisierul .asm Pentru varianta pentru numar cu semn, se foloseste strol in loc

```
; %1 the string with the number
; %2 the base of the number
; result is stored in EAX
%macro STRING2INT 2
    push ebx
    push ecx
    push edx
    push esi
    push edi

    push %2  ; the base
    push 0   ; NULL
    push %1 ; the string
    call strtoul
    add esp, 12
```

```
    pop edi
    pop esi
    pop edx
    pop ecx
    pop ebx
%endmacro
```

Utilizare:

```
section .data
    str: db "0x0000000F", 0
section .text
    extern printf
    extern strtoul
main:
    STRING2INT str, 16
    PRINTF32 `%d\n\x0`, eax
    ret
```

## 5. Afisare reprezentare binara a unui numar

Afiseaza toti cei 32 de biti ai numarului.
Pentru afisarea unui nr pe 2 sau 1 byte vezi mai jos

```
; prints the 32bit binary representation
; of the number %1
%macro PRINT_BINARY32 1
    pusha
    mov eax, %1
    mov ecx, 0
%%bit_loop:
    cmp ecx, 32
    je %%end_loop

    shl eax, 1 ; shift the leftmost bit out
    setc dl ; shl sets CF to this bit so retrieve it
    movzx edx, dl ; perform zero extend to turn it to 32bit
    PRINTF32 `%d\x0`, edx ; print it with macro

    inc ecx
    jmp %%bit_loop

%%end_loop:
    PRINTF32 `%c\n\x0`, 0
    popa
%endmacro
```

Utilizare:

```
PRINT_BINARY32 696969    ; afiseaza 00000000000010101010001010001001
```

## 6. Afisare a cei mai semnificativi n biti ai unui nr pe 32 de biti

Utilizare:

```
mov eax, 20
PRINT_NBITS32 696969696, 5 ; afiseaza 00101
```

Se poate folosi pt a afisa numere pe 1 sau un byte astfel

```
; pentru un byte stocat in registrul al
mov al, 20           ; punem numarul dorit
movzx eax, al        ; il transformam pe 32 biti
shl eax, 24          ; scoatem bitii in plus (32-8)
PRINT_NBITS32 eax, 8 ; afisam doar cei 8 biti
```

Pentru un numar pe 2 bytes stocat in ax

```
mov ax, 20000         ; punem nr dorit
movzx eax, ax         ; il transformam pe 32 biti
shl eax, 16           ; scoatem bitii in plus (32-16)
PRINT_NBITS32 eax, 16 ; afisam doar cei 16 biti
```

```
; prints the most significant
; %2 bits of the 32bit binary representation
; of the number %1
%macro PRINT_NBITS32 2
    pusha
    mov eax, %1
    mov ecx, 0
%%bit_loop:
    cmp ecx, %2
    je %%end_loop

    shl eax, 1 ; shift the leftmost bit out
    setc dl ; shl sets CF to this bit so retrieve it
    movzx edx, dl ; perform zero extend to turn it to 32bit
    PRINTF32 `%d\x0`, edx ; print it with macro

    inc ecx
    jmp %%bit_loop
```

```
%%end_loop:
    PRINTF32 `%c\n\x0`, 0
    popa
%endmacro
```

## 7. Extragere al n-ulea cel mai semnificativ octet (MSB) dintr un numar pe 32biti

*n<=4* Important: functioneaza doar pentru numere pe 32 de biti.
Pentru numar de 2 bytes (dar pus intr-un registru de 32 de biti) se poate shifta la stanga cu 16 biti numarul si apoi apela acest macro.
Utilizare:

```
NTH_MSB 0xDEADBEEF, 2
PRINTF32 `0x%X\n\x0`, eax   ; afiseaza 0xAD
```

```
; returns the %2 (<=4) most significant BYTE
; of the number %1
; result is stored in EAX
%macro NTH_MSB 2
    push ebx
    push ecx
    push esi

    xor esi, esi
    mov eax, %1 ; number
    mov ecx, %2 ; extract the nth MSB
    cmp ecx, 4
    jg %%end_byte_loop ; oops

%%byte_loop:
    cmp ecx, 0
    je %%end_byte_loop
    rol eax, 8
    mov ebx, eax  ; backup eax
    and eax, 0xFF ; apply bit mask

    mov esi, eax  ; store the byte
    mov eax, ebx  ; restore the number

    dec ecx
    jmp %%byte_loop

%%end_byte_loop:
    mov eax, esi

    pop esi
```

```
    pop ecx
    pop ebx
%endmacro
```