

UNIVERSITATEA DE STAT DIN MOLDOVA

**FACULTATEA MATEMATICĂ ȘI INFORMATICĂ
DEPARTAMENTUL INFORMATICĂ**

CEMÎRTAN CRISTIAN

Lucrarea individuală nr. 7
la disciplina *Arhitectura Calculatoarelor și Limbaje de Asamblare*

Coordonator: Sturza Greta, lector universitar

Chișinău, 2021

Cuprins

Varianta „a”	3
Sarcină	3
Cod sursă	3
Rezultat	4
Varianta „b”	5
Sarcină	5
Cod sursă	5
Rezultat	6
Concluzie	8
Anexe	9

Varianta „a”

Sarcină

Calculează valoarea unei expresii variabilele cărora sunt numere fără semn definite în program. Rezultatul nu se va afișa pe ecran, dar se va arăta în Turbo Debugger.

- $80 + (45 - 5) \cdot 3 + 45 / 9$

Cod sursă

```
COMMENT *
    Lucrare individuala nr. 7a, varianta 3
    Copyright Cemirtan Cristian 2021
    Grupa I 2101
*

.MODEL small
.STACK

.DATA
a DB 80
b DB 45
c DB 5
d DB 3
e DB 45
f DB 9

.CODE
; init
    mov dx, @data
    mov ds, dx
    xor ax, ax

; 45 - 5
    mov al, b
    sub al, c

; (45 - 5) * 3
    mov cl, d
    mul cl

; 80 + (45 - 5) * 3
    add al, a
    mov dl, al

; 45 / 9
    mov al, e
    mov cl, f
    div cl
```

```

; 80 + (45 - 5) * 3 + 45 / 9
    add al, dl
    int 03h

; iesire
    mov ax, 4C00h
    int 21h

END

```

Rezultat

File Edit View Run Breakpoints Data Options Window Help										READY
[CPU 80486]										1
cs:0000	BAAA4E	mov	dx,4EAA							ax 00CD c=0
cs:0003	8EDA	mov	ds,dx							bx 0000 z=0
cs:0005	33C0	xor	ax,ax							cx 0009 s=1
cs:0007	A00D00	mov	al,[000D]							dx 4EC8 o=0
cs:000A	2A060E00	sub	al,[000E]							si 0000 p=0
cs:000E	8A0E0F00	mov	cl,[000F]							di 0000 a=0
cs:0012	F6E1	mul	cl							bp 0000 i=1
cs:0014	02060C00	add	al,[000C]							sp 0400 d=0
cs:0018	8AD0	mov	dl,al							ds 4EAA
cs:001A	A01000	mov	al,[0010]							es 4E98
cs:001D	8A0E1100	mov	cl,[0011]							ss 4EAC
cs:0021	F6F1	div	cl							cs 4EA8
cs:0023	02C2	add	al,dl							ip 0025
cs:0025	CC	int	03							
cs:0026	BB004C	mov	ax,4C00							
es:0000	CD 20 FF 9F 00 9A F0 FE	=	f Ü							ss:0408 0000
es:0008	1D F0 E0 01 7F 1B AA 01									ss:0406 0000
es:0010	7F 1B 89 02 DA 15 4C 07									ss:0404 0000
es:0018	01 04 01 02 03 FF FF FF									ss:0402 0000
es:0020	FF FF FF FF FF FF FF									ss:0400 0000

Figura 1. Rezultatul stocat în registrul ax. Valoarea 0CDh este echivalentă cu 205.
Întreruperea 03h dă știre depanatorului să oprească după invocarea sa.

Varianta „b”

Sarcină

Calcularea valorii unei expresii valorile variabilelor căroră sunt numere cu semn introduse de la tastatură. Rezultatul se va afișa pe ecran.

- $(10 \cdot a - 2 \cdot b) \cdot c - d$

Cod sursă

```
COMMENT *
    Lucrare individuala nr. 7b, varianta 3
    Copyright Cemirtan Cristian 2021
    Grupa I 2101
*

INCLUDE stdio.inc

.MODEL small
.386
.STACK

.DATA
txt DB 'Introduceti a, b, c si d:', 0Dh, 0Ah, '$'
fmt1 DB '%b%b%d%d$'
fmt2 DB 'Rezultatul este: %d.$'
a DB ?
b DB ?
c DW ?
d DW ?

.CODE
; init
    .STARTUP

; interupere speciala pentru revarsari numerice
    enable_overflow_check

; afisare text
    puts txt

; argumente pentru scanf
    push OFFSET d OFFSET c OFFSET b OFFSET a
    lea si, fmt1
    call scanf
    add sp, 8

; se incarca variabilele
    movsx eax, a
```

```

        movsx ebx, b
        movsx ecx, c
        movsx edx, d

; 10 * a - 2 * b
        add eax, eax
        add ebx, ebx
        lea eax, [eax + 4 * eax]
        sub eax, ebx

; (10 * a - 2 * b) * c
        imul eax, ecx

; (10 * a - 2 * b) * c - d
        sub eax, edx

; verificam daca rezultatul incapa intr-un cuvant
        movsx edx, ax
        cmp edx, eax
        je afisare
        int 04h

afisare:
; afisare text
        push ax
        lea si, fmt2
        call printf
        add sp, 2

; iesire
        .EXIT 0
END

```

Rezultat

```

E:\>tasm i7b
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file:   i7b.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  449k

E:\>tlink i7b stdio
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

E:\>i7b
Introduceti a, b, c si d:
5
-30
12
7
Rezultatul este: 1313.
E:\>

```

Figura 2. Programul executat cu succes.

```
E:\>i7b
Introduceti a, b, c si d:
-128
Overflow
E:\>i7b
Introduceti a, b, c si d:
100
100
100
100
Overflow
E:\>
```

Figura 3. Protecție împotriva revărsărilor numerice.

Concluzie

Pe parcursul realizării a programelor pentru varianta „b”, am utilizat instrucțiunile pe 32-biți, pentru a beneficia de instrucțiuni performante ce reduc major numărul de cicluri parcurse de microprocesor, comparat cu 16-biți.

De exemplu, în biblioteca `stdio`, ca să transform un număr din ASCII în nativ, am utilizat instrucțiunile `lea` în loc de `imul`, ca să înmulțesc cu 10. Conform documentațiilor ce prevăd procesoarele arhaice x86¹, pe microprocesorul x386, instrucțiunea `lea` consumă 2 cicluri², dar `imul` – între 9 și 38³. În cazul împărțirii la 10, adică din nativ în ASCII, în loc să utilizez instrucțiunea `idiv`, care consumă 43 de cicluri⁴, am utilizat `imul`. De aceea, procesul se transformă în înmulțirea cu numărul real 0.1, reprezentat ca 0CCCCdh⁵, și deplasarea la dreapta cu 19 biți, aproximativ $16 + \log_2 10$.

Analizând formele de adresare pe 32-biți⁶, instrucțiunea `lea` are capacitatea de înmulțire a unui registru doar cu numerele 2, 4 și 8. Adicional, se poate de incrementat coeficientul cu 1 prin adunarea cu registrului respectiv.

Instrucțiunea `lea`, deși este utilizată pentru a calcula adresele relative, nu efectuează niciun acces la memorie, deci poate fi folosită liber în scopuri aritmetice.

Pentru a ameliora procesul de colectare și stocare a datelor, am creat procedurile `printf` și `scanf`, ce permite la o afișare formatată pe ecran. Procedurile iau un număr variabil de argumente de pe stivă, dar se eliberează manual de programator, după apel.

Protecția împotriva revărsărilor numerice, realizată prin înlocuirea comportamentului `int 04h` de funcția `25h` din `int 21h`, este crucială pentru integritatea programului. În caz de revărsări numerice, programul se termină cu motiv, în loc să dea rezultate eronate, care pot cauza un comportament nedorit a programului.

Se anexează coduri sursă a bibliotecii `stdio`.

¹ Zack Smith, The Intel 8086 / 8088 / 80186 / 80286 / 80386 / 80486 Instruction Set, <https://zsmith.co/intel.php>

² Zack Smith, L Instructions, https://zsmith.co/intel_l.php#lea

³ Zack Smith, I Instructions, https://zsmith.co/intel_i.php#imul

⁴ Zack Smith, I Instructions, https://zsmith.co/intel_i.php#idiv

⁵ Félix Cloutier, How to compilers optimize divisions?, GitHub, 2017, _

<https://zneak.github.io/fcd/2017/02/19/divisions.html>

⁶ Wikipedia, x86 – Addressing modes, https://en.wikipedia.org/wiki/X86#Addressing_modes

Anexe

stdio.inc

```
COMMENT *
    Biblioteca stdio pentru x86
    !Implementata partiala!
    Copyright Cemirtan Cristian 2021
*

EXTRN digitcount : PROC
EXTRN wtos : PROC
EXTRN stow : PROC
EXTRN strcpy : PROC
EXTRN strlen : PROC
EXTRN printf : PROC
EXTRN sprintf : PROC
EXTRN scanf : PROC

enable_overflow_check MACRO
    .DATA
        __cemirtan_stdio_overflow_text DB 'Overflow', 7, '$'

        __cemirtan_stdio_overflow_exit LABEL
        mov ah, 09h
        lea dx, __cemirtan_stdio_overflow_text
        int 21h
        .EXIT 1

    .CODE
        push ax dx
        mov ax, 2504h
        lea dx, __cemirtan_stdio_overflow_exit
        int 21h
        pop dx ax
ENDM enable_overflow_check

puts MACRO s
    push ax dx

    mov ah, 09h
    lea dx, s
    int 21h

    pop dx ax
ENDM puts

putc MACRO c
    push ax dx

    mov ah, 02h
```

```

        mov dl, c
        int 21h

        pop dx ax
ENDM putc

putnl MACRO
    push ax dx

    mov ah, 02h

    mov dl, 0Dh
    int 21h

    mov dl, 0Ah
    int 21h

    pop dx ax
ENDM putnl

gets MACRO s
    push ax dx

    mov ah, 0Ah
    mov dx, s
    int 21h

    pop dx ax
ENDM gets

getc MACRO c
    push ax

    mov ah, 01h
    int 21h

    mov BYTE PTR [c], al
    pop ax
ENDM getc

```

stdio.asm

```
COMMENT *
    Biblioteca stdio pentru x86
    !Implementata partiala!
    Copyright Cemirtan Cristian 2021
*

.MODEL tiny
.386

buffer_size EQU 128

; se incarca adresa parametrului
arg_sca MACRO r
    mov r, ss:[bx]
    add bx, 2
ENDM arg_sca

; se incarca in valoarea parametrului
arg_mov MACRO r
    LOCAL ignore
    mov di, ss:[bx]
    cmp di, -1
    je ignore
    mov [di], r
    add bx, 2
ignore:
ENDM arg_sca

.CODE
PUBLIC digitcount
PUBLIC wtos
PUBLIC stow
PUBLIC strcpy
PUBLIC strlen
PUBLIC sprintf
PUBLIC printf
PUBLIC scanf

; nr. de cifre a unui numar fara semn
digitcount PROC
    cmp ax, 10
    jge digitcount$if_1

    mov ax, 1
    ret

digitcount$if_1:
    cmp ax, 100
    jge digitcount$if_2

    mov ax, 2
    ret
```

```

digitcount$if_2:
    cmp ax, 1000
    jge digitcount$if_3

    mov ax, 3
    ret

digitcount$if_3:
    cmp ax, 10000
    jge digitcount$finish

    mov ax, 4
    ret

digitcount$finish:
    mov ax, 5
    ret
digitcount ENDP

; intrare ax - numar, di - sir (lungimea de cel putin 7 octeti)
; iesire ax - nr. de caractere scrise, exclusiv '$'
wtos PROC
    push ecx edx di

    test ax, ax
    jns wtos$init

    mov BYTE PTR [di], '-'
    neg ax
    inc di

; initializare
wtos$init:
    movzx edx, ax
    call digitcount
    add di, ax
    mov eax, edx

    mov BYTE PTR [di], '$'
    push di

wtos$loop:
    mov edx, eax

; impartire la 10
    imul eax, 0CCCDh
    shr eax, 19

; inmultirea la 10 si aflarea restului
    lea ecx, [eax + 4 * eax]
    lea ecx, [2 * ecx - '0']
    sub edx, ecx

    dec di

```

```

        mov [di], dl

        test eax, eax
        jnz wtos$loop

wtos$finish:
        pop ax di edx ecx
        sub ax, di
        ret
wtos ENDP

; intrare cx - nr. bitului superior (max. 15), si - sir
; iesire (e)ax - numar
; apeleaza intreruperea 04h in caz de revarsare numerica
stow PROC
        push ecx ebx edx esi

; bitmask
        and cl, 15
        mov ebx, -1
        shl ebx, cl
        mov ecx, ebx

; initializare
        xor eax, eax
        xor ebx, ebx
        xor edx, edx

; testare daca e negativ
        cmp BYTE PTR [si], '-'
        sete dl
        add si, dx

stow$loop:
        movzx ebx, BYTE PTR [si]

        mov dh, bl
        xor dh, 30h

        cmp dh, 9
        jg stow$finish

; * 10
        lea eax, [eax + 4 * eax]
        lea eax, [ebx + 2 * eax - '0']

; testare daca nu depaseste cx
        test eax, ecx
        jnz stow$overflow
        inc si
        jmp stow$loop

stow$finish:
; negativ daca minus
        test dl, dl
        jz stow$finish_2

```

```

        neg eax

stow$finish_2:
        pop esi edx ebx ecx
        ret

stow$overflow:
        int 04h
        jmp stow$finish_2
stow ENDP

; si - sursa, di - destinatie
; ax - nr. de caractere copiate
strcpy PROC
        push dx si di

        strcpy$loop:
            mov dl, [si]

            cmp dl, '$'
            je strcpy$finish

            mov [di], dl

            inc si
            inc di

            jmp strcpy$loop

strcpy$finish:
        mov BYTE PTR [di], '$'
        mov ax, di
        pop di si dx
        sub ax, di
        ret
strcpy ENDP

; si - sursa
; ax - lungimea sirului
strlen PROC
        push si

        strlen$loop:
            cmp BYTE PTR [si], '$'
            je strlen$finish
            inc si
            jmp strlen$loop

strlen$finish:
        mov ax, si
        pop si
        sub ax, si
        ret
strlen ENDP

; si - sursa, di - destinatie, bx - varful stivei cu argumente

```

```

; ax - nr. de caractere scrise, in afara de '$'
sprintf_base PROC
    push dx si di

    sprintf_base$loop:
        mov dl, [si]
        inc si

        cmp dl, '$'
        je sprintf_base$ret

        cmp dl, '%'
        je sprintf_base$fmt

    sprintf_base$putc:
        mov [di], dl
        inc di
        jmp sprintf_base$loop

    sprintf_base$fmt:
        mov dl, [si]
        inc si

        cmp dl, '%'
        je sprintf_base$putc

        cmp dl, '$'
        je sprintf_base$putc

        cmp dl, 'c'
        je sprintf_base$fmt_char

        cmp dl, 'd'
        je sprintf_base$fmt_decimal

        cmp dl, 's'
        je sprintf_base$fmt_string

        cmp dl, 'n'
        jne sprintf_base$ret

    sprintf_base$fmt_count:
        mov static_var, si
        arg_sca dx

        mov si, sp
        mov si, [si]

        mov ax, di
        sub ax, si

        mov si, dx
        mov [si], ax

```

```

        mov si, static_var
        jmp sprintf_base$loop

sprintf_base$fmt_char:
        mov static_var, si
        arg_sca si
        mov dl, [si]
        mov si, static_var
        jmp sprintf_base$putc

sprintf_base$fmt_decimal:
        arg_sca ax
        call wtos
        add di, ax
        jmp sprintf_base$loop

sprintf_base$fmt_string:
        mov dx, si
        arg_sca si

        call strcpy
        add di, ax
        mov si, dx

        jmp sprintf_base$loop

sprintf_base$ret:
        mov BYTE PTR [di], '$'

        mov ax, di
        pop di
        sub ax, di

        pop si dx
        ret
sprintf_base ENDP

; si - sursa
sprintf PROC
        push si bx

        cmp si, -1
        jne sprintf$call

        lea si, buffer

sprintf$call:
        mov bx, sp
        add bx, 6

        call sprintf_base

        pop bx si
        ret
sprintf ENDP

```



```

; si - sursa
printf PROC
    push bx

    mov bx, sp
    add bx, 4

    push dx di

    lea di, buffer
    call sprintf_base

    mov bx, ax

    mov ah, 09h
    mov dx, di
    int 21h

    mov ax, bx

    pop di dx bx
    ret
printf ENDP

; si - sursa, bx - varful stivei cu argumente
; ax - nr. de argumente citite
scanf PROC
    push bx

    mov bx, sp
    add bx, 4

    push cx dx si di

    mov static_var, 0
    xor cx, cx

scanf$loop:
    mov dl, [si]
    inc si

    cmp dl, '$'
    je scanf$ret

    cmp dl, '%'
    je scanf$fmt

    jmp scanf$loop

scanf$fmt:
    inc static_var

    mov dl, [si]

```

```

    inc si

    cmp dl, 'c'
    je scanf$fmt_char

    cmp dl, 'b'
    je scanf$fmt_byte

    cmp dl, 'd'
    je scanf$fmt_decimal

    cmp dl, 'n'
    je scanf$fmt_count

    mov ax, buffer_size - 4

    cmp dl, 's'
    je scanf$fmt_string

    xor dl, 30h
    cmp dl, 9
    jbe scanf$fmt_stringlen

    dec static_var
    jmp scanf$ret

scanf$fmt_char:
    mov ah, 01h
    int 21h

    mov ah, 09h
    lea dx, newline
    int 21h

    arg_mov al
    inc cx
    jmp scanf$loop

scanf$fmt_byte:
    mov buffer, 5

    mov ah, 0Ah
    lea dx, buffer
    int 21h

    mov ah, 09h
    lea dx, newline
    int 21h

    movzx dx, BYTE PTR [buffer + 1]
    add cx, dx

    mov di, si
    mov dx, cx

```

```

        lea si, [buffer + 2]
        mov cx, 7
        call stow

        mov cx, dx
        mov si, di

        arg_mov al

        jmp scanf$loop

scanf$fmt_decimal:
        mov buffer, 7

        mov ah, 0Ah
        lea dx, buffer
        int 21h

        mov ah, 09h
        lea dx, newline
        int 21h

        movzx dx, BYTE PTR [buffer + 1]
        add cx, dx

        mov di, si
        mov dx, cx

        lea si, [buffer + 2]
        mov cx, 15
        call stow

        mov cx, dx
        mov si, di

        arg_mov ax

        jmp scanf$loop

scanf$fmt_string:
        mov buffer, al

        mov ah, 0Ah
        lea dx, buffer
        int 21h

        mov ah, 09h
        lea dx, newline
        int 21h

        mov dx, si

        movzx di, [buffer + 1]

```

```

        mov [buffer + di + 2], '$'

        lea si, [buffer + 2]
        arg_sca di

        cmp di, -1
        je scanf$fmt_string_ignore

        call strcpy
        jmp scanf$fmt_string_finish

scanf$fmt_string_ignore:
        movzx ax, BYTE PTR [buffer + 1]

scanf$fmt_string_finish:
        add cx, ax
        mov si, dx
        jmp scanf$loop

scanf$fmt_stringlen:
        dec si

        mov dx, cx
        mov cx, 15

        call stow
        mov cx, ax

        call digitcount
        inc ax
        add si, ax

        inc cx
        mov ax, cx

        mov cx, dx
        jmp scanf$fmt_string

scanf$fmt_count:
        arg_mov cx
        jmp scanf$loop

scanf$ret:
        mov ax, static_var
        pop di si dx cx bx
        ret
scanf ENDP

newline DB 0Dh, 0Ah, '$'
buffer DB buffer_size - 2 DUP (?)
static_var DW ?
END

```