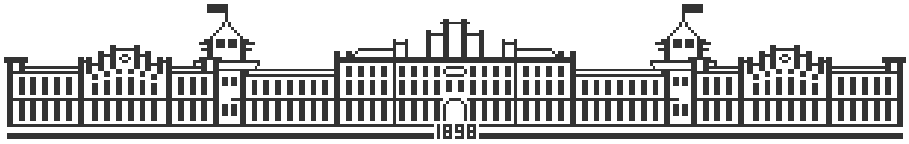


Міністерство освіти та науки України

Національний технічний університет України “Київський політехнічний інститут”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №7
з дисципліни «Системне програмування – 1»
на тему: «Програмування операцій ділення чисел»

Виконав:
студент 2-го курсу ФІОТ
групи ІВ-71
Мазан Я. В.
Перевірив:
Старший викладач
Порєв В. М.

Київ – 2019

Мета:

Навчитися програмувати на асемблері ділення чисел, вивчити перетворення з двійкової у десяткову систему числення.

Завдання:

H = 9

n = 30 + 2H = 48

H mod 3 = 0 → ділення групами по 8 бітів при перетворенні двійкового запису числа у десятковий

$y = F(x, m) = \frac{5}{x+1}2^m$

Код програми:

main.asm:

```
%include "io.inc"
%include "functions.asm"
%include "longop.asm"

section .bss
; my function
result_buffer : resd 1
result_string : resb 10
;factorial
factorial : resd 9
buffer : resd 9
factorial_byte : resb 36
text_buffer : resb 100
section .data
inscription db "Лабораторна робота №7. Виконав Мазан Ян, група ІВ-71", 0
inscription0 db "Моя функція: y = 5/(x+1) * 2^m. Результат обчислень (hex): ", 0
inscription1 db "n = 48", 0
inscription2 db "n! (hex) = ", 0
inscription3 db "n! (dec) = ", 0
var dd 1
x dd 1
m db 3
section .text
global CMAIN
CMAIN:


PRINT_STRING inscription
NEWLINE
NEWLINE
push x
push m
push result_buffer
call MyFunction
PRINT_STRING inscription0
PRINT_HEX 4, result_buffer
NEWLINE
push result_buffer
```

```
push result_string
push 1
call StrDec
PRINT_STRING "Результат обчислень (dec): "
PRINT_REVERSED_STRING 2, result_string
NEWLINE
NEWLINE
mov dword [factorial + 8*4], 1
mov ebx, 1
@factorial:
push 9
push factorial
push dword [var]
push buffer
call MulFactorial
;add esp, 16
mov eax, dword [var]
inc eax
mov dword [var], eax
cmp eax, 48
jle @factorial
; converting dword factorial into byte factorial
mov esi, 0
@convertFactorialCycle:
mov eax, dword [factorial + esi*4]
mov byte [factorial_byte + esi*4 + 3], al
shr eax, 8
mov byte [factorial_byte + esi*4 + 2], al
shr eax, 8
mov byte [factorial_byte + esi*4 + 1], al
shr eax, 8
mov byte [factorial_byte + esi*4], al
;mov dword [factorial_byte + esi*4], ebx
inc esi
cmp esi, 9
jl @convertFactorialCycle
NEWLINE
PRINT_STRING inscription1
NEWLINE
PRINT_STRING inscription2
PRINT_LONG 9, factorial
NEWLINE
NEWLINE
push factorial_byte
push text_buffer
push 36
call StrDec
PRINT_STRING inscription3
PRINT_REVERSED_STRING 100, text_buffer
```

functions.asm:

```
%include "io.inc"


%macro PRINT_REVERSED_STRING 2
mov esi, %1
dec esi
%%print:
PRINT_CHAR [%2 + esi]
dec esi
cmp esi, 0
jge %%print
%endmacro


%macro PRINT_BYTES_HEX 2 ; length, byte long_number
```

```
mov ebp, %1
mov ecx, 0
%%print:
dec ebp
PRINT_HEX 1, [%2 + ecx]
PRINT_STRING " "
inc ecx
cmp ebp, 0
jnz %%print
%endmacro


%macro PRINT_LONG 2 ; length, dword long_number
mov ebp, %1
mov ecx, 0
%%print:
dec ebp
PRINT_HEX 4, [%2 + ecx*4]
PRINT_STRING " "
inc ecx
cmp ebp, 0
jnz %%print
%endmacro


section .text
global MulFactorial
global MyFunction


MyFunction: ; 5/(x+1) * 2^m
push ebp
mov ebp, esp
mov edi, dword [ebp + 16] ; x 32 bit
mov ecx, dword [ebp + 12] ; m 8 bit unsigned
mov eax, dword [ebp + 8] ; result
push eax
mov edi, dword [edi] ; prevent x value to be changed outside
inc edi
cmp edi, 0
je @error
mov eax, 5
cdq
idiv edi
mov cl, byte [ecx]
shl eax, cl
mov ecx, eax
pop eax
mov dword [eax], ecx
jmp @return
@error:
PRINT_STRING "Division by zero! Return from function"
NEWLINE
@return:
leave
ret 12


MulFactorial: ; size, A, B 32 bit, res
push ebp
mov ebp, esp
mov esi, dword [ebp + 20] ; A's size
mov edi, dword [ebp + 16] ; A
mov ebx, dword [ebp + 12] ; B 32 bit
mov ecx, dword [ebp + 8] ; buffer
push ecx ; save ecx if was used before
push esi
dec esi
@mulFact:
mov eax, dword [edi + 4*esi]
```

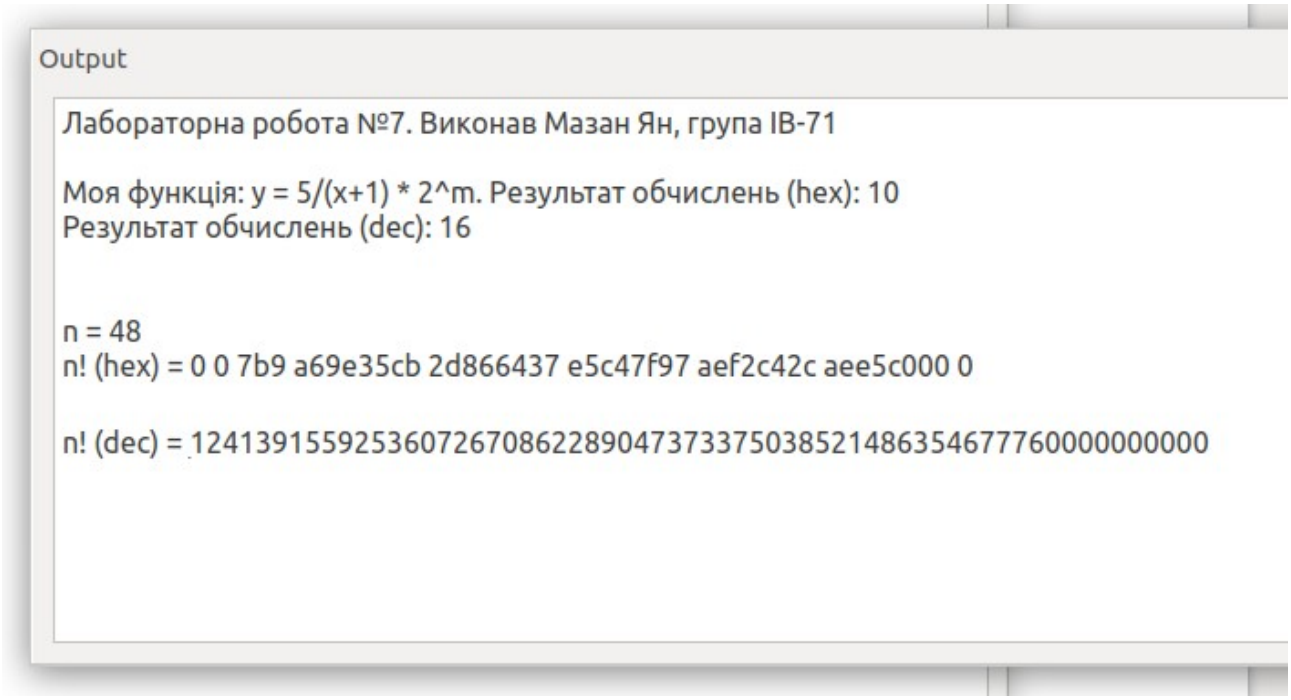
```
mul ebx
add dword [ecx + 4*esi], eax
add dword [ecx + 4*esi - 4], edx
dec esi
cmp esi, 0
jnz @mulFact
pop esi
dec esi
@swap: ; mov result into A
mov edx, dword [ecx + 4*esi]
mov dword [ecx + 4*esi], 0
mov dword [edi + esi*4], edx
dec esi
cmp esi, 0
jnz @swap
pop ecx
leave
ret 16
```

longop.asm:

```
section .bss
partial_division : resb 1000
convert_residue : resb 1
number_size : resb 1
greater_than_ten : resd 1
section .text
global Div10


Div10: ; division by bits groups
push ebp
mov ebp, esp
mov edx, dword [ebp + 20] ; size
mov ebx, dword [ebp + 16] ; divided number
mov edi, dword [ebp + 12] ; partial result
mov ecx, dword [ebp + 8] ; residue
push ebx ; ebx (bl) serves as divisor
mov esi, 0
and ax, 0xff
@division10Cycle:
pop ebx
mov al, byte [ebx + esi]
push ebx
mov bl, 10
div bl
;mov cl, ah
mov byte [edi + esi], al
inc esi
cmp esi, edx
jne @division10Cycle
pop ebx
mov dl, ah
;pop ecx
;mov byte [residue_buffer], dl
;mov ecx, dword [ebp + 8]
mov byte [ecx], dl
leave
ret 16


StrDec:
push ebp
mov ebp, esp
;sub esp, 16
mov eax, dword [ebp + 16] ; number
mov edi, dword [ebp + 12] ; text buffer
mov ecx, dword [ebp + 8] ; number size
mov esi, 0
```

```
push esi
@convertCycle:
;push edi
;push ecx
mov ecx, dword [ebp + 8]
push ecx
push dword [ebp + 16]
push partial_division
push convert_residue
call Div10
;pop ecx
;pop edi
mov ecx, dword [ebp + 8]
mov edi, dword [ebp + 12]
mov dl, byte [convert_residue]
add dl, 48
pop esi
mov byte [edi + esi], dl
inc esi
push esi
mov dword [greater_than_ten], 1 ; dword [ebp - 4] == 1 => partial_division < 10
mov esi, dword [ebp + 8]
dec esi
cmp byte [partial_division + esi], 10
jge @greaterThanTen
@swapCycle1:
mov dl, byte [partial_division + esi]
mov ebx, dword [ebp + 16]
mov byte [ebx + esi], dl
dec esi
;jmp @end
@swapCycle:
mov dl, byte [partial_division + esi]
mov ebx, dword [ebp + 16]
mov byte [ebx + esi], dl
cmp dl, 0
jz @zeroByte
mov dword [greater_than_ten], 0
@zeroByte:
dec esi
cmp esi, 0
jge @swapCycle
;push ebp
;push ecx
;PRINT_BYTES_HEX 3, partial_division
;pop ecx
;pop ebp
;NEWLINE
mov edi, dword [ebp + 12] ; text buffer
mov ecx, dword [ebp + 8] ; number size


cmp dword [greater_than_ten], 0
jz @convertCycle
jmp @end
@greaterThanTen:
mov dword [greater_than_ten], 0
jmp @swapCycle1
@end:
mov esi, dword [ebp + 8]
dec esi
mov al, byte [partial_division + esi]


pop esi
add al, 48
mov byte [edi + esi], al
leave
ret 12
```

Результати виконання програми:



Висновок:

Під час виконання даної лабораторної роботи мною були закріплені навички програмування процедур в асемблері NASM, вивчено та імплементовано в програму алгоритми цілочисельного ділення чисел підвищеної розрядності для асемблера та перетворення двійкових чисел у десяткові. Під час виконання роботи лабораторної проблем великих не виникло.