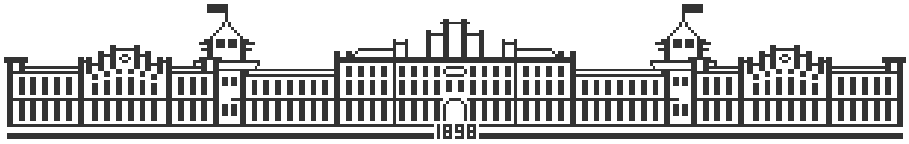


Міністерство освіти та науки України

Національний технічний університет України “Київський політехнічний інститут”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №8
з дисципліни «Системне програмування – 1»
на тему: «»

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

Київ – 2019

Мета:

Навчитися програмувати операції з плаваючою точкою на асемблері.

Завдання:

Варіант	Що потрібно обчислити	Формат даних для A,B,X
9	$Res = (...(A_{n-1}X+A_{n-2})X + A_{n-3})X + ...A_1)X + A_0$	32-бітний

Код програми:

main.asm:

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


section .bss
res : resd 1
text_res : resb 10
text_res2 : resb 20
section .data
a dd 0.0, 2.0, 1.0, 1.0, 2.1
x dd 2.0
test_convert dd -12.24
global CMAIN
CMAIN:
;write your code here
xor eax, eax
push 5
push a
push x
push res
call myFunction ; result = 49.6
PRINT_STRING "Результат обчислення функції у стандарті IEE-754: "
PRINT_HEX 4, res
NEWLINE
push res
push text_res
call floatToDec
NEWLINE
PRINT_STRING "Результат обчислення функції у десятковому форматі: "
PRINT_STRING text_res
NEWLINE
push test_convert
push text_res2
call floatToDec
NEWLINE
PRINT_STRING "Тестове конвертування числа test_convert: "
PRINT_STRING text_res2
ret
```

functions.asm:

```
section .text

global myFunction
global floatToDec
```

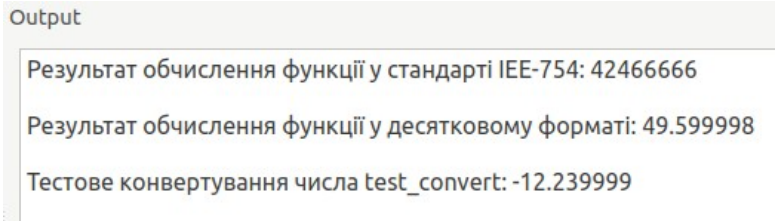
```
; f(A0, A1 ... An-1, X) = (...(An-1X+An-2)X + An-3)X + ...A1)X + A0
myFunction: ; A array size, A, X, result buffer
push ebp
mov ebp, esp
mov eax, dword [ebp + 20] ; size
mov ebx, dword [ebp + 16] ; A
mov ecx, dword [ebp + 12] ; X
mov edx, dword [ebp + 8] ; result
dec eax
cmp eax, 0 ; res = A[0] if A is one-element array
jz @oneElement
fld dword [ebx + 4*eax]
@calculationCycle:
fmul dword [ecx]
dec eax
fadd dword [ebx + 4*eax]
cmp eax, 0
jnz @calculationCycle
fstp dword [edx]
jmp @end
@oneElement:
mov esi, dword [ebx]
mov dword[edx], esi
@end:
leave
ret 16


; converts floating point 4-byte number from IEE-754 standart into perceptible human-understandable decimal
floatToDec: ; A float, result string
push ebp
mov ebp, esp
mov eax, dword [ebp + 12] ; A
mov ebx, dword [ebp + 8] ; result
mov eax, dword [eax] ; do not change outer A
mov esi, 0 ; esi - string symbol pointer
mov edi, eax ; edi - exponent
and edi, 0x7f800000 ; selecting exponent from dword A
shr edi, 23 ; make edi value belong to range [0, 255]
mov ecx, eax ; ecx - mantissa
and ecx, 0x007fffff ; selecting mantissa from dword A
@specialCases:
cmp edi, 0 ; exponent = 00000000
je @zeroExponent
cmp edi, 0xff ; exponent = 11111111
je @maxExponent
or ecx, 0x00800000 ; adding "hidden" bit to mantissa
push eax
and eax, 0x80000000 ; extract sign bit
cmp eax, 0
jz @noMinusSign
mov dword [ebx + esi], '-'
inc esi
@noMinusSign:
pop eax
; integer part conversion
cmp edi, 127
jl @lessThanOneByModule
jmp @greaterThanOneByModule
@lessThanOneByModule:
mov dword [ebx + esi], '0' ; move zero
inc esi
mov dword [ebx + esi], '.' ; move point
inc esi
jmp @floatPartConversion
@greaterThanOneByModule:
push edi
sbb edi, 127
push ecx
```

```
mov edx, ecx
mov ecx, 23
sbb ecx, edi
shr edx, cl ; selecting integer part of edx
pop ecx ; edx - integer part of mantissa
push eax
push ecx
mov ecx, 10 ; we are dividing on ecx == 10 to convert
push esi ; the integer part is reversed so we need to save esi in spite
; reversing the integer part
@integerPartConversionCycle:
mov eax, edx ; eax = edx
xor edx, edx ; edx = 0
div ecx ; {edx, eax} / (ecx == 10)
; edx - new residue
; eax - new dividen number
mov byte [ebx + esi], dl ; move a partial residue into a result text
add byte [ebx + esi], 48
inc esi
mov edx, eax ; edx - dividen part of a number
cmp edx, 10
jge @integerPartConversionCycle
mov byte [ebx + esi], dl ; move the final residue into a result text
add byte [ebx + esi], 48
mov ecx, esi ; save the end of integer part in ecx
pop esi ; save the start of integer part in esi
push ecx ; save the end of integer part in stack
@reverseCycle:
mov dh, byte [ebx + ecx] ; dh - lower number discharges that were saved in the end of a string
mov dl, byte [ebx + esi] ; dl - upper number discharges that were saved in the beginning of a string
mov byte [ebx + ecx], dl
mov byte [ebx + esi], dh
dec ecx
inc esi
cmp esi, ecx
jl @reverseCycle
pop esi ; esi now points at the end of the integer part
inc esi
mov dword [ebx + esi], '.' ; add point at the end of the integer part
inc esi
pop ecx
pop eax
pop edi ; eax contains initial number, ecx - mantissa, edi - exponent again
; float part conversion
@floatPartConversion:
shl ecx, 8 ; the last byte of ecx is empty, so we shift ecx left on 1 byte
@selectFloatPart:
cmp edi, 127
jl @hasOnlyFloatPart
jmp @hasIntegerPart
@hasOnlyFloatPart: ; shift right on absolute value of number's power
push edi
mov edx, edi
mov edi, 128
sbb edi, edx ; edi = abs(number's power). EG: A == 0.25, edi = |-2| = 2
;sbb edi, 127
mov edx, ecx
mov ecx, edi
shr edx, cl
mov ecx, edx
pop edi
;mov dword [tests], ecx
jmp @conversion
@hasIntegerPart: ; shift left on number's power
sbb edi, 127
;push ecx
mov edx, ecx
mov ecx, edi
shl edx, cl
```

```
mov ecx, edx
@conversion:
shl ecx, 1 ; correcting an error that ecx is 2 times smaller than must be
shr ecx, 4 ; use last 4 bits for the result decimal digit
mov edi, 0
@floatPartConversionCycle:
;mov dword [tests], ecx
and ecx, 0x0fffffff ; empty the last 4 bits for the result decimal digit
mov edx, ecx ; edx = ecx
shl edx, 1 ; edx = 2*ecx
shl ecx, 3 ; ecx = 8*ecx
add ecx, edx ; ecx = edx + ecx (== 2*ecx + 8*ecx = 10*ecx)
push ecx
shr ecx, 28 ; make cl contain the result decimal digit
mov byte [ebx + esi], cl
add byte [ebx + esi], 48
inc esi
inc edi
pop ecx
cmp edi, 6 ; find first 6 digits after the fraction point of decimal number
jl @floatPartConversionCycle
jmp @endConversion
@zeroExponent:
cmp ecx, 0 ; mantissa == 0
je @zero
jmp @unnormalized
@zero:
mov byte [ebx], '0'
mov byte [ebx + 1], '.'
mov byte [ebx + 2], '0'
jmp @endConversion
@unnormalized:
;%error "Unnormalized number!"
jmp @endConversion
@maxExponent:
cmp ecx, 0 ; mantissa == 0
je @infinity
jne @nan
@infinity:
push eax
and eax, 0x80000000 ; extract sign bit
cmp eax, 0
pop eax
je @plusInfinity
mov byte [ebx], '-'
@plusInfinity:
mov byte [ebx], '+'
mov dword [ebx + 1], '∞'
jmp @endConversion
@nan:
mov byte [ebx], 'N'
mov byte [ebx + 1], 'a'
mov byte [ebx + 2], 'N'
jmp @endConversion
@endConversion:
leave
ret 8
```

Результати виконання програми:



Висновок:

Під час виконання лабораторної роботи були вдосконалені навички роботи з модулями та здобуто навички виконувати операції над числами з плаваючою точкою за допомогою модуля FPU x87, розв’язуючи завдання лабораторної роботи. Лабораторна робота виконувалась в асемблера NASM, під час виконання роботи проблем не виникло.