# ЭВМ и периферийные устройства

## Отчёт по лабораторной работе №4

## «ВВЕДЕНИЕ В АРХИТЕКТУРУ ARM»

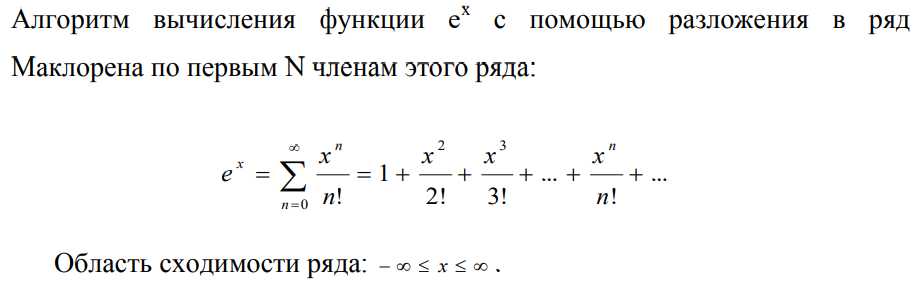
## Выполнил: Скопинцев Н.А. ФИТ НГУ 2 курс

## Преподаватель:

## Цель работы:

1. Знакомство с программной архитектурой ARM.
2. Анализ ассемблерного листинга программы для архитектуры ARM

## Вариант задания



## Листинг программы

#include <stdio.h>

#include <time.h>

#define N 400000000

double Exp(int x) {

double tmp = x;

double res = 1;

long long n = N;

for (long long i = 1; i <= n; i++) {

res += tmp;

tmp = (tmp \* x) / (i + 1);

}

return res;

}

int main() {

double e = Exp(7);

printf("%lf\n", e);

return 0;

}

Ключи компиляции:

-fverbose-asm для комментариев справа,

-O1 или другие для уровней оптимизации

**Компилируемый листинг на ассемблере для ARM с ключами:**

**-O0**

**-O1**

**-Os и -fverbose-asm**

Exp:

        cvtsi2sdl       %edi, %xmm2   # tmp91, tmp

        movsd   .LC0(%rip), %xmm1     #, <retval>

        movl    $1, %eax        #, i

        movaps  %xmm2, %xmm0        # tmp, tmp

.L2:

        addsd   %xmm0, %xmm1  # tmp, <retval>

        mulsd   %xmm2, %xmm0  # tmp, \_2

        incq    %rax    # i

        cvtsi2sdq       %rax, %xmm3   # i, tmp89

        divsd   %xmm3, %xmm0  # tmp89, tmp

        cmpq    $400000001, %rax        #, i

        jne     .L2       #,

        movaps  %xmm1, %xmm0        # <retval>,

        ret

.LC1:

        .string "%lf\n"

main:

        pushq   %rax  #

        movl    $7, %edi        #,

        call    Exp     #

        movl    $.LC1, %edi     #,

        movb    $1, %al #,

        call    printf  #

        xorl    %eax, %eax      #

        popq    %rdx    #

        ret

.LC0:

        .long   0

        .long   1072693248

## Вывод

1. Мы познакомились с программной архитектурой x86/x86-64.

2. Мы проанализировали ассемблерный листинг программы для архитектуры x86/x86-64 с разными уровнями оптимизации:

В x86, в отличии от х86-64, функции берут свои значения только со стека, когда в х86-64 они могут браться с регистров, что значительно ускоряет работу программы. O2, O3, Ofast значительно уменьшают размеры кода засчет удаления мертвого кода в данном случае. Даже такие оптимизации, как O1 тоже превносят вклад в упрощение программы, при нем меньше обращений к стеку и регистрам.s