ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

МИКРОПРОЕКТ ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ АССЕМБЛЕРА

Пояснительная записка

Исполнитель
Студентка группы БПИ193 (подгруппы 2)
Шевко Марина Николаевна
31 октября 2020 г

Москва 2020

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Постановка задачи на разработку программы

Разработать программу, определяющей максимальное значение параметра числа линейной рекуррентной последова-тельности $t_n = t_{n-1} + t_{n-2} + t_{n-3} + t_{n-4}$ при $n \geq 4$ со стартовой последовательностью чисел [0,0,0,1], которое не выходит за пределы беззнакового машинного слова

1.2. Анализ задачи

1.2.1. Беззнаковое машинное слово – 2 байта, то есть 16 битов. Значит, максимальное число, которое входит в пределы $2^{16} - 1 = 65535$.

1.2.2. Ниже приведена таблица (см. рис. 1), в которой для $n \geq 4$ рассчитаны значения рекуррентной функции. Видно, что максимальное n = 20. Будем использовать ее для самопроверки.

	tn-4	tn-3	tn-2	tn-1	tn	n
init	0	0	0	1	1	4
	0	0	1	1	2	5
	0	1	1	2	4	6
	1	1	2	4	8	7
	1	2	4	8	15	8
	2	4	8	15	29	9
	4	8	15	29	56	10
	8	15	29	56	108	11
	15	29	56	108	208	12
	29	56	108	208	401	13
	56	108	208	401	773	14
	108	208	401	773	1490	15
	208	401	773	1490	2872	16
	401	773	1490	2872	5536	17
	773	1490	2872	5536	10671	18
	1490	2872	5536	10671	20569	19
	2872	**5536**	**10671**	**20569**	**39648**	**20**
	5536	10671	20569	39648	76424	21
	10671	20569	39648	76424	147312	22
	20569	39648	76424	147312	283953	23

Рисунок 1 – Таблица расчета максимального значения параметра для заданной рекуррентной функции

2. ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ АЛГОРИТМОВ

2.1. Словесное описание алгоритма

Для вычисления значения рекуррентной функции из 4 слагаемых используется массив из 4 элементов, имитирующий очередь (FIFO).

Ниже приведен пример работы (см. рис. 2).

1. Значения в массиве суммируются (sum) — это значение функции, при текущем значении n (t_n).
2. Вместо самого старого значения записывается только что вычисленное.
3. Действие повторяется до тех пор, пока значение суммы выйдет за пределы беззнакового машинного слова (65535)

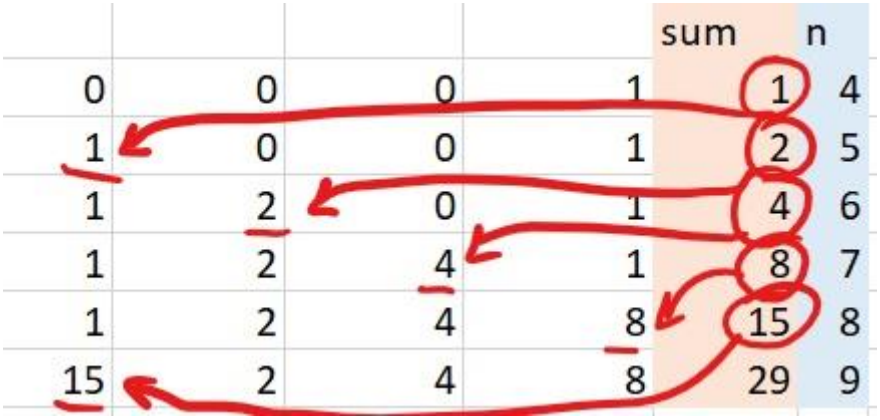

Рисунок 2 – Алгоритм вычисления значений рекуррентной функции посредствам очереди из 4 элементов

2.2. Назначение используемых переменных и регистров

2.2.1. Переменные:

VSIZE – константа, равная 4 – размер очереди.

resStr – строка для вывода итогового значения n.

vec – массив для очереди, инициализирован 0, 0, 0, 1 (по усл).

N – текущее значение параметра n, инициализирован 3 (после вычисления первого значения станет 4, как в условии)

2.2.2. Регистры

ax – хранит текущее значение функции t_n в зависимости от n. Его размер соответствует беззнаковому машинному слову – 2 байтам. На нем проверяется переполнение по флагу переноса CF – символизирует переполнение у беззнаковых чисел.

ebx –хранит ссылку на текущий элемент очереди при подсчете сумму элементов очереди. После – хранит ссылку на начало очереди.

edx – хранит число, на которое нужно увеличить ссылку на первый элемент очереди, чтобы получить самый старый в ней (смещение). Соответственно, перезаписываемый элемент будет храниться по ссылке ebx (начало очереди) + edx. Если смещение указывает на последний элемент очереди, то регистр обнуляется перед следующим циклом.

eax – при выводе значения в консоль хранит итоговое значение параметра n.

2.3. Словесное описание алгоритма для ассемблера

1. Внешний цикл mainLoop идет по значениям функции (t_n) и заканчивается при переполнении машинного слова.
2. Внутренний цикл sumLoop считает сумму значений в очереди для текущего t_n, пока не произойдет переполнение регистра ax, в котором и хранится t_n. Для этого используется инструкция JC (тестируется флаг CF)
3. Если ax не переполнился, то мы записываем вычисленное t_n вместо самого старого в очереди. Для того, чтобы знать, какое именно значение перезаписывать, в edx хранится число, обозначающее смещение от начала очереди на перезаписываемый элемент.
4. Подготовка к следующему циклу: увеличиваем edx – смещение, если он указывает на последний элемент очереди – обнуляем.

3. ТЕСТИРОВАНИЕ ПРОГРАММЫ

1. Т.к функция в условии неизменна, а ее стартовые значения фиксированы – [0, 0, 0, 1], то единственный результат, которые она выводит, это значение параметра n = 20, что совпадает со значением в пункте 1.2. Анализ задачи в Введении.

	tn-4	tn-3	tn-2	tn-1	tn	n
init	0	0	0	1	1	4
	0	0	1	1	2	5
	0	1	1	2	4	6
	1	1	2	4	8	7
	1	2	4	8	15	8
	2	4	8	15	29	9
	4	8	15	29	56	10
	8	15	29	56	108	11
	15	29	56	108	208	12
	29	56	108	208	401	13
	56	108	208	401	773	14
	108	208	401	773	1490	15
	208	401	773	1490	2872	16
	401	773	1490	2872	5536	17
	773	1490	2872	5536	10671	18
	1490	2872	5536	10671	20569	19
	2872	**5536**	**10671**	**20569**	**39648**	**20**
	5536	10671	20569	39648	76424	21
	10671	20569	39648	76424	147312	22

```
VSIZE equ dword 4     ;const size
resStr          db 'Parameter n = %d', 0
vec             dd 0,0,0,1
N               dd 3
```

```
C:\Users\User\Deskt
Parameter n = 20
```

2. Для дополнительной проверки работы алгоритма зададим **другие** стартовые значения – [1, 2, 3, 4]. Ниже приведена расчётная таблица значений функции для этих значений. Видно, что результат совпадает.

tn-4	tn-3	tn-2	tn-1	tn	n
1	2	3	4	10	4
2	3	4	10	19	5
3	4	10	19	36	6
4	10	19	36	69	7
10	19	36	69	134	8
19	36	69	134	258	9
36	69	134	258	497	10
69	134	258	497	958	11
134	258	497	958	1847	12
258	497	958	1847	3560	13
497	958	1847	3560	6862	14
958	1847	3560	6862	13227	15
1847	3560	6862	13227	25496	16
3560	**6862**	**13227**	**25496**	**49145**	**17**
6862	13227	25496	49145	94730	18
13227	25496	49145	94730	182598	19

```
VSIZE equ dword 4     ;const size
resStr          db 'Parameter n = %d', 0
vec             dd 1,2,3,4
N               dd 3
```

```
C:\Users\User\Deskt
Parameter n = 17
```

3. Для дополнительной проверки работы алгоритма зададим **другие** стартовые значения – [100, 200, 300, 400]. Ниже приведена расчётная таблица значений функции для этих значений. Видно, что результат совпадает.

tn-4	tn-3	tn-2	tn-1	tn	n
100	200	300	400	1000	4
200	300	400	1000	1900	5
300	400	1000	1900	3600	6
400	1000	1900	3600	6900	7
1000	1900	3600	6900	13400	8
1900	3600	6900	13400	25800	9
3600	6900	13400	25800	49700	10
6900	13400	25800	49700	95800	11
13400	25800	49700	95800	184700	12

```
VSIZE equ dword 4     ;const size
resStr          db 'Parameter n = %d', 0
vec             dd 100,200,300,400
N               dd 3
```

```
C:\Users\User\Desktop\Stu
Parameter n = 10
```

4. СПИСОК ИСПЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1) Ассемблер. Условия [Электронный ресурс] // Режим доступа: https://ravesli.com/assembler-usloviya/#toc-0, свободный (Дата обращения: 27.10.2020)
2) Мануал программера [Электронный ресурс] // Режим доступа: http://flatassembler.narod.ru/fasm.htm#1-2-6 , свободный (Дата обращения: 29.10.2020)
3) Типы данных в ассемблере [Электронный ресурс] // Режим доступа: https://prog-cpp.ru/asm-datatypes/, свободный (Дата обращения: 27.10.2020)
4) Флаг переноса [Электронный ресурс] // Режим доступа: https://ru.wikipedia.org/wiki/%D0%A4%D0%BB%D0%B0%D0%B3_%D0%BF%D0%B5%D1%80%D0%B5%D0%BD%D0%BE%D1%81%D0%B0, свободный (Дата обращения: 27.10.2020)

ПРИЛОЖЕНИЕ 1

ТЕКСТ ПРОГРАММЫ

```
format PE console
entry start

include 'win32a.inc'

section '.data' data readable writable

        VSIZE equ dword 4 ; константный размер очереди

        resStr         db 'Parameter n = %d', 0
        vec            dd 0,0,0,1
        N              dd 3

section '.code' code readable executable
start:
        mov edx, dword 0; смещение ссылки для перезаписываемого элемента

; внешний цикл
mainLoop:
        ; суммируем элементы очереди - находим значение функции
        xor eax, eax    ; очищаем для хранения текущей суммы
        mov ecx, VSIZE  ; в ecx кладем размер очереди - 4
        mov ebx, vec    ; в ebx находится ссылка на первый элемент очереди

; внутренний цикл
sumLoop:
        add ax, [ebx]   ; увеличиваем ax на текущий элемент очереди
        jc endProc      ; проверка на переполнение
        add ebx, 4      ; i++, ссылаемся на следующий элемент очереди
        loop sumLoop

        ; конец секции суммирования
        inc [N]         ; если переполнения не произошло, то мы увеличиваем параметр n
        mov ebx, vec    ; смещаемся в начало очереди
        mov [ebx+edx], eax ; перезаписываем самый старый элемент в очереди

        ; подготовка к след циклу
        add edx, 4      ; запоминаем смещение от начала очереди
        cmp edx, 12
        jle mainLoop
        xor edx, edx    ; если был перезаписан последний элемент, то обнуляем смещение
        jmp mainLoop

endProc:
        mov eax, [N]    ; запоминаем в регистр итоговое значение параметра n
        push eax
        push resStr
        call [printf]   ; печать значения параметра
        call [getch]

        push 0
        call [ExitProcess]

;----------------------------Importing-libraries---------------------------
section '.idata' import data readable
    library kernel, 'kernel32.dll',\
            msvcrt, 'msvcrt.dll'


include 'api\kernel32.inc'
    import kernel,\
           ExitProcess, 'ExitProcess'

  include 'api\kernel32.inc'
    import msvcrt,\
           printf, 'printf',\
           getch, '_getch'
```