Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

ПРОГРАММА ДЛЯ ОПРЕДЕЛЕНИЯ КОЛИЧЕСТВА ЧИСЕЛ ФЕРМА ОТ 1 ДО БЕЗЗНАКОВОГО ДВОЙНОГО МАШИННОГО СЛОВА

Пояснительная записка

Исполнитель
Студент группы БПИ196
Д.Е. Калмыков
«30» октября 2020 г.

Листов 12

Москва 2020

СОДЕРЖАНИЕ

1. ТЕКСТ ЗАДАНИЯ

Разработать программу для определения количества чисел Ферма от 1 до беззнакового двойного машинного слова.

2. МЕТОДЫ

2.1. Расчетные методы

1. Метод для инициализации полей начальными значениями (здесь также считается первое число Ферма)

```
macro SetInitialValues power, topExpressionValues, allExpresionValues{
        mov [power], 0
        mov [topExpressionValue], 1
        shl [topExpressionValue], 0

        mov [allExpressionValue], 1
        shl [allExpressionValue], 1
        inc [allExpressionValue]
}
```

2. Метод для вычисления следующего числа Ферма

```
macro GetNextFarmNumber result, power {

        ; Initial value.
        mov [temp], 1
        mov [cur_power], 0

        shl [power], 1

        ; Loop for calculating 2 ^ power.
        getNextPowerLoop:

                ; temp = temp * 2 (bit shift).
                shl [temp], 1

                ; Increase current power.
                add [cur_power], 1

                ; Write require power to register.
                mov ecx, [power]

                ; Check number of operation (exit condition - require power = current power).
                cmp ecx, [cur_power]
                jne getNextPowerLoop

        ; Write 2 ^ power to ebx and increase it
        mov ebx, [temp]
        inc ebx

        ; Write data from ebx to the result.
        mov result, ebx
}
```

2.2. Процедуры

1. Метод для вывода числа Ферма в формате F[index] = value

```
macro PrintCurrentFarmNumber index, value {
        push [index]
        push [value]
        push strFarmNumber
        call[printf]
}
```

2. Метод для вывод определения числа Ферма.

```
macro PrintFarmNumberDefinition  {
         push strDefinition
         call[printf]
}
```

3. Метод для вывода информации о переполнении и общем количестве чисел Ферма в заданном диапазоне

```
macro PrintResult power, counter {

         ; Print empty str just for handsome output.
        push strEmpty
        call[printf]

         ; Print message about overflow.
        push[power]
        push strOutOfRange
        call[printf]

         ; Print total count of Farm numbers.
        push [counter]
        push strTotalCount
        call[printf]

         ; Last phrase: "Press any key to cointinue...".
        push strExit
        call[printf]
}
```

3. СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1) Программирование на языке ассемблера. Микропроект. Требования к оформлению. 2020-2021 уч. г. [Электронный ресур]// URL: http://www.softcraft.ru/edu/comparch/tasks/mp01 (Дата обращения: 30.10.2020, режим доступа: свободный)
2) FLAT ASSEMBLER 1.64 – МАНУАЛ ПРОГРАММЕРА [Электронный ресурс] // URL: http://flatassembler.narod.ru/fasm.htm (Дата обращения: 30.10.2020, режим доступа: свободный)
3) Число Ферма – Википедия [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Число_Ферма (Дата обращения: 30.10.2020, режим доступа: свободный)

4. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

```
Farm number: F[n] = 2 ^ (2 ^ n) + 1

F[0] = 3
F[1] = 5
F[2] = 17
F[3] = 257
F[4] = 65537

Out of range of dword for n = 5

Total count: 5

Press any key to exit...
```

5. ПРИЛОЖЕНИЕ

6.1. FarmNumberMain.asm

```
;-------------------------------------------------------------------------
format PE console

entry start

include 'win32a.inc'
include 'FarmNumberMethods.inc'

;-------------------------------------------------------------------------
section '.data' data readable writable

        strDefinition db 'Farm number: F[n] = 2 ^ (2 ^ n) + 1', 10, 10, 0
        strEmpty db '', 10, 0
        strOutOfRange  db 'Out of range of dword for n = %d', 10, 10, 0
        strExit db 'Press any key to exit...', 0
        strFarmNumber  db 'F[%d] = %d', 10, 0
        strTotalCount db 'Total count: %d', 10, 10, 0


        ; Power = n: F[n] = 2 ^ (2 ^ n) + 1.
        power           dd ?

        ; Temp power for calculating new power of 2.
        cur_power       dd ?

        ; Count amount of Farm numbers.
        counter         dd ?

        ; Temp value for calculate new power of 2.
        temp            dd ?

        ; F[n] = 2 ^ (2 ^ n) + 1
        ; Top expression = (2 ^ n)
        topExpressionValue dd ?

        ; AllExpression = F[n]
        allExpressionValue dd ?

;-------------------------------------------------------------------------
section '.code' code readable executable

        start:

                ; Current index of Farm number.
                mov [counter], 0

                ; Print definition of Farm number.
                PrintFarmNumberDefinition
```

```
        ; Initialize fields with start values.
        SetInitialValues power, topExpressionValue, allEaxpressionValue

    getNextValueLoop:

        ; Print current Farm number.
        PrintCurrentFarmNumber allExpressionValue, power

        ; Get new Farm number.
        GetNextFarmNumber [allExpressionValue], topExpressionValue

        inc [power]
        inc [counter]

        ; Check for overflow.
        ; Compare with 1, because of 2 ^ (2 ^ n) cause overflow,
        ; then 2 ^ (2 ^ n) = 0,
        ; but F[n] = 2 ^ (2 ^ n) + 1 = 0 + 1 = 1
        cmp [allExpressionValue], 1

        jne getNextValueLoop


    finish:

        ; Print info about overflow and total count.
        PrintResult power, counter

        call [getch]

        push 0
        call [ExitProcess]

;-----------------------------------------------------------------------

section '.idata' import data readable
library kernel, 'kernel32.dll',\
        msvcrt, 'msvcrt.dll'

    import kernel,\
        ExitProcess, 'ExitProcess'

    import msvcrt,\
        printf, 'printf',\
        getch, '_getch',\
        scanf, 'scanf'
;-----------------------------------------------------------------------
```

6.2. FarmNumberMethods.inc

```
;-------------------------------------------------------------------------------------------------------
; Summary:
;       Get next Farm number.

; Params:
;       result - variable for recording the result.
;       power - require power of 2 (power = (2 ^ n) for F[n] = 2 ^ (2 ^ n) + 1).

; Return:
;       value equal to F[n] = 2 ^ power + 1 (result).
macro GetNextFarmNumber result, power {

        ; Initial value.
        mov [temp], 1
        mov [cur_power], 0

        shl [power], 1

        ; Loop for calculating 2 ^ power.
        getNextPowerLoop:

                ; temp = temp * 2 (bit shift).
                shl [temp], 1

                ; Increase current power.
                add [cur_power], 1

                ; Write require power to register.
                mov ecx, [power]

                ; Check number of operation (exit condition - require power = current
power).
                cmp ecx, [cur_power]
                jne getNextPowerLoop

        ; Write 2 ^ power to ebx and increase it
        mov ebx, [temp]
        inc ebx

        ; Write data from ebx to the result.
        mov result, ebx
}

;-------------------------------------------------------------------------------------------------------
; Summary:
;       Print current Farm number.

; Params:
```

```
;       index - number of current Farm number.
;       value - value of current Farm number.

; Return:
;       void.
macro PrintCurrentFarmNumber index, value {
        push [index]
        push [value]
        push strFarmNumber
        call[printf]
}

;-------------------------------------------------------------------------------------------------
-------------------------
; Summary:
;       Print definition of Farm number.

; Params:
;       No parameters.

; Return:
;       void.
macro PrintFarmNumberDefinition  {
        push strDefinition
        call[printf]
}

;-------------------------------------------------------------------------------------------------
--------------------------
; Summary:
;       Set up initial values and calculate initial Farm number.

; Params:
;       power - start power of farm number: F[0] = 2 ^ (2 ^ 0) + 1.
;       topExpressionValue - start value (2 ^ n) of F[n] = 2 ^ (2 ^ n) + 1
;       allExpressionValue - value of F[n].

; Return:
;       void.
macro SetInitialValues power, topExpressionValues, allExpresionValues{
        mov [power], 0
        mov [topExpressionValue], 1
        shl [topExpressionValue], 0

        mov [allExpressionValue], 1
        shl [allExpressionValue], 1
        inc [allExpressionValue]
}

;-------------------------------------------------------------------------------------------------
-------------------------
; Summary:
```

```
;       Print result.

; Params:
;       power - power that caused the overflow.
;       counter - result amount of Farm numbers.

; Return:
;       void.
macro PrintResult power, counter {

        ; Print empty str just for handsome output.
        push strEmpty
        call[printf]

        ; Print message about overflow.
        push[power]
        push strOutOfRange
        call[printf]

        ; Print total count of Farm numbers.
        push [counter]
        push strTotalCount
        call[printf]

        ; Last phrase: "Press any key to cointinue...".
        push strExit
        call[printf]
}
```