# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе № 3**

# по дисциплине «Организация ЭВМ и систем»

# Тема: Представление и обработка целых чисел. Организация ветвящихся процессов

Студент гр. 1384 Усачева Д.В.

Преподаватель Кирьянчиков В. А.

Санкт-Петербург 2022

# Цель работы.

# Изучить механизм ветвления на языке Ассемблера. Разработать программу на основе полученных знаний.

# Задание.

# Вариант 26.

/ 2\*(i+1) -4 , при a>b

f6 = <

\ 5 - 3\*(i+1), при a<=b

/ - (6\*i+8) , при a>b

f8 = <

\ 9 -3\*(i-1), при a<=b

/ min(i1,i2), при k=0

f1 = <

\ max(i1,i2), при k/=0

Разработать на языке Ассемблера программу, которая по заданным целочисленным значениям параметров a, b, i, k вычисляет:

а) значения функций i1 = f1(a,b,i) и i2 = f2(a,b,i);

b) значения результирующей функции res = f3(i1,i2,k),

где вид функций f1 и f2 определяется из табл. 2, а функции f3 - из табл.3 по цифрам шифра индивидуального задания (n1,n2,n3), приведенным в табл.4.

Значения a, b, i, k являются исходными данными, которые должны выбираться студентом самостоятельно и задаваться в процессе исполнения программы в режиме отладки. При этом следует рассмотреть всевозможные комбинации параметров a, b и k, позволяющие проверить различные маршруты выполнения программы, а также различные знаки параметров a и b.

Замечания:

1) при разработке программы нельзя использовать фрагменты, представленные на ЯВУ, в частности, для ввода-вывода данных. Исходные данные должны вводиться, а результаты контролироваться в режиме отладки;

2) при вычислении функций f1 и f2 вместо операции умножения следует использовать арифметический сдвиг и, возможно, сложение;

3) при вычислении функций f1 и f2 нельзя использовать процедуры;

4) при разработке программы следует минимизировать длину кода, для чего, если надо, следует преобразовать исходные выражения для вычисления функций.

# Выполнение работы.

1. Из таблицы получен вариант набора функций, которые необходимо

реализовать.

2. Программа протранслирована с различными значениями переменных,

результат выполнения набора функций зафиксирован в разделе тестирование;

3. Программа выполнена в пошаговом режиме под управлением

отладчика с фиксацией значений используемых переменных.

Были созданы три сегмента: сегмент стека (AStack), сегмент данных (DATA) и сегмент кода (CODE). Метки сегментов были записаны в соответствующие регистры с помощью директивы ASSUME. Исходный код программы см. в приложении А.

В сегменте данных заданы метки для переменных a, b, i, k, i1, i2, res

использующихся в программе, значения которых можно менять в режиме

отладки.

В сегменте CODE была создана процедура Main, в которой сначала адрес сегмента данных помещается в регистр ds, а дальше происходит работа с функциями.

В программе запрещено использовать процедуры, поэтому функции были реализованы с помощью фрагментов кода, размеченных метками, с

безусловными переходами на них. Вызов функции безусловным переходом jmp к метке данной функции.

Для выполнения задания при реализации функций использовались следующие команды:

* SAL – команда, осуществляющая сдвиг всех битов операнда влево.
* JG – команда, выполняющая короткий переход, если первый операнд больше второго операнда при выполнении операции сравнения с помощью команды cmp.
* JNE – команда, выполняющая короткий переход, если первый операнд не равен второму при выполнении сравнения с помощью команды cmp.
* JGE– команда, выполняющая короткий переход, если первый операнд больше или равен второму при выполнении сравнения с помощью команды cmp.
* NEG – команда, инвертирующая все биты числа.
* ADD – команда выполняет целочисленное сложение двух операндов и флага переноса CF.
* JMP – команда безусловного перехода, то есть прыжок может быть, как дальним, так и ближним.

В функциях есть ветвление. То есть их поведение зависит от состояния

переменных a, b, следовательно, функции f1 и f2 логически подразделяются на

три части: сравнение переменных a и b и две ветви, переход в которые осуществляется непосредственно после сравнения.

Исполнение функции f3 происходит аналогичным образом, но использует значения, рассчитанные в двух предыдущих функциях.

# Тестирование.

Результаты тестирования представлены ниже в таблице. Тестирование проводилось в отладчике AFDPRO.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Выходные данные | Комментарии |
|  | a = 1  b = 2  i = 4  k = 1 | i1 = FFF616 = -1010;  i2 = 000016 = 010;  res = 000016 = 010; | Ответ правильный с учетом проверки функций при a<=b, k!=0 |
|  | a = 2  b = -1  i = 3  k = 0 | i1 = 000416 = 410;  i2 = FFE616 = -2610;  res = FFF616 = (-26)10; | Ответ правильный с учетом проверки функций при a>b, k=0 |
|  | a = 1  b = 1  i = 2  k = 0 | i1 = FFFC16 = -410;  i2 = 000616 = 610;  res = FFFC16 = -410; | Ответ правильный с учетом проверки функций при a<=b, k=0 |
|  | a = 2  b = 1  i = 1  k = -1 | i1 = 000016 = 010;  i2 = FFF216 = 1210;  res = 000016 = 010; | Ответ правильный с учетом проверки функций при a>b, k!=0 |

**Вывод.**

В результате работы была написана программа, реализующая вычисление значения функции по некоторым параметрам. Были изучены способы организации ветвления в языке Ассемблер, условные и безусловные переходы.

# Исходный код LR3.ASM

;  / 2\*(i+1) -4 , a>b

;f6 = <

;  \ 5 - 3\*(i+1), a<=b

;  / - (6\*i+8) , a>b

;f8 = <

;  \ 9 -3\*(i-1), a<=b

;  / min(i1,i2), k=0

;f1 = <

;  \ max(i1,i2), k/=0

; Стек программы

AStack SEGMENT STACK

    DW 12 DUP(?)

AStack ENDS

; Данные программы

DATA SEGMENT

  a   DW 0

  b   DW 0

  i   DW 0

  k   DW 0

  i1  DW 0

  i2  DW 0

  res DW 0

DATA ENDS

; Код программы

CODE SEGMENT

  ASSUME CS:CODE, DS:DATA, SS:AStack

; Головная процедура

Main PROC FAR

  push DS

  sub AX,AX

  push AX

  mov AX,DATA

  mov DS,AX

  mov cx, i

  mov ax, a

  cmp ax, b

  jg f\_true ; если а>b

  ; иначе:

  f\_false:

  ;f1(f6)

  sal cx, 1 ; i << 1 = i\*2

  add cx, i ; i\*2 + i = i\*3

  neg cx    ; -i\*3

  add cx, 2 ; 2 - i\*3 = 5 - 3(i + 1)

  mov i1, cx

  ;f2(f8)

  add cx, 10 ; 12 - i\*3 = 9 - 3\*(i - 1)

  mov i2, cx

  jmp F3 ; переход к f3

  f\_true:

  ;f1(f6)

  mov ax, 2

  neg ax     ;-2

  sal cx, 1  ; i << 1 = i\*2

  add ax, cx ; -2 + 2\*i = 2\*(i + 1) - 4

  mov i1, ax

  ;f2(f8)

  mov ax, 8

  add cx, i  ; i\*2 + i = 3\*i

  sal cx, 1  ; 3i\*2

  add ax, cx ; 8 + 6\*i

  neg AX     ;= -(6\*i + 8)

  mov i2, ax

  F3:

  mov ax, k

  mov bx, i2

  cmp ax, 0

  JNe f3\_not\_zero ; если k!=0, переход к f3\_not\_zero

  ;k=0

  mov ax, i1

  cmp ax, i2

  JGe MIN       ; при i1 >= i2, переходим к MIN

  mov res, ax ; res = ax = i1

  jmp EXIT

  MIN:

  mov res, bx   ; res = bx = i2

  jmp EXIT

  f3\_not\_zero:

  mov ax, i1

  cmp ax, i2

  JGe MAX       ; при i1 >= i2 переходим к MAX

  mov res, bx ; res = bx = i2

  jmp EXIT

  MAX:

  mov res, ax   ; res = ax = i1

  EXIT:

  int 20h

Main ENDP

CODE ENDS

END Main