# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №3**

# по дисциплине «Организация систем и ЭВМ»

**Тема:**

«**Изучение организации ветвлений в программах на языке ассемблера.**»

Студент гр. 3388 Снигирёв А.А.

Преподаватель Фирсов М.А.

Санкт-Петербург 2024

# Цель работы.

Изучить способ реализации ветвлений (if-else) в ассемблере Intel-8086. Реализовать программу, выполняющую различные арифметические операции и находящую значение математических функций в зависимости от некоторых условий.

# Задание.

Разработать на языке Ассемблер iX86 программу, которая по заданным целым значениям a,b,i,k, размером 1 слово, вычисляет:

а) значения i1 = fn1(a,b,i) и i2 = fn2(a,b,i);

b) значения res= fn3(i1,i2,k),

где вид функций fn1,fn2 определяется из табл. 1, а функции fn3 — из табл.2 по цифрам шифра индивидуального задания (n1.n2.n3).

Значения a,b,i,k являются исходными данными, которые должны выбираться студентом самостоятельно и задаваться в процессе исполнения программы в нужные ячейки памяти в режиме отладки. При этом следует рассмотреть все возможные комбинации параметров a, b и k, позволяющие проверить различные маршруты выполнения программы.

Замечания (требования):

1) При разработке программы не 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 fn1 и fn2 вместо операции умножения следует использовать арифметический сдвиг и, возможно, сложение.

3) Не использовать процедуры (в том числе при вычислении функций fn1 и fn2).

4) Случаи, когда число-результат выходит за пределы одного слова, учитывать не требуется.

5) При разработке программы следует минимизировать длину кода; для этого могут потребоваться преобразования формул и введение промежуточных результатов.

6) В коде должна быть выделена главная вычислительная часть, именно её нужно минимизировать. В главную часть не входят команды для инициализации сегментного регистра и для корректного завершения программы.

Для выполнения задания:

1. Преобразовать формулы, по которым будут выполняться вычисления, согласно замечанию 5.

2. В табличном процессоре (например, LibreOffice Calc) создать документ, в котором в 4 ячейки вводятся входные данные (a, b, i, k), в 3 ячейках появляются результаты i1, i2, res, вычисленные по исходным формулам, и ещё в одной ячейке — res, вычисленный по преобразованной формуле. Проверить, что результаты совпадают для разных входных данных.

3. Разработать программу; в коде должны быть выделены части, посвящённые вычислению итоговых значений (3 штуки) и ключевых промежуточных значений, если таковые были определены при преобразовании формул. Код должен быть с подробными комментариями: после каждой вычислительной команды должно быть указано полученное к текущему моменту значение. Подсчитать количество команд в главной вычислительной части и указать в комментарии.

4. Протестировать программу, оформив таблицу с входными данными и результатами. Тесты должны проверять все пути выполнения вычислений, и их точно не менее 4.

Вариант 19: 4.5.7

/-(6\*i-4), при a>b

f4 = <

\ 3\*(i+2), при a<=b

/20-4\*i, при a>b

f5 = <

\ -(6\*i-6), при a<=b

/(|i1|+|i2|), при k<0

f7 = <

\ max(6,|i1|), при k>=0

# Основные теоретические положения.

БВ зависимости от способа указания адреса очередной МК различают устройства МПУ с **принудительным** и **естественным** порядком следования МК. В случае принудительной адресации МК адрес следующей МК указывется непосредственно в текущей МК. Этот адрес может задаваться безусловно или выбираться в зависимости от условия, определяемого по признакам (флажкам) выполнения текущей МК, что позволяет реализовать ветвления в микропрограммах.

В данном ассемблере есть несколько команд, позволяющих перепрыгивать на определенную метку при выполнении условий. При выполнении лабораторной работы были использованы следующие команды:

1. jmp – безусловный переход
2. jge – выполняет переход, если первый операнд больше или равен второму при сравнении командой cmp
3. jg – переход, если первый операнд строго больше второго
4. jne – выполняет переход, если операнды не равны
5. jle – переход если первый меньше или равен второму

Для различных действий с числами был использован стандартный набор команд:

1. add – сложение
2. sub – вычитание
3. sal – сдвиг байт влево (умножение на 2)
4. neg – изменение знака
5. cmp – сравнение операндов

# Выполнение работы

1. Для упрощения было решено раскрыть все скобки у функций

/-6\*i+4, при a>b

f4 = <

\ 3\*i+6, при a<=b

/20-4\*i, при a>b

f5 = <

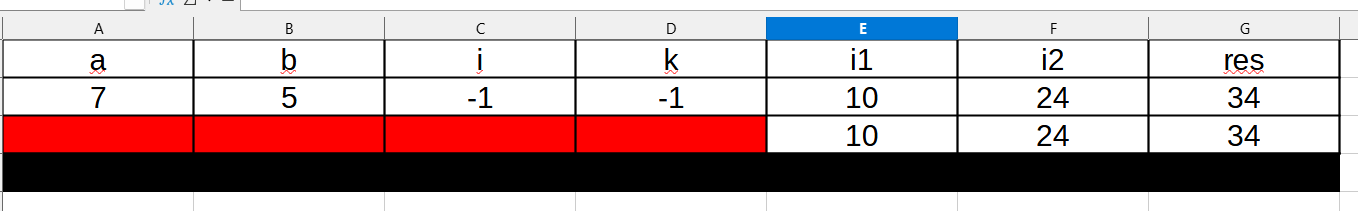
\ -6\*i+6, при a<=b

/|i1|+|i2|, при k<0

f7 = <

\ max(6,|i1|), при k>=0

1. Таблица была создана, формулы вбиты

Рисунок 1

1. Программа была написана в соответствии с требованиями, исходный код см. в **Приложении.**

Основная идея работы — переход с помощью команд jne, jg, jle и jge к нужным, в зависимости от результатов сравнения, модулям кода, каждый из которых обрабатывает свой случай. По сути, это ни что иное как конструкция if-else в более высокоуровневых языках программирования.

1. Программа была протранслирована без ошибок и предупреждений, а также протестирована для различных наборов входных данных.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Переменная** | **A** | **b** | **i** | **k** | **I1** | **I2** | **Res** |
| **Адрес в памяти** | **DS:0002** | **DS:0004** | **DS:0000** | **DS:0006** | **DS:0008** | **DS:000A** | **DS:000C** |
| **Тест 1** | **10**  **(000A)** | **1**  **(0001)** | **3**  **(0003)** | **0**  **(0000)** | **-14**  **(FFF2)** | **8**  **(0008)** | **14**  **(000E)** |
| **Тест2** | **10**  **(000A)** | **1**  **(0001)** | **3**  **(0003)** | **-1**  **(0001)** | **15**  **(000F)** | **-12**  **(FFF4)** | **27**  **(001B)** |
| **Тест3** | **1**  **(0001)** | **10**  **(000A)** | **3**  **(0002)** | **0**  **(0000)** | **15**  **(000F)** | **-12**  **(FFF4)** | **15**  **(000F)** |
| **Тест 4** | **1**  **(0001)** | **10**  **(000A)** | **3**  **(0002)** | **-1**  **(FFFF)** | **15**  **(000F)** | **-12**  **(FFF4)** | **27**  **(001B)** |
| **Тест 5** | **1**  **(0001)** | **10**  **(000A)** | **-1**  **(FFFF)** | **0**  **(0000)** | **3**  **(0003)** | **12**  **(000C)** | **6**  **(0006)** |

Таблица 1

Различные соотношения a и b учтены, также как и различные значения k и i.

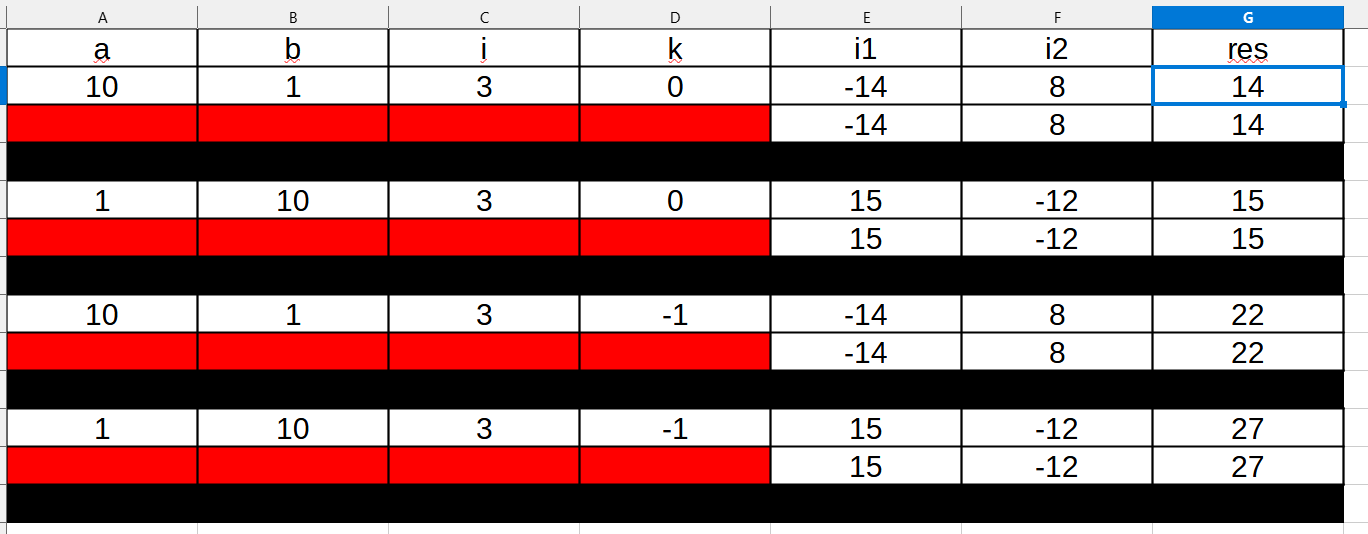
Каждый из пяти случаев был протестирован в табличном процессоре, результаты полностью совпали с экспериментальными (см. Рисунок 2)

Рисунок 2

**Вывод**

Были изучены способы реализации логических ветвлений в Assembler Intel 8086. Это знание было использовано в реализации программы, вычисляющей значения некоторых кусочно-заданных математических функций. Также дополнительно были изучены полезные команды для арифметических операций и сравнений.

# ПРИЛОЖЕНИЕ

# ИСХОДНЫЙ КОД ПРОГРАММЫ

1. lr3.ASM

­

ASSUME CS:CODE, SS:AStack, DS:DATA

AStack SEGMENT STACK

DW 32 DUP(0)

AStack ENDS

DATA SEGMENT

a DW 2

b DW 1

i DW 3

k DW -9

i1 DW ?

i2 DW ?

res DW ?

DATA ENDS

CODE SEGMENT

Main PROC FAR

push ds

sub ax, ax

push ax

mov ax, DATA

mov ds, ax

mov ax, a ;ax = a

mov bx, i ;bx = i

sal bx,1 ;bx = 2i

mov cx,bx ;cx = 2i

add cx,i ;cx = 3i

cmp ax, b

jle solve\_I11 ;if a <= b, go to A1

sal cx,1 ; cx = 6i

neg cx ; -6 \* i

add cx, 4 ; -6 \* i + 4

mov i1, cx ; result of f1 i1=cx

jns solve\_I2 ; if i1>0 go to next

neg cx ; else make above 0

solve\_I2:

mov dx,20

sal bx,1 ;bx = 4i

sub dx,bx ; dx = 20-4i

mov i2, dx ; result of f2

jns B1 ; if i2>0 go to next part

neg dx ; else make above 0

jmp B1

solve\_I11: ; a <= b

add cx, 6 ; 3i + 6

mov i1, cx ; result of f1

mov ax,cx ; ax = 3i+6

jns solve\_I22 ; if i1>0 goto next

neg cx ; В cx лежит модуль i1

solve\_I22:

mov dx,18

sal ax,1 ; ax=6i+12

sub dx,ax ; dx = 18-6i-12 = -6i+6

mov i2, dx ; result of f2

jns B1 ; if i2 > 0 goto next

neg dx ; В dx лежит модуль i2

B1:

mov ax, k ; ax = k

cmp ax, 0

jge B2 ; k >= 0, go to B2

add cx, dx ; dx = |i1| + |i2|

jmp B3

B2:

cmp cx, 6

jg B3 ; |i1| > 6

mov cx,6

B3:

mov res, cx ; res = |i1|

;39

Exit:

int 20h

Main ENDP

CODE ENDS

END Main