**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Организация ЭВМ и систем»**

**Тема:**

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

|  |  |
| --- | --- |
| Студент гр. 4344 | Байдаков Г.И. |
| Преподаватель | Фирсов М.А. |

Санкт-Петербург

2025

**Цель работы.**

Изучить организацию ветвлений на языке Assembler.

**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.  
Вариант №1 (шифр 1.2.1):

/15-2\*i, при a>b

fn1 = <

\ 3\*i+4, при a<=b

/-(4\*i+3), при a>b

fn2 = <

\ 6\*i-10 , при a<=b

/ min(i1,i2), при k=0

fn3 = <

\ max(i1,i2), при k/=0

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, введение промежуточных результатов и инициализация ячеек памяти нужными значениями.

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 трёх ячейках — i1, i2, res, вычисленные по преобразованным формулам. Преобразованные формулы должны соответствовать процессу вычислений, происходящему в написанной программе.

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 (к последней команде главной вычислительной части).

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

**Основные теоретические положения.**

Ветвление (разветвление) программы представляет собой изменение последовательного порядка выполнения команд в зависимости от выполнения определенных условий. В ассемблере организация ветвлений реализуется через систему переходов, которые изменяют адрес следующей исполняемой команды.

Различают два типа переходов: безусловные и условные. Безусловный переход (JMP) выполняется всегда, независимо от состояния флагов процессора, и передает управление команде по указанному адресу. Условные переходы выполняются только при выполнении заданного условия, которое определяется состоянием флагов процессора.

Перед использованием условных переходов необходимо выполнить инструкцию CMP (сравнение), которая вычитает второй операнд из первого без сохранения результата, но устанавливает соответствующие флаги. Анализ этих флагов позволяет определить отношение между операндами.

Основные условные переходы для знаковых чисел включают:

* JE/JZ - переход при равенстве (ZF=1)
* JNE/JNZ - переход при неравенстве (ZF=0)
* JG/JNLE - переход если больше (для знаковых чисел)
* JL/JNGE - переход если меньше (для знаковых чисел)
* JGE/JNL - переход если больше или равно
* JLE/JNG - переход если меньше или равно

Для работы с беззнаковыми числами используются переходы JA/JB, а для анализа отдельных флагов - JC/JNC (анализ флага переноса CF) и JNS/JS (анализ флага знака SF).

В данной лабораторной работе условные переходы применяются для реализации логики функций fn1, fn2 и fn3 на основе сравнения значений a, b и k с использованием соответствующих условий переходов.

**Выполнение работы.**

1. Были преобразованы формулы, по которым будут выполняться вычисления:

i1 = 15-2\*i = 15-(i << 1), при a > b

i1 = 3\*i + 4 = 4 + i + (i << 1), при a <= b

i2 = -(4\*i+3) = -3 - (i << 2), при a > b

i2 = 6\*i-10 = -10 + (i << 2) + (i << 1), при a <= b

res = min(i1, i2), при k == 0: res = i1, если i1 < i2, иначе res = i2

res = max(i1,i2), при k != 0: res = i1, если i1 > i2, иначе res = i2

2. В табличном процессоре (LibreOffice Calc) был создан 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 трёх ячейках — i1, i2, res, вычисленные по преобразованным формулам

Рисунок 1 — Тест №1 тестирование преобразования формул

Рисунок 2 — Тест №2 тестирование преобразования формул

Рисунок 3 — Тест №3 тестирование преобразования формул

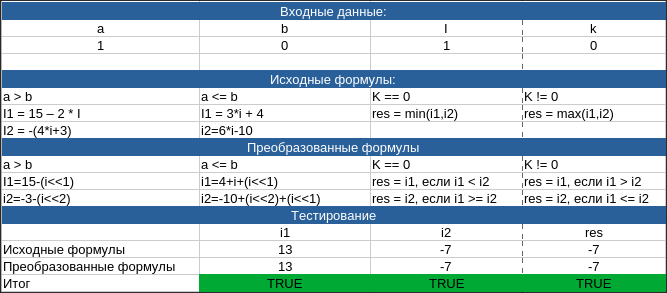
Рисунок 4 — Тест №4 тестирование преобразования формул

Рисунок 5 — Тест №5 тестирование преобразования формул

Рисунок 6 — Тест №6 тестирование преобразования формул

Рисунок 7 — Тест №7 тестирование преобразования формул

Результаты по исходным и преобразованным формулам совпадают.

3. Была разработана программа, в коде были выделены части, посвящённые вычислению итоговых значений. После каждой вычислительной команды было указано полученное к текущему моменту значение. Количество команд в главной вычислительной части составило 27.

4. Программа была протестирована

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Входные данные: | | | | Результаты работы программы: | | |
| a | b | i | k | i1 | i2 | res |
| 0 | 1 | 1 | 1 | 7 | -4 | 7 |
| 0 | 1 | 1 | 0 | 7 | -4 | -4 |
| 1 | 0 | 1 | 1 | 13 | -7 | 13 |
| 1 | 0 | 1 | 0 | 13 | -7 | -7 |
| 5 | 3 | 2 | 0 | 11 | -11 | -11 |
| 5 | 5 | 2 | 0 | 10 | 2 | 2 |
| 7 | 7 | 2 | 1 | 10 | 2 | 10 |

**Вывод**

В ходе лабораторной работы была успешно разработана программа на языке ассемблера iX86, реализующая вычисление условных выражений с ветвлениями. Программа корректно вычисляет функции fn1, fn2 и fn3 с использованием арифметических сдвигов вместо умножения, обрабатывает все возможные комбинации параметров и прошла полное тестирование. Главная вычислительная часть содержит 27 команд и демонстрирует эффективную организацию ветвлений в соответствии с требованиями задания.

**Приложение**

*Код программы LR3.ASM:*

; Лабораторная работа №3, вариант 1 (шифр 1.2.1)

; Исходные данные: a, b, i, k (16-бит)

; Результат: res

.MODEL SMALL

.STACK 100h

.DATA

a DW 4 ; Значение a

b DW 5 ; Значение b

i DW 1 ; Значение i

k DW 0 ; Значение k

i1 DW 4 ; Результат fn1

i2 DW -10 ; Результат fn1

res DW ? ; Результат fn3

.CODE

start:

; Инициализация сегментного регистра

mov ax, @DATA

mov ds, ax

; ========== Начало вычислительной части ==========

mov ax, i ; ax = i

shl ax, 1 ; ax = 2\*i = i<<1

; Сравнение a и b

mov cx, a ; cx = a

cmp cx, b ; сравнить a и b

jg a\_greater\_b ; если a > b, перейти к a\_greater\_b

; Ветка a <= b: Вычисление i1 = 3\*i + 4, i2 = 6\*i - 10

add ax, i ; ax = 3 \* i

add i1, ax ; i1 = 3\*i + 4

shl ax, 1 ; ax = 6 \* i

add i2, ax ; i2 = 6 \* i - 10

; Перейти к вычислению fn3

jmp calc\_fn3

a\_greater\_b:

; Ветка a > b: Вычисление i1 = 15 - 2\*i, i2 = -(4\*i + 3)

add i1, 11 ; i1 = 15

sub i1, ax ; i1 = 15 - 2\*i

sub i2, 13 ; i2 = -3

shl ax, 1 ; ax = 4\*i

sub i2, ax ; i2 = -3 - 4\*i

calc\_fn3:

; Вычисление fn3 в зависимости от k

mov cx, i1 ; cx = i1

mov ax, i2 ; ax = i2

cmp k, 0 ; сравнить k с 0

jz k\_zero; если k = 0, перейти к k\_zero

; Ветка k != 0

; Вычисление fn3 = max(i1,i2)

cmp cx, i2 ; сравнить i1 с i2

jg store\_i1 ; i1 > i2 сохраняем i1

jmp store\_res ; i1 <= i2 сохраняем i2

k\_zero:

; Ветка k == 0

; Вычисление fn3 = min(i1,i2)

cmp cx, i2 ; сравнить i1 с i2

jl store\_i1 ; i1 < i2 сохраняем i1

jmp store\_res ; i1 >= i2 сохраняем i2

store\_i1:

mov ax, i1 ; ax = i1

store\_res:

mov res, ax ; сохранить конечный результат

; ========== Конец вычислительнйо части (КОЛИЧЕСТВО КОМАНД = 27) ==========

end\_prog:

; Завершение программы

mov ax, 4C00h ; Функция завершения программы DOS

int 21h ; Вызов прерывания DOS

end start