**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №3**

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

Тема: Изучение организации ветвлений в программах на языке ассемблера

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Дубровин Д.Н. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2024

**Цель работы.**

Изучение организации ветвлений в программах на языке Ассемблера процессора Intel X86.

**Основные теоретические положения.**

**label (метка)** - это символическое имя, присваиваемое определенному месту в коде. Метки используются как цели для переходов и как ссылки на адреса памяти.

**Безусловный переход (JMP):**

Инструкция JMP безусловно передает управление на указанную метку или адрес памяти. Она не зависит ни от какого условия и всегда выполняет переход.

**Условные переходы:**

Условные переходы передают управление на указанную метку или адрес памяти в зависимости от результата предыдущего сравнения (например, CMP).

**Примеры инструкций условного перехода:**

**JE (Jump if Equal):** Выполняет переход, если предыдущее сравнение привело к равенству.

**JNE (Jump if Not Equal):** Переход, если предыдущее сравнение привело к неравенству.

**JG (Jump if Greater):** Переход, если в результате предыдущего сравнения первый операнд оказался больше второго операнда.

**JL (Jump if Less):** Переход, если в результате предыдущего сравнения первый операнд оказался меньше второго операнда.

**JGE (Jump if Greater or Equal):** Переход, если в результате предыдущего сравнения первый операнд оказался больше или равен второму операнду.

**JLE (Jump if Less or Equal):** Выполняет переход, если в результате предыдущего сравнения первый операнд меньше или равен второму операнду.

**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 режиме отладки. При этом следует рассмотреть все возможные комбинации параметров a,b и k, позволяющие проверить различные маршруты выполнения программы.

Замечания:

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

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 7. Шифр 1.8.7

/15-2\*i, при a>b

f1 = <

\ 3\*i+4, при a<=b

/-(6\*i+8), при a>b

f8 = <

\ 9-3\*(i-1),при a<=b

/ (|i1|+|i2|), при k<0

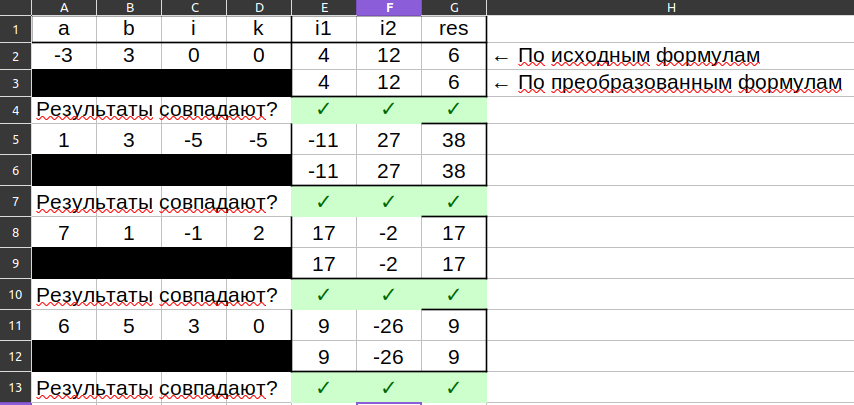
f7 = <

\ max(6,|i1|), при k>=0

**Выполнение работы.**

1. Было выполнено преобразование формулы 9-3\*(i-1) в 12-3\*i для уменьшения количества операций.

2. С помощью программы LibreOffice Calc были вычислены тестовые значения и проверена замена формулы (см. Рисунок 1)

Рисунок 1

3. Разработана программа с минимальной длиной кода. Код программы находится в приложении.

4. Программа протестирована на всех возможных путях выполнения вычислений. Результаты занесены в Табл. 1

Табл. 1

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| a= -3  b=3  i=0  k=0 | i1 = 0004 i2 = 000C (1210) res = 0006 |
| a=1  b=3  i=-5  k=-5 | i1 = FFF5 (-1110)  i2 = 001B (2710)  res = 0026 (3810) |
| a=7  b=1  i=-1  k=2 | i1 = 0011 (1710)  i2 = FFFE (-210)  res = 0011 (1710) |
| a=6  b=5  i=3  k=0 | i1 = 0009  i2 = FFE6 (-2610)  res = 0009 |

**Вывод.**

В результате выполнения лабораторной работы были изучены основные арифметические операции, а также операции условного перехода, за счёт которых было реализовано ветвление. Программа была протестирована на различных входных данных.

**ПРИЛОЖЕНИЕ**

Файл LR3.asm

DOSSEG

.MODEL SMALL

.STACK 100H

.DATA

a DW 6

b DW 5

i DW 3

k DW 0

i1 DW ?

i2 DW ?

res DW ?

.CODE

mov ax, @data

mov ds, ax

; fn1 = if a>b: 15-2\*i else: 3\*i+4

; fn2 = if a>b: -(6\*i+ 8) else: 9-3\*(i-1) = 12-3\*i

; fn3 = if k<0: |i1|+|i2| else: max(6, |i1|)

; 37 операций в вычислительной части

mov ax, i ; ax = i

sal ax, 1 ; ax = 2\*i

mov dx, a ; dx = a

cmp dx, b

jg A\_gt\_B

;a<=b

add ax, i ; ax = 3\*i

;fn1: 3\*i+4

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

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

;fn2: 9-3\*(i-1) = 12-3\*i

sub ax, 16 ; bx = 3\*i - 12

neg ax ; bx = 12 - 3\*i

mov i2, ax ; i2 = 12 - 3\*i

jmp f1\_f2\_end

;a>b

A\_gt\_B:

;fn1: 15-2\*i

mov bx, 15 ; bx = 15

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

mov i1, bx ; i1 = 15 - 2\*i

;fn2: -(6\*i+8) = -6\*i-8

sub bx, 19 ; bx = -4 - 2\*i

sub bx, i ; bx = -4 - 3\*i

sal bx, 1 ; bx = -8 - 6\*i

mov i2, bx ; i2 = -(6\*i + 8)

f1\_f2\_end:

mov bx, i1 ; bx = i1

get\_abs\_i1:

neg bx

js get\_abs\_i1 ; bx = |i1|

mov ax, k ; ax = k

cmp ax, 0

jl K\_lt\_0

;k>=0

;fn3: max(6,|i1|)

cmp bx, 6

jl six\_is\_max

mov res, bx ; res = |i1|

jmp quit

six\_is\_max:

mov ax, 6 ; ax = 6

mov res, ax ; res = 6

jmp quit

;k<0

K\_lt\_0:

;fn3: |i1|+|i2|

mov ax, i2 ; ax = i2

get\_abs\_i2:

neg ax

js get\_abs\_i2 ; ax = |i2|

add bx, ax ; bx = |i1| + |i2|

mov res, bx ; res = |i1| + |i2|

quit:

mov ax, i1 ; Проверка

mov bx, i2 ; итоговых

mov cx, res ; значений

mov ah, 4ch

int 21h

END