**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. 3382 | Копасова К. А. |
| Преподаватель | Фирсов М. А. |

Санкт-Петербург

2024

**Цель работы:** Изучить организацию ветвлений в программах на ассемблере Intel X86 и разработать программу, которая по заданным целым значениям a, b, I и k вычисляет значения i1, i2 и res.

**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. (шифр задания 1.7.6)

Таблица 1:

/ 15 - 2 \* i, при a > b

f1 = <

\ 3 \* i + 4, при a <= b

/ - (4 \* i - 5), при a > b

f7 = <

\ 10 - 3 \* i, при a <= b

Таблица 2:

/ (|i1| - |i2|), при k < 0

f6 = <

\ max(7, |i2|), при k >= 0

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.

**Теоретический материал**

Организация ветвлений — это один из ключевых аспектов программирования на языке ассемблера, который позволяет изменять поток выполнения программы в зависимости от определённых условий. Ветвление делает программы гибкими и позволяет им принимать решения на основе значений данных или состояния процессора.

*Основные виды ветвлений:*

Безусловные переходы (Jumps) — изменения потока выполнения без проверки условий.

Условные переходы (Conditional Jumps) — изменяют выполнение программы на основе флагов процессора (например, флага нуля, переноса и т.д.). Эти флаги устанавливаются в результате выполнения инструкций, таких как арифметические операции или сравнения.

*Безусловные переходы*

Команда **JMP** заставляет процессор перейти к другой части программы без каких-либо условий.

*Условные переходы*

Команда **CMP** сравнивает два операнда, выполняя по сути вычитание одного из другого, но не сохраняет результат. Она только изменяет флаги процессора.

Пример:

CMP AX, BX

После выполнения команды CMP, флаги процессора устанавливаются в зависимости от результата сравнения:

1. Флаг нуля ZF устанавливается, если AX = BX
2. Флаг знака SF устанавливается, если результат отрицательный. При этом установка флага НЕ означает, что AX обязательно меньше BX. Если AX = 7FFFh (32767) и BX = 0FFFFh (-1), то вычитание AX - BX дает отрицательное значение 8000h (поэтому будет установлен флаг знака). Другой пример: есть беззнаковые операнды AX = 0FFFFh и BX = 1. Здесь AX больше, чем BX, но их разница составляет 0FFFEh, что по-прежнему отрицательно. Поэтому для сравнения двух значений со знаком надо использовать вместе два флага - флаг знака и флаг переполнения:

а) Если ((SF = 0) и (OF = 1)) или ((SF = 1) и (OF = 0)), тогда AX меньше, чем BX

б) Если ((SF = 0) и (OF = 0)) или ((SF = 1) и (OF = 1)), тогда AX больше или равно BX

Таким образом, если флаги SF и OF не равны, то AX меньше, чем BX. Если эти флаги равны, тогда AX больше или равно BX

1. Флаг переполнения OF устанавливается, если при вычитании произойдет переполнение знакового бита.
2. Флаг переноса CF устанавливается, если при вычитании AX - BX потребуется заимствование.

Для упрощения работы с переходами в ассемблере есть ряд инструкций, которые позволяют более точно определить результат сравнения и выполнить переход на определённую метку:

**je**: проверяет условие ZF == 1 и выполняет переход, если оба операнда равны. Фактически эквивалентна инструкции jz.

**jne**: проверяет условие ZF == 0 и выполняет переход, если оба операнда НЕ равны. Фактически эквивалентна инструкции jnz.

**ja** / **jnbe**: проверяет одновременно два условия СF == 0 и ZF == 0 (оба условия должны быть истинными). Выполняет переход, если первый операнд больше второго. Оба операнда беззнаковые.

**jae** / **jnb**: проверяет условие СF == 0 и выполняет переход, если первый операнд больше или равен второму. Оба операнда беззнаковые. Аналогичен инструкции jnc.

**jb** / **jnae**: проверяет условие СF == 1 и выполняет переход, если первый операнд меньше второго. Оба операнда беззнаковые. Аналогичен инструкции jc.

**jbe** / **jna**: проверяет одновременно два условия СF == 1 и ZF == 1 (достаточно, чтобы выполнялось хотя бы одно из этих условий). Выполняет переход, если первый операнд меньше или равен второму. Оба операнда беззнаковые.

**jg** / **jnle**: проверяет одновременно два условия SF == OF и ZF == 0 (оба условия должны быть истинными). Выполняет переход, если первый операнд больше второго. Оба операнда со знаком.

**jge** / **jnl**: проверяет условие SF == OF и выполняет переход, если первый операнд больше или равен второму. Оба операнда со знаком.

**jl** / **jnge**: проверяет условие SF != OF (флаги SF и OF не должны быть равны) и выполняет переход, если первый операнд меньше второго. Оба операнда со знаком.

**jle** / **jng**: проверяет одновременно два условия SF != OF и ZF == 1 (достаточно, чтобы выполнялось хотя бы одно из этих условий). Выполняет переход, если первый операнд меньше или равен второму. Оба операнда со знаком.

Данные условия зачастую используются в условных конструкциях и циклах.

**Выполнение работы**

1. Преобразуем формулы, по которым будут выполнятся вычисления, согласно замечанию 5:

Вариант заданий 6: 1.7.6

/ 15 - 2 \* i, при a > b

f1 = <

\ 3 \* i + 4, при a <= b

/ - (4 \* i - 5), при a > b / -4 \* i + 5, при a > b

f7 = < = <

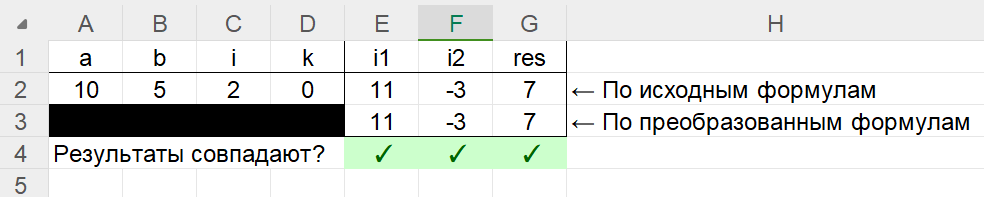
\ 10 - 3 \* i, при a <= b \ 10 - 3 \* i, при a <= b

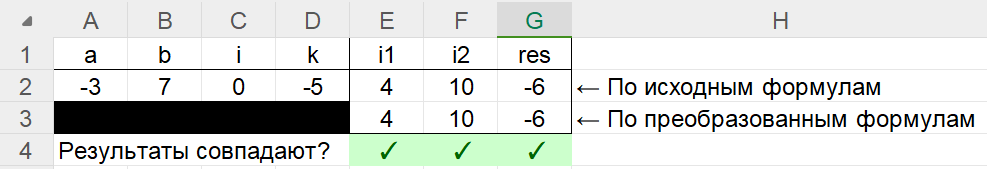
/ (|i1| - |i2|), при k < 0

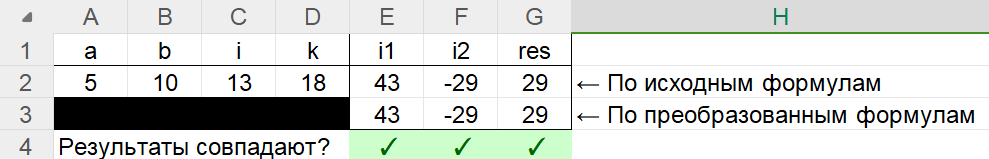
f6 = <

\ max(7, |i2|), при k >= 0

1. Создадим документ, в котором 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. Проверим, что результаты совпадают для разных входных данных.







Результаты для разных входных данных совпадают, следовательно формулы аналогичны.

1. Разработаем программу: в коде выделены части, посвящённые вычислению итоговых значений (3 штуки) и ключевых промежуточных значений (|i1|, |i2|). Код написан с комментариями к каждой команде. Подсчитано количество команд в главной вычислительной части, они указаны в комментарии.
2. Тестирование программы:

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Входные данные | Выходные данные | Комментарий |
| 1 | a = 5, b = A, i = D, k = FFFD | i1 = 2B, i2 = FFE3, res = E | Верно |
| 2 | a = 3, b = 8, i = 2, k = A | i1 = A, i2 = 4, res = 7 | Верно |
| 3 | a = D, b = 7, i = 5, k = 0 | i1 = 5, i2 = FFF1, res = F | Верно |
| 4 | a = 3, b = 2, i = 2, k = 0 | i1 = B, i2 = FFFD, res = 7 | Верно |
| 5 | a = 0, b = 0, i = 0, k = 0 | i1 = 4, i2 = A, res = A | Верно |
| 6 | a = FFE2, b = FFC1, i = FFFD, k = FFE1 | i1 = 15, i2 = 11, res = 4 | Верно |

**Вывод**

В ходе лабораторной работы удалось изучить организацию ветвлений в программах на ассемблере Intel X86 и разработать программу, которая по заданным целым значениям a, b, I и k вычисляет значения i1, i2 и res в соответствии с собственным вариантом.

**Приложение 1**

Файл my\_program.asm:

.model small

.stack 100h

.data

a DW 0

b DW 0

i DW 0

k DW 0

i1 DW 4

i2 DW 5

res DW 0

.code

main proc

;инициализируем сегмент данных

mov ax, @data

mov ds, ax ; устанавливаем регистр ds на сегмент данных

mov ax, i ; помещаем значение i в ax

shl ax, 1 ; ax = 2i

mov bx, a ; перемещаем а в ах

cmp bx, b ; сравниваем a и b

jGe else\_if1 jae else\_if1 ; если a <= b переходим к else\_if1

; если а > b

; ---- СЧИТАЕМ i1 ----

sub i1, ax ; вычитаем 2i из i1 для отрицательного значения (i1 = 4 - 2i)

add i1, 11; прибавляем к значению в i1 11 (i1 = 4 - 2i + 11 = 15 - 2i)

; ---- СЧИТАЕМ i2 ----

shl ax, 1 ; сдвигаем aх на 1 бит (получаем, что ax = 4i)

sub i2, ax ; вычитаем 4i из 5 для отрицательного значения i2 = -4i + 5)

jmp end\_if1 ; переходим на end\_if1

else\_if1:

; если a <= b

; ---- СЧИТАЕМ i1 ----

add ax, i ; получаем, что ax = 3i

add i1, ax ; добавляем к i1 значение ax (3i) (получаем, что i1 = 4 + 3i)

; ---- СЧИТАЕМ i2 ----

sub i2, ax ; инвертируем для отрицательного значения (i2 = 5 - 3i)

add i2, 5 ; прибавляем к i2 5 (i2 = 5 - 3i + 5 = 10 - 3i )

end\_if1:

;--- ОПРЕДЕЛИМ |i1| ----

mov bx, i1 ; перемещаем значение i1 в bx

abs\_i1:

neg bx ; меняем знак

js abs\_i1 ; если значение bx меньше нуля, то снова меняем знак

;---- ОПРЕДЕЛИМ |i2| ----

mov cx, i2 ; перемещаем значение i2 в cx

abs\_i2:

neg cx ; меняем знак

js abs\_i2 ; если значение сх меньше нуля, то снова меняем знак

;---- ОПРЕДЕЛИМ res ----

mov res, cx ; res = |i2|

cmp k, 0 ; сравниваем k и 0

jge else\_if2 ;если k >= 0 - переходим в еlse\_if2

; k < 0

sub res, bx ;вычитаем из res значение bx(|i1|) (res = |i2|-|i1|)

neg res ; берем обратное значение разницы (res = -(|i2|-|i1|) = |i1| - |i2|)

jmp end\_if2 ; перемещаемся в end\_if3

else\_if2:

; k >= 0

cmp cx, 7 ; сравниваем |i2| и 7

jge max ; если значение в cx(|i2|) >= 7, то не меняем cx и переходим сразу в max

mov res, 7 ; если в cx < 7, то меняем cx на 7

max:

;|i2| >= 7

end\_if2: ; 29 команд в вычислительной части

mov ah, 4Ch

int 21h

main endp

end