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

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Лутфулин Д.А. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2024

## **Цель работы.**

Разработать на языке Ассемблера программу, которая по заданным целочисленным значениям параметров вычисляет значения функций.

* 1. 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:13  
Шифр задания:2.8.3

/-(4\*i+3), при a>b

fn1 = f2 = <

\ 6\*i-10 , при a<=b

/-(6\*i+8), при a>b

fn2 = f8 = <

\ 9-3\*(i-1),при a<=b

/ max(i1,10-i2), при k<0

fn3 = f3 = <

\ |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 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.

Основные теоретические положения

Для выполнения данного задания были использованы такие команды как:

1. 1) mov – перемещение значения правого операнда в левый

2) add – добавляет к левому операнду значение правого

3) sub – вычитает из левого операнда значение правого

4) cmp – сравнение

5) neg – смена знака

6) sal – арифметический сдвиг влево (с единицей эквивалентен умножению на 2)

7) sar – арифметический сдвиг вправо (с единицей эквивалентен делению на 2)

8) jmp - безусловный переход

9) jnle(Jump Not Less or Equal), jg (Jump Great) - выполняет короткий переход, если левый операнд больше правого при сравнении.

10) jne (jump greater equal) - выполняет короткий переход, если первый операнд не равен второму при выполнении операции сравнения с помощью команды cmp.

11) jl(jump less) - выполняет короткий переход, если первый операнд меньше второго операнда при выполнении операции сравнения с помощью команды cmp.

12) js (jump sign) – выполняет короткий переход если флаг SF = 1(результат выполнения последней операции имеет знак).

13) int - вызов программного прерывания.

* 1. Выполнение работы

1. Преобразованы следующие формулы:

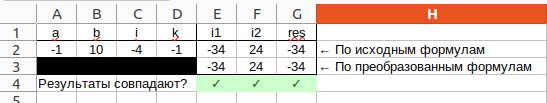
fn1 = 6\*i-10 при a<=b преобразована в -((-3i+5)\*2)

fn2 = 9-3\*(i-1) при a<=b преобразована в -3\*i +12

fn1 = -(4\*i+3), при a>b преобразована в -3i -I -3

fn 2 = -(6\*i+8), при a>b преобразована в -3i \* 2 - 8

2. В табличном процессоре создан документ, проверяющий правильность работы программы. В строке 3 приведены преобразованные формулы для i1 и i2. Формула res копирует изначальную.

 3. Разработана программа, написаны подробные комментарии. Перед вычислением f1 и f2 – 8 команд. В f1 - 5 команд, в f2 — 4, в res — 13. Суммарное количество команд – 30.

4. Программа выполнена в пошаговом режиме под управлением отладчика с фиксацией значений используемых переменных.

Таблица 1 - результат выполнения набора функций.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № теста | Тестируемый случай | Функции для данного случая | Данные | |
| входные | выходные |
| 1 | a > b  k = 0 | f1 -(4\*i+3)  f2 = -(6\*i+8)  f3 = |i1+i2| | a = 2  b = 1  i = 3  k = 0 | i1 = FFF1 (-15)  i2 = FFE6 (-26)  res = 0029 (41) |
| 2 | a > b  k != 0 | f1 -(4\*i+3)  f2 = -(6\*i+8)  f3 = min(i1,i2) | a = 2  b = 1  i = 2  k = 1 | i1 = FFF5 (-11)  i2 = FFEC (-20)  res = FFEC (-20) |
| 3 | a <= b  k = 0 | f1 6\*i-10  f2 = 9-3\*(i-1)  f3 = |i1+i2| | a = 1  b = 1  i = 5  k = 0 | i1 = 0014 (20)  i2 = FFFD (-3)  res = 0011 (17) |
| 4 | a <= b  k != 0 | f1 6\*i-10  f2 = 9-3\*(i-1)  f3 = min(i1,i2) | a = -1  b = 10  i = -4  k = -1 | i1 = FFDE (-34)  i2 = 0018 (24)  res = FFDE |

* 1. Выводы

В ходе выполнения лабораторной работы были получены навыки разработки программы с заданными целочисленными значениями на языке программирования Ассемблер.

1. Приложение А  
   Исходный код программы

Название файла: asm\_lb3.asm

Stack SEGMENT STACK

DW 12 DUP(?)

Stack ENDS

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

CODE SEGMENT

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

\_start:

mov AX,DATA

mov DS,AX

f1\_start:

mov ax, i ;подготавливаем ax (в котором будет i1)

sal ax, 1 ;ax = 2\*i

add ax, i ;ax = 3\*i

neg ax ;ax = -3i

mov bx, a

cmp bx, b ;сравниваем a и b

mov bx, ax ; bx = ax = -3i

jnle a\_more ;если a>b, переход к метке a\_more

a\_not\_more: ;вычисляем f1 = 6\*i - 10 при a <= b

add ax, 5 ;ax = -3\*i +5

sal ax, 1 ;ax = (-3\*i +5)\*2 = -6\*i+10

neg ax ;ax = -(-6\*i+10) = 6i-10 = f1

f2\_a\_not\_more:

add bx, 12 ;bx = -(3\*i) - 12 = f2

jmp f3\_start

a\_more: ;вычисляем f1 = -(4\*i+3) = -4i-3, при a>b

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

sub ax, 3 ;ax = -4i-3 = f1

f2\_a\_more: ;f2 = -(6\*i+8) = -6i-8

sal bx, 1 ;bx = (-3i)\*2 = -6i

sub bx, 8 ;bx = -6i-8

f3\_start:

mov i1, ax ;кладем в i1 и i2 значения из ax и bx

mov i2, bx ;(можно пропустить, это для отображения в отладчике)

cmp [k], 0 ;сравниваем k с 0

jne k\_not\_zero ;если k =/= 0, переход на флак k\_not\_zero

k\_zero: ;значение f3 если k = 0: |i1+i2|

add ax, bx ;i1+i2

k\_zero\_mod:

neg ax ;меняем знак. Если результат отрицательный, SF = 1

js k\_zero\_mod ;Переход в метке если знак (SF = 1). Если произойдёт, знак поменяется обратно

jmp program\_end ;Если на пред.строке не было перехода, то в ax положительное значение

k\_not\_zero:

cmp ax, bx ;значение f3 если k \= 0: f3 = min(i1,i2)

jg k\_not\_zero\_i2\_less ;если i1(в ax) > i2(в bx) то в ax должно оказаться i2

jmp program\_end ;если i1 меньше-ничего не делаем, i1 уже в ax

k\_not\_zero\_i2\_less:

mov ax, i2 ;меняем значение в ax с i1 на i2

program\_end:

mov res, ax ;запись результата в rex

mov ah,4ch ;функциz ДОС завершения программы

int 21h

CODE ENDS

END \_start