**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 |  | Кулач Д.В. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2024

**Цель работы.**

Изучение принципов разработки программ на языке ассемблера с использованием условных переходов, арифметических операций и работы с числами со знаком.

**Основные теоретические положения.**

Инструкция CMP (от слова compare - сравнить) позволяет сравнить значения и установить флаги. Благодаря чему мы можем использовать результат сравнения для выполнения условного перехода.

**Команды условного перехода:**

JE - переход при равенстве операндов

JNE - переход при неравенстве операндов

JG/JNLE - переход если первый операнд больше

JL/JNGE - переход если первый операнд меньше

JGE - переход если первый операнд больше или равен

JLE - переход если первый операнд меньше или равен

Команда NEG выполняет изменение знака числа путем вычисления дополнительного кода. Механизм работы: инвертирование всех битов операнда и добавление единицы к результату.

Команда SHL осуществляет логический сдвиг влево. Биты операнда сдвигаются влево на указанное число позиций. Освободившиеся справа позиции заполняются нулями. Один сдвиг влево умножает число на 2, двойной сдвиг - на 4, тройной - на 8.

**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 11. Шифр 2.6.5

/-(4\*i+3), при a>b

f2 = <

\ 6\*i-10 , при a<=b

/ 2\*(i+1)-4,при a>b

f6 = <

\ 5-3\*(i+1),при a<=b

/ min(|i1|,6), при k=0

f5 = <

\ |i1|+|i2|, при k/=0

**Выполнение работы.**

1. Преобразование формулы не имеет смысла.

2. С помощью программы Microsoft Excel были вычислены тестовые значения. Проверка проводилась на четырех различных случаях, каждый из которых задействует разные варианты функций программы. (см. Рисунок 1)

Рисунок 1

3. Была разработана программа, полностью соответствующая условию: код разбит на части, посвященные вычислению итоговых значений. В коде присутствуют подробные комментарии. Количество строк главной вычислительной части указано в комментариях. Код указан в приложении.

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

Табл. 1

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| a = 1  b = 2  i = 3  k = 0 | i1 = 0008 i2 = FFF9 res = 0006 |
| a = 2  b = 10  i = 3  k = 5 | i1 = 0008  i2 = FFF9  res = 0001 |
| a = 10  b = 2  i = 3  k = -1 | i1 = FFF1  i2 = 0003  res = 0012 |
| a = 2  b = 10  i = 3  k = -1 | i1 = 0008  i2 = FFF9  res = 0001 |

**Вывод.**

Были изучены принципы разработки программ на языке ассемблера с использованием условных переходов, арифметических операций и работы с числами со знаком. Реализована программа, в которой применены полученные знания на практике.

# **ПРИЛОЖЕНИЕ**

Файл main.asm

.8086

code SEGMENT

ASSUME CS:code, DS:data, SS:stack

START:

mov ax, data

mov ds, ax

mov dx, a

cmp dx, b ; Сравниваем a и b

mov cx, i ; Загружаем значение переменной i в регистр cx

shl cx, 1 ; cx = 2\*i

jg fn1\_then ; Если a > b, переход к fn1\_greater

; Случай a <= b

add cx, cx ; ax = 4 \* i (добавляем еще 2 \* i)

sub cx, 10 ; ax = 6 \* i - 10

mov i1, cx ; Сохраняем результат в i1

store\_i2:

; Случай a <= b

add cx, i ; cx = 3\*i

neg cx ; Инвертируем знак значения в регистре cx (делаем его отрицательным)

add cx, 2 ; ax = - 3 \* i + 2

jmp store\_res

fn1\_then: ; Случай a > b

shl cx, 1 ; cx = 4\*i (через сдвиг)

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

neg cx ; cx = -(4\*i+3)

fn2\_then: ; Случай a > b

sub cx, 2 ; cx = 2\*i - 2

mov i2, cx ; Сохраняем результат в i2

store\_res:

; ---- Вычисление res = fn3(i1, i2, k) ----

mov ax, k

cmp ax, 0

je fn3\_if\_k\_zero

; Если k != 0:

; res = |i1| + |i2|

get\_abs\_i1:

mov bx, i1 ; bx = i1

neg bx

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

get\_abs\_i2:

mov ax, i2 ;ax = i2

neg ax

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

find\_res:

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

mov res, ax ; res = ax

jmp fn3\_done

fn3\_if\_k\_zero:

; res = min(|i1|, 6)

cmp bx, 6

jle set\_res\_i1 ; Если |i1| <= 6, то res = |i1|

mov bx, 6 ; Иначе res = 6

set\_res\_i1:

mov res, bx ; res = bx

fn3\_done:

mov ax, 4C00h

int 21h

code ENDS

data SEGMENT

; Входные переменные

a dw 10 ; Первое число для сравнения

b dw 2 ; Второе число для сравнения

i dw 3 ; Значение i для вычислений

k dw 5 ; Значение k для условий

s dw 2

; Результирующие переменные

i1 dw 0 ; Результат первой функции

i2 dw 0 ; Результат второй функции

res dw 0 ; Итоговый результат

data ENDS ; 39 строк в главной вычислительной части

; без учета комментариев и названий

stack SEGMENT STACK

db 256 DUP(?)

stack ENDS

END START