**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 |  | Лексин М.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 12. Шифр 2.7.4

/-(4\*i+3), при a>b

f2 = <

\ 6\*i-10, при a<=b

/-(4\*i-5), при a>b

f7 = <

\ 10-3\*i, при a<=b

/ min(|i1|-|i2|,2), при k<0

f4 = <

\ max(-6,-i2), при k>=0

**Выполнение работы.**

1. В связи с тем, что формула для вычисления первой функции f2 при a > b очень похожа на формулу вычисления второй функции f7, было принято решение сократить ее: -(4\*i-5) == i1+8, где i1 – результат выполнения первой функции f2. Так же с целью сократить код и избежать дублирования в третьей функции f4 было принято решение вынести выражение (3\*i) в регистр cx, так как это выражение используется в двух местах программы.

2. С помощью программы Microsoft Excel были вычислены тестовые значения и проверена замена формулы. 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 = 10  b = 2  i = 3  k = 5 | i1 = FFF1 i2 = FFF9 res = 0007 |
| a = 2  b = 10  i = 3  k = 5 | i1 = 0008  i2 = 0001  res = FFFF |
| a = 10  b = 2  i = 3  k = -1 | i1 = FFF1  i2 = FFF9  res = 0002 |
| a = 2  b = 10  i = 3  k = -1 | i1 = 0008  i2 = 0001  res = 0002 |

**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 main2.asm

.8086

code SEGMENT

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

START:

mov ax, data

mov ds, ax

mov dx, a

cmp dx, b

jle a\_leq\_b

a\_greater\_b:

; a > b

; fn1: i1 = -(4\*i + 3)

mov ax, i

shl ax, 1 ; ax = 2\*i

shl ax, 1 ; ax = 4\*i

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

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

mov i1, ax

; fn2: i2 = -(4\*i - 5)

add ax, 8 ; ax = i1 + 8

mov i2, ax

jmp compute\_fn3

a\_leq\_b:

; a <= b

mov ax, i ; ax = i

add ax, ax ; ax = 2\*i

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

mov cx, ax ; cx = 3\*i (для будущих вычислений)

; i1 = 6\*i - 10

shl ax, 1 ; ax = 3\*i \* 2 = 6\*i

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

mov i1, ax

; i2 = 10 - 3\*i

mov ax, 10

sub ax, cx ; ax = 10 - 3\*i

mov i2, ax

compute\_fn3:

mov ax, k

cmp ax, 0

jge k\_geq\_0

k\_lt\_0:

; k < 0

; res = min(|i1 - i2|, 2)

mov ax, i1

sub ax, i2 ; ax = i1 - i2

jge abs\_done ; если ax >= 0, пропускаем neg

neg ax ; ax = |i1 - i2|

abs\_done:

cmp ax, 2

jle fn3\_end

mov ax, 2

jmp fn3\_end

k\_geq\_0:

; k >= 0

; res = max(-6, -i2)

mov ax, i2

neg ax ; ax = -i2

cmp ax, -6

jge fn3\_end

mov ax, -6

fn3\_end:

mov res, ax

mov ax, 4C00h

int 21h

code ENDS ; 47 строк в главной вычислительной части

; без учета комментариев и названий меток

data SEGMENT

; Входные переменные

a dw 2 ; Первое число для сравнения

b dw 10 ; Второе число для сравнения

i dw 3 ; Значение i для вычислений

k dw -1 ; Значение k для условий

; Результирующие переменные

i1 dw 0 ; Результат первой функции

i2 dw 0 ; Результат второй функции

res dw 0 ; Итоговый результат

data ENDS

stack SEGMENT STACK

db 256 DUP(?)

stack ENDS

END START