**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. 3342 |  |  | Мохамед Махмуд |
| Преподаватель |  |  | Кирьянчиков В.А. |

Санкт-Петербург

2024

**Цель работы.**

Изучить организацию ветвлений в программах на языке Ассемблера.

**Задание.**

**Вариант 24.**

Разработать на языке Ассемблер iX86 программу, которая по заданным значениям размером 1 слово вычисляет:

а) значения и

б) значения ,

где вид функций определяется из табл. 1, а функции из табл. 2 по цифрам шифра индивидуального задания: вариант 24 – 5.7.5

Значения 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 , 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 вместо операции умножения следует использовать арифметический сдвиг и, возможно, сложение;

3) не использовать процедуры (в том числе при вычислении функций );

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

**Основные теоретические положения.**

Для корректной обработки данных в ассемблере нужна разветвленность программы. Это достигается использованием условных переходов. Условный переход – это такая команда процессору, при которой в зависимости от состояния регистра флагов производится передача управления по некоторому адресу, иначе говоря прыжок. Этот адрес может быть ближним или дальним. Прыжок считается ближним, если адрес, на который делается прыжок, находится не дальше, чем 128 байт назад и 127 байт вперёд от следующей команды. Дальний прыжок – это прыжок дальше, чем на [-128,127] байт.

Каждая команда изменяет регистр флагов в зависимости от результата своей операции. Обычно перед командой прыжка идёт команда сравнения, которая изменяет регистр флагов в зависимости от результата. Команд сравнения две: cmp и test. Наиболее универсальная cmp. Команда cmp сравнивает два значения (регистр, память, непосредственное значение) и устанавливает флаг нуля Z (zeroflag) если они равны.

Переходы (прыжки) делятся на условные и безусловные:

1. Безусловный переход — это переход, который выполняется всегда. Безусловный переход осуществляется с помощью команды JMP. У этой команды один операнд, который может быть непосредственным адресом, регистром или ячейкой памяти, содержащей адрес.

2. Условный переход осуществляется, если выполняется определённое условие, заданное флагами процессора. Состояние флагов изменяется после выполнения арифметических, логических и некоторых других команд. Если условие не выполняется, то управление переходит к следующей команде. Некоторые команды условных переходов см. в табл. 1.



**Выполнение работы.**

1. Исходные формулы:

Для случая нужно значение , для и , поэтому в регистре будет храниться промежуточное значение .

2. В ds загружается адрес начала сегмента данных. Через отладчик afd в регистры вносятся значения с помощью команды . Командой значения переносятся из регистров в переменные. Промежуточное значение вычисляется с помощью арифметического сдвига влево.

Сравниваются значения и . Если производится переход к метке . Если , то вычисляется , затем в переменную заносится число 10 и из него вычитается . Так вычисляется значение функции . Далее с помощью арифметического сдвига влево вычисляется , в заносится число 6 и вычисляется .

Метка . С помощью арифметического сдвига влево вычисляется , в заносится число 20 и вычисляется . В заносится число 20 и вычисляется .

Далее сравнивается с нулем. Если , то команда меняет знак , то есть берет модуль.

сравнивается с 0. Если , то происходит переход к метке . Если , то сравнивается с 0 и при необходимости команда меняет знак . Далее в регистр заносится значение и к нему прибавляется , полученная сумма записывается в переменную .

Если , то сравнивается с числом 6 и в переменную записывается минимальное из них.

В конце программы в регистр ah вносится номер функции DOS 4ch (завершение программы -- EXIT), затем вызывается прерывание int 21h.

3. Программа была протестирована с рассмотрением всех возможных случаев.

Результаты тестирования см. в табл. 2.

Табл.2 Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Выходные данные | Комментарии |
| 1 | a:=6 (0004h), b:=5(0005h), i:=1(0001h), k:=0(0000h) | 6 (0006h) |  |
| 2 | a:=6 (0004h), b:=5(0005h), i:=2(0002h), k:=-1(FFFFh) | 15 (000Fh) |  |
| 3 | a:=5 (0005h), b:=6(60006h), i:=3(0003h), k:=0(0000h) | 6 (0006h) |  |
| 4 | a:=5 (0005h), b:=5(0005h), i:=4(0004h), k:=-1(FFFFh) | 20 (0014h) |  |

**Вывод.**

Во время выполнения лабораторной работы была изучена организация ветвления в языке Ассемблера, и реализована программа, выполняющая вычисления с использованием ветвлений.

**ПРИЛОЖЕНИЕ 1.**

**ИСХОДНЫЙ КОД.**

Название файла: lr3.asm

.MODEL SMALL

.STACK 100H

.DATA

a DW 0

b DW 0

i DW 0

k DW 0

i1 DW 0

i2 DW 0

res DW 0

.CODE

START:

mov ax, @data

mov ds, ax

mov a, ax

mov b, bx

mov i, cx

mov k, dx

cmp ax, b

jg a\_greater\_b

sal cx, 1 ; i\*2

add cx, i ; 2\*i+i

mov i2, 10

sub i2, cx ; fn2 = 10 - 3\*i

sal cx, 1 ; i\*6

mov i1, 6

sub i1, cx ; fn1 = -(6\*i-6) = 6 - 6\*i

jmp count\_fn3

a\_greater\_b:

sal cx, 1 ; i\*2

sal cx, 1 ; i\*4

mov i1, 20

sub i1, cx ; fn1 = 20 - 4\*i

mov i2, 5

sub i2, cx ; fn2 = -(4\*i-5) = 5-4\*i

count\_fn3:

cmp i1, 0

jge i1\_greater\_0

neg i1 ; |i1|

i1\_greater\_0:

cmp k, 0

je min\_i1\_0

cmp i2, 0

jge i2\_greater\_0

neg i2 ; |i2|

i2\_greater\_0:

mov ax, i1

add ax, i2

mov res, ax

jmp ending

min\_i1\_0:

cmp i1, 6

jl min\_i1

mov res, 6

jmp ending

min\_i1:

mov ax, i1

mov res, ax

ending:

mov ah, 4ch

int 21h

END START