**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 |  | Павлов А.Р. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2024

## Цель работы

Изучить организацию ветвлений в программах на языке ассемблера. Разработать программу с ветвлением.

## 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.

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.

Вариант 14.

Шифр 3.4.2

/ 7-4\*i, при a > b

f3 = <

\ 8-6\*i, при a <= b

/ -(6\*i - 4), при a > b

f4 = <

\  3\*(i+2), при a <= b

/ max(i1, 10-i2), при k < 0

f2 = <

\ |i1-i2|, при k>=0

**Основные теоретические положения**

Алгоритм выполнения программ, заложенный в процессор, обязывает его выполнять инструкции последовательно, следуя порядку, определенному в исходном коде и в исполняемом модуле. Однако программистам нередко требуется изменить этот порядок: обойти определенные участки кода, перейти к другой ветви исполнения или передать управление подпрограмме с намерением вернуться на исходное место после ее завершения. Для реализации этих операций используются команды перехода. Переходы делятся на безусловные, которые осуществляются независимо от каких-либо условий, и условные, выполнение которых зависит от различных условий, таких как результаты сравнений и анализов. Безусловные переходы further подразделяются на простые переходы (без возвращения в исходную точку) и вызовы подпрограмм (с возвратом после завершения).

Регистр флагов (часто именуемый FLAGS) представляет собой эквивалент регистра состояния процессора в других вычислительных системах и содержит информацию о текущем состоянии процессора. Он состоит из шести флагов состояния и трех бит управления, также называемых флагами.

CF (Carry Flag): Флаг переноса, устанавливается, если происходит переполнение при сложении или заимствование при вычитании.

PF (Parity Flag): Флаг четности, устанавливается, если количество единиц в результирующем числе четное.

AF (Adjust Flag): Флаг корректировки, используется для операций BCD (двоично-десятичного кодирования), устанавливается, если при сложении происходит заимствование из младшей половины байта.

ZF (Zero Flag): Флаг нуля, устанавливается, если результат последней операции равен нулю.

SF (Sign Flag): Флаг знака, устанавливается, если результат последней операции отрицателен (старший бит равен 1).

OF (Overflow Flag): Флаг переполнения, устанавливается, если произошло переполнение при выполнении арифметической операции со знаковыми числами.

Кроме того, регистр флагов содержит следующие биты управления:

DF (Direction Flag): Определяет направление обработки строк (инкрементирование или декрементирование адресов).

IF (Interrupt Flag): Разрешает или запрещает обработку аппаратных прерываний.

TF (Trap Flag): Включает режим пошагового выполнения, позволяя процессору приостанавливать выполнение после каждой инструкции.

Флаги состояния обновляются процессором после выполнения каждой инструкции, позволяя в определенной степени оценить результат этой команды (например, определять, является ли результат нулем). Флаги управления дают возможность изменять некоторые условия работы процессора, например, разрешать или блокировать аппаратные прерывания.

Команды условных переходов, основанные на значениях флагов, включают:

jc: выполняет переход к метке, если установлен флаг переноса, что указывает на наличие переполнения при выполнении операции сложения.

jnc: выполняет переход к метке, если флаг переноса не установлен, что говорит о том, что операции не привели к переполнению.

jo: выполняет переход к метке, если установлен флаг переполнения, указывая на то, что произошло переполнение знакового числа.

jno: выполняет переход к метке, если флаг переполнения не установлен, что свидетельствует о нормальном выполнении операций со знаковыми числами.

js: выполняет переход к метке, если установлен флаг знака, что указывает на то, что результат операции отрицательный.

jns: выполняет переход к метке, если флаг знака не установлен, что означает, что результат положительный.

jz: выполняет переход к метке, если установлен флаг нуля, указывая на то, что результат операции равен нулю.

jnz: выполняет переход к метке, если флаг нуля не установлен, что говорит о том, что результат операции не равен нулю.

Таким образом, эти команды позволяют управлять выполнением программы в зависимости от результатов вычислений, что существенно расширяет функциональные возможности программирования на ассемблере.

**ВЫПОЛНЕНИЕ РАБОТЫ**

1. Преобразование формул, по которым будут выполняться вычисления, для минимизации длины кода.

x = (i << 1) + i = 3i

При a > b

f3 = 7 - 4 \* i = 7 - (i << 2)

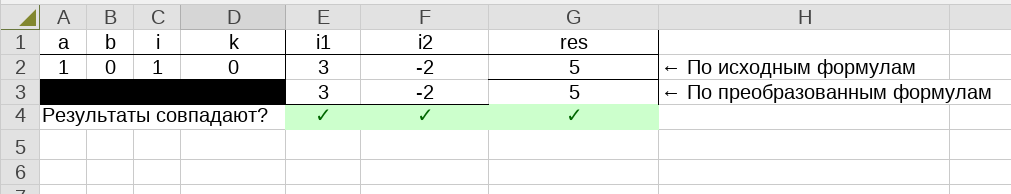
f4 = -(6 \* i - 4) = 4 - 6\*i = (2 - x) << 1

При a <= b

f3 = 8-6\*i = (4 – x) << 1

f4 = 3\*(i+2) = 3\*i + 6 = x + 6

1. Создан 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 трёх ячейках — i1, i2, res, вычисленные по преобразованным формулам.



1. Разработана программа; в коде выделены части, посвящённые вычислению итоговых значений и ключевых промежуточных значений. Код с комментариями: после каждой вычислительной команды указано полученное к текущему моменту значение. Количество команд в главной вычислительной части составило 35.



1. Тестирование программы на различных входных данных. Тесты проверяют все пути выполнения вычислений. Результаты тестирования представлены в Таблице 1.

Таблица 1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Исходные данные | Ожидаемые результаты | Полученный результат |
| 1 | a = 310 = 000316  b = 210 = 000216  i = 510 = 000516  k = -410 = FFFC16 | i1 = -1310 = FFF316  i2 = -2610 = FFE616  res = 3610 = 002416 | i1 = -1310 = FFF316  i2 = -510 = FFE616  res = 210 = 002416 |
| 2 | a = 310 = 000316  b = 410 = 000416  i = 210 = 000216  k = -410 = FFFC16 | i1 = -410 = FFFC16  i2 = 1210 = 000C16  res = -210 = FFFE16 | i1 = -410 = FFFC16  i2 = 1210 = 000C16  res = -210 = FFFE16 |
| 3 | a = 310 = 000316  b = 410 = 000416  i = 210 = 000216  k = 010 = 000016 | i1 = -410 = FFFC16  i2 = 1210 = 000C16  res = 1610 = 001016 | i1 = -410 = FFFC16  i2 = 1210 = 000C16  res = 1610 = 001016 |
| 4 | a = 510 = 000516  b = 210 = 000216  i = 310 = 000316  k = 410 = 000416 | i1 = -510 = FFFB16  i2 = -1410 = FFF2 16  res = 910 = 000916 | i1 = -510 = FFFB16  i2 = -1410 = FFF2 16  res = 910 = 000916 |

## Выводы

В ходе лабораторной работы были изучены организации ветвлений в программах на языке ассемблера. Разработана программа с ветвлением.

# Приложение

Название файла: lr3.asm

DOSSEG

.MODEL SMALL

.STACK 100H

.DATA

a DW 3 ; Значение a

b DW 2 ; Значение b

i DW 5 ; Значение i

k DW -4 ; Значение k

i1 DW ? ; Промежуточное значение i1

i2 DW ? ; Промежуточное значение i2

res DW ? ; Итоговый результат

.CODE

mov AX, @data

mov DS, AX ; Инициализация сегмента данных

; 0

compute\_i\_values: ; Первая часть: вычисление i1 и i2

mov AX, i ; AX = i

shl AX, 1 ; AX = 2 \* i

mov BX, b ; BX = b

cmp a, BX ; Сравнение a и b

jg a\_is\_greater ; Если a > b, переход к a\_is\_greater

a\_less\_or\_equal\_b: ; Если a <= b

add AX, i ; AX = 2i + i = 3i

mov CX, 4 ; CX = 4

sub CX, AX ; CX = 4 - 3i

shl CX, 1 ; CX = 8 - 6i

mov i1, CX ; Сохранить i1 = 8 - 6i

add AX, 6 ; AX = 3i + 6

mov i2, AX ; Сохранить i2 = 3i + 6

jmp compute\_res ; Переход к вычислению res

a\_is\_greater: ; Если a > b

mov CX, 2 ; CX = 2

sub CX, AX ; CX = 2 - 2\*i

sub CX, i ; CX = 2 - 3\*i

shl CX, 1 ; CX = 4 - 6\*i

mov i2, CX ; Сохранить i2 = 4 - 6\*i

shl AX, 1 ; AX = 4i

mov CX, 7 ; CX = 7

sub CX, AX ; CX = 7 - 4\*i

mov i1, CX ; Сохранить i1 = 7 - 4\*i

;

; На этом этапе CX всегда хранит i1

;

compute\_res: ; Вторая часть: вычисление res

mov AX, i2 ; AX = i2

cmp k, 0 ; Сравнение k с 0

jge compute\_abs ; Если k >= 0, переход к compute\_abs

compute\_max: ; Если k < 0

add AX, CX ; AX = i2 + i1

cmp AX, 0Ah ; Сравнение 10 и i2 + i1

jg store\_res ; Если 10 меньше, res = СX = i1

; Если больше (10 - i2 >= i1), res = 10 - i2

; CX = i1 AX = i2 + i1

sub CX, AX ; CX = -i2

add CX, 0Ah ; CX = 10 - i2

jmp store\_res

compute\_abs: ; Если k >= 0

sub CX, AX ; CX = i1 - i2

jns store\_res ; Если результат неотрицательный, store\_res

neg CX ; Если отрицательный, берем модуль |i1 - i2|

store\_res: ; Сохранение результата

mov res, CX ; Сохранить результат res

; 35

mov AH, 4Ch ; Завершение программы

int 21h ; Вызов DOS для завершения программы

END