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 a, b, i, k размером 1 слово вычисляет:

а) значения $i1 = fn1(a, b, i)$ и $fn2(a, b, i)$

б) 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: вариант 24 – 5.7.5

$$f_{5_1} = \begin{matrix} 20 - 4i, & \text{при } a > b \\ -(6i - 6), & \text{при } a \leq b \end{matrix}$$

$$f_{7_2} = \begin{matrix} -(4i - 5), & \text{при } a > b \\ 10 - 3i, & \text{при } a \leq b \end{matrix}$$

$$f_{5_3} = \begin{matrix} \min(|i1|, 6), & \text{при } k = 0 \\ |i1| + |i2|, & \text{при } k \neq 0 \end{matrix}$$

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, i, 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 $fn1, fn2$ вместо операции умножения следует использовать арифметический сдвиг и, возможно, сложение;

3) не использовать процедуры (в том числе при вычислении функций $fn1, fn2$);

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.

Табл. 1. Команды условных переходов.

Команда	Переход, если	Условие перехода
JA	Jump if above	CF = 0 & ZF = 0
JAE	Jump if above or equal	CF = 0
JB	Jump if below	CF = 1
JBE	Jump if below or equal	CF = 1 or ZF = 1
JE	Jump if equal	ZF = 1
JG	Jump if greater (signed)	ZF = 0 & SF = OF
JGE	Jump if greater or equal (signed)	SF = OF
JL	Jump if less (signed)	SF != OF
JLE	Jump if less or equal (signed)	ZF = 1 or SF != OF
JNA	Jump if not above	CF = 0 or ZF = 1
JNAE	Jump if not above or equal	CF = 1
JNB	Jump if not below	CF = 0
JNBE	Jump if not below or equal	CF = 1 & ZF = 0
JNE	Jump if not equal	ZF = 0
JNG	Jump if not greater (signed)	ZF = 1 or SF !=OF
JNGE	Jump if not greater or equal (signed)	SF != OF
JNL	Jump if not less (signed)	SF = OF
JNLE	Jump if not less or equal (signed)	ZF = 0 & SF = OF
JZ	Jump if zero (X = Y)	ZF = 1
JNZ	Jump if not zero (X != Y)	ZF = 0

Выполнение работы.

1. Исходные формулы:

$$f_{5_1} = \begin{matrix} 20 - 4i, & \text{при } a > b \\ -(6i - 6), & \text{при } a \leq b \end{matrix}$$

$$f_{7_2} = \begin{matrix} -(4i - 5), & \text{при } a > b \\ 10 - 3i, & \text{при } a \leq b \end{matrix}$$

$$f_{5_3} = \begin{matrix} \min(|i1|, 6), & \text{при } k = 0 \\ |i1| + |i2|, & \text{при } k \neq 0 \end{matrix}$$

Для случая $a > b$ нужно значение $4i$, для $a \leq b - 3i$ и $6i$, поэтому в регистре cx будет храниться промежуточное значение $2i$.

2. В ds загружается адрес начала сегмента данных. Через отладчик afd в регистры ax, bx, cx, dx вносятся значения a, b, i, k с помощью команды $r\ register\_name = value$. Командой mov значения переносятся из регистров в переменные. Промежуточное значение $2i$ вычисляется с помощью арифметического сдвига влево.

Сравниваются значения a и b. Если $a > b$ производится переход к метке $a\_greater\_b$. Если $a \leq b$, то вычисляется $3i$, затем в переменную $i2$ заносится число 10 и из него вычитается $3i$. Так вычисляется значение

функции $fn2$. Далее с помощью арифметического сдвига влево вычисляется $6i$, в $i1$ заносится число 6 и вычисляется $fn1 = 6 - 6i$.

Метка $a\_greater\_b$. С помощью арифметического сдвига влево вычисляется $4i$, в $i1$ заносится число 20 и вычисляется $fn1 = 20 - 4i$. В $i2$ заносится число 20 и вычисляется $fn2 = 5 - 4i$.

Далее $i1$ сравнивается с нулем. Если $i1 < 0$, то команда neg меняет знак $i1$, то есть берет модуль.

k сравнивается с 0. Если $k = 0$, то происходит переход к метке min _$i1$_0. Если $k \neq 0$, то $i2$ сравнивается с 0 и при необходимости команда neg меняет знак $i2$. Далее в регистр ax заносится значение $|i1|$ и к нему прибавляется $|i2|$, полученная сумма записывается в переменную res.

Если $k = 0$, то $|i1|$ сравнивается с числом 6 и в переменную res 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)	$a > b, k = 0$
2	a:=6 (0004h), b:=5(0005h), i:=2(0002h), k:=-1(FFFFh)	15 (000Fh)	$a > b, k \neq 0$
3	a:=5 (0005h), b:=6(60006h), i:=3(0003h), k:=0(0000h)	6 (0006h)	$a < b, k = 0$
4	a:=5 (0005h), b:=5(0005h), i:=4(0004h), k:=-1(FFFFh)	20 (0014h)	$a = b, k \neq 0$

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
```