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 Ассемблер.

Студент гр. 3384 _____ Рудаков А.Л.

Преподаватель _____ Ковалев А.Д.

Санкт-Петербург

2024

Цель работы.

Познакомиться с уловными операторами в языке Ассемблер. Разработать программу на языке Ассемблер, которая решает поставленные задачи.

Задание.

Разработать на языке Ассемблер iX86 программу, которая по заданным целым значениям a,b,i,k, размером 1 слово, вычисляет:

a) 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.

Функции:

f1 =<
- / -(6*i-4), при a > b
- \ 3*(i+2), при a <= b

f2 =<
- / 20-4*i, при a > b
- \ -(6*i-6), при a <= b

f3 =<
- / (|i1| + |i2|), при k < 0
- \ max(5, |i1|), при k >= 0

Вариант 19

Основные теоретические положения.

Условные переходы в ассемблере реализуются с помощью команд `j..`, где `..` - код условия (например, `j`, `jn`, `jg`, `jl`). Эти команды сравнивают значение регистра флагов с нулем и переходят на указанную метку, если условие выполняется.

Регистр флагов содержит биты, которые устанавливаются в результате выполнения арифметических и логических операций. Например, бит нуля (ZF) устанавливается в 1, если результат операции равен нулю.

Для проверки условия в ассемблере сначала выполняется соответствующая операция, а затем используется команда `j..` для перехода, если условие выполняется.

Команда	**Переход, если**	**Условие перехода**
JZ/JE	нуль или равно	ZF=1
JNZ/JNE	не нуль или не равно	ZF=0
JC/JNAE/JB	есть переполнение/не выше и не равно/ниже	CF=1
JNC/JAE/JNB	нет переполнения/выше или равно/не ниже	CF=0
JP	число единичных бит чётное	PF=1
JNP	число единичных бит нечётное	PF=0
JS	знак равен 1	SF=1
JNS	знак равен 0	SF=0
JO	есть переполнение	OF=1
JNO	нет переполнения	OF=0
JA/JNBE	выше/не ниже и не равно	CF=0 и ZF=0
JNA/JBE	не выше/ниже или равно	CF=1 или ZF=1
JG/JNLE	больше/не меньше и не равно	ZF=0 и SF=OF
JGE/JNL	больше или равно/не меньше	SF=OF
JL/JNGE	меньше/не больше и не равно	SF≠OF

JLE/JNG	меньше или равно/не больше	ZF=1 или SF≠OF
JCXZ	содержимое CX равно нулю	CX=0

2) Арифметические операции в ассемблере реализуются с помощью команд `dd`, `sub`, `mul`, `div` и т.д. Эти команды выполняют соответствующую операцию над значениями в регистрах. Для выполнения арифметической операции в ассемблере сначала загружаются значения в регистры, а затем используется соответствующая команда для выполнения операции.

Выполнение работы.

В начале программы задаются расположение сегментов под DOS, модель памяти small и размер стека.

Далее в сегменте *.data* Объявляются переменные типа *dw: a, b, i, k, i1, i2*, res, которым не задаются начальные значения, а также сообщения для ввода переменных типа *db: msg_a, msg_b, msg_i, msg_k* и сообщения для вывода: *out_i1, out_i2, out_res.*

Далее идет сегмент кода *.code*. Внутри первой и основной процедурой является *main proc*. В ней инициализируется сегмент данных, после чего идет ввод данных.

Ввод осуществляется при помощи процедуры *read_string_to_int*, перед которой вызывается прерывание *int 21h*, со значением 9, благодаря чему происходит вывод строки *msg_** на экран.

Внутри процедуры *read_string_to_int* происходит посимвольное считывание строки, пока не будет достигнут ее конец. Полученные данные в процессе этого преобразуются в число. После процедуры получившееся число записывается в соответствующую ей переменную.

Далее идет сравнение *a* и *b*, если a больше то выполняется блок *f1_a_greater*, внутри которой благодаря сдвигам, вычитанию и замене знака *i1* становится равным *–(6*i-4*), после чего идет переход к блоку *calculate_i2*. Если же *a <= b*, то выполняется блок *f1_a_not_greater*, внутри которого *i1* становится равным *3*(i+2),* после чего переход к *calculate_i2*.

В блоке *calculate_i2* происходит сравнение *a* и *b*, если *a > b*, то выполняется блок *f2_a_greater*, в котором значение *i2* становится *20-4i*, после чего переход к блоку *calculate_res*. В противном случае выполняется блок *f2_a_not_greater*, в котором значение *i2* становится равным *–(6*i-6)*, после чего переход к *calculate_res*.

В блоке *calculate_res* в начале берется модуль *i2*, через вызов *call* блока *abs_i1*, в котором происходит сравнение *i1* с 0. Если меньше, то переходит к *i1_negative*, в котором значение меняется на *-i1* и возвращается к *call* при помощи *ret*, если не меньше, то переходит к *i1_positive* и возвращается через ret. После этого идет сравнение k с 0, если *k < 0*, то выполняется блок *f3_0_greater*, в котором через *call abs_i2* (аналогично, что с *abs_i1*), после чего модули *i1* и *i2* складываются и значение передается в res, после чего переход к блоку output. Если *k >= 0*, то выполняется *f3_0_not_greater*, в котором сравнивается модуль *i1* с 6, если 6 больше, то переходит к *value_6_greater*, записывает в res 6 и переходит к output, если меньше, то в *i1_greater res* присваивается знаенчие *i1* и переходит к output.

В блоке output происходит вывод значений *i1, i2*, res и сообщений перед ними. Сообщения выводятся при помощи прерываний *int 21h* с функцией 9, а значения при помощи процедуры *print_value*, в которой числа преобразуются в строку и выводятся посимвольно.

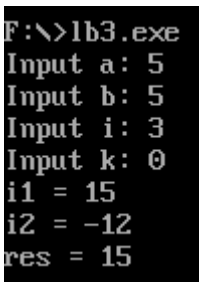

Рисунок 1. Пример ввода/вывода

Программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Input a: 5 Input b: 5 Input i: 3 Input k: 0	i1 = 15 i2 = -12 res = 27	ok
2.	Input a: 0 Input b: 0 Input i: 0 Input k: 0	i1 = 6 i2 = 6 res = 6	ok
3.	Input a: 10 Input b: 20 Input i: 5 Input k: 5	i1 = 21 i2 = -24 res = 21	ok
4.	Input a: 10 Input b: 5 Input i: -5 Input k: -5	i1 = 34 i2 = 40 res = 74	ok
5.	Input a: -1 Input b: -2 Input i: -3 Input k: -4	i1 = 22 i2 = 32 res = 54	ok

Выводы.

В ходе лабораторной работы была написана программа на языке ассемблер, выполняющая функции, указанные в задании при помощи условных операторов, а также использования сдвигов/сложений/вычитаний.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lr3.asm

```
DOSSEG                          ; Задание сегментов под ДОС
.model small                      ; Модель памяти-SMALL(Малая)
.stack 100h                       ; Отвести под Стек 256 байт


.data
    a dw ?
    b dw ?
    i dw ?
    k dw ?
    i1 dw ?
    i2 dw ?
    res dw ?
    msg_a db 'Input a: $'
    msg_b db 'Input b: $'
    msg_i db 'Input i: $'
    msg_k db 'Input k: $'
    out_i1 db 'i1 = $'
    out_i2 db 'i2 = $'
    out_res db 'res = $'

.code
main proc
    mov ax, @data
    mov ds, ax

    ; Input a,b,i,k
    lea dx, msg_a
    mov ah, 9               ; 9 - Display String
    int 21h
    call read_string_to_int
    mov a, ax

    lea dx, msg_b
    mov ah, 9
    int 21h
    call read_string_to_int
    mov b, ax

    lea dx, msg_i
    mov ah, 9
    int 21h
    call read_string_to_int
    mov i, ax

    lea dx, msg_k
    mov ah, 9
    int 21h
    call read_string_to_int
    mov k, ax

    mov ax, a
    mov bx, b
    cmp ax, bx                ; сравнение a и b
    jg f1_a_greater           ; если a > b
    jmp f1_a_not_greater    ; если a <= b
```

```
; Функции задания
f1_a_greater:
    mov ax, i
    shl ax, 1               ; сдвиг на 1 влево (*2)
    mov bx, ax
    shl ax, 1               ; сдвиг на 1 влева (*2) (суммарно *4)
    add ax, bx              ; сумма *4 + *2 = *6
    sub ax, 4               ; вычитание 4
    neg ax                  ; смена знака на -
    mov i1, ax
    jmp calculate_i2

f1_a_not_greater:
    mov ax, i
    add ax, 2               ; добавление 2
    mov bx, ax
    shl ax, 1               ; сдвиг влево на 1 (*2)
    add ax, bx              ; *2 + *1 = *3
    mov i1, ax
    jmp calculate_i2

calculate_i2:
    mov ax, a
    mov bx, b
    cmp ax, bx
    jg f2_a_greater         ; если a > b
    jle f2_a_not_greater    ; если a <= b

f2_a_greater:
    mov ax, i
    shl ax, 1               ; сдвиг на 1 (*2)
    shl ax, 1               ; сдвиг на 1 (*4)
    neg ax                  ; смена знака
    add ax, 20              ; добавлениме 20
    mov i2, ax
    jmp calculate_res

f2_a_not_greater:
    mov ax, i
    shl ax, 1               ; сдвиг на 1 влево (*2)
    mov bx, ax
    shl ax, 1               ; сдвиг на 1 влева (*2) (суммарно *4)
    add ax, bx              ; сумма *4 + *2 = *6
    sub ax, 6               ; вычитание 4
    neg ax                  ; смена знака на -
    mov i2, ax
    jmp calculate_res

calculate_res:
    call abs_i1             ; модуль i1
    cmp k, 0
    jl f3_0_greater         ; если k < 0
    jmp f3_0_not_greater    ; если k >= 0

abs_i1:
    cmp i1, 0
    jl i1_negative
    jmp i1_positive

i1_negative:                ; смена знака
    mov ax, i1
    neg ax
```

```
    mov i1, ax
    ret                     ; возврат к call

i1_positive:
    ret                     ; возврат к call

f3_0_greater:
    call abs_i2             ; модуль i2
    mov ax, i1
    add ax, i2
    mov res, ax
    jmp output

abs_i2:
    cmp i2, 0
    jl i2_negative
    jmp i2_positive

i2_negative:
    mov ax, i2
    neg ax                  ; смена знака
    mov i2, ax
    ret                     ; возврат к call

i2_positive:
    ret                     ; возврат  к call

f3_0_not_greater:
    cmp i1, 6
    jl value_6_greater
    jmp i1_greater

value_6_greater:
    mov res, 6
    jmp output

i1_greater:
    mov ax, i1
    mov res, ax
    jmp output

; Вывод результатов
output:
    lea dx, out_i1
    mov ah, 9
    int 21h
    mov ax, i1
    call print_value

    lea dx, out_i2
    mov ah, 9
    int 21h
    mov ax, i2
    call print_value

    lea dx, out_res
    mov ah, 9
    int 21h
    mov ax, res
    call print_value

    mov ah, 4ch
    int 21h
```

```
main endp

; Ввод числа
read_string_to_int proc
        xor bx, bx              ; обнуление bx
        xor cx, cx              ; обнуление cx (знак)

    read_loop:
        mov ah, 1               ; 1 - Read Character
        int 21h
        cmp al, 13              ; проверка на завершение строки
        je end_read
        cmp al, '-'             ; проверка на знак -
        jne digit_value
         mov cx, 1                ; запись в cx 1 означает, что число
будет отрицательным
        jmp read_loop

    digit_value:
        sub al, '0'             ; вычитание ascii-кода символа '0'
        xor ah, ah              ; обнуление ah
        push ax
        mov ax, 10
        mul bx                  ; умножение bx на 10
        pop bx
        add bx, ax              ; соединяет числа
        jmp read_loop

    end_read:
        mov ax, bx
        cmp cx, 1
        jne exit_read
        neg ax                 ; меняется знак, если число отрицательное

    exit_read:
        ret                     ; возвращается к call
read_string_to_int endp



print_value proc
        push ax
        push bx
        push cx
        push dx

        xor cx, cx               ; очищаем cx
        mov bx, 10              ; добавляем 10 в bx для деления
        test ax, ax             ; проверяем на полложительное
        jns positive_value
        push ax
        mov dl, '-'             ; добавляем -
        mov ah, 2               ; 2 - Вывод на экран
        int 21h
        pop ax
        neg ax                  ; изменение на положительное

    positive_value:
        xor dx, dx              ; обнуление dx
        div bx                  ; деление 10
        push dx                 ; добавляем остаток в стек
```

```
        inc cx                  ; увеличиваем счетчик цифр
        test ax, ax             ; провеярем достижение 0
        jnz positive_value          ; продолжение цикла

    print_loop:
        pop dx                  ; получение текущего значения
        add dl, '0'             ; прибавление ascii-код 0
        mov ah, 2               ; вывод
        int 21h
        loop print_loop

        mov dl, 13              ; символ Carriage Return
        mov ah, 2
        int 21h
        mov dl, 10              ; символ Line Feed
        mov ah, 2
        int 21h

        pop dx
        pop cx
        pop bx
        pop ax
        ret
print_value endp
end main
```