Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №3

Работа со знаковыми числами

Студент: гр. 753502

Саттарова Александра Сергеевна

Руководитель: ассистент Шиманский В.В.

Минск 2018

СОДЕРЖАНИЕ

1. Введение
2. Постановка задачи
3. Программная реализация
4. Выводы

Литература

Приложение 1. Текст программы

1. Введение
   1. Данная программа реализует ввод чисел и вывод результата знаковых вычислений. Вводимые данные могут быть как положительными, так и отрицательными числами.
   2. Цель лабораторной работы — ознакомиться с командами знакового вычисления, а также c командами знакового преобразования значения.
   3. В ходе выполнения лабораторной работы будет использован ассемблер NASM версии 2.13.03-1, работающий на процессорах архитектуры Intel x86. NASM (*Netwide Assembler*) — это свободный (LGPL и лицензия BSD) ассемблер для архитектуры Intel x86, использующийся для написания 16-, 32- и 64-разрядных программ. В NASM используется Intel-синтаксис записи инструкций.
   4. Исполняться написанная работа будет в консоли системы Arch Linux с версией ядра 4.18.16.
2. Постановка задачи
   1. Текст задания

В каждом из заданий переменные a, b, c, d определяются в сегменте данных и имеют размерность двойное слово. Необходимо выполнить над ними заданные арифметические и логические операции с учетом знака (+ или -), а результат вывести в консоль.

При выполнении умножения считаем, что результат вмещается в двойное слово. При выполнении деления считаем, что оно целочисленное, и делитель не равен нулю.

При вводе данных осуществить их проверку на переполнение и отсутствие посторонних символов.

* 1. Условие задания

if (a\*c + b\*d = a\*d + b\*c)

if (a > b)

result = a^2

else if (a > c)

result = (c && b) + min(a, b, c)/(a^2 - c)

else result = a - (b OR c)

else result = a/d

1. Программная реализация
   1. Значения переменных не устанавливаются при объявлении сегмента данных, а вводятся с клавиатуры. Программа разбита при помощи меток на несколько логических частей, которые отвечают за обработку ввода\вывода, а также их обработку.
   2. Результат можно видеть в консоли.
   3. Примеры:
      1. Примеры вычислений:

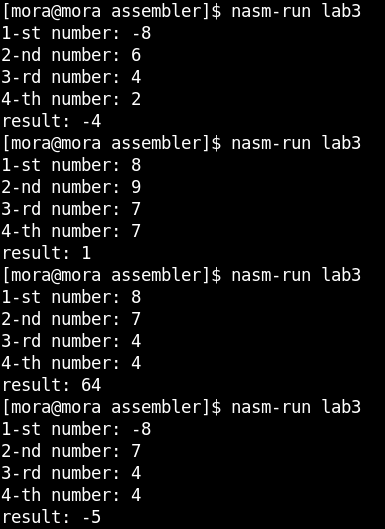


Рис. 1. Примеры вычислений

3.3.2. Случай введения слишком большого числа

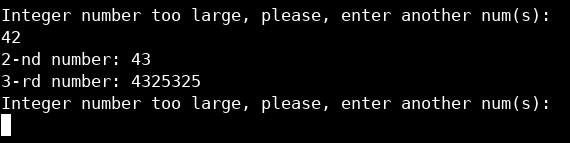


Рис. 2. Ввод слишком большого числа

3.3.3. Проверка на ноль

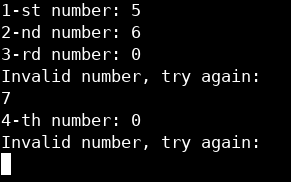
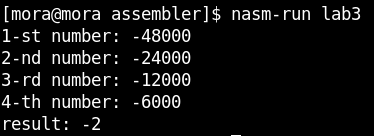


Рис. 3. Ввод нуля

3.3.4. Проверка переполнения при вводе со знаком:

 Рис. 4. Проверка переполнения со знаком

3.3.5. Ввод со знаком «плюс» и «минус»

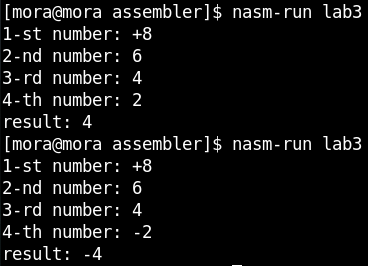


Рис. 5. Ввод со знаком «плюс» и «минус»

4. Выводы

В ходе выполнения лабораторной работы была создана программа на языке ассемблера, поддерживающая консольный ввод и вывод данных, а также знаковые вычисления. Были освоены команды для знакового умножения\деления чисел (imul\idiv), команда neg для изменения числа с положительного на отрицательное (или наоборот), команда cwd, расширяющая знаковый бит AX до DX.

Также были реализованы: проверка вводимых чисел на отсутствие лишних символов, проверка переполнения результата, проверка переполнения числа; был учтен случай введения ведущих нулей. Для каждого случая были написаны соответствующие процедуры.

Литература

1. Learn Assembly Language - <http://asmtutor.com/>

2. Столяров А. В. Программирование на языке ассемблера NASM для ОС Unix: Уч. пособие. - 2-е изд. - М.: МАКС Пресс, 2011. - 188 с.

3. Stack Overflow — https://stackoverflow.com/

4. NASM - Википедия — https://ru.wikipedia.org/wiki/NASM

5. X86 Assembly/Interfacing with Linux - Wikibooks — https://en.wikibooks.org/wiki/X86\_Assembly/Interfacing\_with\_Linux

Приложение 1. Текст программы

; NASM assembly x64, lab 2 (lab 1 with i/o)

;

; nasm -f elf64 lab2.asm && ld -o lab2 lab2.o

;

%define STDIN dword 0

%define STDOUT dword 1

%define BUFSIZE 256

section .data

string\_one: db '1-st number: ', 0

string\_two: db '2-nd number: ', 0

string\_three: db '3-rd number: ', 0

string\_four: db '4-th number: ', 0

string\_res: db 'result: ', 0

string\_error: db 'Invalid number, try again: ', 0

string\_overflow: db 'Integer number too large, please, enter another num(s): ', 0

ten: dw 10

nl: db 10, 0

section .bss

var\_one resb 256

var\_two resb 256

var\_three resb 256

var\_four resb 256

output\_buffer resb 1

readbuf resb BUFSIZE ; buffer for read operations

section .text

global \_start

\_start:

mov rsi, string\_one

call print

; reads first num

call readint

mov [var\_one], rax

mov rsi, string\_two

call print

; reads second num

call readint

mov [var\_two], rax

mov rsi, string\_three

call print

; reads third num

call readint

mov [var\_three], rax

mov rsi, string\_four

call print

; reads fourth num

call readint

mov [var\_four], rax

call calculate

call output

; exit

xor rbx, rbx

mov rax, 1

int 80h

strlen:

mov rax, rsi

.nextchar:

cmp byte [rax], 0

je strlen.finished

inc rax

jmp strlen.nextchar

.finished:

sub rax, rsi

ret

; rax checknumber(rsi) - 1 if number in ESI, 0 if not

checknumber:

push rsi

mov rax, rsi

cmp byte [rax], '-' ; Check whether first symbol is '-'

jne checknumber.loopy

inc rsi

inc rax ; if true, then skip minus

.loopy:

cmp byte [rax], 0 ; if met end

je checknumber.exityes

cmp byte [rax], 0xA ; if met newline

je checknumber.exityes

cmp byte [rax], '0'

jl checknumber.exitno

cmp byte [rax], '9'

jg checknumber.exitno

inc rax

jmp checknumber.loopy

.exityes:

cmp rax, rsi

mov rax, 1

jne checknumber.exit

.exitno:

mov rax, 0

.exit:

pop rsi

ret

; void print(rsi) - prints buffer at ESI

print:

call strlen

mov rdx, rax ; number of bytes

mov rax, 1 ; system call for write

mov rdi, 1 ; file handle 1 is stdout

;mov rsi, message ; address of string to output

syscall

ret

; rax read(rdi) - reads from terminal at EDI, rax shows how many was read

read:

mov rax, 0 ; system call for read

mov rdi, 0 ; file handle 0 is stdin

mov rsi, readbuf ; address of string to input

mov rdx, BUFSIZE ; number of bytes

syscall

ret

; rax calculate(var\_one, var\_two, var\_three , var\_four) - calculates expression (see description)

calculate:

mov ax, [var\_one]

mov cx, [var\_three]

imul cx ;ax = a\*c

mov bx, ax ;bx = a\*c

mov ax, [var\_two]

mov cx, [var\_four]

imul cx ;ax = b\*d

add bx, ax ;bx = a\*c + b\*d

push bx

mov ax, [var\_one]

imul cx ;ax = a\*d

mov bx, ax ;DX = a\*d

mov ax, [var\_two]

mov cx, [var\_three]

imul cx ;ax=b\*c

add bx, ax ;DX = a\*d + b\*c

mov ax, [var\_one]

mov cx, bx

pop bx

cmp bx, cx ;if (a\*c + b\*d == a\*d + b\*c)

jnz else1 ;if they are not equal, go to label else1

mov bx, [var\_two]

cmp ax, bx

jng else2 ;if a < b, go to label else2

imul ax ;print a^2

ret

else2:

mov cx, [var\_three]

cmp ax, cx

jng else3 ;if a < c, go to label else3

;print ((c AND b) + min (a, b, c)/(a^2-c))

call findmin

push ax

mov ax, [var\_one]

imul ax

sub ax, cx ;ax = a^2-c

and cx, bx ;cx = c and b

mov bx, ax

pop ax ;min(a,b,c)

cwd

idiv bx

add ax, cx

ret

else3:

or bx, cx

sub ax, bx ;a-(b or c)

ret

else1:

mov ax, [var\_one]

mov cx, [var\_four]

;mov rdx, 0

cwd

;mov ah, 0

;mov dl, 0

idiv cx ;a/d

ret

; rax findmin(a, b, c) - finds minimal num

findmin:

cmp ax, bx

jg findmin.comp\_b\_c ;if a > b, go to label comp\_b\_c

cmp ax, cx

jg findmin.min\_c ;if a > c, go to min\_c

ret

.comp\_b\_c:

cmp bx, cx

jg findmin.min\_c ;if b > c go to min\_c

mov ax, bx

ret

.min\_c:

mov ax, cx

ret

; void output(rax) - prints result

output:

jo output.fail

mov [output\_buffer], rax

; result

mov rsi, string\_res

call print

mov rax, [output\_buffer]

mov rdi, readbuf

call itoa

mov rsi, rdi

call print

mov rsi, nl

call print

ret

.fail:

mov rsi, string\_overflow

call print

mov rsi, nl

call print

jmp \_start

; rax atoi(esi) - converts string at ESI to number at EAX

atoi:

push rsi

push rcx

push rbx

push rdx

call strlen

mov rcx, rax

mov rdx, 0 ; is negative

mov rax, 0

mov rbx, 0

cmp byte [rsi], '-'

jne .loopy

inc rsi

dec rcx

mov rdx, 1

.loopy:

movzx rcx, byte [esi] ; get a character

inc esi ; ready for next one

cmp rcx, '0' ; valid?

jb .continue

cmp rcx, '9'

ja .continue

sub rcx, '0' ; "convert" character to number

imul rax, 10 ; multiply "result so far" by ten

add rax, rcx ; add in current digit

loop .loopy

.continue:

cmp rdx, 0

je .finish

neg rax

.finish:

pop rdx

pop rbx

pop rcx

pop rsi

ret

; rdi itoa(rax) - convert number from EAX and store it to EDI

itoa:

push rdi

push rdx

push rcx

push rbx

push rax

mov rcx, 0

mov rbx, 0 ; is negative

test ax, ax

jns .loopy

neg rax

inc rcx

mov rbx, 1

.loopy:

inc     rcx

mov     rdx, 0

idiv    word [ten]

add     rdx, '0'

push    rdx

cmp     rax, 0

jg     .loopy

cmp     rbx, 0

je     .loopo

push '-'

.loopo:

pop rax

mov [rdi], al

inc rdi

loop .loopo

    mov     [rdi], byte 0;

pop rax

pop rbx

pop rcx

pop rdx

pop rdi

ret

readint:

;mov rdi, readbuf

;mov rax, BUFSIZE

call read

mov rsi, readbuf

call checknumber

cmp rax, 0

jne .success

mov rsi, string\_error

call print

jmp readint

.success:

mov rsi, readbuf

call atoi

cmp rax, 0

jz .zero\_fail

cmp rax, 0xFFFF

jg .overflow

ret

.overflow:

mov rsi, string\_overflow

call print

mov rsi, nl

call print

jmp readint

.zero\_fail:

mov rsi, string\_error

call print

mov rsi, nl

call print

jmp readint