*«*Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ_____
КАФЕДРА _____КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ_____

О Т Ч Е Т

по лабораторной работе № 3

Дисциплина: Машинно-зависимые языки и основы компиляции

Название лабораторной работы: Программирование ветвлений и итерационных циклов

Студент гр. ИУ6-45Б ____19.03.2024_____ ___И.А.Дулина_____
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ ____С.С.Данилюк_____
(Подпись, дата) (И.О. Фамилия)

Москва, 2024

Вариант 5.6

Цель работы: изучение средств и приемов программирования ветвлений и итерационных циклов на языке ассемблера

Задание: вычислить целочисленное выражение

$$f = \begin{cases} \dfrac{b^2 - 2}{3 + b} & \text{если } \dfrac{a}{b} > 5 \\ \dfrac{a}{b} & \text{иначе} \end{cases}$$

В листинге 1 показан код программы.

Листинг 1 – Код программы

```
section .data
InputA db "Input A", 10
lenMsgA equ $-InputA
InputB db "Input B not equal 0 or -3", 10
lenMsgB equ $-InputB
ResultMsg db "Result = ", 10
lenMsgResult equ $-ResultMsg
ErrorStr db "Error: Invalid input format", 10
lenError equ $-ErrorStr

section .bss
InBuf resb 10 ; буфер для вводимой строки
lenIn equ $-InBuf
OutBuf resb 10
lenOut equ $-OutBuf
A resd 1
B resd 1
F resd 1
section .text

global _start
_start:
;input A
;write
mov eax, 4 ; системная функция 4 (write)
mov ebx, 1 ; дескриптор файла stdout=1
mov ecx, InputA ; адрес выводимой строки
mov edx, lenMsgA ; длина выводимой строки
int 80h ; вызов системной функции
; read
call Buffer
mov esi, InBuf
call StrToInt
cmp ebx, 0
jne Error
mov [A], eax
```

```
;input B
;write
mov eax, 4 ; системная функция 4 (write)
mov ebx, 1 ; дескриптор файла stdout=1
mov ecx, InputB ; адрес выводимой строки
mov edx, lenMsgB ; длина выводимой строки
int 80h ; вызов системной функции
; read
call Buffer
mov esi, InBuf
call StrToInt
cmp ebx, 0
jne Error
cmp eax, 0
je Error
cmp eax, -3
je Error
mov [B], eax

;program
mov eax, [A]
mov ecx, [B]
cdq
idiv ecx ; eax = a/b
cmp eax, 5
jle Cont
mov eax, [B]
imul eax ; B^2
sub eax, 2 ; B^2-2
add ecx, 3 ; 3+B
idiv ecx ; B^2-2/3+B
Cont:
mov [F], eax
;int 80h ; вызов системной функции

;output
mov eax, 4 ; системная функция 4 (write)
mov ebx, 1 ; дескриптор файла stdout=1
mov ecx, ResultMsg ; адрес выводимой строки
mov edx, lenMsgResult ; длина строки
int 80h ; вызов системной функции
;вывод F
mov eax, [F]
mov esi, OutBuf
call IntToStr
mov esi, eax
mov eax, 4 ; системная функция 1 (write)
mov ebx, 1 ; дескриптор файла stdout=1
mov ecx, OutBuf ; адрес буфера
mov edx, esi ; длина буфера
int 80h ; вызов системной функции
jmp Exit
Error:
```

```
    mov eax, 4 ; системная функция 4 (write)
    mov ebx, 1 ; дескриптор файла stdout=1
    mov ecx, ErrorStr ; адрес сообщения об ошибке
    mov edx, lenError ; длина сообщения об ошибке
    int 80h ; вызов системной функции
    jmp Exit
; Выход
Buffer:
mov eax, 3 ; системная функция 3 (read)
mov ebx, 0 ; дескриптор файла stdin=0
mov ecx, InBuf ; адрес буфера ввода
mov edx, lenIn ; размер буфера
int 80h
ret
Exit:
mov eax, 1 ; системная функция 1 (exit)
xor ebx, ebx ; код возврата 0
int 80h ; вызов системной функции
%include "../lib.asm"
```

На рисунке 1 показана схема алгоритма.

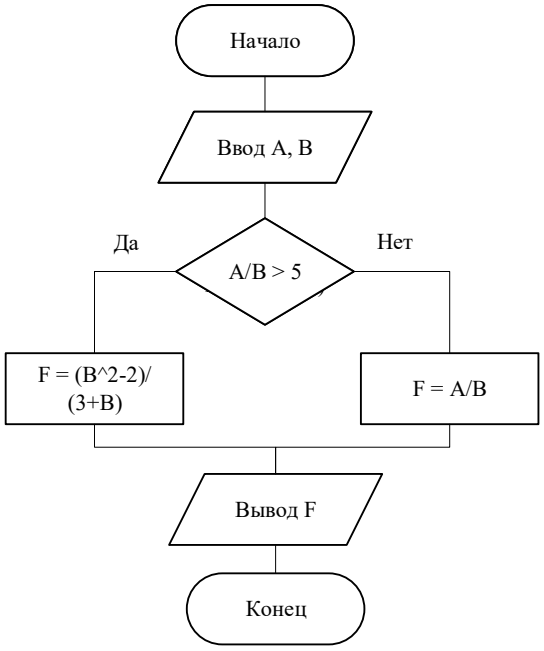

Рисунок 1 – Схема алгоритма

На рисунке 2 показан первый проведенный тест в консоли.

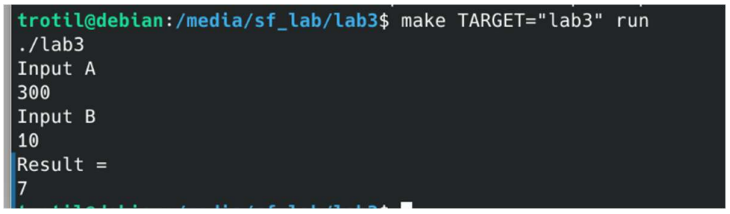

Рисунок 2 – 1 пример работы программы

На рисунке 3 показан второй проведенный тест в консоли.

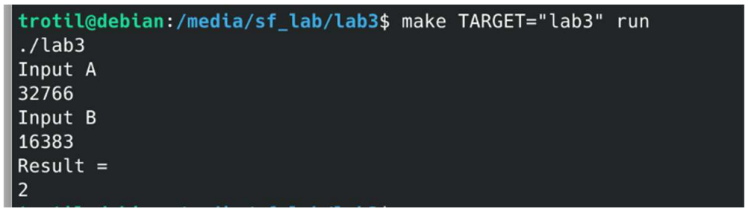

Рисунок 3 – 2 пример работы программы

На рисунке 4 показан второй проведенный тест в консоли.

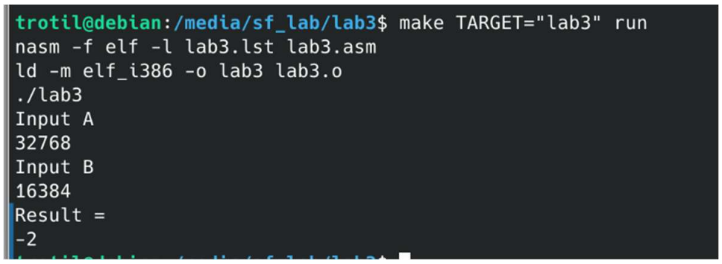

Рисунок 4 – 3 пример работы программы

На рисунке 5 проиллюстрирован пример 3 в дебаггере, а именно состояние регистра EAX после выполнения функции StrToInt, а также data damp, где мы можем видеть число-результат выполнения функции. К сожалению, в библиотеке используются 16-битные регистры (AX: а именно проблема заключается в строчке mul di, при которой мы умножаем наше число 32 768 на 10), поэтому при работе с числами большими, чем 32 767 происходит заполнение единицей знакового бита и на выходе мы получаем не положительное число 00 00 80 0016 = 32 76810, как если бы мы работали с беззнаковыми числами, а ff ff 80 00, или же -32 768.

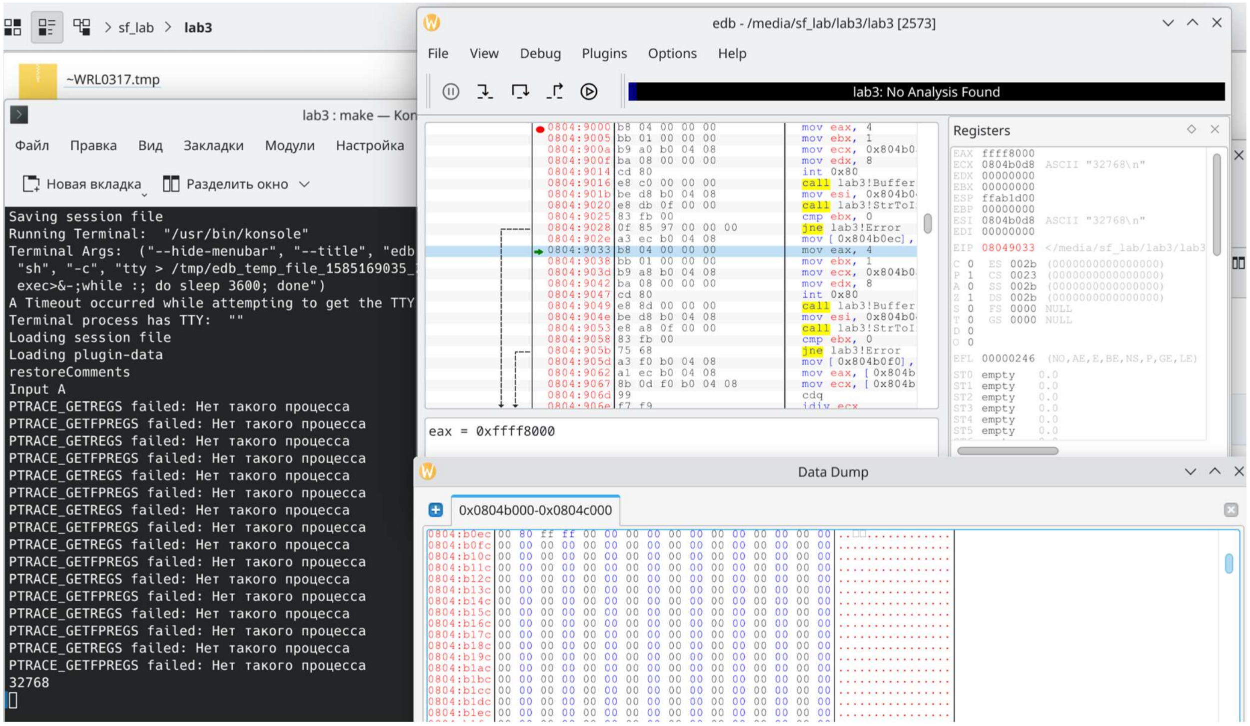

Рисунок 5 – Вид программы в дебаггере edb

В таблице 1 представлено представление чисел в памяти.

Таблица 1 – Представление чисел со знаком и без

Двоичная система	Шестнадцатеричная	Беззнаковые	Знаковые
0000 0000 0000 0000	00 00	0	0
…		…	…
0111 1111 1111 1111	7f ff	32 767	32 767
1000 0000 0000 0000	80 00	32 768 = 2^{15}	-32 768
…		…	…
1111 1111 1111 1111	ff ff	65 535 = 2^{16}-1	-1

В таблице 2 представлены тесты.

Таблица 2 – Тесты

Номер	Исходные данные	Ожидаемый результат	Полученный результат
1	A = 300, B=10	7,538	7
2	A=32766, B=16383	2	2
3	A=32768, B=16384	2	-2

Вывод: были изучены средства и приемы программирования ветвлений и итерационных циклов на языке ассемблера

Контрольные вопросы:

1) Какие машинные команды используют при программировании ветвлений и циклов?

a) Условный переход (Conditional Jump): Эта команда выполняет переход к определенной инструкции в зависимости от значения флагов процессора. Мнемоники условного перехода:

(a) JZ – переход по "ноль";

(b) JE – переход по "равно";

(c) JNZ – переход по "не нуль";

(d) JNE – переход по "не равно";

(e) JL – переход по "меньше";

(f) JNG, JLE – переход по "меньше или равно ";

(g) JG – переход по "больше";

(h) JNL, JGE – переход по "больше или равно ";

(i) JA – переход по "выше" (беззнаковое “больше”);

(j) JNA, JBE – переход по "не выше"(беззнаковое “не больше”);

(k) JB – переход по "ниже" (беззнаковое “меньше”);

(l) JNB, JAE – переход по "не ниже" (беззнаковое “не меньше”).

b) Безусловный переход (Unconditional Jump): Команда выполняет безусловную передачу управления по указанному адресу. Наиболее распространенная команда - JMP.

c) Сравнение (Compare): Команда CMP сравнивает два операнда и устанавливает флаги процессора на основе результата сравнения. Эти флаги могут использоваться для последующих условных переходов.

d) Инструкции цикла (Loop Instructions): Команды цикла, такие как:

(i) LOOP

Выполнение команды: ECX:=ECX-1, если ECX=0, то происходит переход на следующую команду, иначе – короткий (-128..127 байт) переход на метку

(ii) LOOPE – организация цикла с условием

Помимо регистра ECX команды проверяют значение флага ZF: LOOPE осуществляет переход на метку, если ZF=1 & ECX!=0, LOOPNE – если ZF=0 & ECX!=0

е) Ветвление на подпрограмму (Call Instructions): Команды CALL и RET используются для вызова и возврата из подпрограммы или процедуры.

2) Выделите в своей программе фрагмент, реализующий ветвление. Каково назначение каждой машинной команды фрагмента?

В листинге 2 показан фрагмент программы.

Листинг 2 – Фрагмент программы, реализующий ветвление

```
cmp eax, 5
jle Cont
mov eax, [B]
imul eax ; B^2
sub eax, 2 ; B^2-2
add ecx, 3 ; 3+B
idiv ecx ; B^2-2/3+B
Cont:
mov [F], eax
;int 80h ; вызов системной функции
```

Cmp сравнивает значение, находящееся в eax, с литералом 5 и устанавливает флаги, значения которых используются дальше командой jle (меньше или равно) для условной передачи управления на метку Cont, в таком случае реализуется ветвление при не удовлетворении условия, при удовлетворении условия A/B>5 переход не осуществляется и программа выполняет последующие вычисления с общим кодом после метки Cont.

3) Чем вызвана необходимость использования команд безусловной передачи управления?

Необходимость использования команд безусловной передачи управления (Unconditional Jump) обусловлена несколькими факторами:

а) Реализация условий и циклов: Безусловные переходы позволяют программистам реализовывать условные конструкции (if-else) и циклы (for, while, do-while) на уровне ассемблерного кода. Если условие не удовлетворяется, или цикл должен быть завершен, безусловный переход обеспечивает переход к нужному участку кода.

b) Обработка исключений: при обработке ошибок или исключительных ситуаций, возникает необходимость немедленного перехода к определенной обработке ошибки, игнорируя остальной код программы.

4) Поясните последовательность команд, выполняющих операции ввода-вывода в вашей программе. Чем вызвана сложность преобразований данных при выполнении операций ввода-вывода?

- Ввод данных:

В листинге 3 показан фрагмент программы.

Листинг 3 – Фрагмент программы с чтением введенной строки

```
; read
call Buffer
mov esi, InBuf
call StrToInt
cmp ebx, 0
jne Error
mov [A], eax
…
Buffer:
mov eax, 3 ; системная функция 3 (read)
mov ebx, 0 ; дескриптор файла stdin=0
mov ecx, InBuf ; адрес буфера ввода
mov edx, lenIn ; размер буфера
int 80h
ret
```

mov ecx, InBuf: Загружается адрес буфера InBuf в регистр ecx. Буфер используется для хранения введенных пользователем данных.

mov edx, lenIn: В регистр edx загружается размер буфера, который указан в переменной lenIn. Это позволяет операционной системе знать о размере буфера и сколько данных нужно прочитать.

После прочтения строки вызывается функция StrToInt, которая на вход получает в регистре ESI адрес хранимой строки, а на выходе записывает в EAX полученное число, при этом если при преобразовании произошли ошибки то регистр EBX принимает значение отличное от нуля.

- Вывод данных:

В листинге 4 показан фрагмент программы.

Листинг 4 – Фрагмент программы с выводом результата в виде строки

```
    mov eax, [F]
    mov esi, OutBuf
    call IntToStr
    mov esi, eax
    mov eax, 4 ; системная функция 1 (write)
    mov ebx, 1 ; дескриптор файла stdout=1
    mov ecx, OutBuf ; адрес буфера
    mov edx, esi ; длина буфера
    int 80h ; вызов системной функции
    jmp Exit
```

Для корректного выполнения функции IntToStr в регистр EAX записывается значение преобразуемого числа, а в регистр ESI записывается адрес памяти, куда необходимо вывести строку результата, на выходе в регистр EAX записывается длина строки, а в буфере окажется наша преобразованная строка. Длину буфера мы получили из регистра EAX.

Сложность преобразования данных при выполнении операций ввода-вывода вызвана необходимостью работы с различными типами данных (строки, числа), а также требованием обработки возможных ошибок (некорректный ввод, деление на ноль и т.д.).