



**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)**

$$s = r^2 -$$

---

---

**ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ  
КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

## **О т ч е т**

**по лабораторной работе № 2**

**Название:** Программирование целочисленных вычислений

**Дисциплина:** Машинно-зависимые языки и основы компиляции

Студент гр. ИУ6-41Б

01.03.2023

(Подпись, дата)

Т. Е. Старжевский

(И.О. Фамилия)

Преподаватель

01.03.2023

(Подпись, дата)

С. С. Данилюк

(И.О. Фамилия)

Москва, 2023

## Введение

**Цель работы:** изучение форматов машинных команд, команд целочисленной арифметики ассемблера и программирование целочисленных вычислений.

### Задачи работы:

1) Разработать программу, вычисляющую заданное выражение. Просмотреть в отладчике и зафиксировать в отчете ход выполнения вычислений (покомандно). Убедиться в правильности программы. Выражение согласно варианту:

2) Посмотреть в отладчике форматы 3-4 команд mov и расшифровать двоичные коды этих команд, используя материалы теоретической части.

## Ход работы

### Задание 1

Код программы, вычисляющий данное выражение:

```
%include "../lib64.asm"
section .data; сегмент инициализированных переменных
Hello1Msg dq "Input r: "
lenHello1 equ $-Hello1Msg
Hello2Msg dq "Input a: "
lenHello2 equ $-Hello2Msg
Hello3Msg dq "Input q: "
lenHello3 equ $-Hello3Msg

ResMsg dq "Result: "
lenRes equ $-ResMsg

section .bss
InBuf resq 10
lenIn equ $-InBuf
OutBuf resq 10
lenOut equ $-OutBuf
r resq 1
a resq 1
q resq 1
S resq 1

section .text ; сегмент кода
```

```

global _start

_start:
; input
; Output r
mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, Hello1Msg ; адрес выводимой строки
mov rdx, lenHello1 ; длина строки
syscall; вызов системной функции
;end

; read data to InBuf
mov rax, 0; системная функция 0 (read)
mov rdi, 0 ; дескриптор файла stdout=0
lea rsi, InBuf ; передаем указатель на буфер
mov rdx, lenIn ; длина строки
syscall
; end

; InBuf To string
mov RSI, InBuf
call StrToInt64; Вход: ESI Выход: EAX, EBX содержит 0 if errors = 0
cmp EBX, 0
mov [r], RAX
; end

; Output a
mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, Hello2Msg ; адрес выводимой строки
mov rdx, lenHello2 ; длина строки
syscall; вызов системной функции
;end

; read data to InBuf
mov rax, 0; системная функция 0 (read)
mov rdi, 0 ; дескриптор файла stdout=0
lea rsi, InBuf ; передаем указатель на буфер
mov rdx, lenIn ; длина строки
syscall
; end

; InBuf To string
mov RSI, InBuf
call StrToInt64; Вход: ESI Выход: EAX, EBX содержит 0 if errors = 0
cmp EBX, 0
mov [a], RAX
; end

```

```

; Output q
mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, Hello3Msg ; адрес выводимой строки
mov rdx, lenHello3 ; длина строки
syscall; вызов системной функции
;end

; read data to InBuf
mov rax, 0; системная функция 0 (read)
mov rdi, 0 ; дескриптор файла stdout=0
lea rsi, InBuf ; передаем указатель на буфер
mov rdx, lenIn ; длина строки
syscall
; end

; InBuf To string
mov RSI, InBuf
call StrToInt64; Вход: ESI Выход: EAX, EBX содержит 0 if errors = 0
cmp EBX, 0
mov [q], RAX
; end
; end

; Count result
mov RAX, [a]
imul rax
mov RBX, [a]
imul rbx

mov RBX, [q]
idiv RBX

mov RBX, RAX; Сохранили третий рез-т в BX
mov RAX, [a]
mov RDX, [q]
imul RDX
mov RDX, 2
imul RDX
sub RBX, RAX
mov RAX, [r]
imul RAX
add RAX, RBX
mov [S], RAX; Success!
; end

; Result to string
mov rsi, OutBuf

```

```

mov rax, [S]
cwde
call IntToStr64
; end

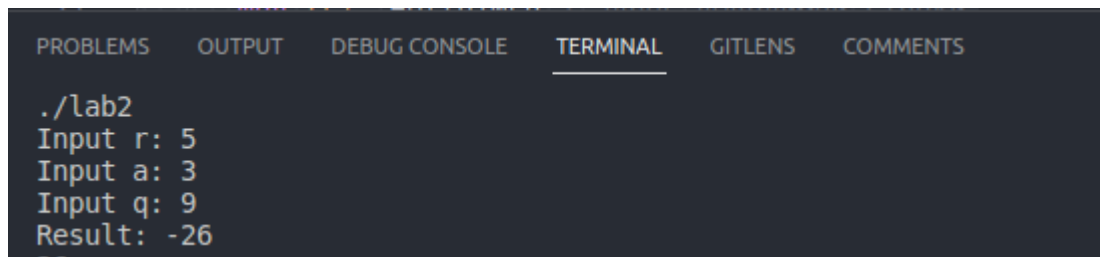
; Output
mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, ResMsg ; адрес выводимой строки
mov rdx, lenRes ; длина строки
syscall; вызов системной функции

mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, OutBuf ; адрес выводимой строки
mov rdx, lenOut ; длина строки
syscall; вызов системной функции
;end

; close program
mov rax, 60; системная функция 60 (exit)
xor rdi, rdi; return code 0
syscall; вызов системной функции
; end

```

Результат работы программы представлен на рисунке 1:



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS COMMENTS
./lab2
Input r: 5
Input a: 3
Input q: 9
Result: -26

```

Рисунок 1 - результат работы программы

Ход выполнения вычислений:

Все значения ( $r = 5$ ,  $a = 3$ ,  $q = 9$ ) введены. Отладчик до арифметических действий показан на рисунке 1:

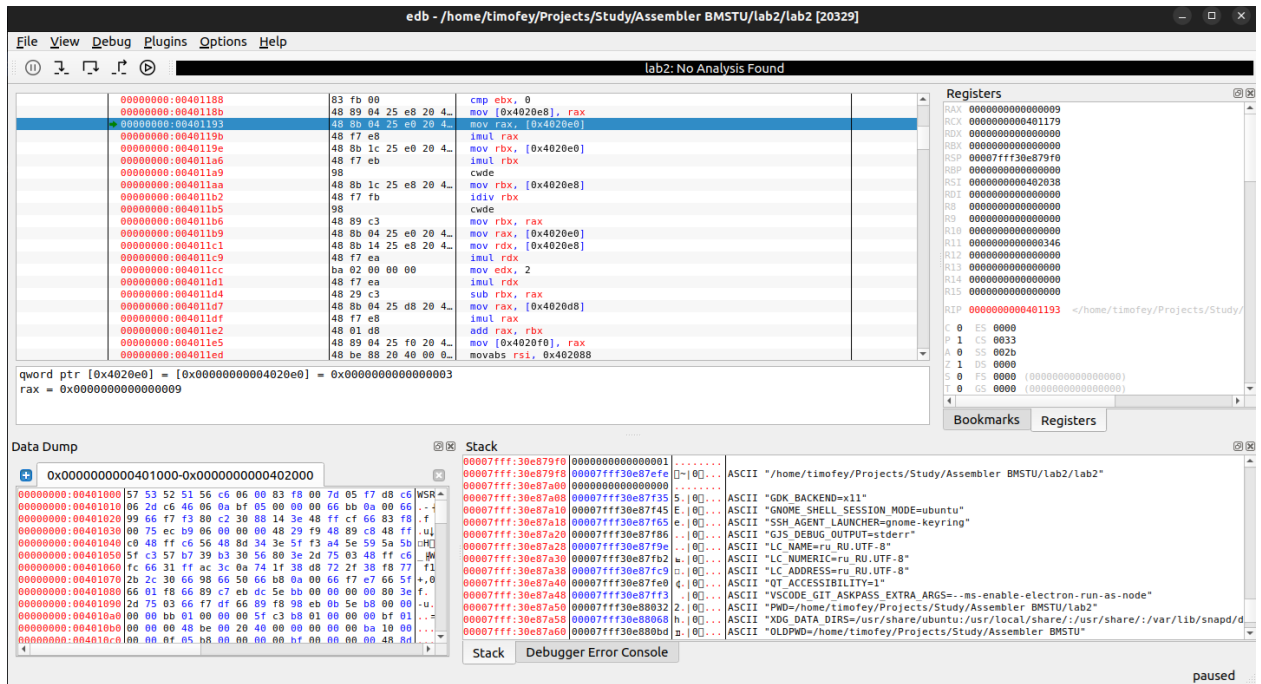


Рисунок 2 – отладчик до действий

1) Действия `mov RAX, [a]`,  $RAX = a$  и `imul rax` -  $RAX = RAX * RAX$ , показаны на рисунке 3:

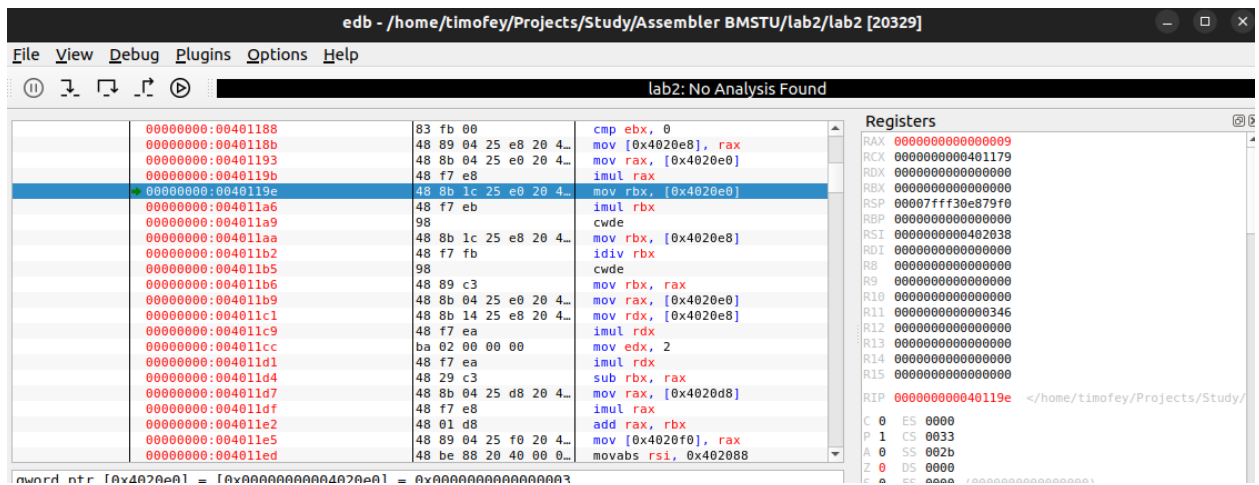


Рисунок 3 - действия `mov RAX, [a]`,  $RAX = a$ , и `imul rax` -  $RAX = RAX * RAX$

2) Действия `mov rbx, [a]` -  $rbx = [a]$  и `imul rbx` —  $rax = rax * rbx$ , показаны на рисунке 4:

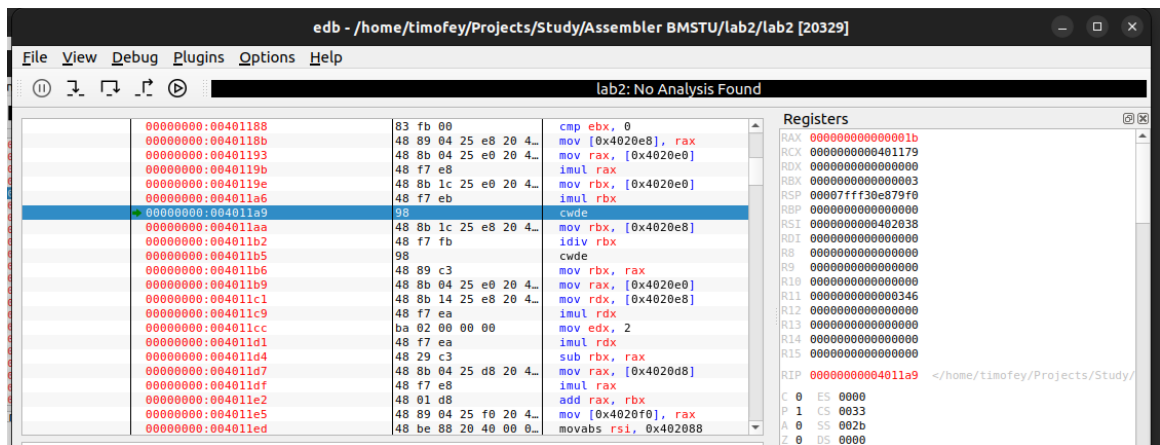


Рисунок 4 – действие `mov rbx, [a]; imul rbx`

3) Действие `mov rbx - rbx = rbx, [q]`, показано на рисунке 5:

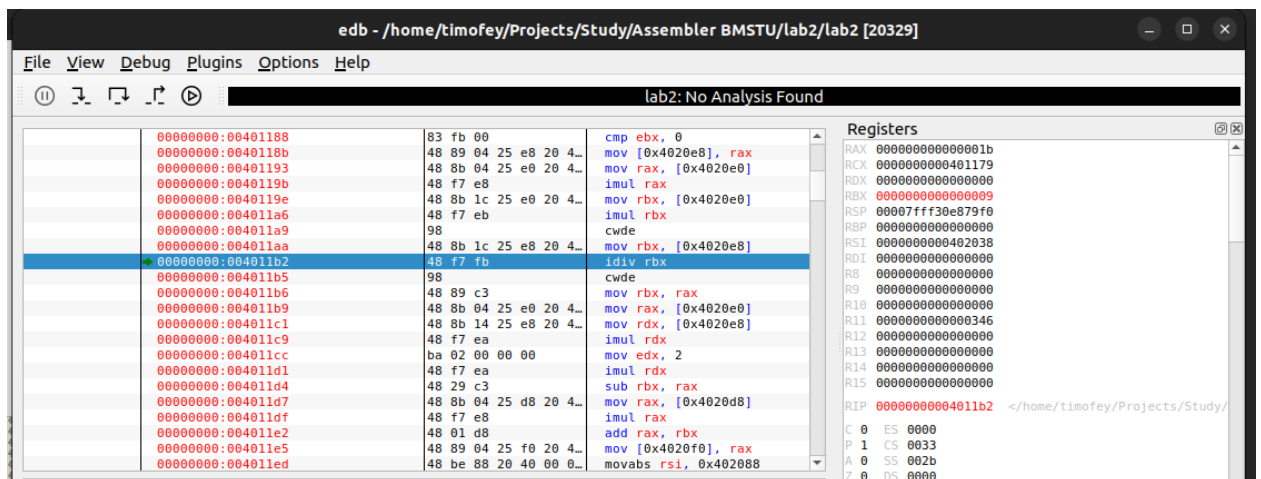


Рисунок 5 – действие `mov rbx, [q]`

4) Действие `idiv rbx` — `rax = rax/q`, показано на рисунке 6:

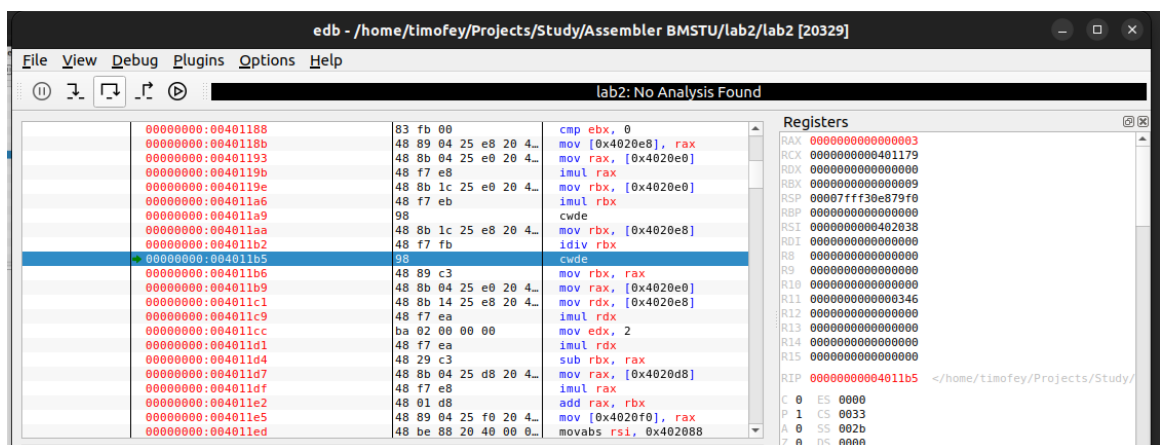


Рисунок 6 - действие `idiv rbx`



5) Действие `cdwe`, показано на рисунке 7:

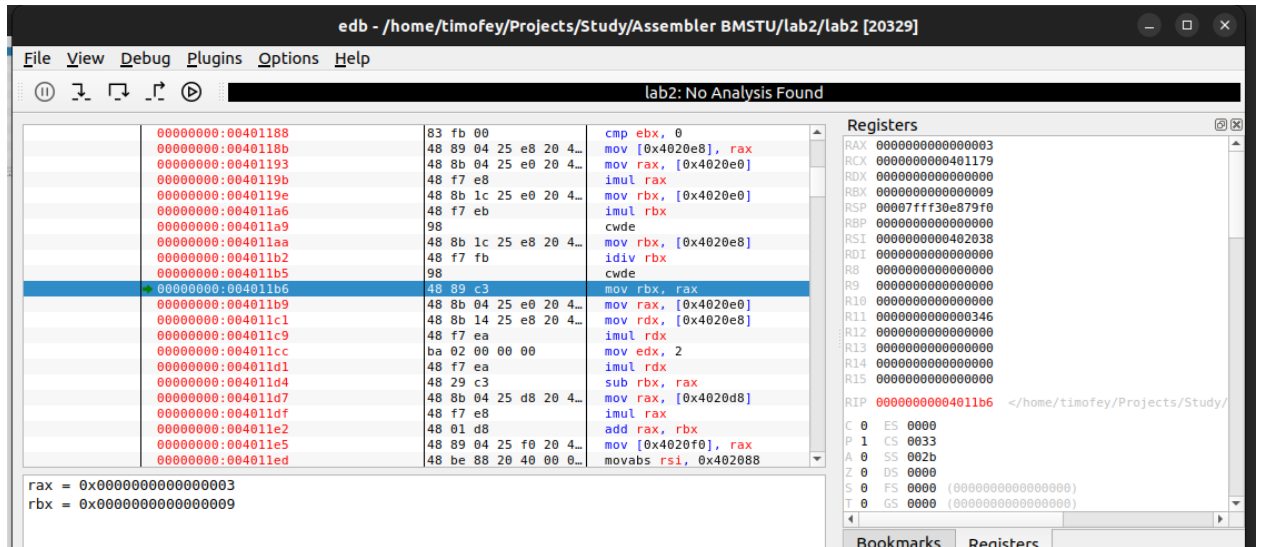


Рисунок 7 - действие `cdwe`

6) Действие `mov rbx, rax` — `rbx = rax`, показано на рисунке 8:

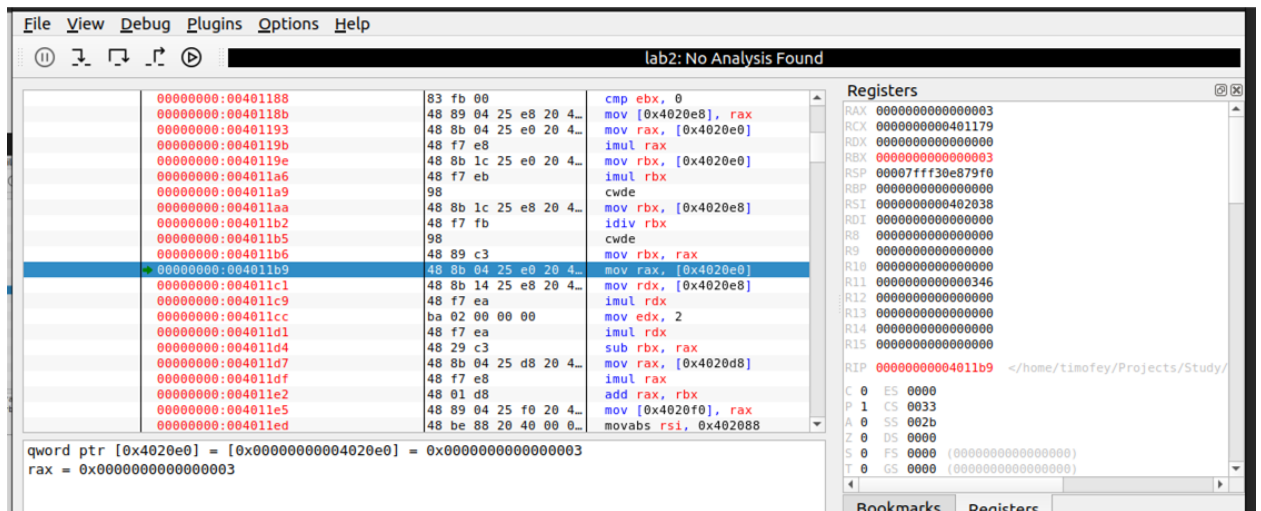


Рисунок 8 - действие `mov rbx, rax`

7) Действие `mov rax, [a]` — `rax = [a]`, показан на рисунке 9:

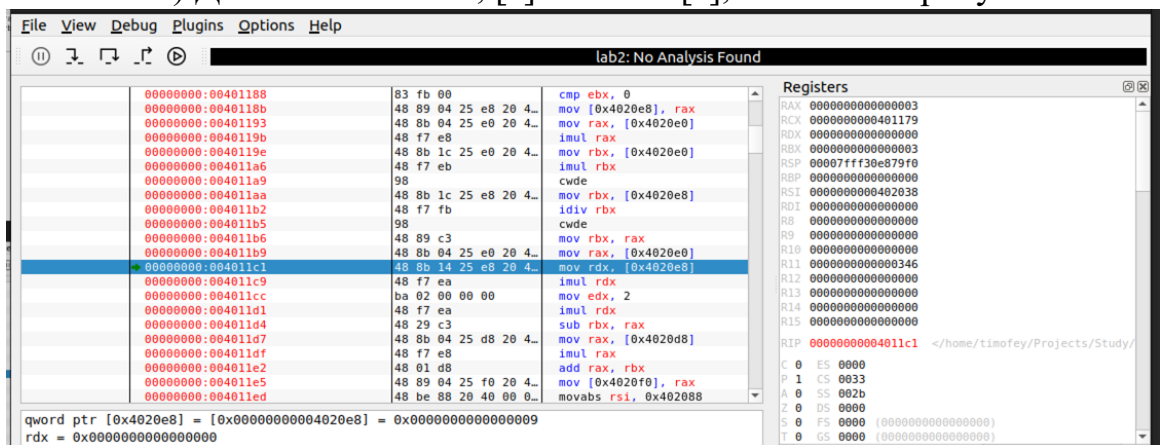


Рисунок 9 - действие `mov rax, [a]`

8) Действие `mov rdx, [q]` -  $rdx = [q]$ , показан на рисунке 10:

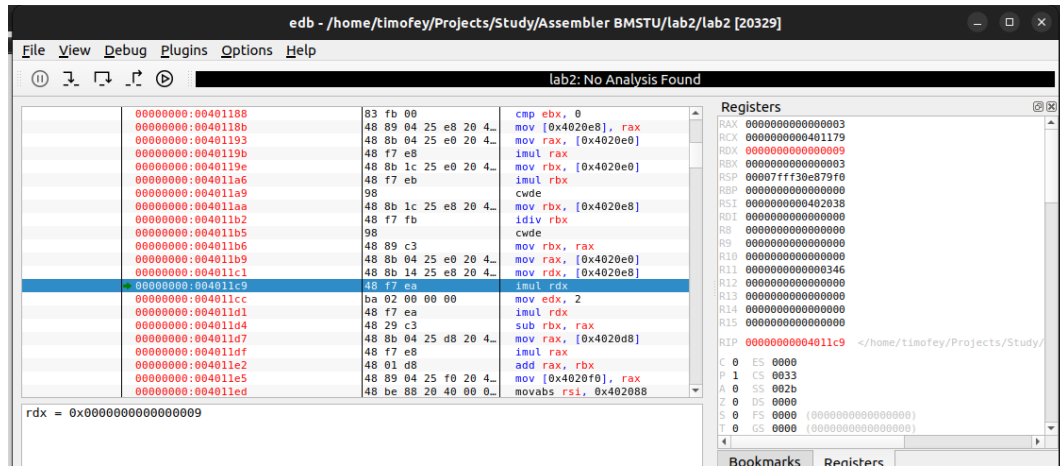


Рисунок 10 - действие `mov rdx, [q]`

9) Действие `imul rdx` —  $rax = rax * rdx$ , показано на рисунке 11:

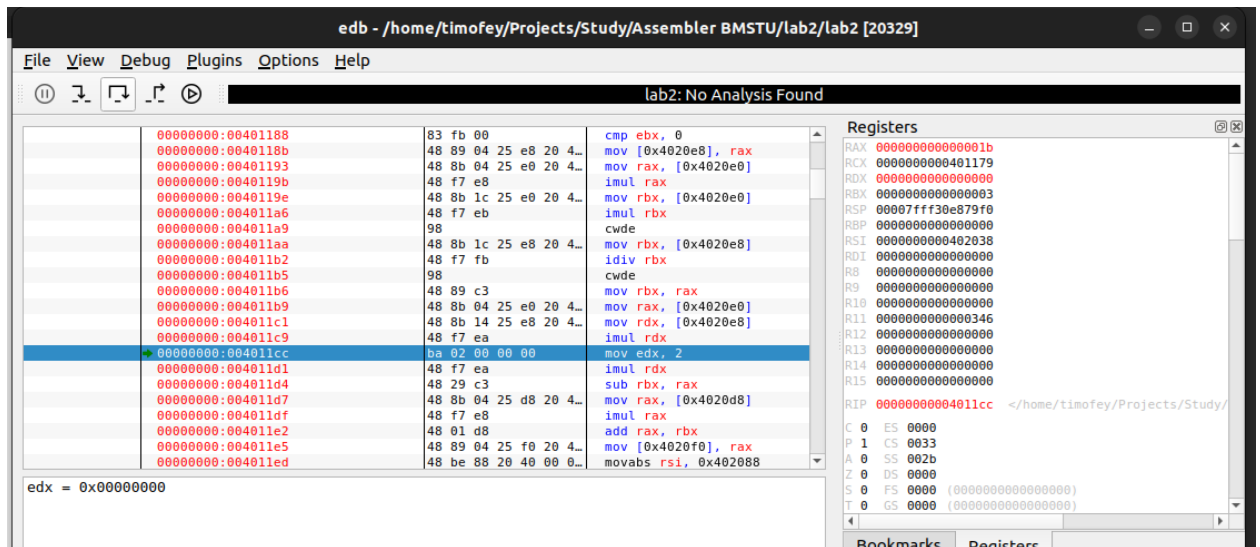


Рисунок 11 - действие `imul rdx`

10) Действие `mov edx, 2` —  $edx = 2$ , показан на рисунке 12:

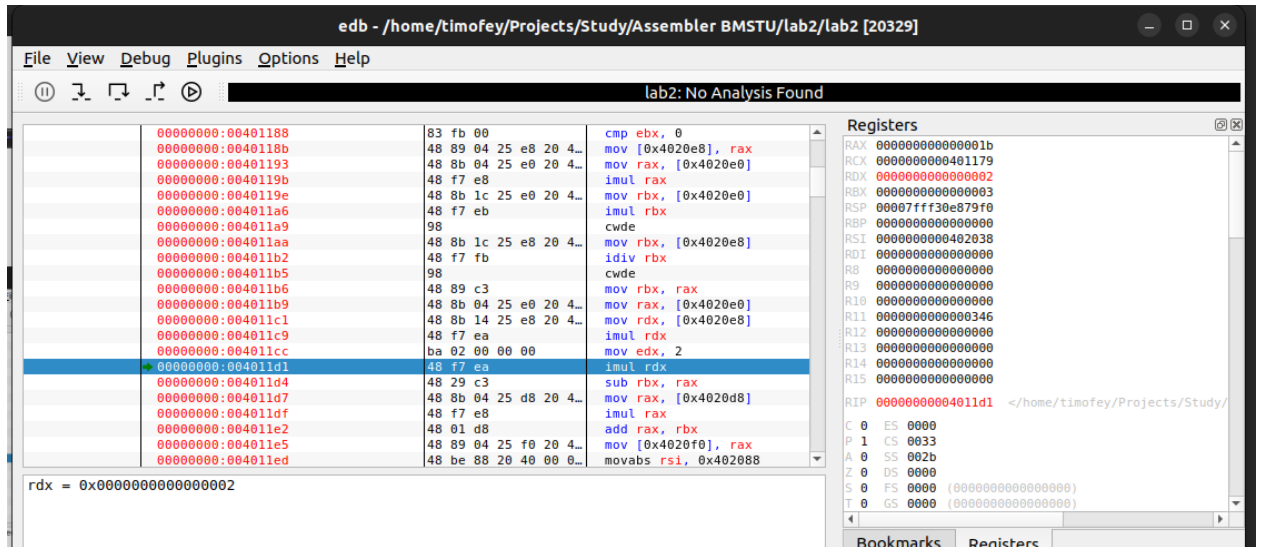


Рисунок 12 - действие `mov edx, 2`

11) Действие `imul rdx` —  $rax = rax * rdx$ , показан на рисунке 13:

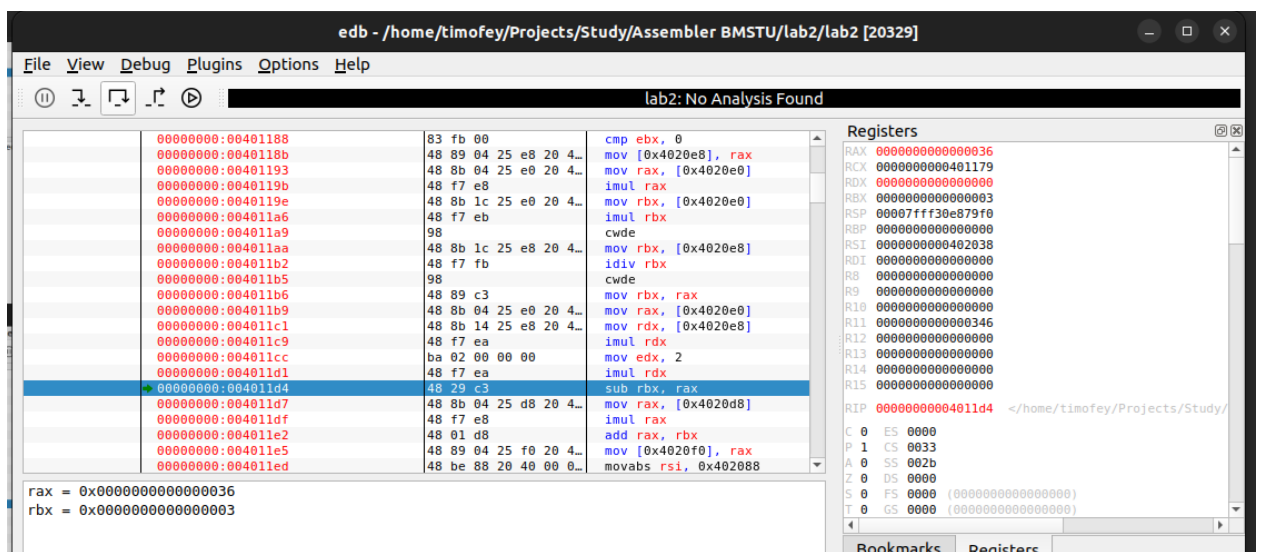


Рисунок 13 - действие `imul rdx`

12) Действие `sub rbx, rax` —  $rbx = rbx - rax$ , показан на рисунке 14:

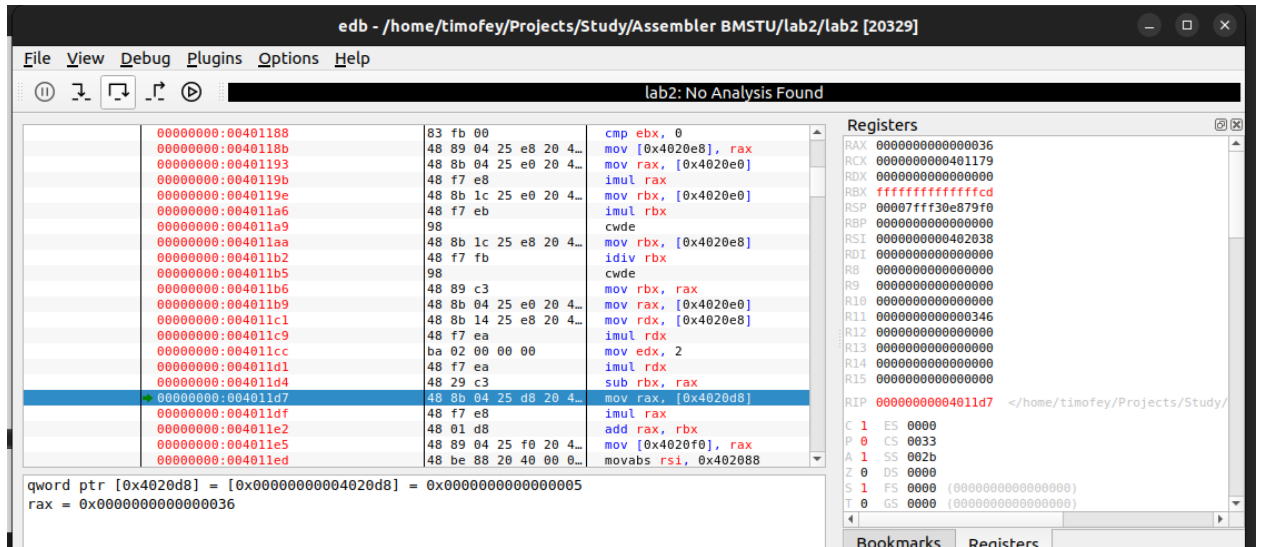


Рисунок 14 - действие `sub rbx, rax`

13) Действие `mov rax, [r]` —  $rax = [r]$ , показан на рисунке 15:

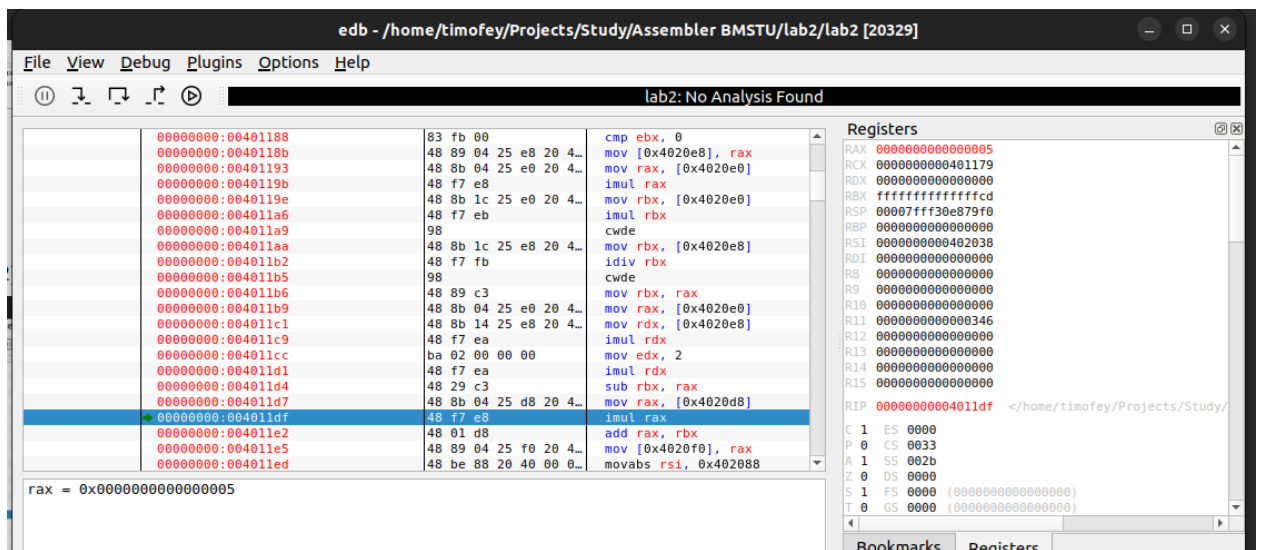


Рисунок 15 - действие `mov rax, [r]`

14) Действие `imul, rax` —  $rax = rax * rax$ , показан на рисунке 16:

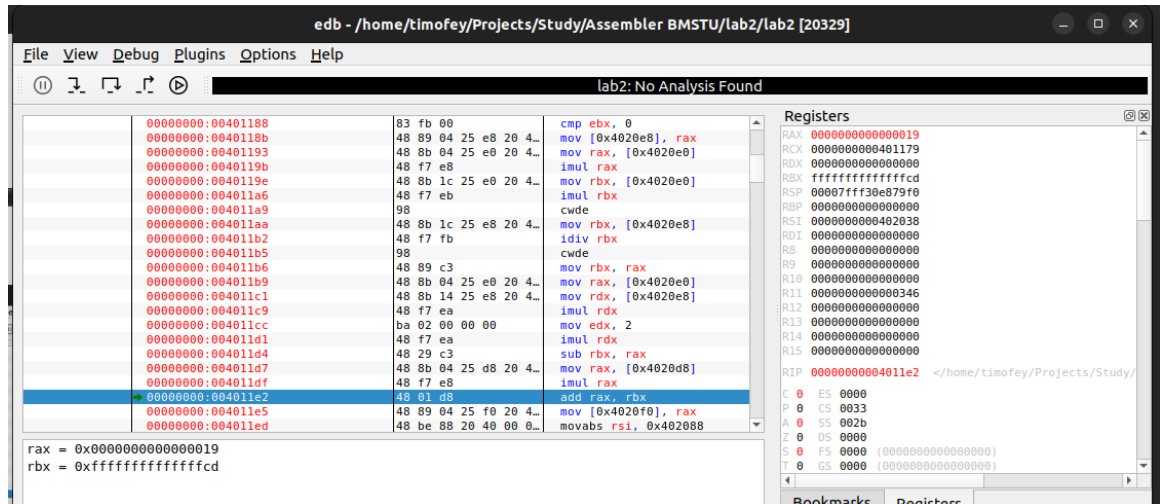


Рисунок 16 - действие `imul rax`

15) Действие `add rax, rbx` —  $rax = rax + rbx$ , показан на рисунке 17:

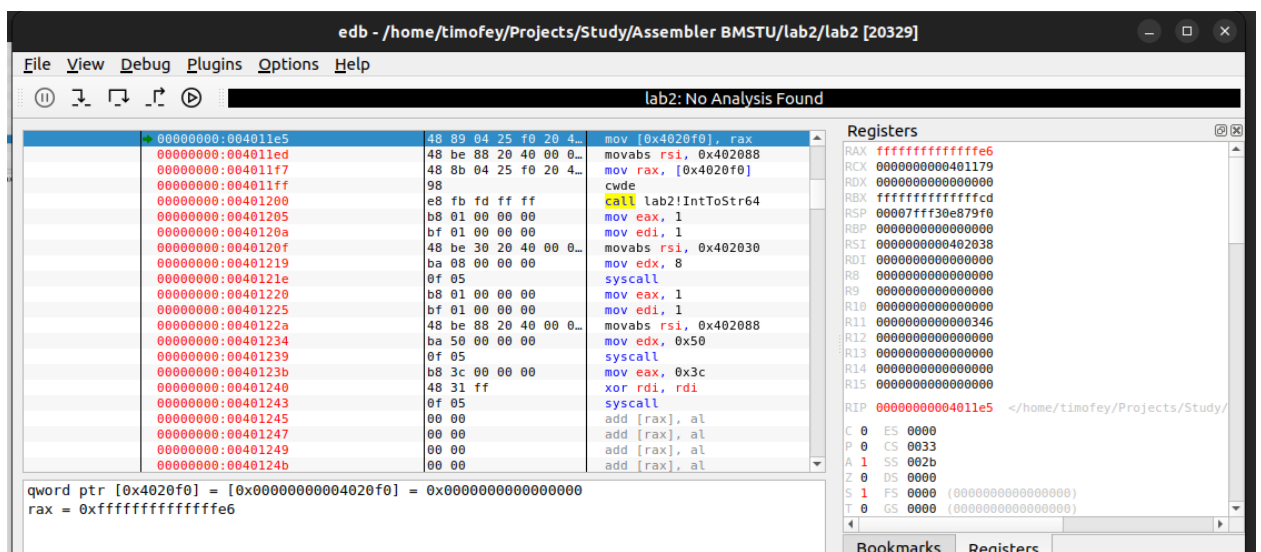


Рисунок 17 - действие `add rax, rbx`

16) Действие `mov [s], rax` —  $s = rax$ , показан на рисунке 18:

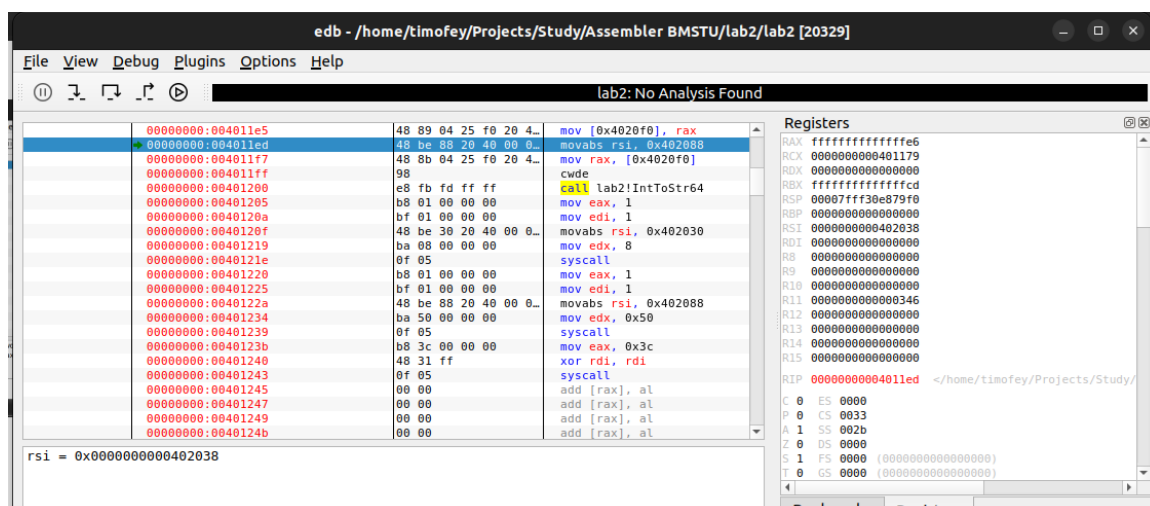


Рисунок 18 - действие `mov [s], rax`

## Задание 2

1) `mov rbx, rax`

Отображение этой команды в отладчике показана на рисунке 19:



Рисунок 19 – команда `mov rbx, rax`

0110 0110 100010 11 11 001 000

66h mov DW mod reg reg

D = 1 – в регистр, W = 1 – двойное слово

2) `mov rax, [a]`

Отображение этой команды в отладчике показана на рисунке 20:

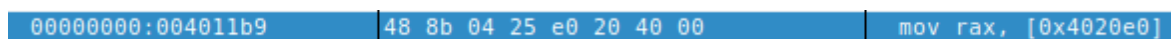


Рисунок 20 – команда `mov rax, [a]`

0110 0110 101000 11 01100110001100000010000000000000

66h mov DW 32-ух разрядная адресация

D = 1 – в регистр, W = 1 – двойное слово

3) `mov edx, 2`

Отображение этой команды в отладчике показана на рисунке 21:





Рисунок 18 – команда mov AX,S

0110 0110 101000 01 01100110001100000100000000000000

66h      mov   DW      32-ух разрядная адресация

D = 0 – из регистра, W = 1 – двойное слово

**Вывод:** была написана программа на языке ассемблер, вычисляющая математическое выражение и была просмотрена работа арифметических действий в отладчике.

### **Контрольные вопросы:**

#### **1) Что такое машинная команда? Какие форматы имеют машинные команды процессора IA32? Чем различаются эти форматы?**

Машинная команда представляет собой код, определяющий операцию вычислительной машины и данные, участвующие в операции.

Префиксы разделяют на:

- префикс повторения – используется только для строковых команд;

- префикс размера адреса (67h) – применяется для изменения размера смещения: 16 бит при 32-х разрядной адресации;

- префикс размера операнда (66h) – указывается, если вместо 32-х разрядного регистра для хранения операнда используется 16-ти разрядный;

- префикс замены сегмента – используется при адресации данных любым сегментом кроме DS.

d – направление обработки, например, пересылки данных: 1 – в регистр, 0 – из регистра;

w – размер операнда: 1 – операнды - двойные слова, 0 – операнды - байты;

mod – режим:

- 1) 00 - Disp=0 – смещение в команде 0 байт;
- 2) 01 - Disp=1 – смещение в команде 1 байт;
- 3) 10 - Disp=2 – смещение в команде 2 байта;
- 4) 11 - операнды-регистры. Регистры кодируются в зависимости от размера операнда:

Регистры кодируются в зависимости от размера операнда:

1) для w = 1	2) для w = 0
– 000 EAX;	– 000 AL;
– 001 ECX;	– 001 CL;
– 010 EDX;	– 010 DL;
– 011 EBX;	– 011 BL;
– 100 ESP;	– 100 AH;
– 101 EBP;	– 101 CH;
– 110 ESI;	– 110 DH;
– 111 EDI.	– 111 BH.

## **2) Назовите мнемоники основных команд целочисленной арифметики. Какие форматы для них можно использовать?**

Мнемоники: add, sub, div/ldiv, mul/imul.

Арифметические операции, такие как сложение, вычитание, деление и умножение можно выполнять над однобайтовыми, двухбайтовыми и четырёхбайтовыми целыми числами.

## **3) Сформулируйте основные правила построения линейной программы вычисления заданного выражения.**

- 1) Объявить инициализированные переменные.
- 2) Объявить неинициализированные переменные.



3) Написать команды для вычисления выражений арифметических действий и команды перемещения, объявленных ранее переменных как в регистры, так и из них.

4) Написать процедур ввода и вывода, если этого требует условие поставленного выражения.

**4) Почему ввод-вывод на языке ассемблера не программируют с использованием соответствующих машинных команд? Какая библиотека используется для организации ввода-вывода в данной лабораторной?**

В данной лабораторной работе для организации ввода-вывода используется библиотека MASM32.lib, которая имеет специальные возможности для ввода и вывода.

**5) Расскажите, какие процедуры используют для организации ввода вывода. Какие операции выполняет каждая процедура?**

Ввод:

Invoke StdIn,ADDR Buffer,LengthOf Buffer – вызов процедуры ввода

Invoke StripLF,ADDR Buffer – вызов процедуры замены символов конца строки нулем (буфер ввода)

Invoke atol,ADDR Buffer – вызов процедуры преобразования завершающейся строки нулем (результат в EAX)

Вывод:

Invoke dwtoa,X,ADDR ResStr – вызов процедуры помещения значения X в регистр AX, а AX в ResStr

Invoke StdOut,ADDR Result – вызов процедуры вывода