|  |  |
| --- | --- |
| Gerb-BMSTU_01 | ***«*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\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по лабораторной работе №** 2

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

**Название лабораторной работы:** Программирование целочисленных вычислений

Студент гр. ИУ6 - 42Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_**Г. Д. Нефедов

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Москва, 2024

**Цель работы:** изучение форматов машинных команд, команд

целочисленной арифметики ассемблера и программирование целочисленных вычислений.

**Задание**

Вычислить целочисленное выражение:

**Код программы:**

Разберем код программы по секциям:

Секция данных:

|  |
| --- |
| section .data  ib db "Enter B: ", 0  ic db "Enter C: ", 0  id db "Enter D: ", 0  ia db "Your result: ", 0 |

Листинг 1

В секции данных (.data) объявляем строки и инициализируем их значениями. Каждая строка заканчивается нулевым байтом (0), что означает конец строки в ассемблере.

Листинг секция .bss:

|  |
| --- |
| section .bss  msgB resb 10  lenMsgB equ $-msgB    msgC resb 10  lenMsgC equ $-msgC    msgD resb 10  lenMsgD equ $-msgD  msgA resb 13  lenMsgA equ $-msgA  inbuf resb 10  lenIn equ $-inbuf  outbuf resb 10  lenout equ $-outbuf    A resw 1  B resd 1  C resd 1  D resd 1 |

Листинг 2

В секции BSS мы зарезервировали память для переменных, которые будут использоваться в программе, но им еще не присвоены начальные значения. В строке «msgB resb 10», резервируется 10 байт для вывода сообщения ib, аналогично и для сообщений ic, id, ia. В строке «lenMsgB equ $-msgB» определяется длина сообщения lenMsgB как разница между текущим адресом $ и адресом начала строки msgB, аналогично происходит вычисление длины для вывода сообщений ic, id, ia. В данных строках «inbuf resb 10», «lenIn equ $-inbuf» мы резервируется память для вводимых значений и считаем длину вводимого сообщения, аналогично и для outbuf. Далее резервируем память для переменных A, B, C и D.

Секция .text

Секция .text разбита для наглядности на несколько меток, рассмотрим код каждой метки.

Метка InputB:

Листинг

|  |
| --- |
| InputB:  ;; вывовдим сообщение ib  mov eax, 4  mov ebx, 1  mov ecx, ib  mov edx, lenMsgB  int 0x80  ;; считываем введеное число  mov eax, 3  mov ebx, 0  mov ecx, inbuf  mov edx, 10  int 0x80  ;; передаем параметры в функцию  mov esi, inbuf  call StrToInt  cmp EBX, 0  mov [B], ax  jmp InputC |

Листинг 3

Рассмотрим каждую строку кода по отдельности:

1. «mov eax, 4» - устанавливаем в регистр eax значение 4, что представляет собой системный вызов для вывода строки.

2. «mov ebx, 1» - устанавливаем в регистр ebx значение 1, что указывает на стандартный вывод (stdout).

3. «mov ecx, ib» - устанавливаем в регистр ecx адрес строки ib.

4. «mov edx, lenMsgB» - устанавливаем в регистр edx длину строки сообщения B.

5. «int 0x80» - системный вызов, который приводит к выводу строки на экран.

6. «mov eax, 3» - системный вызов для ввода.

7. «mov ebx, 0» - устанавливаем в регистр ebx значение 0, что указывает на стандартный ввод (stdin).

8. «mov ecx, inbuf» - устанавливаем в регистр ecx адрес буфера ввода.

9. «mov edx, 10» - устанавливаем в регистр edx максимальную длину ввода.

10. «int 0x80» - системный вызов, который считывает введенное значение и сохраняет его в буфер ввода.

11. «mov esi, inbuf» - устанавливаем в регистр esi адрес буфера ввода.

12. «call StrToInt» - вызов подпрограммы для преобразования строки в число.

13. «cmp EBX, 0» - сравниваем значение в регистре ebx с 0, чтобы проверить, произошла ли ошибка преобразования.

14. «mov [B], ax» - сохраняем результат преобразования в переменную B.

15. «jmp InputC» - переход к следующему блоку кода.

Аналогичный код для меток InputС, InputD.

Метка Calc:

|  |
| --- |
| Calc:  ;;записываем значения переменных в регистры  mov bx, [B]  mov ax, [C]  mov dx, [D]  xor ecx, ecx  mov cx, ax  sub cx, 5  imul cx, dx  add cx, bx  mov ax, cx  imul bx, bx  add bx, 1  xor edx, edx  div bx  shl edx, 16  or eax, edx  mov [A], ax  jmp Output |

Листинг 4

Рассмотрим каждую строку кода по отдельности:

1. «mov bx, [B]» - устанавливает значение переменной B в регистр bx.
2. «mov ax, [C]» - устанавливает значение переменной C в регистр ax.
3. «mov dx, [D]»-устанавливает значение переменной D в регистр dx.
4. «xor ecx, ecx» - обнуляет регистр ecx.
5. «mov cx, ax» - устанавливает значение регистра ax в регистр cx.
6. «sub cx, 5» - вычитаем 5 из значения в регистре cx (С- 5).
7. «imul cx, dx» - умножаем значение регистра cx на значение в регистре dx, получаем (С - 5)\*D.
8. «add cx, bx» - прибавляем значение регистра bx к значению в регистре cx, получаем (C - 5) \* D.
9. «mov ax, cx» - устанавливаем значение регистра cx в регистр ax.
10. «imul bx, bx» - умножаем значение регистра bx само на себя, получаем (b^2).
11. «add bx, 1» - прибавляем 1 к значению в регистре bx, получаем (b^2 + 1).
12. «xor edx, edx» - обнуляем регистр edx.
13. «div bx» - делим значение в регистре dx:ax на значение в регистре bx, результат сохраняется в регистре ax, а остаток - в регистре dx.
14. «shl edx, 16» - сдвигаем биты в регистре edx на 16 позиций влево.
15. «or eax, edx» - применяем логическую операцию ИЛИ между значениями в регистрах eax и edx.
16. «mov [A], ax» - загружаем значение регистра ax в переменную A.
17. «jmp Output» - переход к следующему блоку кода.

Метка Output:

|  |
| --- |
| Output:  ;;конвертирум из A в строку  mov esi, outbuf  mov ax, [A]  call IntToStr  mov eax, 4  mov ebx, 1  mov ecx, ia  mov edx, lenMsgA  int 0x80  ;; выводим результат  mov eax, 4  mov ebx, 1  mov ecx, outbuf  mov edx, lenout  int 0x80  jmp Exit |

Листинг 5

1. «mov esi, outbuf» - устанавливаем указатель строки esi на начало буфера outbuf.
2. «mov ax, [A]» - устанавливаем значение переменной A в регистр ax.
3. «call IntToStr» - вызываем процедуру IntToStr для конвертации значения регистра ax в строку и сохранения ее в буфере outbuf.
4. «mov eax, 4» - устанавливаем номер системного вызова для вывода строки в регистр eax.
5. «mov ebx, 1» - устанавливаем дескриптор файла (stdout) в регистр ebx.
6. «mov ecx, ia» - устанавливаем указатель на строку ia в регистр ecx.
7. «mov edx, lenMsgA» - устанавливаем длину строки lenMsgA в регистр edx.
8. «int 0x80» - системный вызов для вывода строки на экран.
9. «mov eax, 4» - устанавливаем номер системного вызова для вывода строки в регистр eax.
10. «mov ebx, 1» - устанавливаем дескриптор файла (stdout) в регистр ebx.
11. «mov ecx, outbuf» - устанавливаем указатель на строку outbuf в регистр ecx.
12. «mov edx, lenout» - устанавливаем длину строки lenout в регистр edx.
13. «int 0x80» - системный вызов для вывода результата строки на экран.
14. «jmp Exit» - переход к метке Exit.

|  |
| --- |
| Exit:  mov eax, 1  xor ebx, ebx  int 0x80 |

Листинг 6

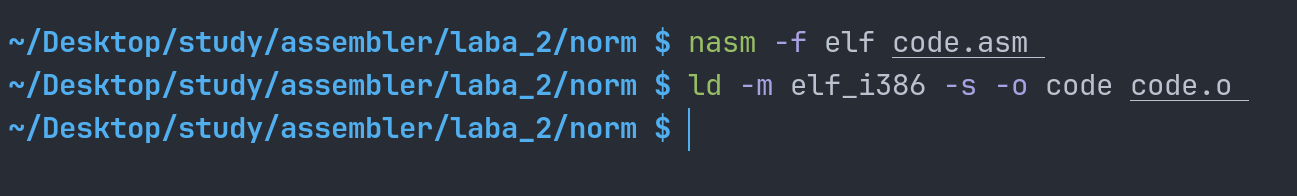
1. «mov eax, 1» - загружаемй номер системного вызова для завершения программы в регистр eax.
2. «xor ebx, ebx» - обнуляем регистр ebx.
3. «int 0x80» - системный вызов для завершения программы.

Полный код программы:

|  |
| --- |
| %include "lib.asm"    section .data  ib db "Enter B: ", 10  ic db "Enter C: ", 10  id db "Enter D: ", 10  ia db "Your result: ", 10  section .bss  msgB resb 10  lenMsgB equ $-msgB    msgC resb 10  lenMsgC equ $-msgC    msgD resb 10  lenMsgD equ $-msgD  msgA resb 13  lenMsgA equ $-msgA  inbuf resb 10  lenIn equ $-inbuf  outbuf resb 10  lenout equ $-outbuf    A resd 1  B resd 1  C resd 1  D resd 1  section .text  global \_start  \_start:  jmp InputB  InputB:  ;; вывовдим сообщение ib  mov eax, 4  mov ebx, 1  mov ecx, ib  mov edx, lenMsgB  int 0x80  ;; считываем введеное число  mov eax, 3  mov ebx, 0  mov ecx, inbuf  mov edx, 10  int 0x80  ;; передаем параметры в функцию  mov esi, inbuf  call StrToInt  cmp EBX, 0  mov [B], eax  jmp InputC  InputC:  ;; вывовдим сообщение ic  mov eax, 4  mov ebx, 1  mov ecx, ic  mov edx, lenMsgC  int 0x80  ;; считываем введеное число  mov eax, 3  mov ebx, 0  mov ecx, inbuf  mov edx, 10  int 0x80  ;;передаем параметры в функцию  mov esi, inbuf  call StrToInt  cmp EBX, 0  mov [C], eax  jmp InputD  InputD:  ;; вывовдим сообщение id  mov eax, 4  mov ebx, 1  mov ecx, id  mov edx, lenMsgD  int 0x80  ;; считываем введеное число  mov eax, 3  mov ebx, 0  mov ecx, inbuf  mov edx, 10  int 0x80  ;;передаем параметры в функцию  mov esi, inbuf  call StrToInt  cmp EBX, 0  mov [D], eax  jmp Calc  Calc:  mov ebx, [B]    mov eax, [C]  mov edx, [D]  xor ecx, ecx  mov ecx, [C]  sub ecx, 5  mov eax, ecx    imul edx  add eax, ebx  xor ecx, ecx  mov ecx, eax  mov eax, ebx  imul ebx  add eax, 1  mov ebx, eax  mov eax, ecx  cdq  idiv ebx    mov [A], eax  jmp Output  Output:  mov eax, 4  mov ebx, 1  mov ecx, ia  mov edx, lenMsgA  int 0x80  ;;конвертирум из A в строку  mov esi, outbuf  cwde  mov eax, [A]  call IntToStr  ;; выводим результат  mov eax, 4  mov ebx, 1  mov ecx, outbuf  mov edx, lenout  int 0x80  jmp Exit  Exit:  mov eax, 1  xor ebx, ebx  int 0x80 |

Листинг 7

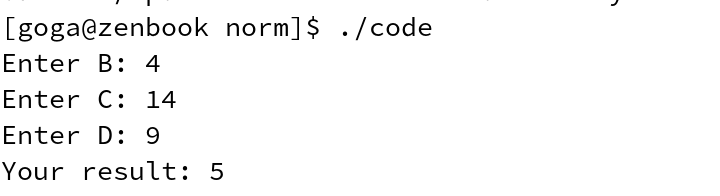
Трансляция и компановка программы представлены на рисунке 1:

Рисунок 1 - Трансляция и компановка программы

Посчитаем пример вручную:

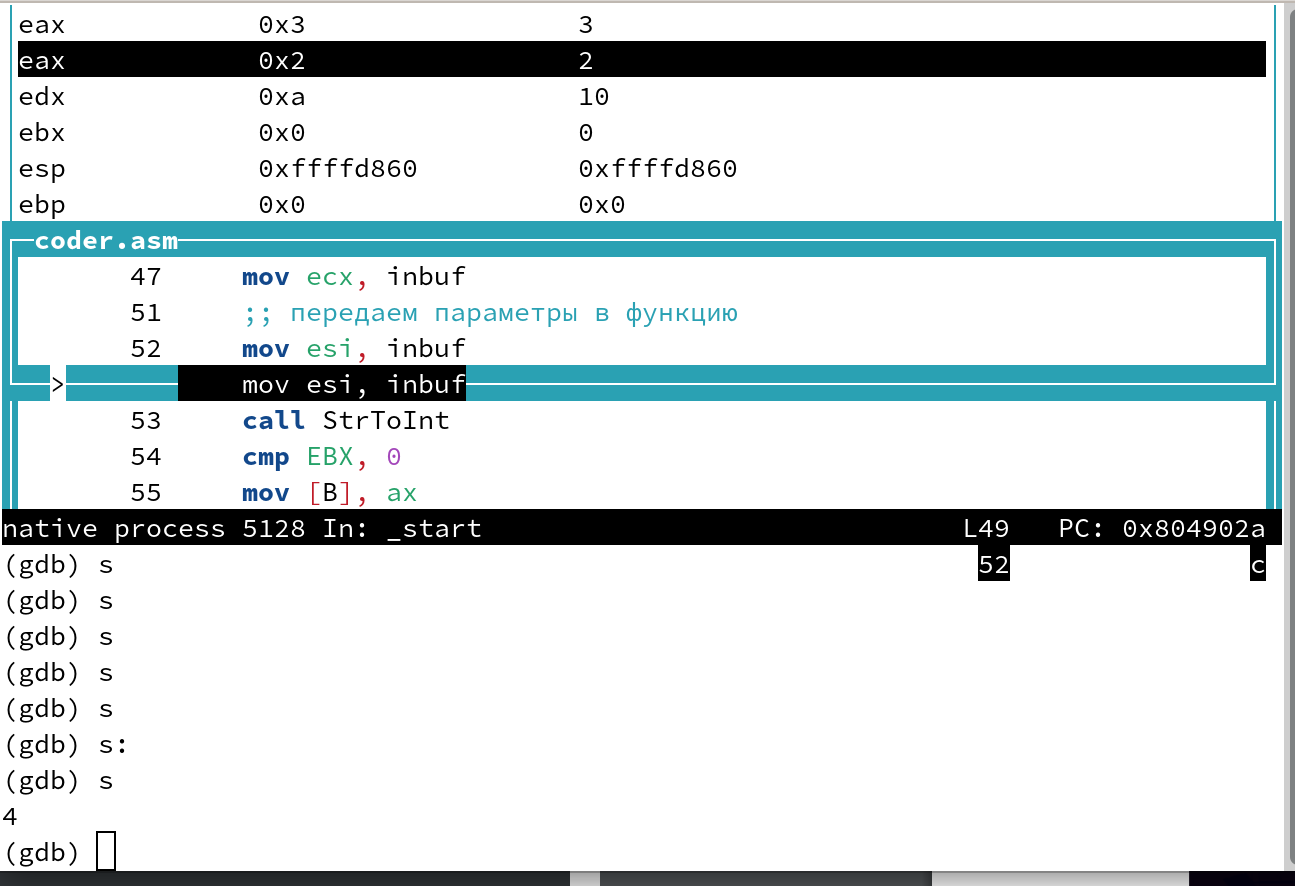
Пусть b = 4, c = 14, d = 9, тогда:

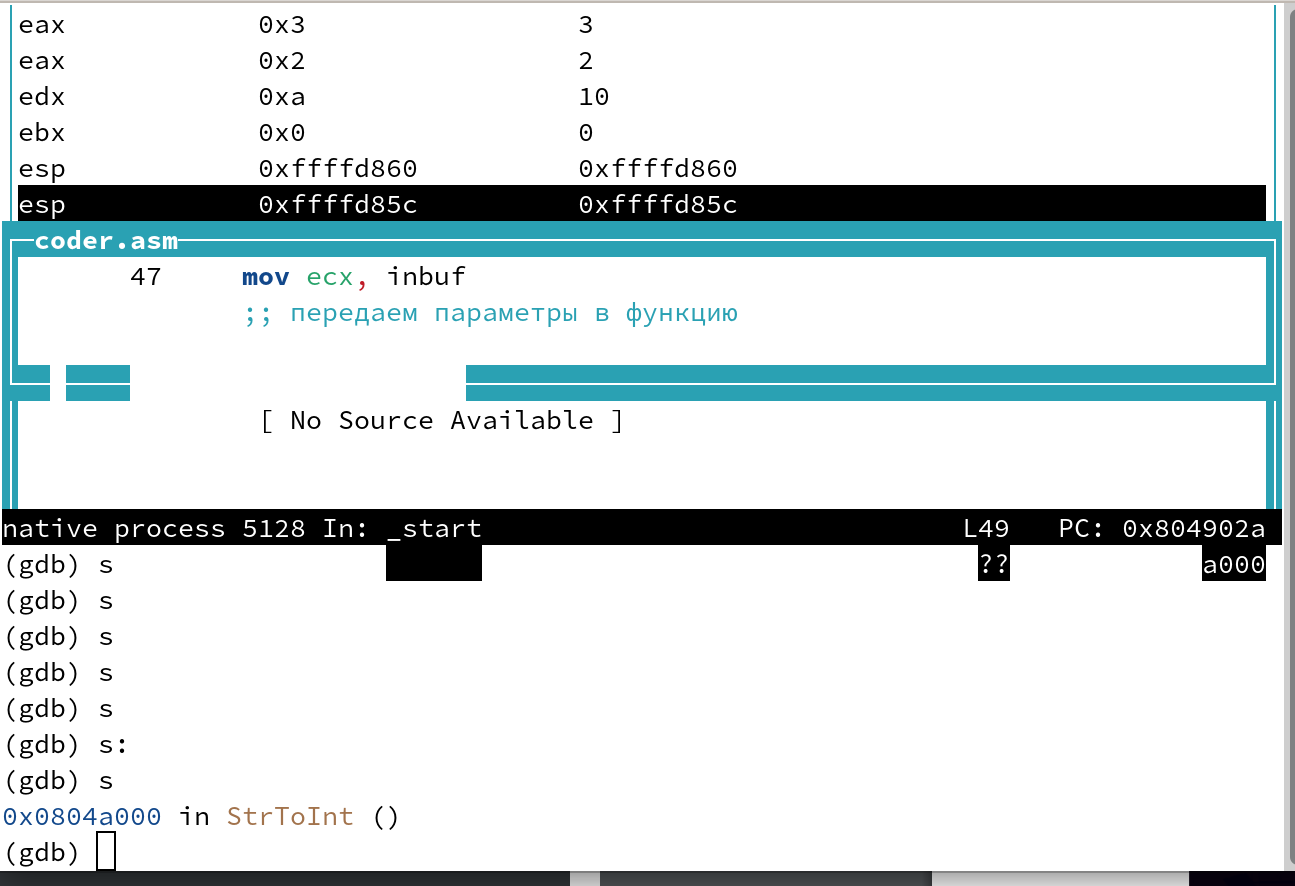
Теперь запустим программы и проверим корректность выводимых данных представлены на рисунке 2:

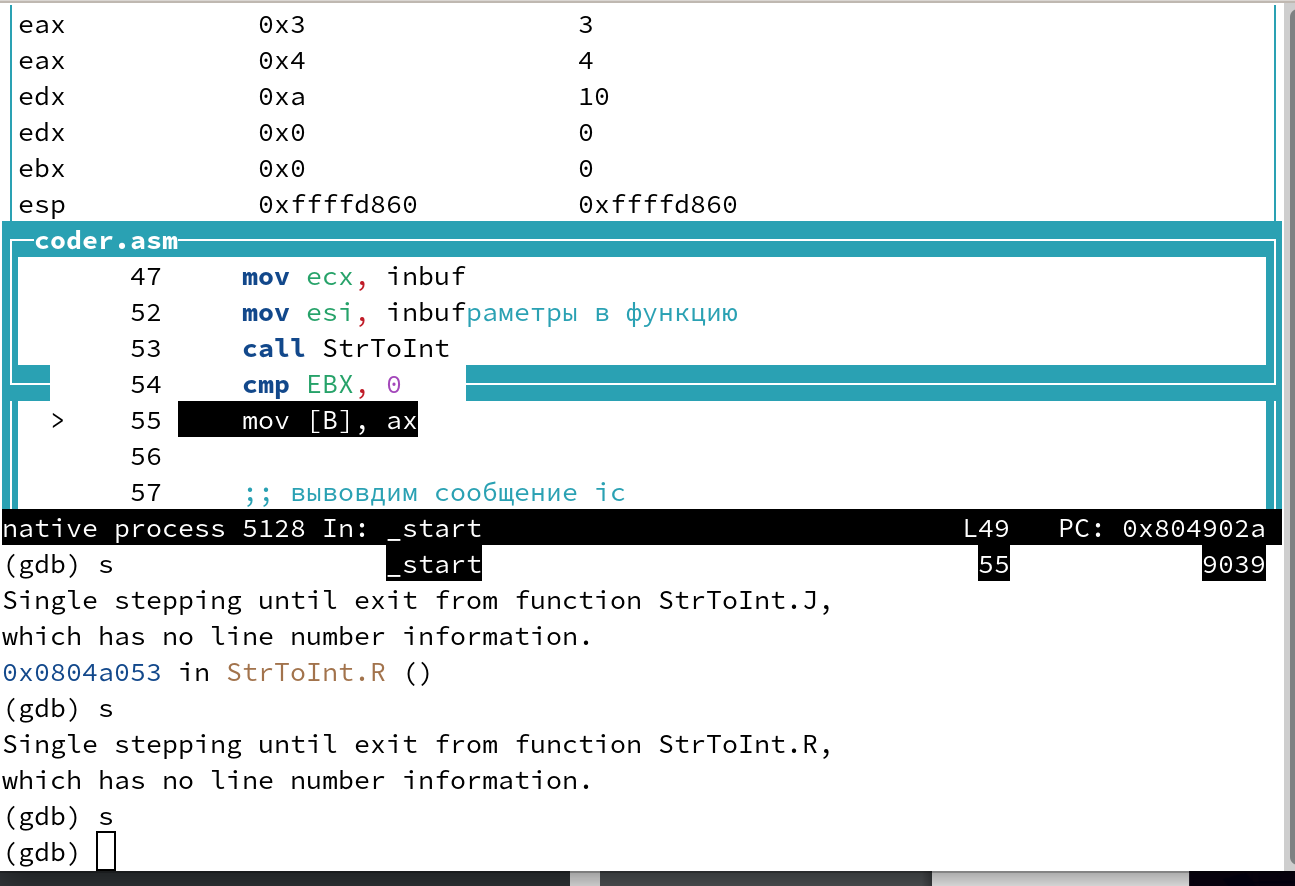
Рисунок 2 — Запуск программы

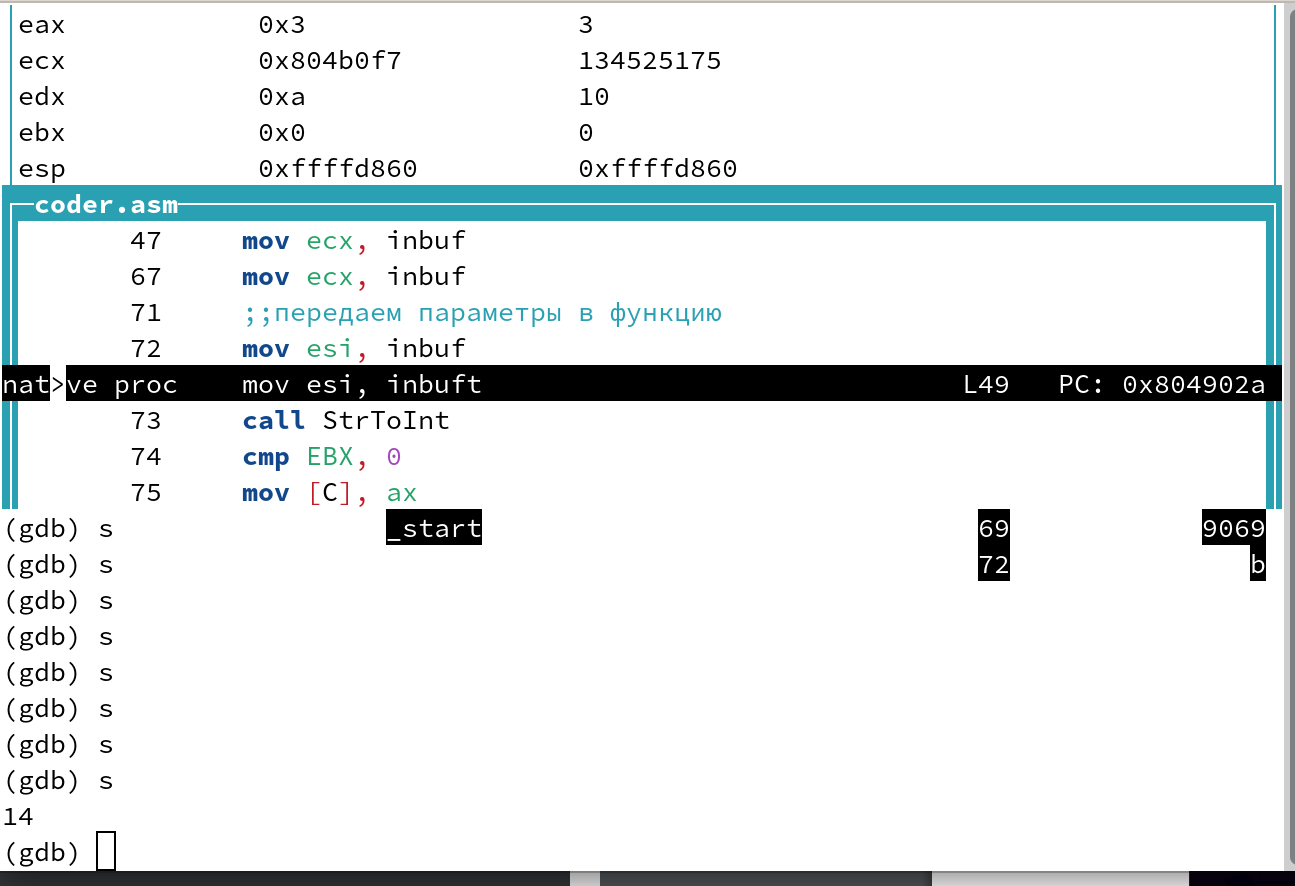
Отладка программы представлены на рисунках 3 - 12:

Рисунок 3 — Запуск отладчика gdb.

Рисунок 4 — Записываем в качестве параметра B цифру 4.

Рисунок 5 — Заход в функцию StrToInt.

Рисунок 6 — Вышли из функции StrToInt и записали преобразованное число в регистр eax.

Рисунок 7 — Записываем в качестве параметра C цифру 14.

После ввода сообщения для инициализации C, аналогично инициализации переменной B, программа переходит в функцию StrToInt для конвертации строки в число.

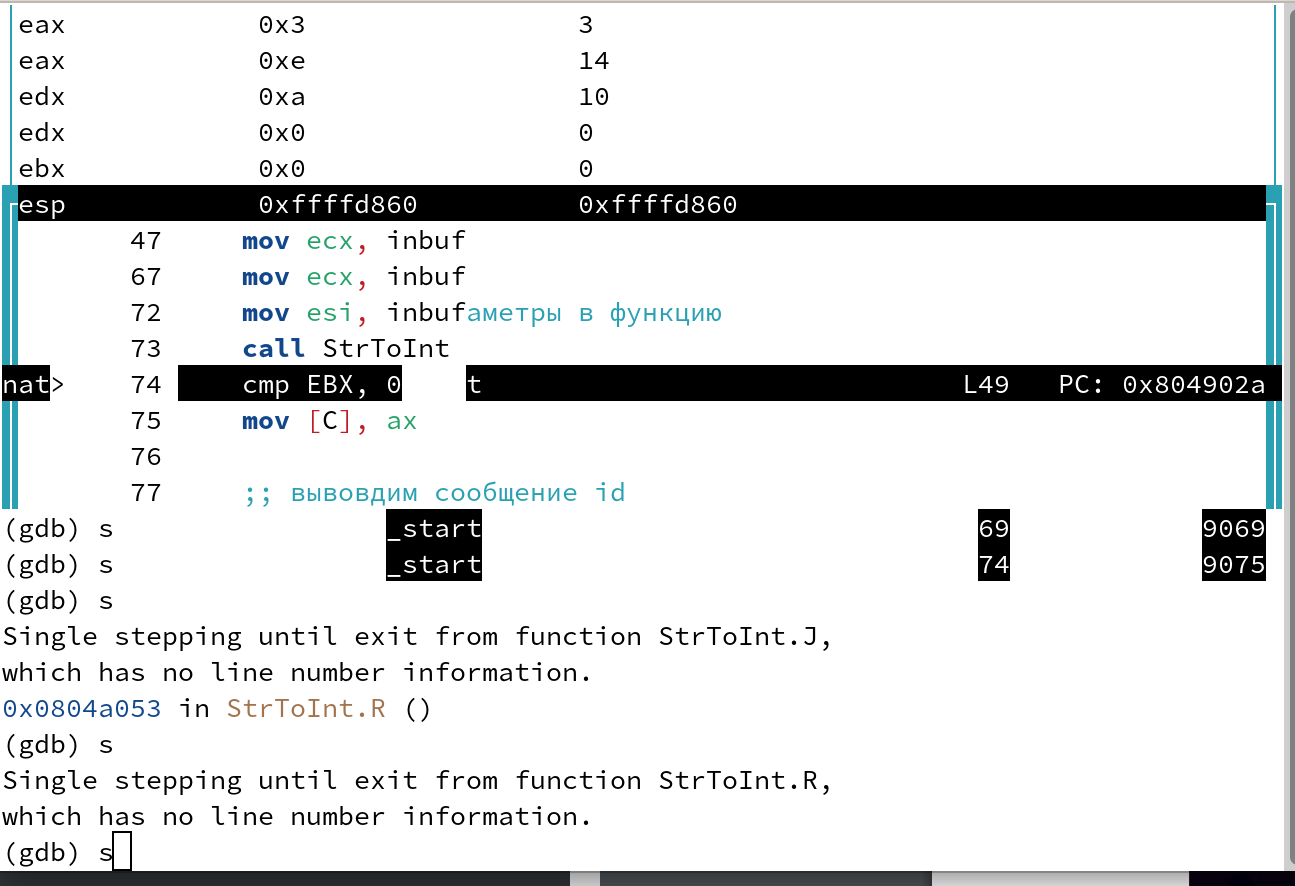
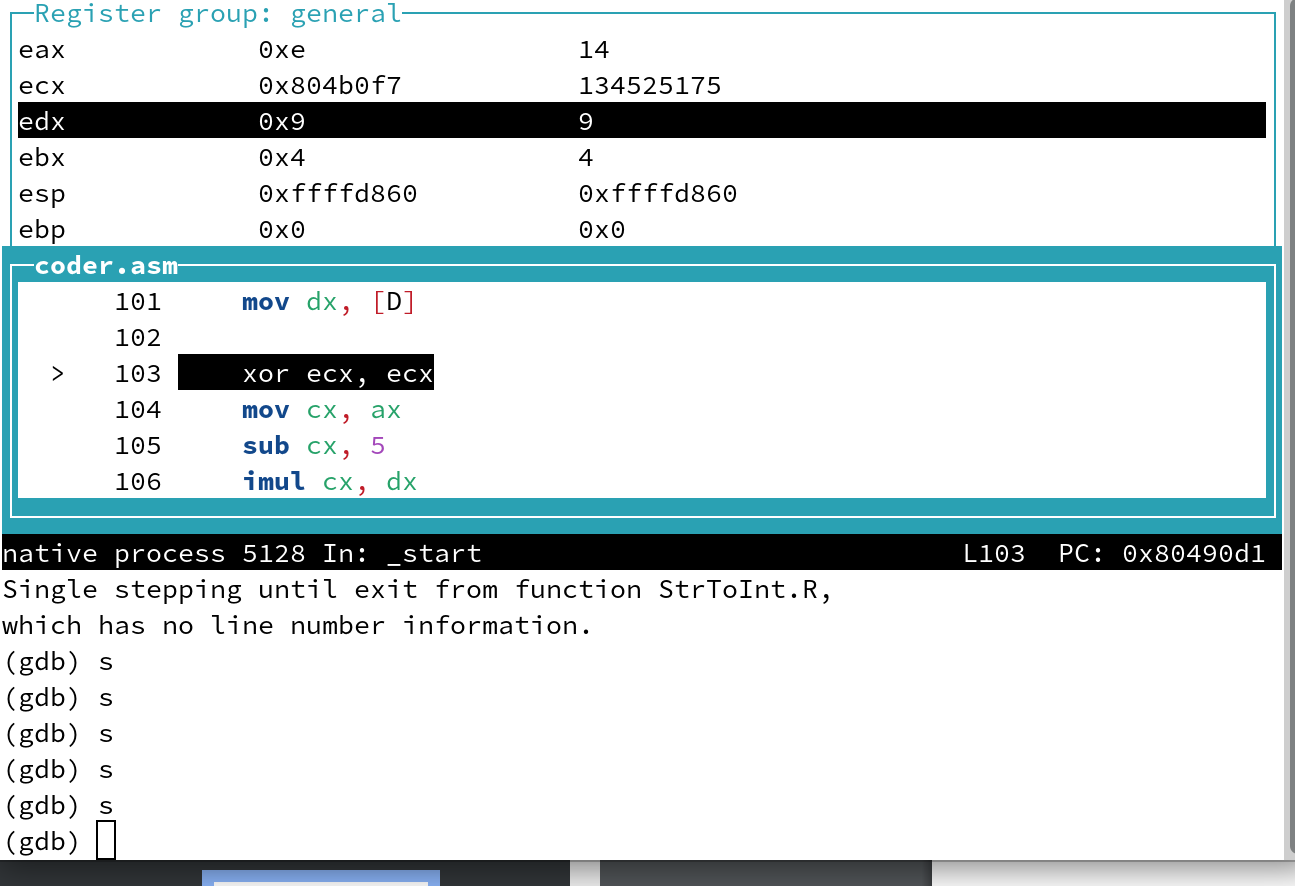
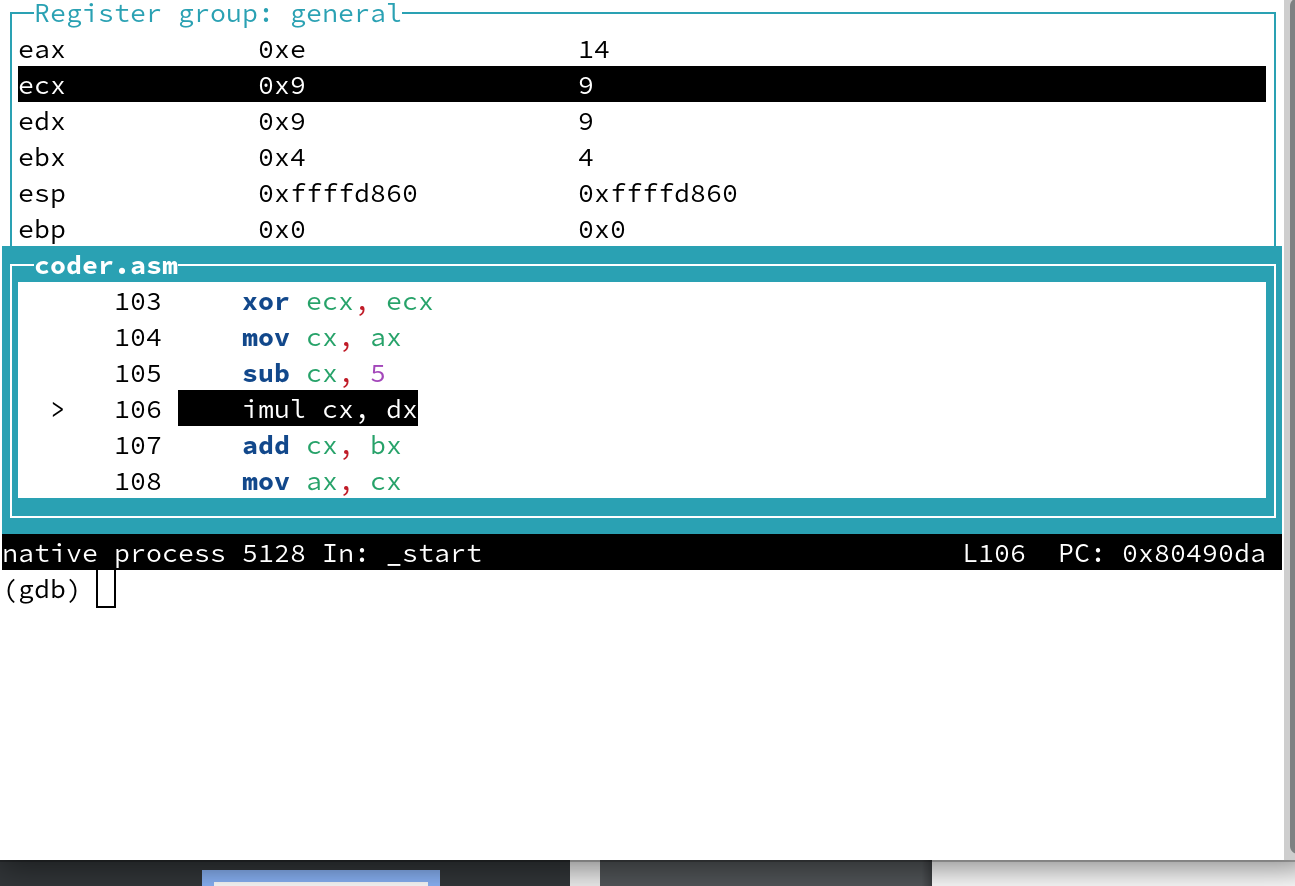
Рисунок 8 — Возврат переменной из функции и запись ее в регистр eax.

Рисунок 9 — Записываем в качестве параметра D цифру 9.

После ввода сообщения для инициализации D, аналогично инициализации переменной B, программа переходит в функцию StrToInt для конвертации строки в число.

Рисунок 10 — Перемещаем значения переменных B, C, D в регистры ebx, eax, edx соответсвенно.

Рисунок 11 — Выполнение арифметической операций (С — 5)

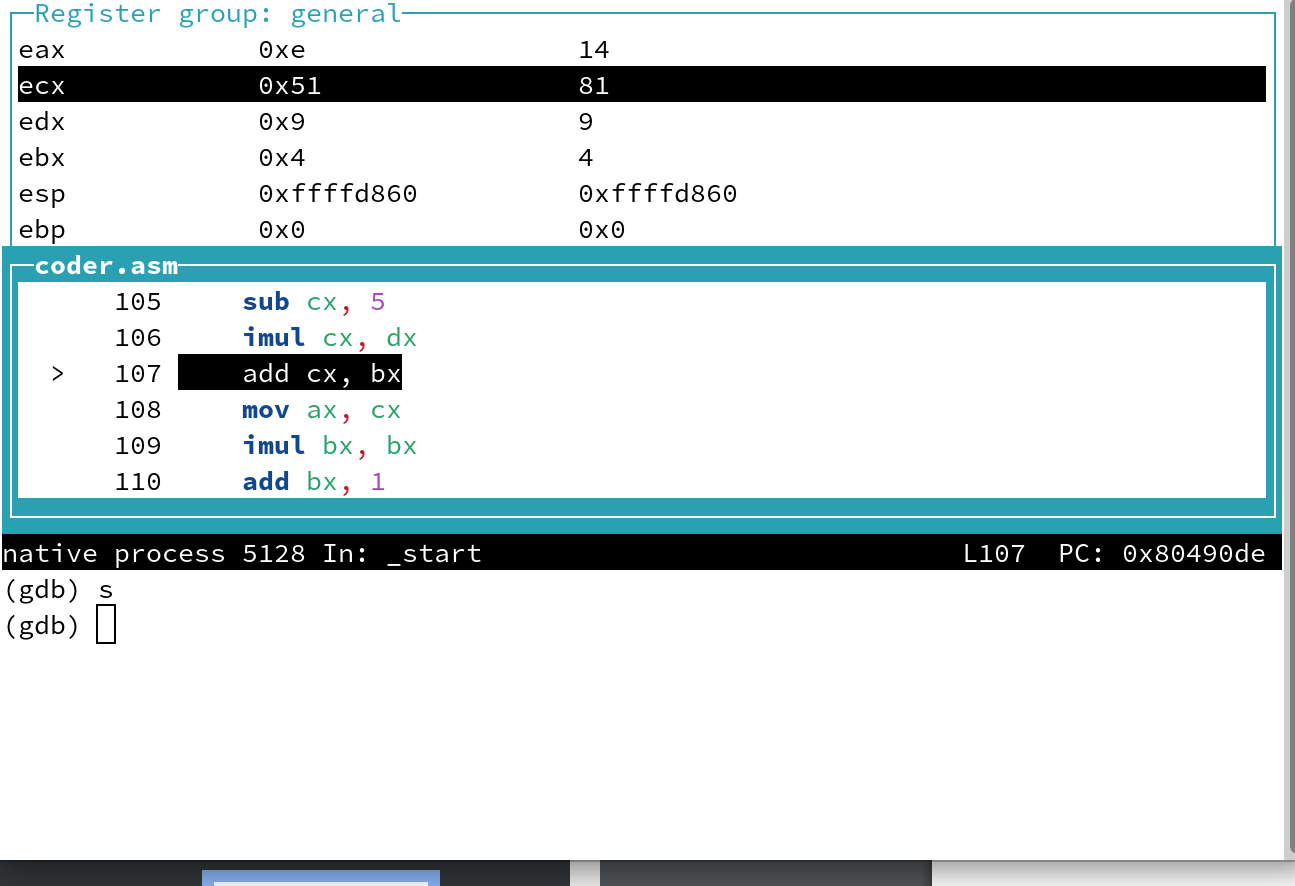
Рисунок 12 — Выполнение арифметической операций ((С — 5) \* d)

Рисунок 13

89c8 переведем из 16-ти ричную, получим двоичный код 1000100111001000. Разбор двоичного кода:

- Первые 6 бит (100010)- это команда `mov`.

- Следующие биты указывают на регистры и другие параметры команды, `d = 0` и `w = 1`, что означает копирование 4 байт (32 бит) данных между регистрами. Таким образом, команда `mov eax, ecx` копирует содержимое регистра `ecx` в регистр `eax`, перенося 4 байта данных (32 бита) между этими регистрами.

**Вывод:** в результате выполнения работы были изучены работа с арифметическими операциями, такие как: деление, умножение, сложения, вычитание.

**Контрольные вопросы:**

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

Машинная команда - это элементарная инструкция, которая задается и исполняется процессором. Примеры машинных команд в вашем ответе включают mov, add, sub, imul и idiv.

mov <регистр, память>, <регистр, память, непосредственное значение>:

Это формат команды перемещения данных. Он позволяет копировать данные из регистра или памяти в регистр, память или использовать непосредственное значение. Однако, перемещение данных непосредственно из памяти в память не разрешено.

add <регистр, память>, <регистр, память>:

Это команда сложения. Она выполняет сложение данных, находящихся в регистре и в памяти, и записывает результат обратно в регистр или в память. Однако операция сложения между данными, хранящимися непосредственно в памяти, не допускается.

sub <регистр, память>, <регистр, память>:

Это команда вычитания. Она аналогична команде сложения, но выполняет вычитание данных.

imul <регистр, память, непосредственное значение>:

Это команда целочисленного умножения. Она умножает значение в регистре или памяти на непосредственное значение и сохраняет результат.

idiv <регистр, память, непосредственное значение>:

Это команда целочисленного деления. Она делит значение в регистре или памяти на непосредственное значение и сохраняет частное.

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

Мнемоники основных команд целочисленной арифметики включают в себя следующие:

1. **ADD**: Сложение.
2. **SUB**: Вычитание.
3. **IMUL**: Умножение целых чисел.
4. **IDIV**: Целочисленное деление с остатком.

Форматы которые можно использовать с этими мнемониками:

1. **Регистр-регистр (register-register)**: Обе операнды хранятся в регистрах процессора.
2. **Регистр-память (register-memory)**: Один операнд хранится в регистре процессора, а другой - в памяти.
3. **Память-регистр (memory-register)**: Один операнд хранится в памяти, а другой - в регистре процессора.
4. **Непосредственное значение (immediate)**: Один операнд содержит непосредственное значение, а другой - регистр или память.

**3. Сформулируйте  основные  правила  построения  линейной  программы вычисления заданного выражения.**  
**Основные правила:**

1. **Объявление переменных**: В секции неинициализированных данных объявите переменные, которые будут использоваться в программе.
2. **Ввод данных**: Выполните системные команды ввода данных для получения символов из буфера.
3. **Преобразование символов в числа**: Используйте целочисленные машинные команды для преобразования символов в численные значения.
4. **Выполнение вычислений**: Напишите часть программы, отвечающую за выполнение вычислений с использованием целочисленных машинных команд.
5. **Преобразование числа в символьное представление**: Используйте системную команду для преобразования полученного числа в символьное представление.
6. **Вывод результата**: Выведите символьное представление результата с помощью системной команды вывода.

**4. Почему  ввод­вывод  на  языке  ассемблера  не  программируют  с  использованием  соответствующих  машинных  команд?  Какая  библиотека  используется  для  организации  ввода  вывода  в  данной  лабораторной?**  
Ввод-вывод на языке ассемблера часто не программируются с использованием машинных команд из-за сложности аппаратных интерфейсов. Вместо этого, для этой цели используются системные команды syscall, которые обеспечивают удобный доступ к операционной системе.

**5. Расскажите, какие операции используют при организации вводавывода.**

 Используются команды пересылки данных и системная команда  syscall .