|  |  |
| --- | --- |
| **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 (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.03 Прикладная информатика.**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 1 |

Название: Изучение среды и отладчика ассемблера

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-45Б |  |  | Л.Э. Барсегян |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Я. С. Петрова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2022

**Цель работы:** изучение процессов создания, запуска и отладки программ на ассемблере Nasm под управлением операционной системы Linux, а также особенностей описания и внутреннего представления данных.

**Задания 1.2.1-1.2.2**

Создадим каталог и подкаталог, используя команду mkdir и объявим подкаталог labs/lab1 текущим (см. Рисунок 1)

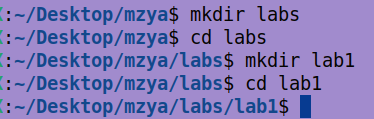


Рисунок 1 — создание каталога labs и подкаталога lab1

**Задание 1.2.3**

Введем заготовку 32-х или 64-х разрядной программы на ассемблере.

Сохраним программу с именем lab1.asm в подкаталоге labs/lab1. (см. Рисунок 2)

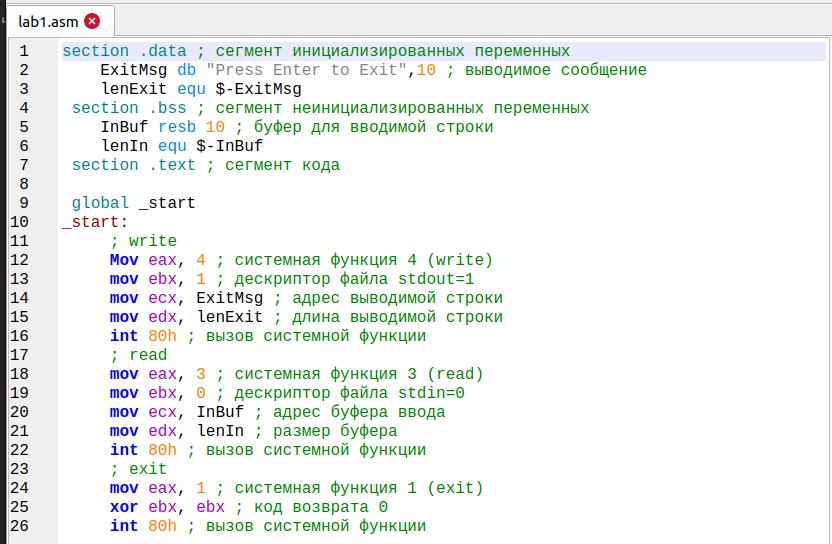


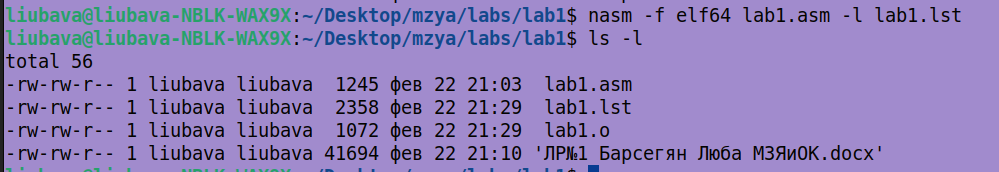
Рисунок 2 - Текст 64-разрядной программы

**Задание 1.2.4**

Выполним трансляцию программы с листингом. Для 64-разрядной программы введем в терминале команду:

**nasm -f elf64 lab1.asm -l lab1.lst**

Убедимся, что операция прошла без ошибок, воспользовавшись командой ls. (см. Рисунок 3)

 Рисунок 3 — Проверка трансляции программы с листингом.

**Задание 1.2.5**

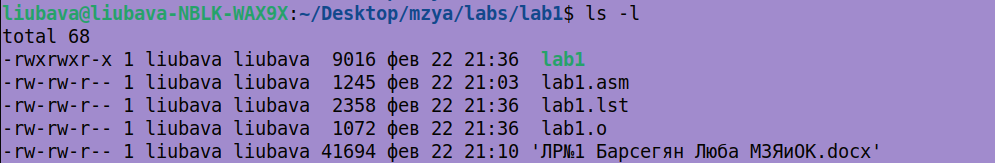
Для компоновки 64-х разрядной программы, которая будет

выполняться на компьютере той же размерности, следует ввести следующую команду:

**ld -o lab1 lab1.o**

Если все прошло без ошибок, то в том же каталоге появится файл

исполняемой программы lab1. Проверим с помощью поманды ls. (см. Рисунок 4)

Рисунок 4 — Проверка создания исполняемого файла lab1  
 **Задание 1.2.6**

Запустим программу на выполнение, как указано на рисунке 5. Запущенная программа выводит текст «Press Enter to Exit» и ожидает нажатия клавиши Enter. После нажатия клавиши Enter выполнение программы завершится.

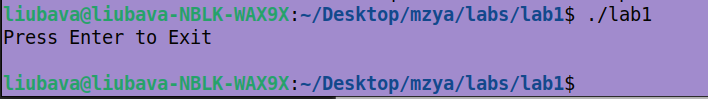
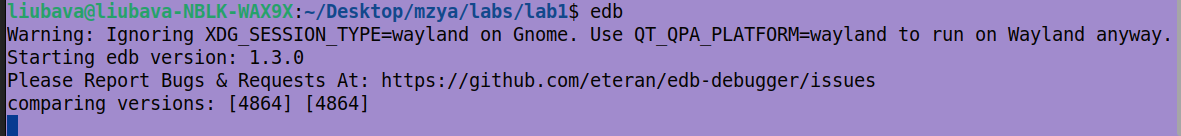


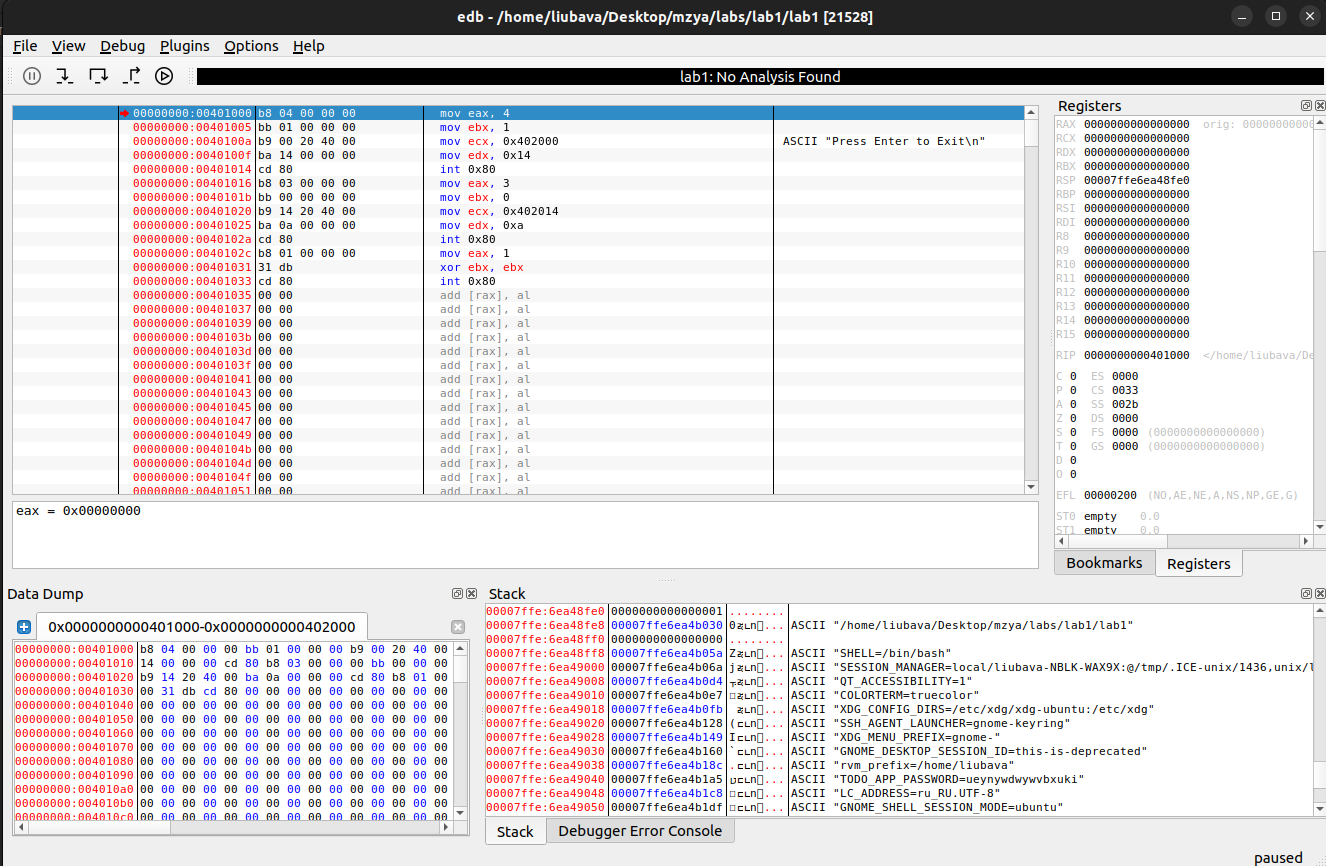
Рисунок 5 — Запуск программы

**Задание 1.2.7**

Запустим отладчик edb. Для этого следует в окне терминала ввести

команду «edb» (см. Рисунок 6) и открыть исполняемую программу lab1 (см Рисунок 7).

 Рисунок 6 — запуск отладчика edb

 Рисунок 7 — открытие исполняемой программы lab1 в «edb»

**Задание 1.2.8**

Для изучения возможностей отладчика добавим в код программы несколько команд для вычисления результата следующего выражения: X = A + 5 — B.

Данные программы зададим константами, поместив их описание в раздел инициализированных данных .data. Для результата вычислений — переменной Х — необходимо зарезервировать место, поместив описание соответствующей неинициализированной переменной в раздел неинициализированных данных .bss. Фрагмент кода программы, выполняющей сложение и вычитание,

поместим в сегмент кодов после метки start. Изменения показаны на рисунке 8.

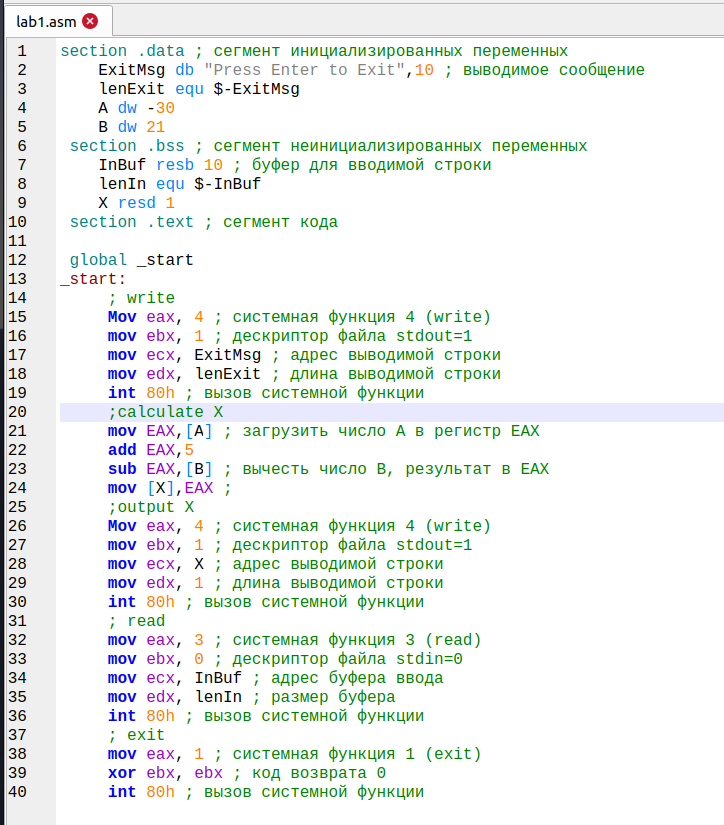


Рисунок 8 — Код программы lab1.asm

Выполним трансляцию, компоновку программы и загрузку в отладчик.

**Задание 1.2.9**

Проследим в отладчике выполнение программы и зафиксируем результаты выполнения каждой выполненной программы в таблицу 1.

|  |  |  |
| --- | --- | --- |
| **Код операции** | **Код операции в отладчике** | **Код операции в шестнадцатеричном представлении** |
| mov EAX,[A] | mov eax, [0x402014] |  |
| add EAX,5 | add eax, 5 |  |
| sub EAX,[B] | sub eax, [0x402016] |  |
| mov [X],EAX | mov [0x402022], eax |  |

Таблица 1

Рассмотрим подробнее первую операцию с переменной А. Как можно заметить, в отладчике А представлено в виде адреса. Перейдем по этому адресу (см. рисунок 9) и переведем содержимое из шестнадцатеричной системы счисления в десятеричную, учитывая обратный порядок записи. (см. рисунок 10).



Рисунок 9 — адрес переменной А

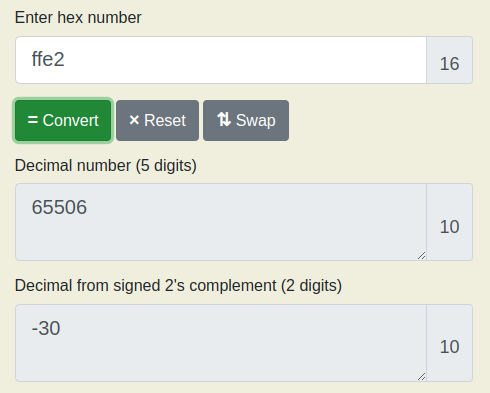


Рисунок 10 — перевод числа А в десятеричную систему счисления

Как можно заметить, получили то же число, что и присвоили переменной А в секции инициализированных переменных.

При выполнении операции mov EAX,[A], в регистр записывается адрес переменной А (см. рисунок 11)



Рисунок 11 — содержимое регистра RAX

после выполнения первой операции

После выполнения операции add EAX,5 содержимое регистра RAX увеличивается на 5, что показано на рисунке 12.



Рисунок 12 — содержимое регистра RAX

после выполнения второй операции

**Задание 1.2.10**

Занесем следующие строки в разделы описания инициализированных и неинициализированных данных и определим с помощью отладчика внутреннее представление этих данных в памяти.

**val db 255**

**chart dw 256**

**lue3 dw -128**

**v5 db 10h**

**db 100101B**

**beta db 23,23h,0ch**

**sdk db "Hello",10**

**min dw -32767**

**ar dd 12345678h**

**valar times 5 db**

**alu resw 10**

**f1 resb 5**

С помощью отладчика определим внутреннее представление этих данных в памяти и занесем результаты в таблицу 2.

|  |  |
| --- | --- |
| Название переменной | Внутреннее представление |
| val |  |
| chart |  |
| lue3 |  |
| v5 |  |
| beta |  |
| sdk |  |
| min |  |
| ar |  |
| valar |  |
| alu |  |
| f1 |  |

**Задание 1.2.11**

Определим в памяти следующие данные:

а) целое число 25 размером 2 байта со знаком;

б) двойное слово, содержащее число -35;

в) символьную строку, содержащую ваше имя (русскими буквами и

латинскими буквами).

Описание данных в сегменте инициализированных данных представлено на рисунке 13.

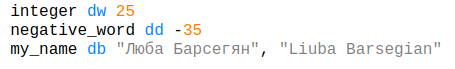


Рисунок 13 — описание данных задания 1.2.11

Внутреннее представление данных запишем в таблицу 3.

|  |  |  |
| --- | --- | --- |
| Описание | Внутреннее представление | Пояснение |
| Integer dw 25 |  | Так как порядок записи байтов обратный, то содержимое переменной — 0019, причем первые два нуля незначащие. Переводя число 19 из шестнадцатеричной СС в десятеричную, получаем число 25. |
| negative\_word dd -35 |  | DD (16) = -35 (10) |
| my\_name db «Люба Барсегян», «Liuba Barsegian» |  | Пояснение ниже |

Таблица 3 — Внутреннее представление переменных задания 1.2.11

На рисунках 14 и 15 можно увидеть перевод строк, содержащих мое имя, на русском и на английском соответственно.

Рисунок 14 — Перевод строки с именем

на русском в шестнадцатеричную СС

Рисунок 15 — Перевод строки с именем

на английском в шестнадцатеричную СС

Как можно заметить по результатам, внесенным в таблицу 3, эти строки записаны последовательно и совпадают с переводом в шестнадцатеричную систему счисления, а также заканчиваются терминирующим нулем, что свойственно строкам.

**Задание 1.2.12**

Определим в программе числа, которые во внутреннем представлении отладчика будут выглядеть как **25 00.**

Для этого вспомним, что порядок записи обратный, следовательно, само число должно быть равно числу **0025.** Переведем это число из шестнадцатеричной системы счисления в десятеричную и получим **37.**

Запишем число 37 несколькими способами (см. рисунок 16)

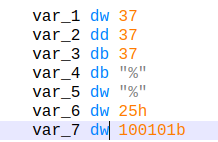


Рисунок 16 — объявление переменных

Внутреннее представление данных занесем в таблицу 4.

|  |  |
| --- | --- |
| Объявление | Внутреннее представление |
| Var\_1 dw 37 |  |
| Var\_2 dd 37 |  |
| Var\_3 db 37 |  |
| Var\_4 db «%» |  |
| Var\_5 dw «%» |  |
| Var\_6 dw 25h |  |
| Var\_7 dw 100101b |  |

Таблица 4 — внутренне представление данных задания 1.2.12

Замечание: в случаях 3 и 4, где используется db, выделяется лишь один байт памяти, поэтому незначащие нули не записываются и число остается в виде 25.

Теперь определим в программе числа, которые во внутреннем представлении отладчика будут выглядеть как **00 25.**

Помня об обратном порядке записи, запишем само число, как равное числу **2500.** Переведем это число из шестнадцатеричной системы счисления в десятеричную и получим **9472.**

Запишем число 9472 несколькими способами (см. рисунок 17)

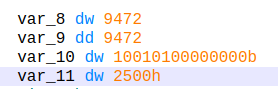


Рисунок 17 — объявление переменных

Внутреннее представление данных занесем в таблицу 5.

|  |  |
| --- | --- |
| Объявление переменных | Внутреннее представление |
| var\_8 dw 9472 |  |
| var\_9 dd 9472 |  |
| var\_10 dw 10010100000000b |  |
| var\_11 dw 2500h |  |

Таблица 5 — внутреннее представление данных задания 1.2.12

**Задание 1.2.13**

Добавим в программу переменную F1=65535 размером слово и

переменную F2= 65535 размером двойное слово. Также вставим в программу

команды сложения этих чисел с 1:

**add[F1],1**

**add[F2],1**

На рисунке 18 код программы сложения переменных F1 и F2 с 1.

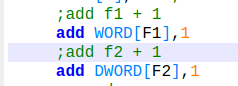


Рисунок 18 — код программы 1.2.13

Для начала найдем обратимся по адресу переменных и посмотрим содержимое (см. рисунки 19 и 20 для переменных F1 и F2 соответственно)



Рисунок 19 — внутреннее представление F1



Рисунок 20 — внутреннее представление F2

Выполним программу и зафиксируем изменения.

В случае с dw произошло переполнение и остались нули, так как изначально был выделен всего лишь один байт (см. рисунок 21)



Рисунок 21 — прибавление 1 к F1

Рассмотрим случай с dd, представленный на рисунке 22.



Рисунок 22 — прибавление 1 к F2

Так как под переменную F2 выделено 4 байта памяти, то рассмотрим 4 байта в обратном порядке. Число 0001000 переведем из шестнадцатеричной СС в десятеричную и получим число 65536. Следовательно, переполнение не произошло из-за достаточного количества выделенной памяти, операция добавления 1 к содержимому переменной прошла успешно.

Обратим внимание на флаги при выполнении данных операций. Флаги при добавлении 1 к F1 показаны на рисунке 23, а флаги при добавлении 1 к F2 — на рисунке 24.

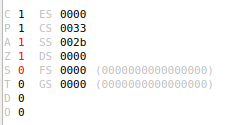


Рисунок 23 — флаги при выполнении первой операции

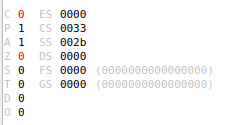
****

Рисунок 24 — флаги при выполнении второй операции

Распишем флаги и их назначения.

С — флаг переноса из старшего байта

Р — флаг четности

А — флаг переноса из младшего байта в старший

Z — флаг, указывающий, что возвращаемый результат равен нулю

O — флаг переполнения

**Контрольные вопросы**

**1. Дайте определение ассемблеру. К какой группе языков он относится?**

**Язык ассемблера —** низкоуровневый язык программирования, состоящий из операций, которые представляют собой команды процессора. Язык ассемблера относится к группе машинно-зависимых языков.

**2. Из каких частей состоит заготовка программы на ассемблере?**

Заготовка программы на языке ассемблера состоит из трех частей:

- .text (сегмент кода)

- .data (сегмент инициализированных данных)

- .bss (сегмент неинициализированных данных)

**3. Как запустить программу на ассемблере на выполнение? Что**

**происходит с программой на каждом этапе обработки?**

Для подготовки программы к выполнению сперва

вызывают транслятор nasm и компоновщик ld следующей командой:

**nasm -f elf64 lab1.asm -l lab1.lst**

В результате работы транслятор создает объектный файл, которые затем подается на вход компоновщика:

**ld -o lab1 lab1.o**

Компоновщик формирует исполняемую программу.

**4. Назовите основные режимы работы отладчика. Как осуществить**

**пошаговое выполнение программы и просмотреть результаты выполнения**

**машинных команд.**

– F7 – выполнить шаг с заходом в тело процедуры;

– F8 – выполнить шаг, не заходя в тело процедуры.

– F9 - выполнить выход из тела процедуры.

Пошаговое выполнение программы можно осуществить с помощью F7.

Результаты выполнения программы

**5. В каком виде отладчик показывает положительные и отрицательные**

**целые числа? Как будут представлены в памяти числа: A dw 5,-5 ? Как те же числа будут выглядеть после загрузки в регистр AX?**

5 в шестнадцатеричной СС будет равно 5 в то время, как -5 равняется FFFB.

Проверим, в каком виде отладчик покажет эти числа. На рисунке 25 первые два байта — это число 5 во внутреннем представлении. На рисунке 26 — число -5.



Рисунок 25 — 5 во внутреннем представлении отладчика

****

Рисунок 26 — -5 во внутреннем представлении отладчика

Загрузим эти числа в регистр АХ и проверим содержимое (см. рисунки 27 и 28 соответственно)



Рисунок 27 — содержимое регистра AX после записи в него числа 5



Рисунок 28 — содержимое регистра АХ после записи в него числа -5

**6. Каким образом в ассемблере программируются выражения? Составьте**

**фрагмент программы для вычисления С=A+B, где A, В и С – целые числа**

**формата BYTE.**

Код программы на рисунке 29.

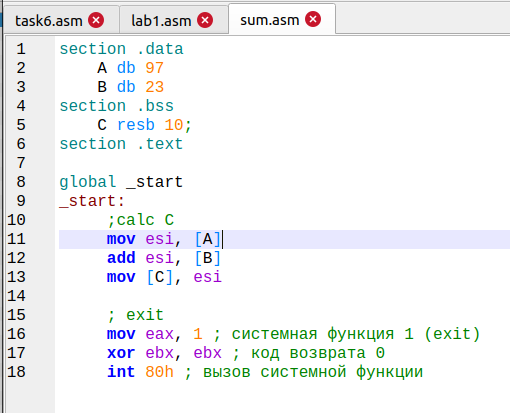


Рисунок 29 — Код программы нахождения суммы

Найдем внутреннее представление переменных A, B, C в отладчике.

Переменная А показана на рисунке 30— 61 в шестнадцатеричной это 97 в десятеричной. Переменная B показана на рисунке 31 — 17 в шестнадцатеричной это 23 в десятеричной.



Рисунок 30 — переменная А



Рисунок 31 — переменная B

Рассмотрим подробнее операции. Сперва содержимое переменной А помещается в регистр ESI, результат чего изображен на рисунке 32.

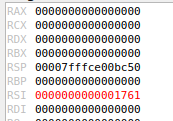


Рисунок 32 — запись содержимого переменной А в регистр ESI

Вторая операция добавляет содержимое переменной В в регистр ESI. Число 23, содержащееся в переменной В, в шестнадцатеричной СС равняется 17. Следовательно, к значению регистра ESI добавилось 17, что наблюдается на рисунке 33.

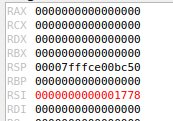


Рисунок 33 — добавление содержимого числа В в регистр ESI

После этого, содержимое регистра ESI помещается в переменную С. Найдем переменную С и посмотрим внутреннее представление отладчика (см. рисунок 33)



Рисунок 34 — содержимое переменной С

Так как размер переменных программы sum.asm всего лишь 1 байт, то нас интересует 1 младший байт, который на рисунке 32 находится слева. 78 переводим в десятеричную систему и получаем 120, что является правильным результатом сложения чисел 97 и 23.

**Вывод:** ознакомилась со средой разработки SASM, с отладчиком edb, научилась пользоваться отладчиком, находить данные как по переменным, так и по адресу. Кроме того, научилась писать простейшие программы на языке ассемблера.