|  |  |
| --- | --- |
| 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 **09.03.01 Информатика и вычислительная техника**

**Отчет**

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

**Название:**

Изучение среды и отладчика ассемблера

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-42Б |  |  | Д.В. Сулейманов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | М.В. Широкова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2022

**Вариант 2.24**

*Задание:*

1. Выполните трансляцию программы с листингом. Для этого

– для 32-х разрядной программы следует ввести команду:

nasm -f elf lab1.asm -l lab1.lst

– для 64-х разрядной:

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

В результате вы должны получить объектный модуль *lab1.o* и файл листинга *lab1.lst*. Убедитесь, что операция прошла без ошибок.

1. Для компоновки 32-х или 64-х разрядной программы, которая будет выполняться на компьютере той же размерности, следует ввести:

ld -o lab1 lab1.o

Для компоновки 32-разрядной программы, которая будет выполняться на 64-х разрядном компьютере, необходимо запросить режим эмуляции:

ld -m elf\_i386 -o lab1 lab1.o

После команды в окне терминала, если они есть, выводятся сообщения об ошибках. Если все прошло без ошибок, то в том же каталоге появится файл исполняемой программы.

1. Запустите программу на выполнение.

Запущенная программа должна вывести текст:

*Press Enter to Exit*

и ожидать нажатия клавиши **Enter**. После нажатия клавиши **Enter** выполнение программы завершится.

Следует иметь в виду, что буфер ввода предусматривает ввод до 10 символов в то время как при нажатии клавиши **Enter** в буфер записывается единственный символ с кодом 1010=0A16. Символ с кодом 1310, заносимый в буфер ввода в операционных системах семейства Windows, в операционных системах типа Linux при нажатии на клавишу **Enter** в буфер не заносится.

1. Запустите отладчик *edb*. Для этого следует в окне терминала ввести команду:

edb

Средствами графического интерфейса отладчика откройте в нем исполняемую программу *lab1* и проанализируйте, что вы видите в его окне. Найдите машинное представление программы, ее дисассемблированный код, содержимое регистров и т.д.

Выполните программу по шагам, контролируя содержимое регистров и оперативной памяти.

1. Для изучения возможностей отладчика добавьте в заготовку несколько команд для вычисления результата следующего выражения:

X=A+5-B

Данные для программы задайте константами, поместив их описание в раздел инициированных данных *.data*:

A DD -30

B DD 21

Для результата вычислений – переменной X – необходимо зарезервировать место, поместив описание соответствующей неинициализированной переменной в раздел неинициализированных данных *.bss*:

X resd 1

Фрагмент кода программы, выполняющей сложение и вычитание, поместите в сегмент кодов после метки *start*:

start: mov EAX,[A] ; загрузить число A в регистр EAX

add EAX,5 ; сложить EAX и 5, результат в EAX

sub EAX,[B] ; вычесть число B, результат в EAX

mov [X],EAX ; сохранить результат в памяти

...

Сохраните программу с тем же именем, затем выполните ее трансляцию, компоновку и загрузку в отладчик.

Зафиксируйте изменение программы в отчете.

1. Найдите в отладчике внутреннее представление исходных данных, отразите его в отчете и поясните.

Проследите в отладчике выполнение программы и зафиксируйте в отчете результаты выполнения каждой добавленной команды (изменение регистров, флагов и полей данных).

1. Введите следующие строки в разделы описания инициализированных и неинициализированных данных и определите с помощью отладчика внутренние представление этих данных в памяти. Результаты проанализируйте и занесите в отчет.
2. val1 db 255
3. chart dw 256
4. lue3 dw -128
5. v5 db 10h
6. db 100101B
7. beta db 23,23h,0ch
8. sdk db "Hello",10
9. min dw -32767
10. ar dd 12345678h
11. valar times 5 db 8
12. alu resw 10
13. f1 resb 5
14. Определите в памяти следующие данные:

– целое число 25 размером 2 байта со знаком;

– двойное слово, содержащее число -35;

– символьную строку, содержащую ваше имя (русскими буквами и латинскими буквами).

Зафиксируйте в отчете описание и внутреннее представление этих данных и дайте пояснение.

1. Определите несколькими способами в программе числа, которые во внутреннем представлении (в отладчике) будут выглядеть как **25 00** и **00 25**. Проверьте правильность ваших предположений, введя соответствующие строки в программу. Зафиксируйте результаты в отчете.
2. Добавьте в программу переменную *F1=65535* размером слово и переменную *F2=65535* размером двойное слово. Вставьте в программу команды сложения этих чисел с 1:

add WORD [F1], 1

add DWORD [F2], 1

Проанализируйте и прокомментируйте в отчете полученный результат (обратите внимание на флаги).

*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.

*Ход работы:*

1) Создадим 64-х разрядную программу, воспользовавшись шаблоном, и скомпилируем ее (см. листинг 1.1). Программа выводит сообщение, предлагающее пользователю нажать клавишу Enter для выхода.

*Листинг 1.1 – текст программы, созданной по шаблону*

1. section .data ; сегмент инициализированных переменных

2. ExitMsg db "Press Enter to Exit", 10 ; выводимое сообщение

3. lenExit equ $-ExitMsg

4. section .bss ; сегмент неинициализированных переменных

5. InBuf resb 10 ; буфер для вводимой строки

6. lenIn equ $-InBuf ; длина буфера для вводимой строки

7.

8. section .text ; сегмент кода

9. global \_start

10.

11. \_start:

12. ; вывод

13. mov rax, 1 ; системная функция 1 (write)

14. mov rdi, 1 ; дескриптор файла stdout=1

15. mov rsi, ExitMsg ; адрес выводимой строки

16. mov rdx, lenExit ; длина строки

17. syscall ; вызов системной функции

18.

19. ; ввод

20. mov rax, 0 ; системная функция 0 (read)

21. mov rdi, 0 ; дескриптор файла stdin=0

22. mov rsi, InBuf ; адрес вводимой строки

23. mov rdx, lenIn ; длина строки

24. syscall ; вызов системной функции

25.

26. ; завершение программы

27. mov rax, 60 ; системная функция 60 (exit)

28. xor rdi, rdi ; return code 0

29. syscall ; вызов системной функции

Посмотрев файл листинга, можем убедиться, что компилятор не выдал ошибок (см. листинг 1.2)

*Листинг 1.2 – вывод транслятора*

1 section .data

2 ; сегмент инициализированных переменных

3 00000000 507265737320456E74- ExitMsg db "Press Enter to Exit",10 ; выводимое сообщение

3 00000009 657220746F20457869-

3 00000012 740A

4 lenExit equ $-ExitMsg

5 ; сегмент неинициализированных переменных

6 section .bss

7 00000000 <res Ah> InBuf resb 10 ; буфер для вводимой строки

8 lenIn equ $-InBuf

9 section .text ; сегмент кода

10 global \_start

11 \_start:

12 ; write

13 00000000 B801000000 mov rax, 1 ; системная функция 1 (write)

14 00000005 BF01000000 mov rdi, 1 ; дескриптор файла stdout=1

15 0000000A 48BE- mov rsi, ExitMsg ; адрес выводимой строки

15 0000000C [0000000000000000]

16 00000014 BA14000000 mov rdx, lenExit ; длина строки

17 ; вызов системной функции

18 00000019 0F05 syscall

19 ; read

20 0000001B B800000000 mov rax, 0 ; системная функция 0 (read)

21 00000020 BF00000000 mov rdi, 0 ; дескриптор файла stdin=0

22 00000025 48BE- mov rsi, InBuf ; адрес вводимой строки

22 00000027 [0000000000000000]

23 0000002F BA0A000000 mov rdx, lenIn ; длина строки

24 ; вызов системной функции

25 00000034 0F05 syscall

26 ; exit

27 00000036 B83C000000 mov rax, 60 ; системная функция 60 (exit)

28 0000003B 4831FF xor rdi, rdi ; return code 0

29 0000003E 0F05 syscall

30 ; вызов системной функции

2) После трансляции запустим компоновщик. Сообщения об ошибках не были выведены

3) Запустим программу и убедимся в правильности ее работы. Результат можно видеть на рисунке 1.

A picture containing text

Description automatically generated

*Рисунок 1 – вывод терминала*

4) Посмотрим на рисунок 2, где изображен отладчик edb, на нем можно увидеть следующую информацию: код программы, значения инициализированных переменных, содержимое регистров.

Graphical user interface, table

Description automatically generated

*Рисунок 2 – работа в отладчике*

Написанный код был транслирован в команды процессора, которые затем были преобразованы в те же буквенные обозначения операторов

5) Дополним программу согласно заданию (см. листинг 5.1)

*Листинг 5.1 – текст программы, созданной по шаблону*

1. section .data ; сегмент инициализированных переменных

2. ExitMsg db "Press Enter to Exit", 10 ; выводимое сообщение

3. lenExit equ $-ExitMsg

4. A dd -30

5. B dd 21

6.

7. section .bss ; сегмент неинициализированных переменных

8. InBuf resb 10 ; буфер для вводимой строки

9. lenIn equ $-InBuf ; длина буфера для вводимой строки

10. X resd 1

11.

12. section .text ; сегмент кода

13. global \_start

14.

15. \_start:

16. ; вычисления

17. mov EAX, [A] ; загрузить число A в регистр EAX

18. add EAX, 5 ; сложить EAX и 5, результат в EAX

19. sub EAX, [B] ; вычесть число B, результат в EAX

20. mov [X], EAX ; сохранить результат в памяти

21.

22. ; вывод

23. mov rax, 1 ; системная функция 1 (write)

24. mov rdi, 1 ; дескриптор файла stdout=1

25. mov rsi, ExitMsg ; адрес выводимой строки

26. mov rdx, lenExit ; длина строки

27. syscall ; вызов системной функции

28.

29. ; ввод

30. mov rax, 0 ; системная функция 0 (read)

31. mov rdi, 0 ; дескриптор файла stdin=0

32. mov rsi, InBuf ; адрес вводимой строки

33. mov rdx, lenIn ; длина строки

34. syscall ; вызов системной функции

35.

36. ; завершение программы

37. mov rax, 60 ; системная функция 60 (exit)

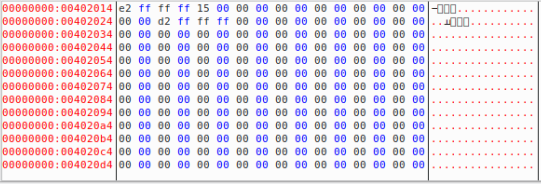
38. xor rdi, rdi ; return code 0

39. syscall ; вызов системной функции

6) Посмотрим с помощью отладчика на участок памяти, используемый программой. После выполнения соответствующих команд в память был записан результат вычислений. В таблице 1 указаны значения и их представления в десятичной системе. Отметим, что младший байт числа хранится в младшем адресе памяти. (см. рисунок 3)

*Таблица 1 – значения переменных A, B и X*

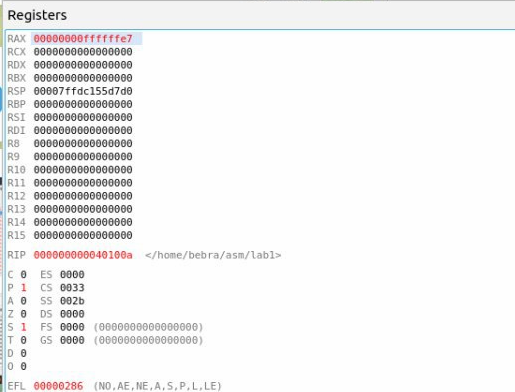
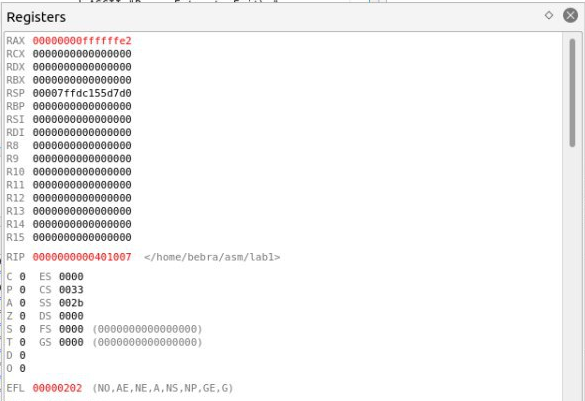
|  |  |  |  |
| --- | --- | --- | --- |
| **Адреса памяти** | **Значение** | **Число** | **Вычисления** |
| 402014–402017 | FF FF FF E2 | -30 | Доп. код 1111 1111 1110 00102  Обр. код 1111 1111 1110 00012  Пр. код 1000 0000 0001 11102  Получим -3010 |
| 402018–402021 | 00 00 00 15 | 21 | 1516 = 2110 |
| 402026-402029 | FF FF FF D2 | -46 | Доп. код 1111 1111 1101 00102  Обр. код 1111 1111 1101 00012  Пр. код 1000 0000 0010 11102  Получим -4610 |



*Рисунок 3 – переменные A, B и X в памяти*

Число D2 FF FF FF при переводе в десятичную систему счисления дает -46 = -30 + 5 - 21

Рассмотрим изменение содержимого регистров на рисунках 4.1 - 4.4.



*Рисунок 4.1, 4.2 – изменение регистров при выполнении строк 17-18 листинга 5.1*



*Рисунок 4.3, 4.4 – изменение регистров при выполнении строк 19-20 листинга 5.1*

Во время отладки подсвечивались флаги P и S.

P – флаг чётности. Его состояние говорит о том, четно ли количество единичных битов в младшем байте результата.

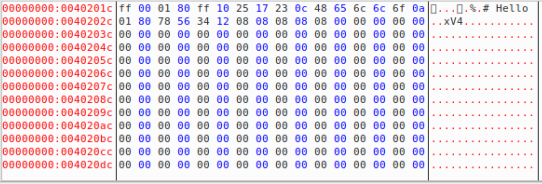
S – знаковый флаг. Он соответствует старшему биту результата последней арифметической операции, т.е. говорит о его знаке.

Обратимся к регистрам, которые тоже меняли свое значение.

Регистр RIP содержит смещение следующей подлежащей выполнению команды.

Регистр RAX используется для хранения данных, которые подсвечивались при изменении

1. Добавим объявление переменных в соответствующие секции и определим внутреннее представление данных (см. рисунок 5)



*Рисунок 5 – представление в памяти новых данных*

Рассмотрим каждое значение (см таблицу 2).

*Таблица 2 – значения, хранящиеся в памяти*

|  |  |  |
| --- | --- | --- |
| **Идентификатор** | **Значение переменной** | **Вычисления** |
| val1 | FF | 25510 = FF16 |
| chart | 01 00 | 25610 = 10016 |
| lue3 | FF 80 | Доп. код 1111 1111 1000 00002  Обр. код 1111 1111 0111 11112  Пр. код 1000 0000 1000 00002  Получим -12810 |
| v5 | 10 | 1016 = 10h. |
|  | 25 | 1001012 = 2516 |
| beta | 17  23  0с | 2310 = 1716  23h = 2316  0ch = 0C16 |
| sdk | 48 65 6C 6C 6F 0A | Представление строки в памяти в соответствии с таблицей ASCII |
| min | 80 01 | Доп. код 1000 0000 0000 00012  Получим 800116 |
| ar | 12 34 56 78 | 1234567816 = 12345678h. |
| valar | 08 08 08 08 08 |  |
| alu | Не инициализирована |  |
| fl | Не инициализирована |  |

8) Определим в разделе инициализированных данных новые переменные:

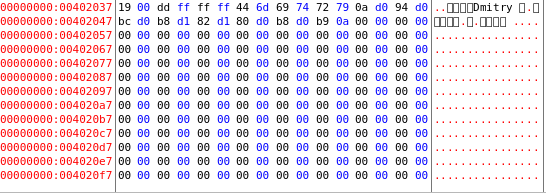
*Листинг 8.1 – определение переменных с числовыми значениями и строками*

value1    dw  25 ; целое число 25 размером 2 байта со знаком

value2    dd -35 ; двойное слово, содержащее число -35

name      db “Dmitry”, 10        ; символьная строка с именем

name\_ru   db “Дмитрий”, 10   ; строка с именем на кириллице



*Рисунок 6 – представление в памяти новых данных*

Рассмотрим, каким образом они хранятся в памяти (смотри таблицу 3).

*Таблица 3 – добавленные значения*

|  |  |  |
| --- | --- | --- |
| **Идентификатор** | **Значение переменной** | **Комментарии** |
| value1 | 00 19 | 2510 = 1916 |
| value2 | FF FF FF DD | Число -3510 представлено в дополнительном коде |
| name | 44 6D 69 74 72 79 0A | Представление строки в соответствии с таблицей ASCII |
| name\_ru | C4 EC E8 F2 F0 E8 E9 0A | Представление строки в соответствии с таблицей кодировки Windows-1251 |

9) Рассмотрим данные, добавленные в программу, для представления в необходимом виде (см. листинг 9.1 и рисунок 7).

*Листинг 9.1 – определение переменных с внутреннем представлением 25 00 или 00 25*

x1      db      25h, 0

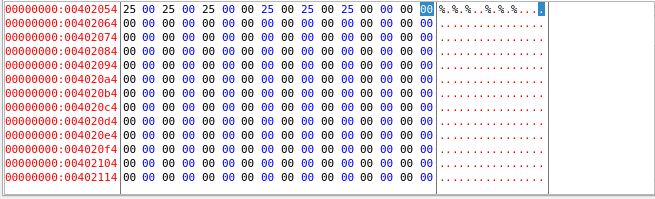
x2      dw      25h

x3      dw      '%'

y1      db      0, 25h

y2      dw      2500h

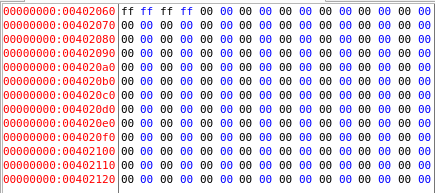
y3      db      0, '%'



*Рисунок 7 – представление в памяти новых данных*

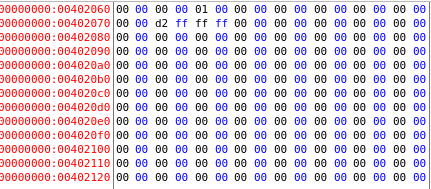
10) Число 65535 полностью заполняет разрядную сетку, состоящую из двух байт, единицами. Очевидно, что при увеличении на любое число произойдет переполнение.

На рисунках 9-10 можно видеть значения переменных F1 и F2, расположенных начиная с адреса 402060.



*Рисунок 8 – переменные F1 (FF FF16) и F2 (00 00 FF FF16) до изменения*

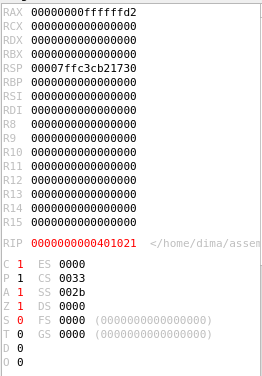
При выполнении операций число F1 должно стать равным 0 из-за переполнения, а число F2 будет равно 65536.



*Рисунок 9 – переменные F1 (00 0016) и F2 (00 01 00 0016) после изменения*

Значения переменных такие же, как и ожидалось.

При увеличении F1 был поднят флаг переноса CF, а не переполнения OF (см. рисунок 10), поскольку для беззнаковых чисел именно CF сигнализирует о переполнении. Также флаг переноса из тетрады AF получил значение 1, так как произошел перенос из тетрады; флаг SF – 0, поскольку полученное число положительное. Флаг ZF получил значение 1 при увеличении F1, потому что результат вычислений – 0, а при увеличении F2 получил значение 0. Флаг CF при увеличении F2 равен 0, поскольку переполнения не случилось.



*Рисунок 10 – выполнение операции сложения к переменной F1*

*Контрольные вопросы*

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

Язык ассемблера, или ассемблер, — язык низкого уровня с командами, обычно соответствующими командам процессора. Программа на языке ассемблера состоит из мнемонических команд – буквенных обозначений для числовых кодов.

2) Из каких частей состоит заготовка программы на ассемблере?

Программа на ассемблере NASM состоит из сегментов (секций) следующих типов:

– *.text* – текст программы;

– *.data* – сегмент инициализированных данных, содержащий их объявление с начальными значениями;

– *.bss* (англ. block starting symbol) – сегмент неинициализированных данных, содержащий их объявление с указанием размерности и количества.

3) Как запустить программу на ассемблере на выполнение? Что происходит с программой на каждом этапе обработки?

Сначала используют транслятор Nasm, результатом работы которого является объектный модуль. Затем компоновщик ld формирует исполняемую программу.

4) Назовите основные режимы работы отладчика. Как осуществить пошаговое выполнение программы и просмотреть результаты выполнения машинных команд.

При помощи отладчика можно выполнять программу по шагам, заходя или не заходя в подпрограммы, и при этом анализировать изменения содержимого регистров, памяти и стека.

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?

Положительные числа хранятся в обычном двоичном формате, отрицательные – в дополнительном коде.

Переменная А типа данных «слово» занимает 2 байта. Следовательно, 5 в двоичном представлении 00000000 000001012, а -5 – 11111111 111110112. В памяти используется обратный порядок байт, т.е. там мы увидим 00000101 00000000 и 11111011 11111111. При загрузке в регистр процессора будет использован прямой порядок.

6) Каким образом в ассемблере программируются выражения? Составьте

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

формата BYTE.

Выражения в ассемблере программируются при помощи мнемонических команд и параметров, передаваемых им.

mov al, [A]

add al, [B]

mov [C], al

***Вывод:*** в рамках лабораторной работы были изучены процесс создания, запуска и отладки программ на ассемблере NASM, была рассмотрена специфика программирования на языке ассемблера.