Лабораторная работа №6

Архитектура ОС

Гаязов Рузаль Ильшатович НКАбд-04-24

16 ноября 2023

Цель лабораторной работы

Освоение арифметических инструкций языка ассемблера NASM.

Теоретическое введение

Адресация в NASM

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обработке. Это могут быть данные хранящиеся в регистре или в ячейке памяти. Далее рассмотрены все существующие способы задания адреса хранения операндов – способы адресации. Существует три основных способа адресации: • Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: mov ax,bx. • Непосредственная адресация – значение операнда задается непосредственно в команде, Например: mov ax,2. • Адресация памяти – операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию. Например, определим переменную intg DD 3 – это означает, что задается область памяти размером 4 байта, адрес которой обозначен меткой intg. В таком случае, команда mov eax,[intg] копирует из памяти по адресу intg данные в регистр eax. В свою очередь команда mov [intg],eax запишет в память по адресу intg данные из регистра eax. Также рассмотрим команду mov eax,intg В этом случае в регистр eax запишется адрес intg. Допустим, для intg выделена память начиная с ячейки с адресом 0x600144, тогда команда mov eax,intg аналогична команде mov eax,0x600144 – т.е. эта команда запишет в регистр eax число 0x600144.

Арифметические операции в NASM

Схема команды целочисленного сложения add (от англ. addition - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда. Команда add работает как с числами со знаком, так и без знака и выглядит следующим образом: add , Допустимые сочетания операндов для команды add аналогичны сочетаниям операндов для команды mov. Так, например, команда add eax,ebx прибавит значение из регистра eax к значению из регистра ebx и запишет результат в регистр eax. Примеры: add ax,5 ; AX = AX + 5 add dx,cx ; DX = DX + CX add dx,cl ; Ошибка: разный размер операндов. Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает аналогично команде add и выглядит следующим

образом: sub , Так, например, команда sub ebx,5 уменьшает значение регистра ebx на 5 и записывает результат в регистр ebx. Довольно часто при написании программ встречается операция прибавления или вычитания единицы. Прибавление единицы называется инкрементом, а вычитание — декрементом. Для этих операций существуют специальные команды: inc (от англ. increment) и dec (от англ. decrement), которые увеличивают и уменьшают на 1 свой операнд. Эти команды содержат один операнд и имеет следующий вид: inc dec Операндом может быть регистр или ячейка памяти любого размера. Команды инкремента и декремента выгодны тем, что они занимают меньше места, чем соответствующие команды сложения и вычитания. Так, например, команда inc ebx увеличивает значение регистра ebx на 1, а команда inc ax уменьшает значение регистра ax на 1. Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака neg: neg Команда neg рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Операндом может быть регистр или ячейка памяти любого размера. mov ax,1 ; AX = 1 neg ax ; AX = -1 Умножение и деление, в отличии от сложения и вычитания, для знаковых и беззнаковых чисел производиться по-разному, поэтому существуют различные команды. Для беззнакового умножения используется команда mul (от англ. multiply – умножение): mul Для знакового умножения используется команда imul: imul Для команд умножения один из сомножителей указывается в команде и должен находиться в регистре или в памяти, но не может быть непосредственным операндом. Второй сомножитель в команде явно не указывается и должен находиться в регистре EAX,AX или AL, а результат помещается в регистры EDX:EAX, DX:AX или AX, в зависимости от размера операнда. Пример использования инструкции mul: a dw 270 mov ax, 100 ; AX = 100 mul a ; AX = AX*a, mul bl ; AX = AL*BL mul ax ; DX:AX = AX*AX

Для деления, как и для умножения, существует 2 команды div (от англ. divide - деление) и idiv: div ; Беззнаковое деление idiv ; Знаковое деление В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом. Местоположение делимого и результата для команд деления зависит от размера делителя. Кроме того, так как в результате деления получается два числа – частное и остаток, то эти числа помещаются в определённые регистры.

Перевод символа числа в десятичную символьную запись

Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно стандарту ASCII каждый символ кодируется одним байтом. Расширенная таблица ASCII состоит из двух частей. Первая (символы с кодами 0-127) является универсальной (см. Приложение.), а вторая (коды 128-255) предназначена для специальных символов и букв национальных алфавитов и на компьютерах разных типов может меняться. Среди инструкций NASM нет такой, которая выводит числа (не в символьном виде). Поэтому, например, чтобы вывести число, надо предварительно преобразовать его цифры в ASCII-коды этих цифр и выводить на экран эти коды, а не само число. Если же выводить число на экран непосредственно, то экран воспримет его не как число, а как последовательность ASCII-символов – каждый байт числа будет воспринят как один

ASCII-символ – и выведет на экран эти символы. Аналогичная ситуация происходит и при вводе данных с клавиатуры. Введенные данные будут представлять собой символы, что сделает невозможным получение корректного результата при выполнении над ними арифметических операций. Для решения этой проблемы необходимо проводить преобразование ASCII символов в числа и обратно. Для выполнения лабораторных работ в файле in_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Это: • iprint – вывод на экран чисел в формате ASCII, перед вызовом iprint в регистр eax необходимо записать выводимое число (mov eax,). • iprintLF – работает аналогично iprint, но при выводе на экран после числа добавляет к символ перевода строки. • atoi – функция преобразует ascii-код символа в целое число и записает результат в регистр eax, перед вызовом atoi в регистр eax необходимо записать число (mov eax,).

Выполнение лабораторной работы

Практическая часть

Создал каталог для программам лабораторной работы № 6, перешел в него и создал файл lab6-1.asm: (рис. [-@fig:001]).

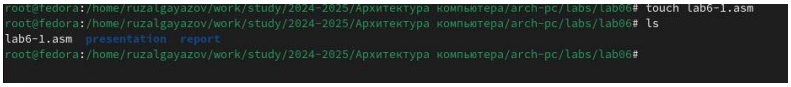

Рис.1

Ввел в файл lab6-1.asm текст программы из листинга 6.1. (рис. [-@fig:002]).

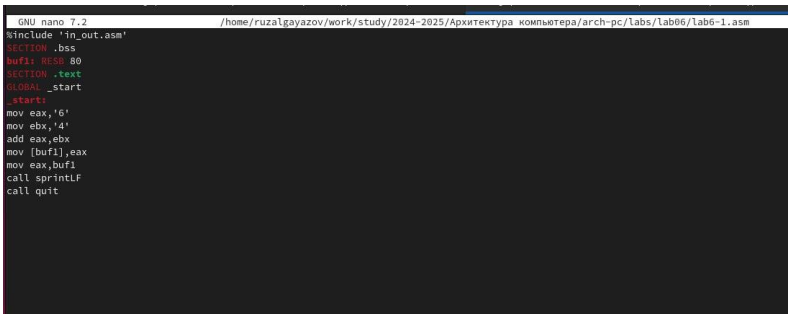

Рис.2

Создал исполняемый файл и запустил его.(Заранее скопировав файл in_out.asm из lab05) (рис. [-@fig:003]).

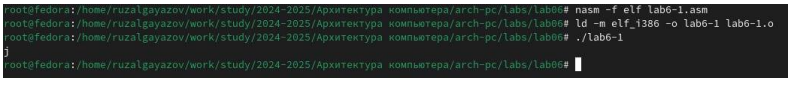

Рис.3

Далее изменим текст программы и вместо символов, запишем в регистры числа. (рис. [-@fig:004]).

```
 GNU nano 7.2                  /home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06/lab6-1.asm
%include 'in_out.asm'
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
mov [buf1],eax
mov eax,buf1
call sprintLF
call quit
```

Рис.4

Создал исполняемый файл и запустил его. Символ не отображается. (рис. [-@fig:005]).

```
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# nasm -f elf lab6-1.asm
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ld -m elf_i386 -o lab6-1 lab6-1.o
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ./lab6-1

root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06#
```

Рис.5

Создал файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и ввел в него текст программы из листинга 6.2. (рис. [-@fig:006]).

```
 GNU nano 7.2                  /home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06/lab6-2.asm
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
call iprintLF
call quit
```

Рис.6

Создал исполняемый файл и запустил его.(рис. [-@fig:007]).

```
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# touch lab6-2.asm
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# nasm -f elf lab6-2.asm
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ld -m elf_i386 -o lab6-2 lab6-2.o
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ./lab6-2
106
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06#
```

Рис.7

Аналогично предыдущему примеру изменил символы на числа.(рис. [-@fig:008]).

```
 GNU nano 7.2                  /home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06/lab6-2.asm
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

Рис.8

Создал исполняемый файл и запустил его. В результате я получил - 10.(рис. [-@fig:009]).

```
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# nasm -f elf lab6-2.asm
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ld -m elf_i386 -o lab6-2 lab6-2.o
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ./lab6-2
10
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# s
```

Рис.9

Заменил функцию iprintLF на iprint. Создал исполняемый файл и запустил его. Разница на лицо.(рис. [-@fig:010]).

```
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# nasm -f elf lab6-2.asm
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ld -m elf_i386 -o lab6-2 lab6-2.o
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ./lab6-2
10root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06#
```

Рис.10

Создал файл lab6-3.asm. Внимательно изучил текст программы из листинга 6.3 и ввел в lab6-3.asm. (рис. [-@fig:011]).

```
 GNU nano 7.2                  /home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06/lab6-3.asm
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax,5 ; EAX=5
mov ebx,2 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,3 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,3 ; EBX=3
div ebx ; EAX=EAX/3, EDX=остаток от деления
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран
mov eax,div ; вызов подпрограммы печати
call sprint ; сообщения 'Результат: '
mov eax,edi ; вызов подпрограммы печати значения
call iprintLF ; из 'edi' в виде символов
mov eax,rem ; вызов подпрограммы печати
call sprint ; сообщения 'Остаток от деления: '
mov eax,edx ; вызов подпрограммы печати значения
call iprintLF ; из 'edx' (остаток) в виде символов
call quit ; вызов подпрограммы завершения
```

Рис.11

Создал исполняемый файл и запустил его.(рис. [-@fig:012]).

```
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# nasm -f elf lab6-3.asm
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ld -m elf_i386 -o lab6-3 lab6-3.o
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ./lab6-3
Результат: 4
Остаток от деления: 1
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06#
```

Рис.12

Изменил текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$. Создал исполняемый файл и проверил его работу..(рис. [-@fig:013]).

```
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# nasm -f elf lab6-3.asm
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ld -m elf_i386 -o lab6-3 lab6-3.o
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06# ./lab6-3
Результат: 5
Остаток от деления: 1
root@fedora:/home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06#
```

Рис.13

Создал файл variant.asm. Внимательно изучил текст программы из листинга 6.4 и ввел в файл variant.asm.(рис. [-@fig:014]).

```
 GNU nano 7.2                  /home/ruzalgayazov/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab06/variant.asm
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите № студенческого билета: ',0
rem: DB 'Ваш вариант: ',0
SECTION .bss
x: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprintLF
mov ecx, x
mov edx, 80
call sread
mov eax,x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
xor edx,edx
mov ebx,20
div ebx
inc edx
mov eax,rem
call sprint
mov eax,edx
call iprintLF
call quit
```

Рис.14

Создал исполняемый файл и запустил его. Решая этот пример аналитически у меня также получается вариант - 5.(рис. [-@fig:015]).

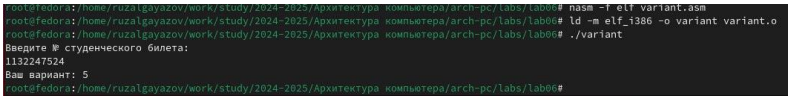

Рис.15

Теоритическая часть

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’? mov eax, rem call sprint

2. Для чего используются следующие инструкции? mov ecx, x: Указывает адрес буфера для записи. mov edx, 80: Указывает максимальную длину строки для ввода. call sread: Вызывает подпрограмму для чтения строки с консоли.

3. Для чего используется инструкция call atoi? Инструкция call atoi выполняет преобразование строки (ASCII-кодов) в число. После вызова значение записывается в регистр eax.

4. Какие строки листинга 6.4 отвечают за вычисления варианта? xor edx, edx mov ebx, 20 div ebx inc edx

5. В какой регистр записывается остаток от деления при выполнении инструкции div ebx? Остаток от деления записывается в регистр edx.

6. Для чего используется инструкция inc edx? Инструкция inc edx увеличивает значение остатка (edx) на единицу, чтобы определить номер варианта (так как номер варианта обычно начинается с 1).

7. Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений? mov eax, edx call iprintLF

Задания для самостоятельной работы

Написал программу для вычесления выражения $(9x - 8)/8$ (5-ый вариант).(рис. [-@fig:016]).

Рис.16

Создал исполняемый файл и проверил его работу для значений $x1$ и $x2$ из 6.3.(рис. [-@fig:017]).

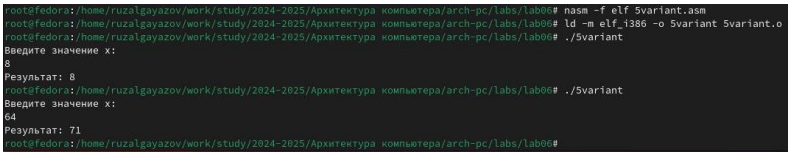

Рис.17

Вывод

Я освоил арифметические инструкции языка ассемблера NASM.

Список источников

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.
2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.
3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.
4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).