

Отчёт по лабораторной работе №6

Архитектура компьютера

Алехин Давид Андреевич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	11
5	Выводы	18
	Список литературы	19

Список иллюстраций

4.1	lab06/lab6-1.asm	11
4.2	lab6-1.asm	11
4.3	lab06/in_out.asm	11
4.4	Запуск lab6-1.asm	12
4.5	Изменённый lab6-1.asm	12
4.6	Запуск изменённого lab6-1.asm	12
4.7	Код lab6-2.asm	13
4.8	Запуск lab6-2.asm	13
4.9	Изменённый lab6-2.asm	13
4.10	Запуск изменённого lab6-2.asm	13
4.11	print на printLF	14
4.12	Текст lab6-3.asm	14
4.13	Запуск lab6-3.asm	14
4.14	Текст изменённого lab6-3.asm	15
4.15	Запуск изменённого lab6-3.asm	15
4.16	Текст variant.asm	15
4.17	Запуск variant.asm	16
4.18	lab6-1s.asm	17
4.19	Запуск lab6-1s.asm	17

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Символьные и численные данные в NASM
2. Выполнение арифметических операций в NASM
3. Задание для самостоятельной работы

3 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`.

Целочисленное сложение `add`

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

Целочисленное вычитание `sub`

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`.

Команды инкремента и декремента

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 `dec ax` уменьшает значение регистра `ax` на 1.

Команда изменения знака операнда `neg`

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` ; $AX = -1$

Команды умножения mul и imul

Умножение и деление, в отличие от сложения и вычитания, для знаковых и беззнаковых чисел производятся 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.

Команды деления div и idiv

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 запишет результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число (`mov eax, ...`).

4 Выполнение лабораторной работы

Создаю папку lab06 и в ней файл lab6-1.asm. (рис. 4.1).

```
alekhin_david@vbox:~$ mkdir ~/work/arch-pc/lab06
alekhin_david@vbox:~$ cd ~/work/arch-pc/lab06~
bash: cd: /home/alekhin_david/work/arch-pc/lab06~: Нет такого файла или каталога
alekhin_david@vbox:~$ cd ~/work/arch-pc/lab06
alekhin_david@vbox:~/work/arch-pc/lab06$ touch lab6-1.asm
alekhin_david@vbox:~/work/arch-pc/lab06$
```

Рис. 4.1: lab06/lab6-1.asm

Ввожу в lab6-1.asm текст программы 6.1. Программа вывода значения регистра eax. (рис. 4.2).

```
/home/alekhin_david/work/arch-pc/lab06/lab6-1.asm 172/172 100%
#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 sprintf
call quit
```

Рис. 4.2: lab6-1.asm

Переношу файл in_out.asm в папку lab06. (рис. 4.3).

Левая панель		Файл	Команда	Настройки	Пр
<-		~/work/arch-pc/lab06	. [^]>		<- ~/work/arch-
.и	Имя	Размер	Дата правки	.и	Имя
/..		-ВВЕРХ-	ноя 23 18:12	/..	
in_out.asm		3942	ноя 6 17:22	in_out.asm	
lab6-1.asm		172	ноя 23 18:17	*lab05-1.1	

Рис. 4.3: lab06/in_out.asm

Компеллирую и запускаю файл lab6-1.asm. (рис. 4.4).

```
alekhin_david@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
alekhin_david@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
alekhin_david@vbox:~/work/arch-pc/lab06$ ./lab6-1
j
```

Рис. 4.4: Запуск lab6-1.asm

Меняю символы на числовые значения. (рис. 4.5).

```
lab6-1.asm [-M--] 8 L: [ 1+ 7 8/ 13] *(103 / 168b) 0052 0x034 [*]
#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 sprintf
call quit
```

Рис. 4.5: Изменённый lab6-1.asm

Компеллирую и запускаю исправленный файл. В результате получается невидимый символ. (рис. 4.6).

```
alekhin_david@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
alekhin_david@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
alekhin_david@vbox:~/work/arch-pc/lab06$ ./lab6-1
```

Рис. 4.6: Запуск изменённого lab6-1.asm

Создаю файл lab6-2.asm и вписываю туда текст команды 6.2. Программа вывода значения регистра eax. (рис. 4.7).

```
lab6-2.asm      [-M--]  0 L:[  1+ 9  10/ 10] *(118 /
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov  eax,'6'
mov  ebx,'4'
add  eax,ebx
call iprintLF
call quit
█
```

Рис. 4.7: Код lab6-2.asm

Запускаю файл lab6-2.asm. (рис. 4.8).

```
alekhin_david@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
alekhin_david@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
alekhin_david@vbox:~/work/arch-pc/lab06$ ./lab6-2
106
alekhin_david@vbox:~/work/arch-pc/lab06$ █
```

Рис. 4.8: Запуск lab6-2.asm

Меняю символы на числовые значения. (рис. 4.9).

```
/home/alekhin_david/work/arch-pc/lab06/lab6-2.asm 11
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov  eax,6
mov  ebx,4
add  eax,ebx
call iprintLF
call quit
```

Рис. 4.9: Изменённый lab6-2.asm

Компелирую и запускаю исправленный файл. В результате получаем 10. (рис. 4.10).

```
alekhin_david@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
alekhin_david@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
alekhin_david@vbox:~/work/arch-pc/lab06$ ./lab6-2
10
█
```

Рис. 4.10: Запуск изменённого lab6-2.asm

Меняю print на printf и запускаю. Различие в том что print не переносит на следующую строку. (рис. 4.11).

```
alekhin_david@vbox:~/work/arch-pc/lab06$ ./lab6-2
10alekhin_david@vbox:~/work/arch-pc/lab06$
```

Рис. 4.11: print на printf

Создаю файл lab6-3.asm и вписываю туда текст команды 6.3. Программа вычисления выражения. (рис. 4.12).

```
/home/alekhin_david/work/arch-pc/lab06/lab6-3.asm
;-----
; Программа вычисления выражения
;-----
#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. 4.12: Текст lab6-3.asm

Запускаю lab6-3.asm. (рис. 4.13).

```
alekhin_david@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
alekhin_david@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
alekhin_david@vbox:~/work/arch-pc/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
```

Рис. 4.13: Запуск lab6-3.asm

Меняю значения переменных в lab6-3.asm. (рис. 4.14).

```

mov eax,4 ; EAX=5
mov ebx,6 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,2 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; EBX=3

```

Рис. 4.14: Текст изменённого lab6-3.asm

Запускаю изменённый файл. (рис. 4.15).

```

alekhin_david@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
alekhin_david@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
alekhin_david@vbox:~/work/arch-pc/lab06$ ./lab6-3
Результат: 5
Остаток от деления: 1

```

Рис. 4.15: Запуск изменённого lab6-3.asm

Создаю файл variant.asm и вписываю текст команды 6.4. Программа вычисления вычисления варианта задания по номеру студенческого билета. (рис. 4.16).

```

/home/alekhin_david/work/arch-pc/lab06/variant.asm 618/618
;-----
; Программа вычисления варианта
;-----
%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 sprintf
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. 4.16: Текст variant.asm

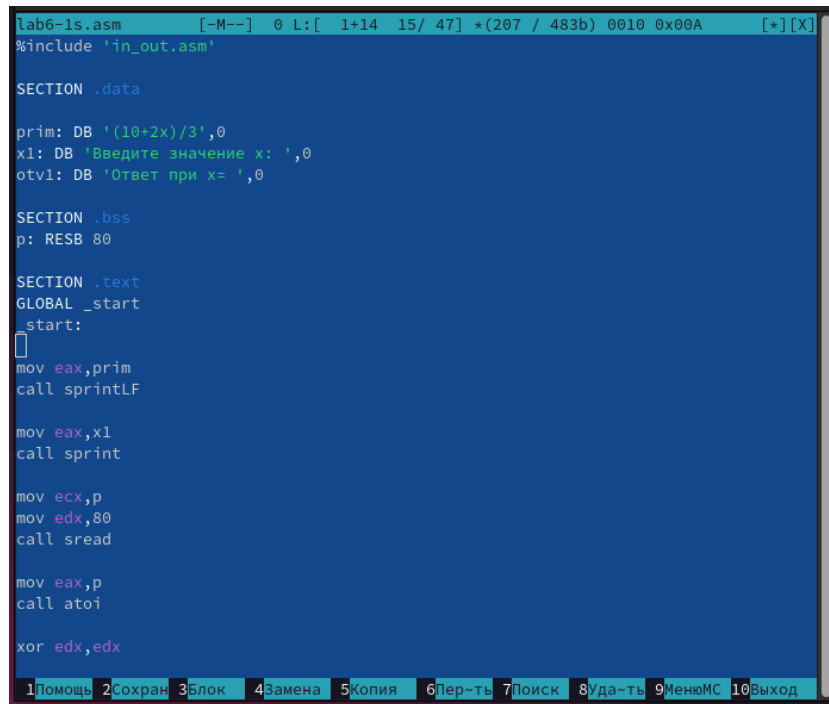
Запускаю команду, вписываю свой студ билет, получаю 1. (рис. 4.17).

```
alekhin_david@vbox:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132246840
Ваш вариант: 1
```

Рис. 4.17: Запуск variant.asm

Ответ на вопросы: 1) Какие строки листинга 7.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’? `mov eax,msg call sprintLF` 2) Для чего используются следующие инструкции? `nasm mov ecx, x mov edx, 80 call sread` Эти инструкции используются для ввода переменной X с клавиатуры и сохранения введенных данных. 3) Для чего используется инструкция “`call atoi`”? Эта инструкция используется для преобразования Кода переменной ASCII в число. 4) Какие строки листинга 7.4 отвечают за вычисления варианта? `mov ebx,20 div ebx inc edx` 5) В какой регистр записывается остаток от деления при выполнении инструкции “`div ebx`”? В регистре `ebx`. 6) Для чего используется инструкция “`inc edx`”? Для увеличения значения `edx` на 1. 7) Какие строки листинга 7.4 отвечают за вывод на экран результата вычислений? `mov eax,edx call iprintLF`

Приступаю к самостоятельному заданию. Создаю файл `lab6-1s.asm` и вписываю туда текст команды для выполнения самостоятельного задания. (рис. 4.18).



```
lab6-1s.asm  [-M--]  0 L: [ 1+14 15/ 47] *(207 / 483b) 0010 0x00A  [*][X]
#include 'in_out.asm'

SECTION .data
prim: DB '(10+2x)/3',0
x1: DB 'Введите значение x: ',0
otv1: DB 'Ответ при x= ',0

SECTION .bss
p: RESB 80

SECTION .text
GLOBAL _start
_start:
mov eax,prim
call sprintfLF

mov eax,x1
call sprintf

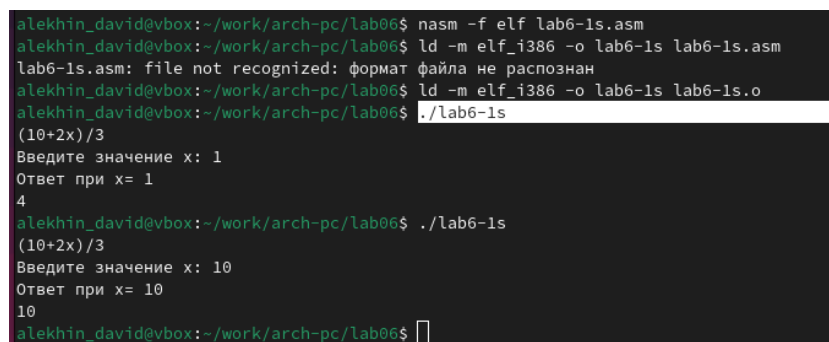
mov ecx,p
mov edx,80
call sread

mov eax,p
call atoi

xor edx,edx
```

Рис. 4.18: lab6-1s.asm

Компеллирую файл и запускаю вписывая разные значения x. (рис. 4.19).



```
alekhin_david@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-1s.asm
alekhin_david@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1s lab6-1s.asm
lab6-1s.asm: file not recognized: формат файла не распознан
alekhin_david@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1s lab6-1s.o
alekhin_david@vbox:~/work/arch-pc/lab06$ ./lab6-1s
(10+2x)/3
Введите значение x: 1
Ответ при x= 1
4
alekhin_david@vbox:~/work/arch-pc/lab06$ ./lab6-1s
(10+2x)/3
Введите значение x: 10
Ответ при x= 10
10
alekhin_david@vbox:~/work/arch-pc/lab06$
```

Рис. 4.19: Запуск lab6-1s.asm

5 Выводы

Выполнив лабораторную я изучил арифметические инструкции языка `asm`.

Список литературы

- Список литературы 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/Learning-bash-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. — БХВ- Петербург, 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).