

лабораторная работа №6.

Арифметические операции в NASM

Четвергова Мария Викторовна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	9
4	Выводы	16

Список иллюстраций

3.1	рис.1 Создание каталога для лаб.работы №6	9
3.2	рис.2 Введём в файл lab6-1.asm текст программы из листинга 6.1 .	10
3.3	рис.3 вывод программы	10
3.4	рис.4 Создание и запуск файла	11
3.5	рис.5 Внесение листинга 6.2	12
3.6	рис.6 Запуск исполняемого файла	12
3.7	рис.7 Результат программы	13
3.8	рис.8 Видоизменённая программа листинга	13
3.9	изменённая программа листинга (ч.2)	13
3.10	Результат программы	14
3.11	рис.9 Создание файла	14
3.12	рис.10 Результат программы	15
3.13	рис.11	15

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM

2 Теоретическое введение

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 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`. Схема команды целочисленного сложения `add` (от англ. addition – добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда.

Команда целочисленного вычитания `sub` (от англ. subtraction – вычитание) работает аналогично команде `add` и выглядит следующим образом

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. 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

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 размер.

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

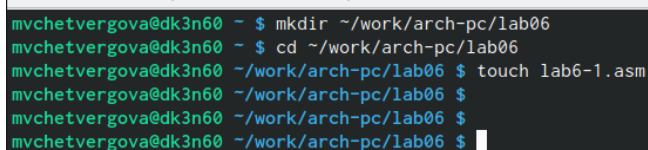
Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от `American Standard Code for Information Interchange` (Американский стандартный код для обмена информацией). Согласно 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 вы-

водить 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, ...`).

3 Выполнение лабораторной работы

1. Создайте каталог для программ лабораторной работы No 6, перейдите в него и создайте файл lab6-1.asm:



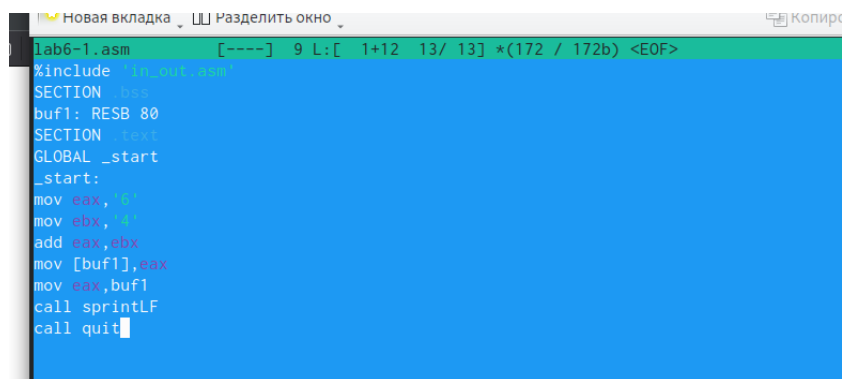
```
mvchetvergova@dk3n60 ~ $ mkdir ~/work/arch-pc/lab06
mvchetvergova@dk3n60 ~ $ cd ~/work/arch-pc/lab06
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ touch lab6-1.asm
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $
```

Рис. 3.1: рис.1 Создание каталога для лаб.работы №6

2. Рассмотрим примеры программ вывода символьных и численных значений.

Программы будут выводить значения записанные в регистр еах.

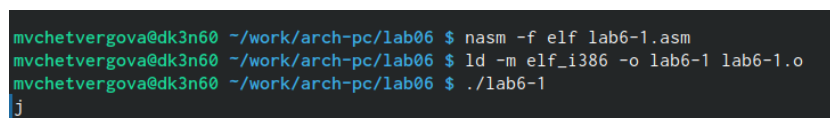
Введём в файл lab6-1.asm текст программы из листинга 6.1. В данной программе в регистр еах записывается символ 6 (mov еах,'6'), в регистр ебх символ 4 (mov ебх,'4'). Далее к значению в регистре еах прибавляем значение регистра ебх (add еах,ебх, результат сложения запишется в регистр еах). Далее выводим результат.



```
lab6-1.asm [----] 9 L: [ 1+12 13/ 13] *(172 / 172b) <EOF>
#include "lab6-1.asm"
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, 'j'
mov ebx, '4'
add eax, ebx
mov [buf1], eax
mov eax, buf1
call sprintf
call quit
```

Рис. 3.2: рис.2 Введём в файл lab6-1.asm текст программы из листинга 6.1

В данном случае при выводе значения регистра `eax` мы ожидаем увидеть число 10. Однако результатом будет символ `j`. Это происходит потому, что код символа `б` равен `00110110` в двоичном представлении (или `54` в десятичном представлении), а код символа `4` – `00110100` (`52`). Команда `add eax, ebx` запишет в регистр `eax` сумму кодов – `01101010` (`106`), что в свою очередь является кодом символа `j` (см. таблицу ASCII в приложении)



```
mychetyergova@dk3n60 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
mychetyergova@dk3n60 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
mychetyergova@dk3n60 ~/work/arch-pc/lab06 $ ./lab6-1
j
```

Рис. 3.3: рис.3 вывод программы

3. Далее изменим текст программы и вместо символов, запишем в регистры числа. Ис- правьте текст программы (Листинг 6.1) следующим образом: замените строки

`mov eax, 'б' mov ebx, '4'` на строки `mov eax, 6 mov ebx, 4` Создайте исполняемый файл и запустите его. Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. Пользуясь таблицей ASCII определите какому символу соответствует код 10.

```
%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. 3.4: рис.4 Создание и запуск файла

4. Как отмечалось выше, для работы с числами в файле in_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно.

Преобразуем текст программы из Листинга 6.1 с использованием этих функций. Создайте файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и введите в него текст программы из листинга 6.2.

```
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ mcedit lab6-1.asm
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ ./lab6-1
```

Рис. 3.5: рис.5 Внесение листинга 6.2

В результате работы программы мы получим число 106. В данном случае, как и в первом, команда add складывает коды символов '6' и '4' (54+52=106). Однако, в отличие от программы из листинга 6.1, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

5. Аналогично предыдущему примеру изменим символы на числа. Замените строки mov eax,'6' mov ebx,'4' на строки mov eax,6 mov ebx,4 Создайте исполняемый файл и запустите его. Какой результат будет получен при исполнении программы? Замените функцию iprintLF на iprint. Создайте исполняемый файл и запустите его.

```
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ ./lab6-2
106
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $
```

Рис. 3.6: рис.6 Запуск исполняемого файла

6. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения $\frac{5 \times (2 + 3)}{3}$. Создайте файл lab6-3.asm в каталоге ~/work/arch-pc/lab06: touch ~/work/arch-pc/lab06/lab6-3.asm Внимательно изучите текст программы из листинга 6.3 и введите в lab6-3.asm.

Создайте исполняемый файл и запустите его. Результат работы программы должен быть следующим

```
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ ./lab6-2
10
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $
```

Рис. 3.7: рис.7 Результат программы

```
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ ./lab6-2
10mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $
```

Рис. 3.8: рис.8 Видоизменённая программа листинга

```
; Программа вычисления выражения
; вычисления выражения и остатка от деления
%include "lab06.asm" ; подключение внешнего файла
SECTION .data
div: DB "Результат: ",0
rem: DB "Остаток от деления: ",0
66 Демидова А. В.
Архитектура ЭВМ
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, [EBX]-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. 3.9: изменённая программа листинга (ч.2)

```

mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $

```

Рис. 3.10: Результат программы

7. В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму

В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символ-ном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может быть использована функция `atoi` из файла `in_out.asm`. Создайте файл `variant.asm` в каталоге `~/work/arch-pc/lab06`:

```

mvchetvergova@dk3n60 ~ $ mkdir ~/work/arch-pc/lab06
mvchetvergova@dk3n60 ~ $ cd ~/work/arch-pc/lab06
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $ touch lab6-1.asm
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $
mvchetvergova@dk3n60 ~/work/arch-pc/lab06 $

```

Рис. 3.11: рис.9 Создание файла

#Задание для самостоятельной работы

Написать программу вычисления выражения $\boxed{x} = \boxed{x}(\boxed{x})$. Программа должна выводить выражение для вычисления, выводить запрос на ввод значения \boxed{x} , вычислять заданное выражение в зависимости от введенного \boxed{x} , выводить результат вычислений. Вид функции $\boxed{x}(\boxed{x})$ выбрать из таблицы 6.3 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы. Создайте исполняемый файл и проверьте его работу для значений $\boxed{x}1$ и $\boxed{x}2$.

```

; ~~~~~
; Программа вычисления варианта
; ~~~~~
#include 'io.h'
SECTION .data
msg: DB 'Введите x: ',0
rem: DB 'F(x) = ',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
dec eax
mul eax
mov ebx, 5
mul ebx
mov edx, eax

mov eax, rem
call sprintf
mov eax, edx
call iprintLF
call quit

```

Рис. 3.12: рис.10 Результат программы

```

mvchettergova@dk3n60 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
mvchettergova@dk3n60 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
mvchettergova@dk3n60 ~/work/arch-pc/lab06 $ ./lab6-3
Введите x:
3
F(x) = 20

```

Рис. 3.13: рис.11

4 Выводы

В ходе выполнения лабораторной работы №6 мы освоили арифметических инструкций языка ассемблера NASN. Научились применять полученные знания на практике и обучились новым навыкам работы с ЭВМ.