

Отчёт по выполнению лабораторной работы №6

Дисциплина: архитектура компьютера

Лысенко Маргарита Олеговна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Ответы на вопросы	12
6	Выполнение самостоятельной работы	13
7	Листинг написанной программы для самостоятельной работы	14
8	Выводы	16

Список иллюстраций

4.1	Создание и запуск файла	8
4.2	Редактирование программы	8
4.3	Запуск программы	8
4.4	Создание и запуск файла	9
4.5	Редактирование программы	9
4.6	Запуск программы	9
4.7	Редактирование файла	9
4.8	Создание и запуск программы	10
4.9	Создание и запуск программы	10
4.10	Редактирование программы	10
4.11	Создание и запуск программы	11
4.12	Запуск программы. Получение номера варианта	11
6.1	Запуск программы. Подстановка разных значений x	13

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Задание

Задать некоторые уровнения через консоль.

3 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 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`.

4 Выполнение лабораторной работы

Создала каталог для программ лабораторной работы No 6, перешла в него и создала файл lab6-1.asm. Ввела в него lab6-1.asm текст программы из листинга 6.1. Создала исполняемый файл и запустила его.(рис. 4.1).

```
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ./lab6-1
j
```

Рис. 4.1: Создание и запуск файла

Далее изменила текст программы и вместо символов, записала в регистры числа. (рис. 4.2).

```
mov eax, 6
mov ebx, 4
```

Рис. 4.2: Редактирование программы

Создала исполняемый файл и запустила его. Символ на экране не отображается.(рис. 4.3).

```
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ./lab6-1
molihsenko@dk8n76 ~/work/arch-pc/lab06 $
```

Рис. 4.3: Запуск программы

Создала файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и ввела в него текст программы из листинга 6.2. Запустила файл. В результате работы программы

получила число 106.(рис. 4.4).

```
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ touch lab6-2.asm
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ mc

molihsenko@dk8n76 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ./lab6-2
106
molihsenko@dk8n76 ~/work/arch-pc/lab06 $
```

Рис. 4.4: Создание и запуск файла

Аналогично предыдущему примеру изменила символы на числа. (рис. 4.5).

```
%include "lab06.inc"

SECTION .text
GLOBAL _start
_start:

mov eax,6
mov ebx,4
add eax,ebx
call iprintLF

call quit
```

Рис. 4.5: Редактирование программы

Создала исполняемый файл и запустила его. Теперь программа складывает сами числа, поэтому получается вывод 10. (рис. 4.6).

```
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ mc

molihsenko@dk8n76 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ./lab6-2
10
```

Рис. 4.6: Запуск программы

Заменяла функцию iprintLF на iprint. (рис. 4.7).

```
mov eax,6
mov ebx,4
add eax,ebx
call iprint

call quit
```

Рис. 4.7: Редактирование файла

Создала исполняемый файл и запустила его. Теперь вывод не пишется на отдельной строке. Iprint не добавляет к выводу символ переноса строки, в отличие от iprintLF. (рис. 4.8).

```
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ mc
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ^C
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ./lab6-2
10molihsenko@dk8n76 ~/work/arch-pc/lab06 $
```

Рис. 4.8: Создание и запуск программы

Создала файл lab6-3.asm. Создала исполняемый файл и запустила его. (рис. 4.9).

```
10molihsenko@dk8n76 ~/work/arch-pc/lab06 $ touch lab6-3.asm
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ mc
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
lab6-3.asm:5: warning: no operand for data declaration [-w+db-empty]
lab6-3.asm:6: warning: no operand for data declaration [-w+db-empty]
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
molihsenko@dk8n76 ~/work/arch-pc/lab06 $
```

Рис. 4.9: Создание и запуск программы

Изменила текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$. (рис. 4.10).

```
mov eax,4
mov ebx,6
mul ebx
add eax,2
xor edx,edx
mov ebx,5
div ebx
```

Рис. 4.10: Редактирование программы

Создала исполняемый файл и проверила его работу. (рис. 4.11).

```

molihsenko@dk6n34 ~/work/arch-pc/lab06 $ mc
molihsenko@dk6n34 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
molihsenko@dk6n34 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
molihsenko@dk6n34 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 5
Остаток от деления: 1
molihsenko@dk6n34 ~/work/arch-pc/lab06 $ █

```

Рис. 4.11: Создание и запуск программы

Создала файл variant.asm и запустила его. Проверила результат работы программы, вычислив номер варианта аналитически. (рис. 4.12).

```

molihsenko@dk6n34 ~/work/arch-pc/lab06 $ mc
molihsenko@dk6n34 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
molihsenko@dk6n34 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant variant.o
./variant
Введите No студенческого билета:
1132236109
Ваш вариант: 10 █

```

Рис. 4.12: Запуск программы. Получение номера варианта

5 Ответы на вопросы

1. За вывод сообщения “Ваш вариант” отвечают строки кода:

```
mov eax,rem call sprint
```

2. `mov ecx,x` используется, чтобы положить адрес вводимой строки `x` в регистр `ecx`. `mov edx,80` - запись в регистр `edx` длины вводимой строки. `call ssread` - вызов подпрограммы из внешнего файла, обеспечивающий ввод сообщения с клавиатуры.
3. `call atoi` используется для вызова подпрограммы из внешнего файла, которая преобразует `ascii`-код символа в целое число и записывает результат в регистр `eax`.

4. За вычисление варианта отвечают:

```
xor edx,edx mov ebx div ebx inc edx
```

5. Остаток от деления записывается в регистр `edx`
6. `inc edx` увеличивает значение регистра `edx` на 1.
7. За вывод на экран результатов вычислений отвечают строки:

```
mov eax,edx call iprintLF
```

6 Выполнение самостоятельной работы

Создала файл variant10.asm. Ввела в него текст программы для вычисления выражения $5(x+18)-28$. Запустила исполняемый файл. Подставила разные значения x и получила ответы. (рис. 6.1).

```
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant10 variant10.o
molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ./variant10
Введите значение переменной x: 2
Результат: 72molihsenko@dk8n76 ~/work/arch-pc/lab06 $ ./variant10
Введите значение переменной x: 3
Результат: 77molihsenko@dk8n76 ~/work/arch-pc/lab06 $
```

Рис. 6.1: Запуск программы. Подстановка разных значений x

7 Листинг написанной программы для самостоятельной работы

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите значение переменной x: ',0
rem: DB 'Результат: ',0
SECTION .bss
x: RESB 80
SECTION .text
GLOBAL _start
_start:
; Вычисление выражения
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
add eax,18; eax = eax+18 = x+18
mov ebx,5
mul ebx; EAX=EAX*EBX = (x+18)*5
```

```
sub eax, 28; eax = eax-28 = (x+18)*5-28
mov edi,eax
; Вывод результата на экран
mov eax,rem ; вызов подпрограммы печати
call sprint ; сообщение 'Результат: '
mov eax,edi ; вызов подпрограммы печати значения
call iprint ; из 'edi' в виде символов
call quit
```

8 Выводы

В ходе выполнения работ я освоила арифметические инструкции языка ассемблера NASM.