Отчёт по лабораторной работе №6

дисциплина: Архитектура компьютера

Аносов Даниил Игоревич

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Написать программу вычисления выражения $y = f(x)$. Программа должна выводить выражение для вычисления, выводить запрос на ввод значения x, вычислять заданное выражение в зависимости от введенного, выводить результат вычислений. Вид функции $f(x)$ 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 x_1 и x_2 из 6.3.

3 Выполнение лабораторной работы

Откроем терминал и создадим каталог для программ лабораторной работы №6 (рис. 3.1).

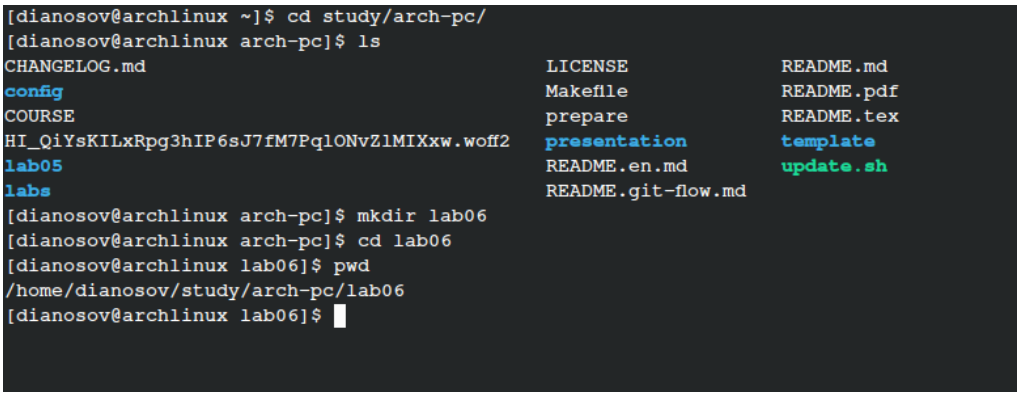

Рис. 3.1: Создание каталога для программ

Создадим файл *lab6-1.asm* и скопируем из каталога предыдущей лабораторной работы файл *in_out.asm* с полезными подпрограммами (рис. 3.2).

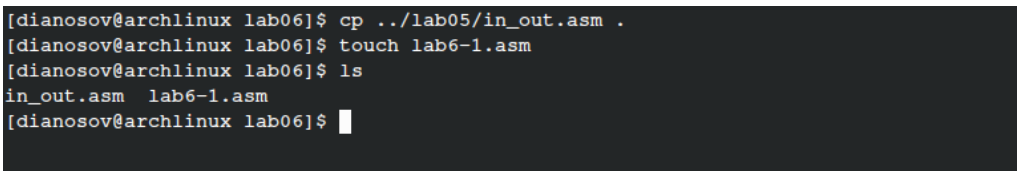

Рис. 3.2: Подготовка файлов

3.1 Вывод чисел в терминал в NASM

Скопируем код из предложенного листинга в файл *lab6-1.asm* (рис. 3.3).

В данной программе в регистр eax записывается символ 6 (`mov eax, '6'`), в регистр *ebx* символ 4 (`mov ebx,'4'`). Далее к значению в регистре *eax* прибавляем значение регистра *ebx* (`add eax,ebx`, результат сложения запишется в регистр eax). Далее выводим результат. Так как для работы функции *sprintLF* в регистр eax должен быть записан адрес, необходимо использовать дополнительную переменную. Для этого запишем значение регистра *eax* в переменную *buf1* (`mov [buf1], eax`), а затем запишем адрес переменной *buf1* в регистр *eax* (`mov eax, buf1`) и вызовем функцию *sprintLF*.

Рис. 3.3: Vim с файлом lab6-1.asm

Скомпилируем и запустим программу *lab6-1.asm* (рис. 3.4). В данном случае при выводе значения регистра *eax* мы ожидаем увидеть число 10. Однако результатом будет символ *j*. Это происходит потому, что код символа 6 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 своюочередь является кодом символа *j*.

```
[dianosov@archlinux lab06]$ nasm -f elf lab6-1.asm
[dianosov@archlinux lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[dianosov@archlinux lab06]$ ./lab6-1
j
[dianosov@archlinux lab06]$
```

Рис. 3.4: Компиляция и первый запуск программы

Изменим текст программы и вместо символов запишем в регистры числа. Исправим текст программы следующим образом: заменим строки

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

на строки

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

Снова скомпилируем программу и запустим её (рис. 3.5).

```
[dianosov@archlinux lab06]$ nasm -f elf lab6-1.asm
[dianosov@archlinux lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[dianosov@archlinux lab06]$ ./lab6-1


[dianosov@archlinux lab06]$
```

Рис. 3.5: Повторная компиляция и запуск программы

Как и в предыдущем случае при исполнении программы мы не получили число 10. В данном случае должен выводиться символ с кодом 10. Он не отображается в терминале.

В этом же каталоге создадим файл *lab6-1.asm*. В него вставим код из предложенного листинга, использующий команду *iprintLF* (рис. 3.6).

Рис. 3.6: Редактирование файла *lab6-2.asm*

Скомпилируем и запустим новую программу (рис. 3.7). 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 отличии от программы из листинга 6.1, функция *iprintLF* позволяет вывести число, а не символ, кодом которого является это число.

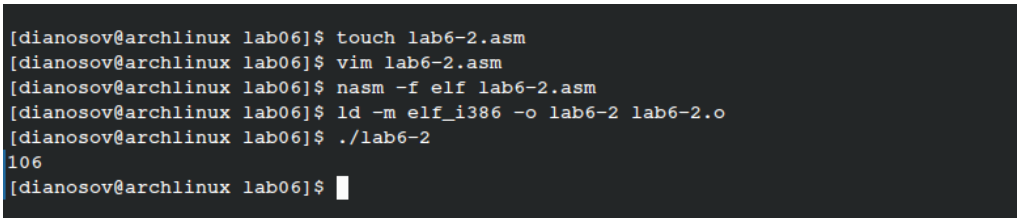

Рис. 3.7: Компиляция и запуск новой программы

Аналогично предыдущему примеру изменим символы на числа. Заменим строки

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

на строки

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

Рис. 3.8: Редактирование файла *lab6-2.asm*

Скомпилируем и запустим программу *lab6-2.asm* (рис. 3.9). Отметим, что команда *iprintLF* не просто выводит число, а ещё и переводит строку после него. Команда *iprint*, напротив, только выводит значение из *eax*.

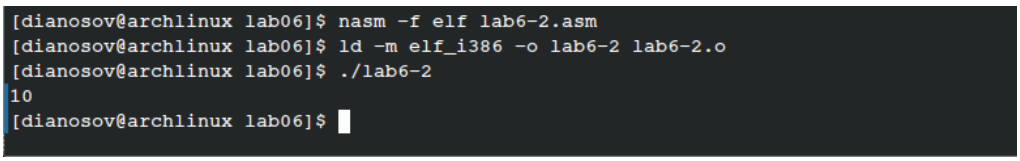

Рис. 3.9: Повторная компиляция и запуск новой программы

3.2 Арифметические операции в NASM

3.2.1 Вычисление значения выражения

В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения $f(x) = \frac{5*2+3}{3}$.

Создадим в рабочем каталоге файл *lab6-3.asm* и запишем в него код из предложенного листинга (рис. 3.10).

```
;-----------------------------
; 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. 3.10: Редактирование третьей программы

Скомпилируем и запустим программу *lab6-3.asm* (рис. 3.11).

```
[dianosov@archlinux lab06]$ nasm -f elf lab6-3.asm
[dianosov@archlinux lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[dianosov@archlinux lab06]$ ./lab6-3
Результат: 4
Остаток от деления: 1
[dianosov@archlinux lab06]$
```

Рис. 3.11: Компиляция и запуск третьей программы

Теперь сделаем так, чтобы программа вычисляла значение выражения $f(x) = \frac{4*6+2}{5}$. Откроем файл с кодом в редакторе **Vim** и отредактируем код (рис. 3.12).

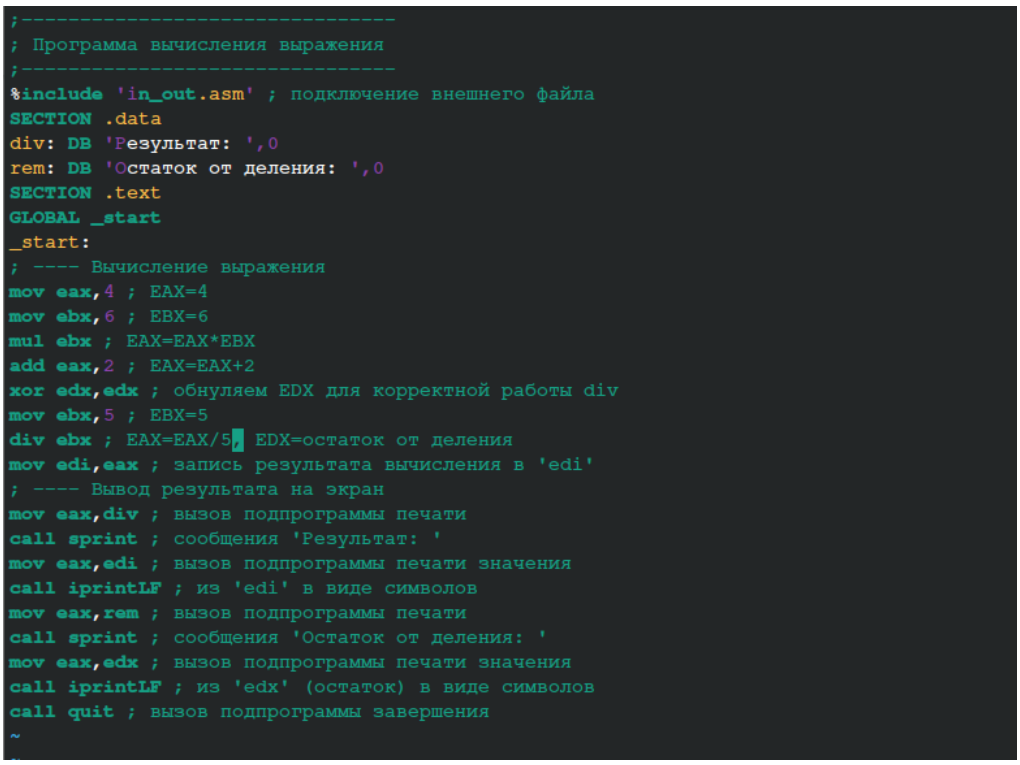

Рис. 3.12: Изменение кода третьей программы

Снова скомпилируем и запустим программу *lab6-3.asm* (рис. 3.13).

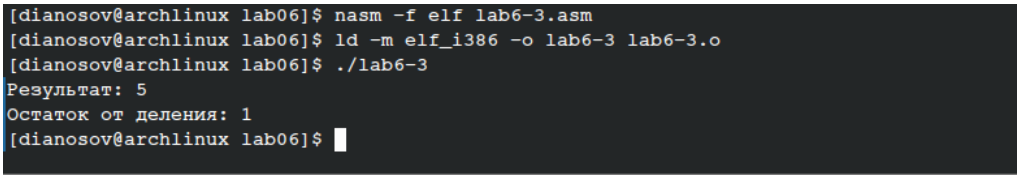

Рис. 3.13: Компиляция и запуск третьей программы после редактирования

Видим, что программа работает корректно. Действительно, $\frac{4*6+2}{5} = \frac{26}{5} = 5 + \frac{1}{5}$.

3.2.2 Программа вычисления номера варианта по номеру студенческого билета

В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму: - вывести запрос на введение № студенческого билета - вычислить

номер варианта по формуле: $(S_n \mod 20) + 1$, где S_n – номер студенческого билета. - вывести на экран номер варианта.

Создадим в рабочем каталоге файл *variant.asm* (рис. 3.14).

```
[dianosov@archlinux lab06]$ touch variant.asm
[dianosov@archlinux lab06]$ ls
in_out.asm  lab6-1.asm  lab6-2      lab6-2.o  lab6-3.asm  variant.asm
lab6-1      lab6-1.o    lab6-2.asm  lab6-3    lab6-3.o
[dianosov@archlinux lab06]$
```

Рис. 3.14: Создание файла *variant.asm*

В новую программу введём код из предложенного листинга. (рис. 3.15).

```
;-----------------------------
; 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 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. 3.15: Редактирование файла *variant.asm*

Проведем компиляцию нового файла и проверим его работу. (рис. 3.16).

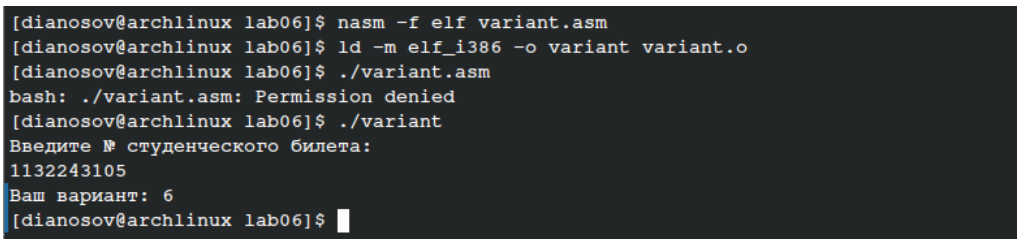

Рис. 3.16: Компиляция и запуск *variant.asm*

Программа верно вычисляет вариант. Действительно, $(1132243105 + 1)\%20 = 6$. Ответы на вопросы: 1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’? За это отвечают строки:

```
mov eax, msg
call sprintLF
```

2. Для чего используется следующие инструкции?

```
mov ecx, x
mov edx, 80
call sread
```

Они отвечают за считывание ввода из терминала. Первая строчка говорит, что считанное значение будет помещено в x. Вторая строчка задает максимальный объём данных для считывания. Третья - вызывает функцию *sread* считывания строки из терминала.

3. Для чего используется инструкция “`call atoi`”?

Для преобразования значения в регистре eax в число.

4. Какие строки листинга 6.4 отвечают за вычисления варианта?

```
mov ebx,20
div ebx
inc edx
```

5. В какой регистр записывается остаток от деления при выполнении инструкции “`div ebx`”?

Остаток от деления помещается в регистр `edx`.

6. Для чего используется инструкция “`inc edx`”?

Эта инструкция используется для увеличения значения в регистре `edx` на 1.

7. Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений?

```
mov eax,rem
call sprint
mov eax,edx
call iprintLF
```

4 Задание для самостоятельной работы

Полученный номер варианта - 6, как указано ранее. Выражение: $f(x) = \frac{x^3}{2} + 1$

Скопируем файл `lab6-3.asm` и дадим новому файлу название `task.asm`. (рис. 4.1). В новом файле сделаем нужные изменения (код прокомментирован).

```
;-----------------------------
; Программа вычисления выражения
;-----------------------------
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg: DB 'Введите значение x',0
div: DB 'Результат: ',0
dot: DB '.',0
SECTION .bss
x: RESB 80    ; резервируем память под значение x
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprintLF ; вывод приглашения к вводу
mov ecx, x
mov edx, 80
call sread ; считывание значения x
mov eax, x ; eax = x
call atoi ; преобразуем eax в число
mov edi, eax
mul edi   ; умножаем eax на edi=x, eax = x^2
mul edi   ; eax = x^3
mov ebx,2 ; ebx = 2
xor edx, edx ; обнуляем edx
div ebx   ; делим eax на ebx, eax = eax//2
; остаток от деления помещается в edx
inc eax ; eax = x^3/2 + 1
mov edi, eax ; результат кладём в edi
mov eax, edx ; остаток от деления перемещаем в eax
mov ecx,10
mul ecx  ; умножаем eax на 10
div ebx ; делим eax на 2
mov edx, eax
; теперь в edx имеем дробную (десятичную) форму остатка
; в edi целая часть частного
; выводить будем [целая часть].[дробная часть]
mov eax,div ; вызов подпрограммы печати
-- INSERT --                                                    21,13
```

Рис. 4.1: Редактирование файла *task.asm*

Проверим коррекность работы программмы для $x_1 = 2, x_2 = 5$ (рис. **??**).

```
[dianosov@archlinux lab06]$ nasm -f elf task.asm
[dianosov@archlinux lab06]$ ld -m elf_i386 -o task task.o
[dianosov@archlinux lab06]$ ./task
Введите значение x
2
Результат: 5.0
[dianosov@archlinux lab06]$ ./task
Введите значение x
5
Результат: 63.5
```

Программа работает правильно.

Ниже приведен весь код новой программы с подробными комментариями.

```
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg: DB 'Введите значение x',0
div: DB 'Результат: ',0
dot: DB '.',0
SECTION .bss
x: RESB 80    ; резервируем память под значение x
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprintLF ; вывод приглашения к вводу
mov ecx, x
mov edx, 80
call sread ; считывание значения x
mov eax, x ; eax = x
call atoi ; преобразуем eax в число
mov edi, eax
mul edi   ; умножаем eax на edi=x, eax = x^2
mul edi   ; eax = x^3
mov ebx,2 ; ebx = 2
```

```
xor edx, edx ; обнуляем edx
div ebx   ; делим eax на ebx, eax = eax//2
; остаток от деления помещается в edx
inc eax ; eax = x^3/2 + 1
mov edi, eax ; результат кладём в edi
mov eax, edx ; остаток от деления перемещаем в eax
mov ecx,10
mul ecx  ; умножаем eax на 10
div ebx ; делим eax на 2
mov edx, eax
; теперь в edx имеем дробную (десятичную) форму остатка
; в edi целая часть частного
; выводить будем [целая часть].[дробная часть]
mov eax,div ; вызов подпрограммы печати
call sprint ; сообщения 'Результат: '
mov eax,edi ; вызов подпрограммы печати значения
call iprint ; из 'edi' в виде символов
mov eax,dot ; точка
call sprint ; печать точки
mov eax,edx ; вызов подпрограммы печати значения
call iprintLF ; из 'edx' (остаток) в виде символов
call quit ; вызов подпрограммы завершения
```

Задание выполнено, загрузим новую версию проекта курса на GitHub.

```
[dianosov@archlinux arch-pc]$ git add .
[dianosov@archlinux arch-pc]$ git commit -am "add files  for lab06"
On branch master
nothing to commit, working tree clean
[dianosov@archlinux arch-pc]$ git push origin master
Enumerating objects: 82, done.
Counting objects: 100% (82/82), done.
Delta compression using up to 4 threads
Compressing objects: 100% (74/74), done.
Writing objects: 100% (80/80), 2.20 MiB | 3.22 MiB/s, done.
Total 80 (delta 20), reused 23 (delta 3), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (20/20), completed with 2 local objects.
To https://github.com/exterminateddd/pc-course-2024-2025
   c5f4f3f..a055603  master -> master
[dianosov@archlinux arch-pc]$
```

5 Выводы

В ходе выполнения лабораторной работы были освоены арифметические инструкции языка ассемблера NASM.