

Содержание

1	ОТЧЕТ	4
1.1	по лабораторной работе №6	4
1.2	«Арифметические операции в NASM»	4
1.3	1. Цель работы	4
1.4	2. Выполнение лабораторной работы	4
1.5	3. Задание для самостоятельной работы	43
1.6	4. Ответы на контрольные вопросы	54
1.7	5. Выводы	62
1.8	6. Список файлов работы	64

Список иллюстраций

Список таблиц

1 ОТЧЕТ

1.1 по лабораторной работе №6

1.2 «Арифметические операции в NASM»

Выполнил: Нхари Хатим

Группа: НБИбд-03-25

Дата: 18.01.2026

1.3 1. Цель работы

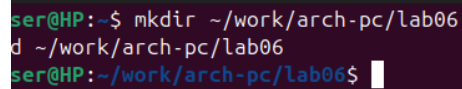
Освоение арифметических инструкций языка ассемблера NASM.

1.4 2. Выполнение лабораторной работы

1.4.1 2.1. Создание каталога для лабораторной работы №6

Команды:

```
mkdir ~/work/arch-pc/lab06  
cd ~/work/arch-pc/lab06
```



```
ser@HP:~$ mkdir ~/work/arch-pc/lab06  
d ~/work/arch-pc/lab06  
ser@HP:~/work/arch-pc/lab06$
```

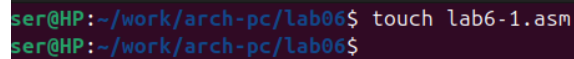
Скриншот 1:

Комментарий: Создали рабочий каталог для шестой лабораторной работы и перешли в него.

1.4.2 2.2. Создание файла lab6-1.asm

Команда:

```
touch lab6-1.asm
```



```
ser@HP:~/work/arch-pc/lab06$ touch lab6-1.asm  
ser@HP:~/work/arch-pc/lab06$
```

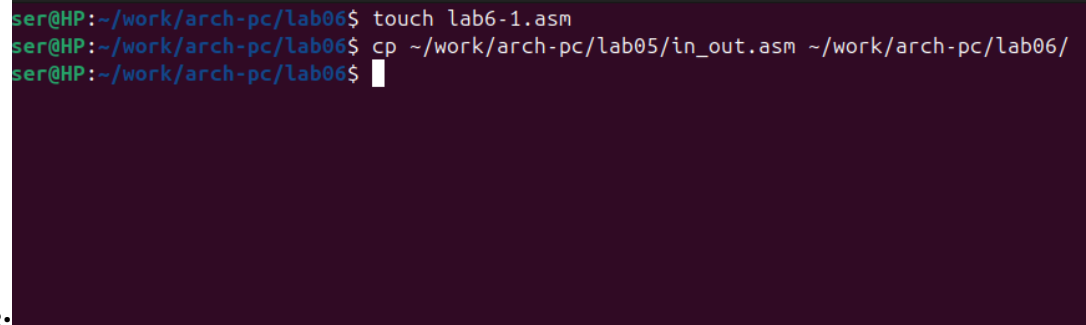
Скриншот 2:

Комментарий: Создали пустой файл для первой программы.

1.4.3 2.3. Копирование файла in_out.asm

Команда:

```
cp ~/work/arch-pc/lab05/in_out.asm ~/work/arch-pc/lab06/
```



```
ser@HP:~/work/arch-pc/lab06$ touch lab6-1.asm
ser@HP:~/work/arch-pc/lab06$ cp ~/work/arch-pc/lab05/in_out.asm ~/work/arch-pc/lab06/
ser@HP:~/work/arch-pc/lab06$
```

Скриншот 3:

Комментарий: Скопировали файл in_out.asm из каталога lab05 в текущий каталог lab06, так как он необходим для работы программ.

1.4.4 2.4. Редактирование файла lab6-1.asm

Действия: - Открыл файл lab6-1.asm в текстовом редакторе - Ввел текст программы из листинга 6.1

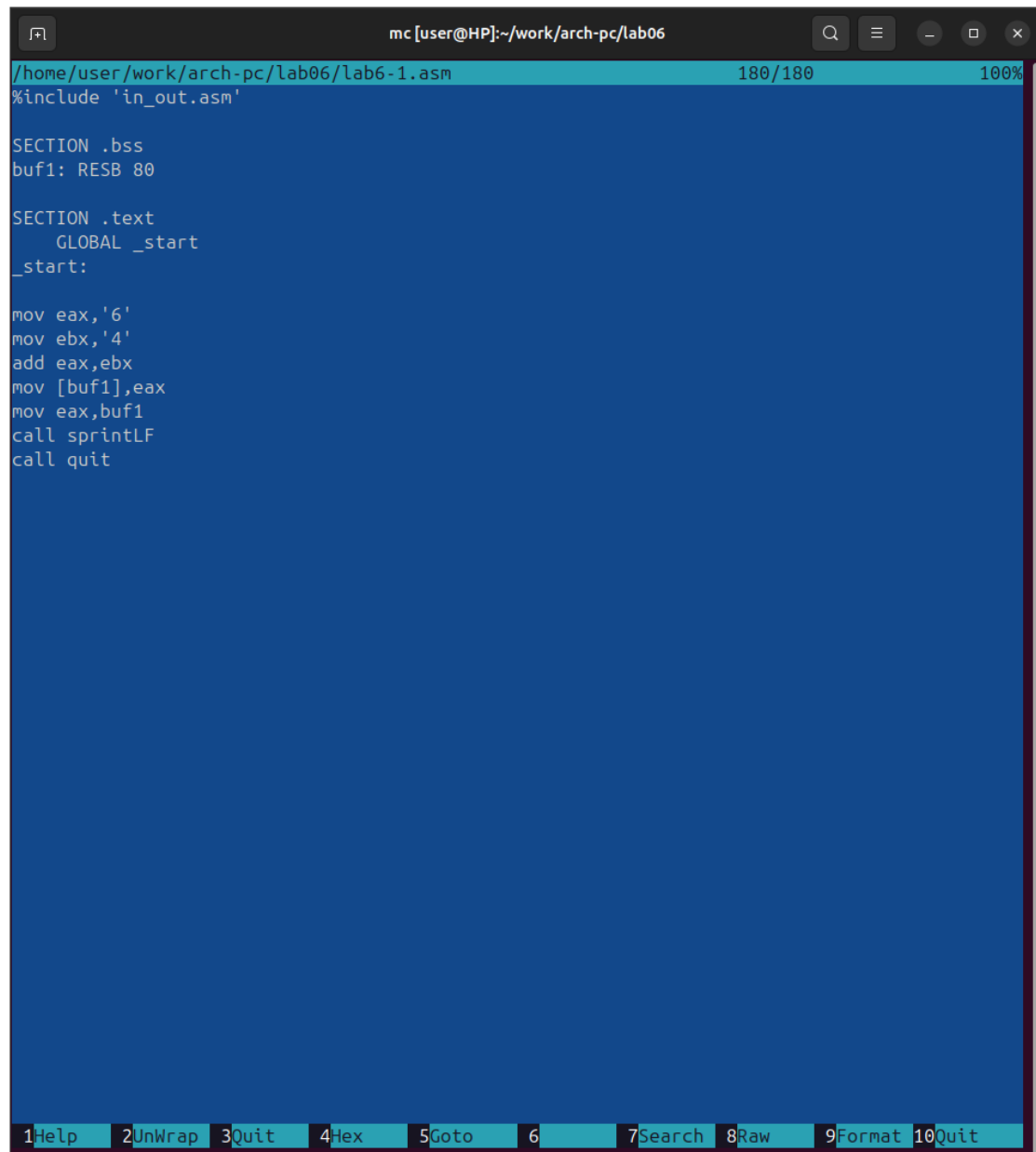
Листинг 6.1. Программа вывода значения регистра eax (символы):

```
%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
```



```
mc [user@HP]:~/work/arch-pc/lab06
/home/user/work/arch-pc/lab06/lab6-1.asm 180/180 100%
#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 sprintLF
call quit

1Help 2UnWrap 3Quit 4Hex 5Goto 6 7Search 8Raw 9Format 10Quit
```

Скриншот 4:

Комментарий: В этой программе в регистр `eax` записывается символ „6“, в регистр `ebx` - символ „4“. Затем происходит сложение значений этих регистров.

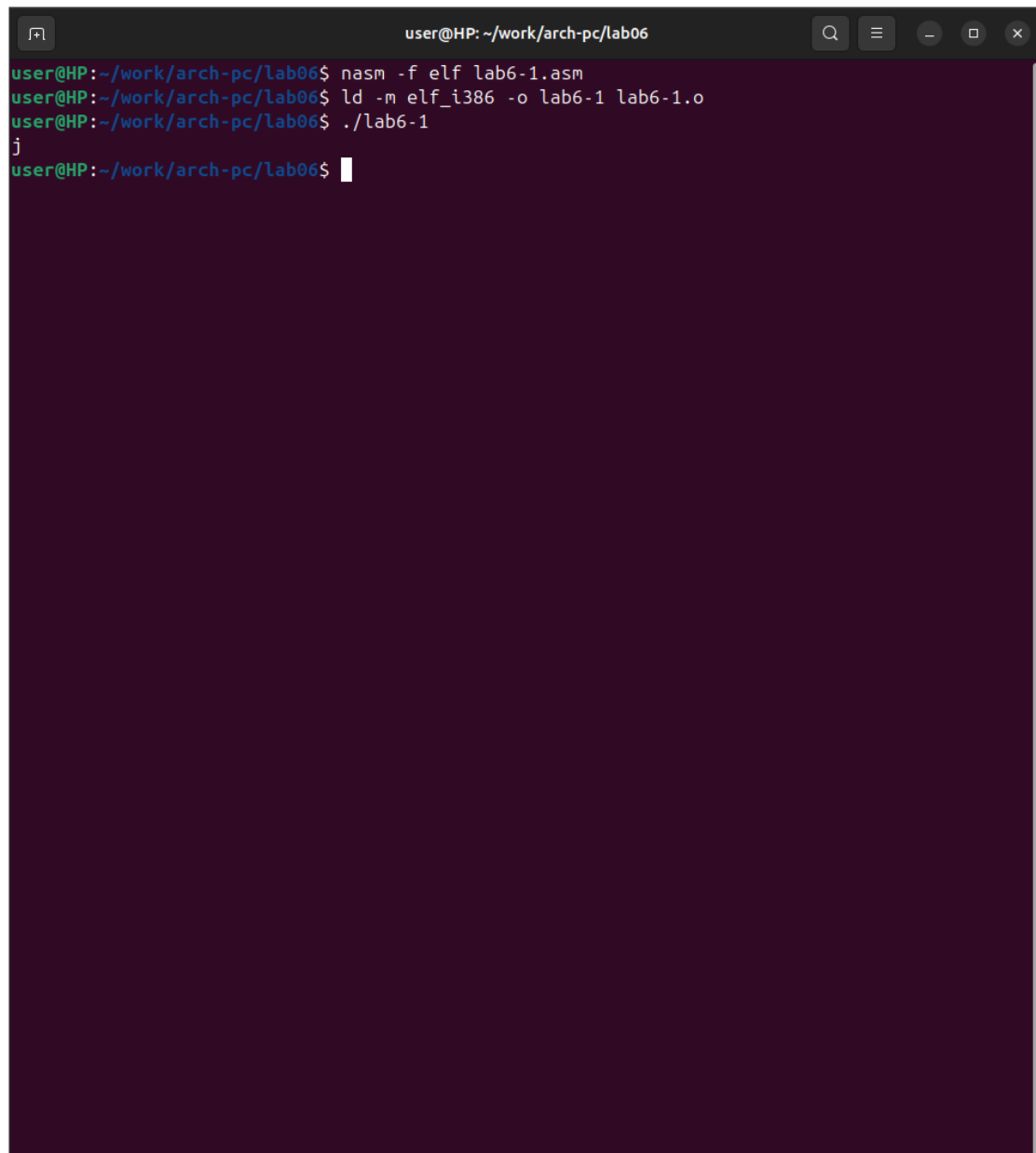
1.4.5 2.5. Компиляция и запуск программы lab6-1.asm

Команды:

```
nasm -f elf lab6-1.asm  
ld -m elf_i386 -o lab6-1 lab6-1.o  
./lab6-1
```

Результат:

j

A screenshot of a terminal window with a dark background. The window title is "user@HP: ~/work/arch-pc/lab06". The terminal shows the following commands and their outputs:

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

Скриншот 5:

Комментарий: Программа вывела символ „j“ вместо ожидаемого числа 10. Это происходит потому, что: - Код символа „6“ в ASCII = 54 (в двоичном: 00110110) - Код символа „4“ в ASCII = 52 (в двоичном: 00110100) - Сумма кодов: $54 + 52 = 106$ (в двоичном: 01101010) - Код 106 соответствует символу „j“ в таблице ASCII

1.4.6 2.6. Изменение программы lab6-1.asm (числа вместо символов)

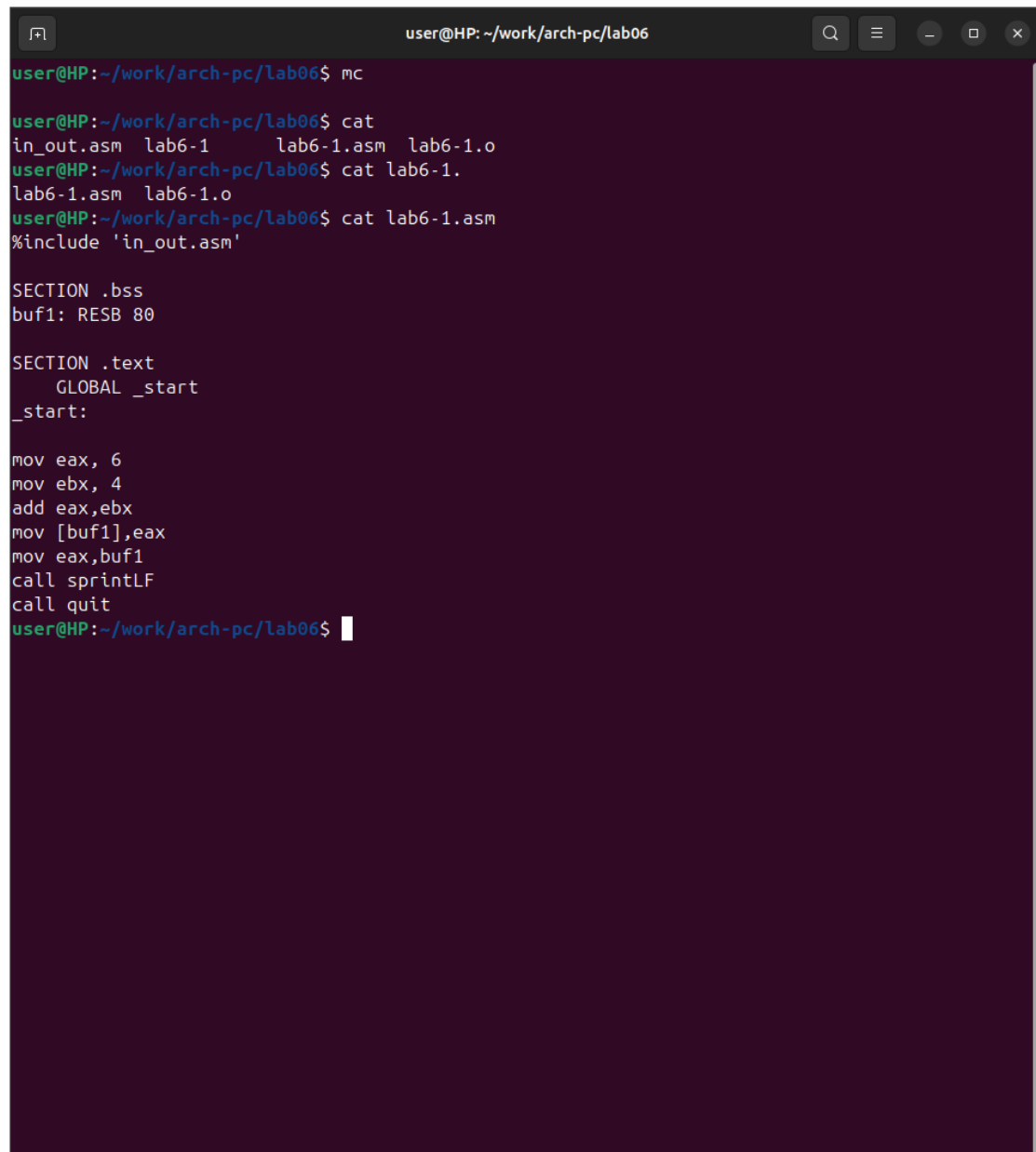
Действия: - Открыл файл lab6-1.asm для редактирования - Заменял строки:

Было:

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

Стало:

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

A terminal window with a dark background and light green text. The window title is 'user@HP: ~/work/arch-pc/lab06'. The terminal shows a series of commands and their outputs. The user runs 'mc', then 'cat' to view the contents of 'lab6-1.o', 'lab6-1.asm', and 'lab6-1.o'. The assembly code is displayed, including a section for '.bss' with a buffer 'buf1' of size 80, and a section for '.text' with a global '_start' function. The function contains assembly instructions: 'mov eax, 6', 'mov ebx, 4', 'add eax, ebx', 'mov [buf1], eax', 'mov eax, buf1', 'call sprintf', and 'call quit'. The terminal ends with a prompt 'user@HP: ~/work/arch-pc/lab06\$' and a cursor.

```
user@HP:~/work/arch-pc/lab06$ mc
user@HP:~/work/arch-pc/lab06$ cat
in_out.asm lab6-1 lab6-1.asm lab6-1.o
user@HP:~/work/arch-pc/lab06$ cat lab6-1.
lab6-1.asm lab6-1.o
user@HP:~/work/arch-pc/lab06$ cat 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 sprintf
call quit
user@HP:~/work/arch-pc/lab06$
```

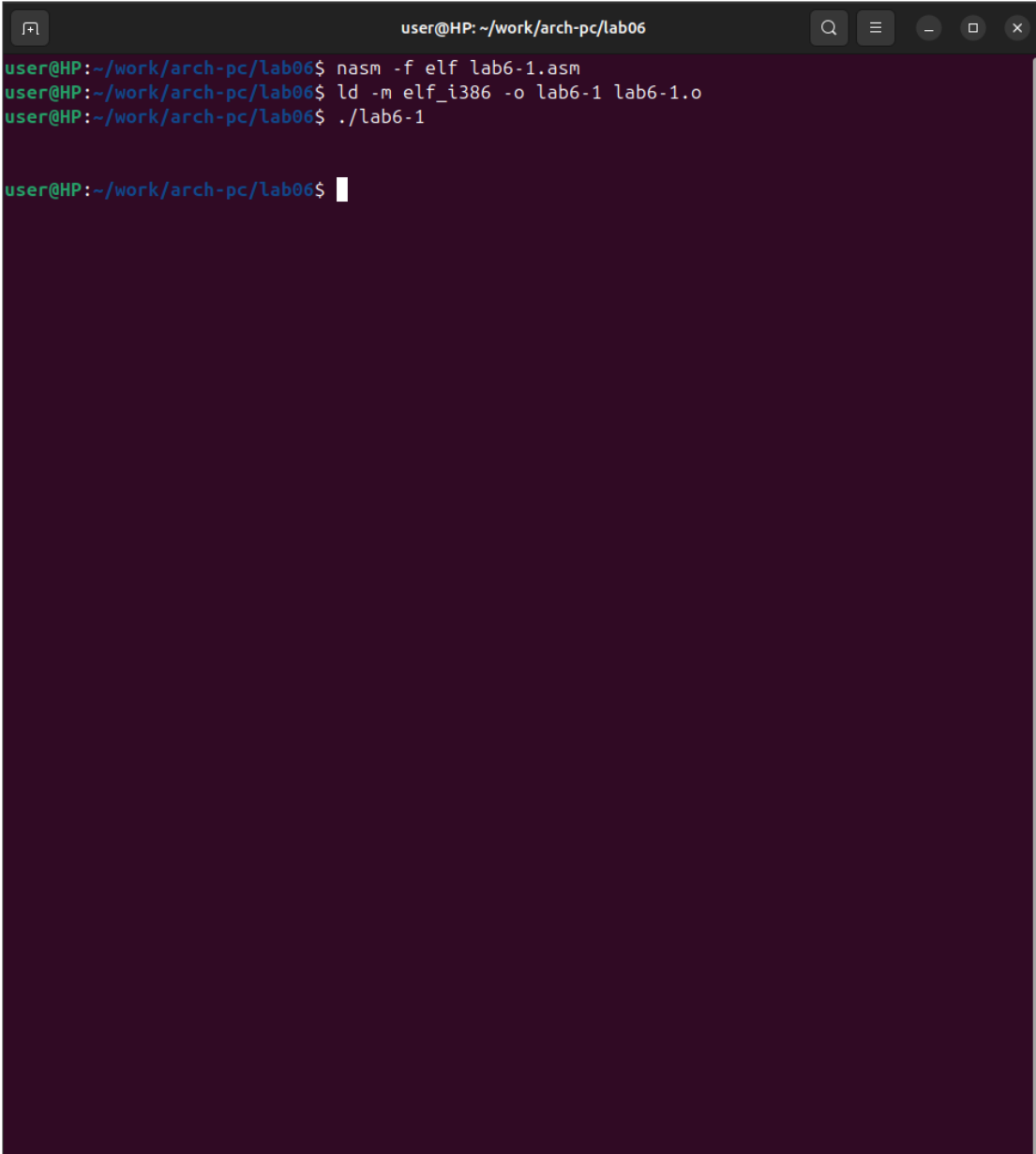
Скриншот 6:

1.4.7 2.7. Компиляция и запуск измененной программы

Команды:

```
nasm -f elf lab6-1.asm
ld -m elf_i386 -o lab6-1 lab6-1.o
./lab6-1
```

Результат:

A screenshot of a terminal window with a dark background. The window title is "user@HP: ~/work/arch-pc/lab06". The terminal shows three lines of commands and their execution: "nasm -f elf lab6-1.asm", "ld -m elf_i386 -o lab6-1 lab6-1.o", and "./lab6-1". Each line is preceded by the prompt "user@HP:~/work/arch-pc/lab06\$". The output of the third command is not visible. A cursor is shown on the line following the last command.

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

user@HP:~/work/arch-pc/lab06$
```

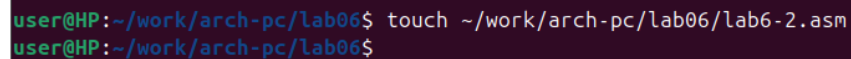
Скриншот 7:

Комментарий: На экран выводится символ с кодом 10. Согласно таблице ASCII, код 10 соответствует символу LF (Line Feed - перевод строки), который является управляющим символом и не отображается визуально на экране, но выполняет перевод на новую строку.

1.4.8 2.8. Создание файла lab6-2.asm

Команда:

```
touch ~/work/arch-pc/lab06/lab6-2.asm
```



```
user@HP:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-2.asm
user@HP:~/work/arch-pc/lab06$
```

Скриншот 8:

1.4.9 2.9. Редактирование файла lab6-2.asm

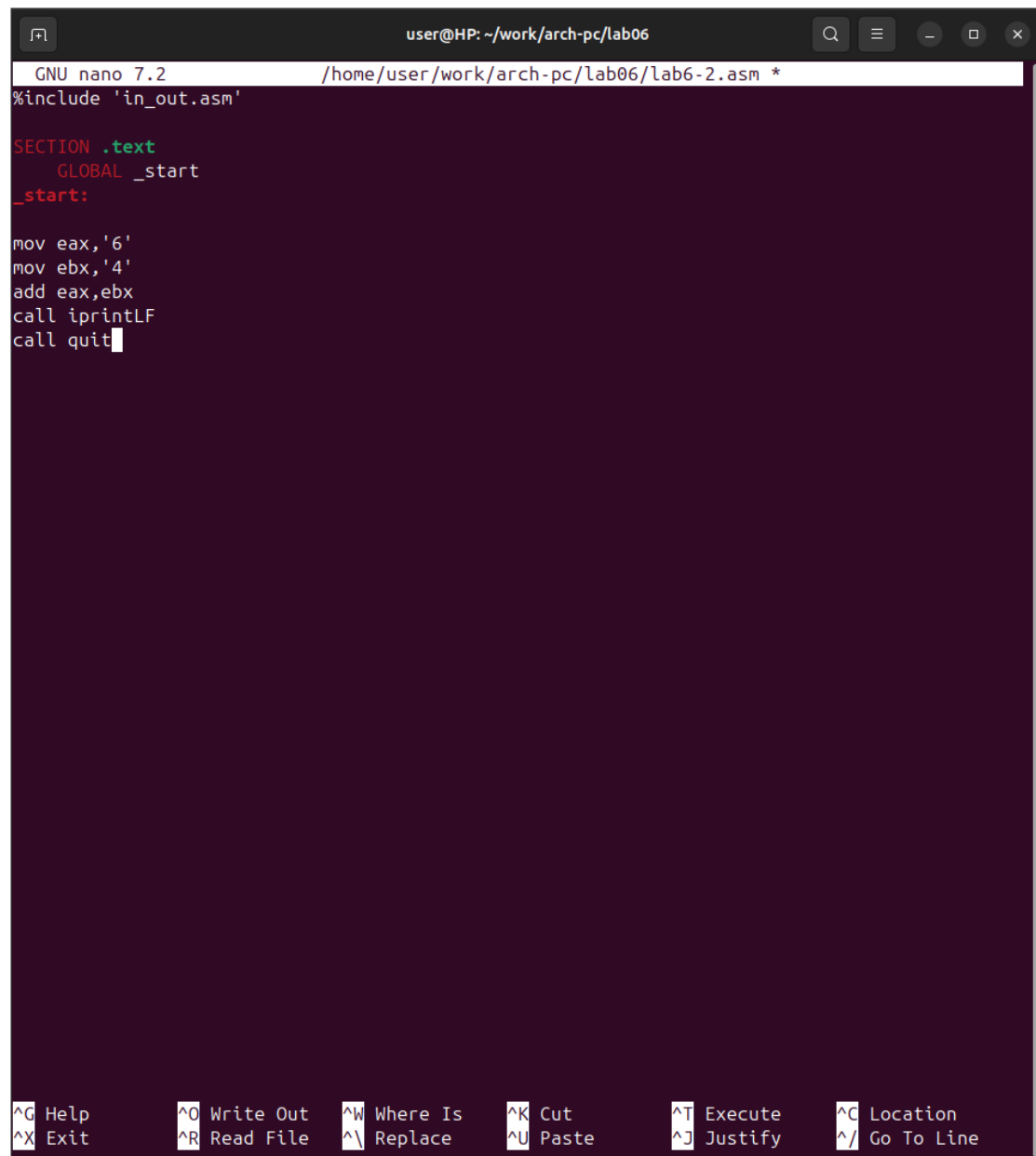
Листинг 6.2. Программа вывода значения регистра `eax` с использованием `iprintLF`:

```
%include 'in_out.asm'

SECTION .text
    GLOBAL _start
_start:

mov eax, '6'
```

```
mov ebx, '4'
add eax, ebx
call iprintLF
call quit
```



```
user@HP: ~/work/arch-pc/lab06
GNU nano 7.2 /home/user/work/arch-pc/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
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^_ Go To Line

Скриншот 9:

Комментарий: В этой программе используется функция `iprintLF` для вывода числового значения регистра `eax`.

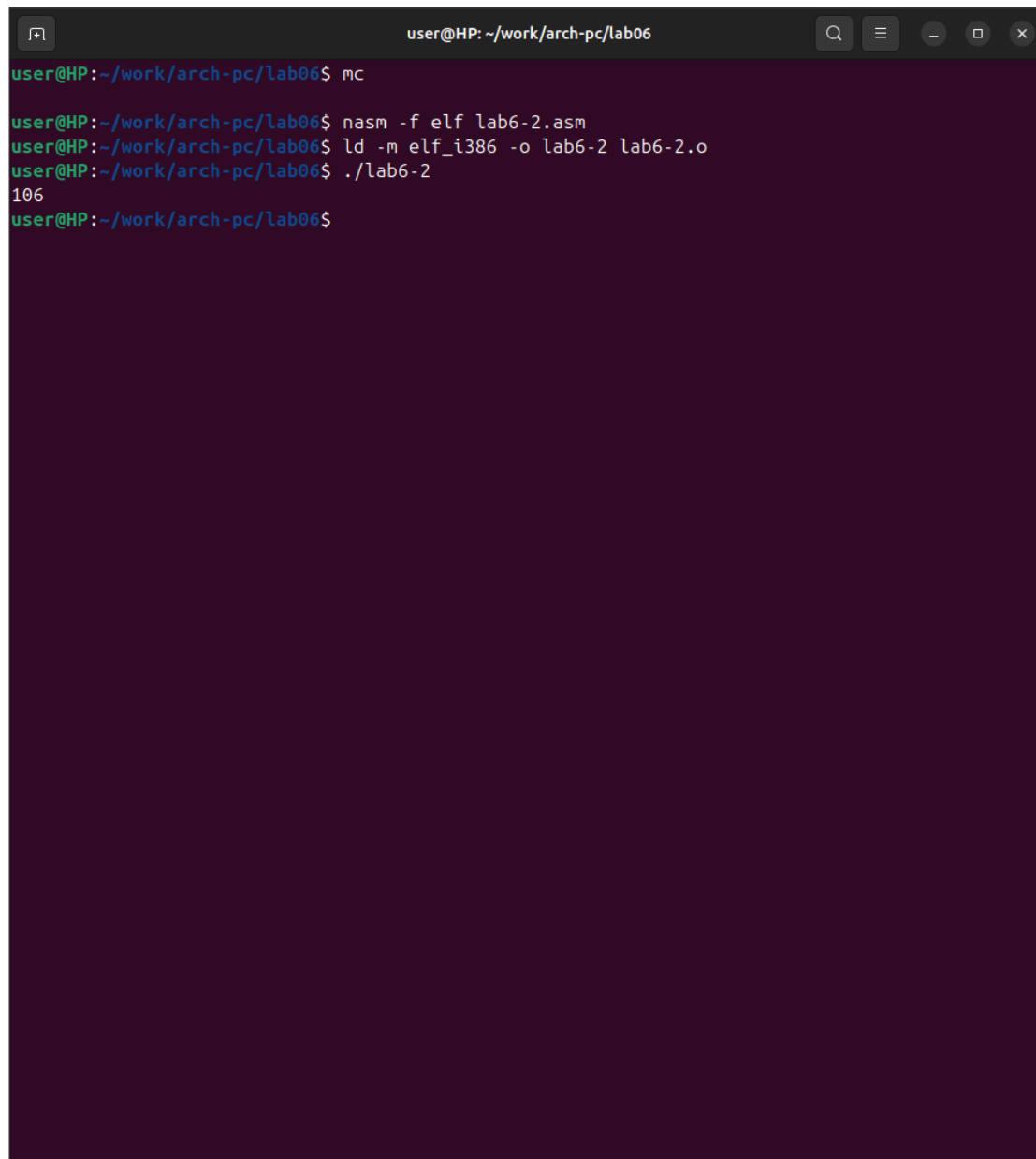
1.4.10 2.10. Компиляция и запуск программы lab6-2.asm

Команды:

```
nasm -f elf lab6-2.asm  
ld -m elf_i386 -o lab6-2 lab6-2.o  
./lab6-2
```

Результат:

106



```
user@HP: ~/work/arch-pc/lab06$ mc
user@HP:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
user@HP:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
user@HP:~/work/arch-pc/lab06$ ./lab6-2
106
user@HP:~/work/arch-pc/lab06$
```

Скриншот 10:

Комментарий: Программа вывела число 106, которое является суммой ASCII-кодов символов „6“ (54) и „4“ (52). Функция `iprintLF` выводит число, а не символ с соответствующим кодом.

1.4.11 2.11. Замена символов на числа в lab6-2.asm

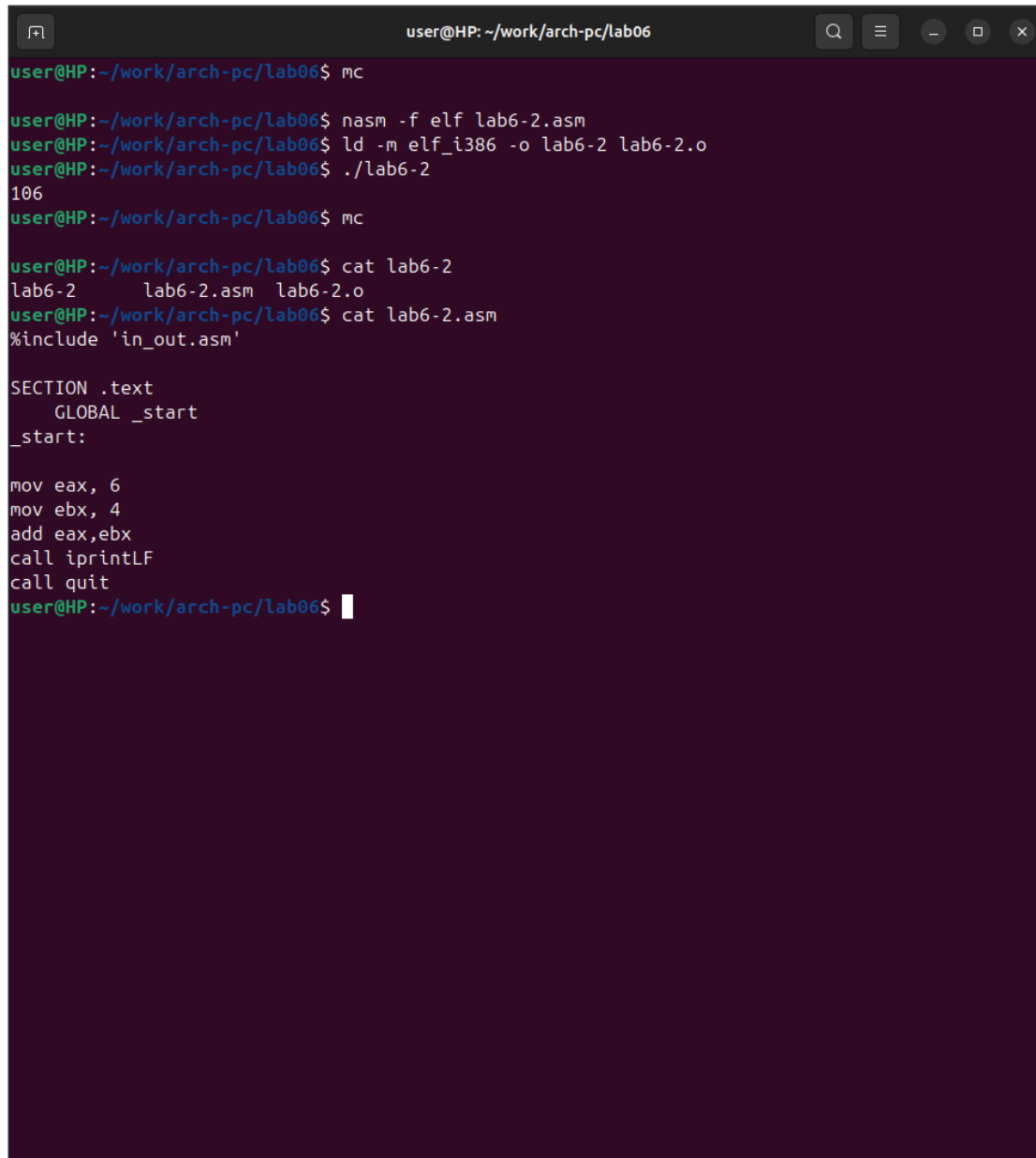
Действия: - Открыл файл lab6-2.asm для редактирования - Заменял строки:

Было:

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

Стало:

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

A terminal window with a dark purple background and light green text. The window title is 'user@HP: ~/work/arch-pc/lab06'. The terminal shows the following commands and output:

```
user@HP:~/work/arch-pc/lab06$ mc
user@HP:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
user@HP:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
user@HP:~/work/arch-pc/lab06$ ./lab6-2
106
user@HP:~/work/arch-pc/lab06$ mc
user@HP:~/work/arch-pc/lab06$ cat lab6-2
lab6-2      lab6-2.asm  lab6-2.o
user@HP:~/work/arch-pc/lab06$ cat 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
user@HP:~/work/arch-pc/lab06$
```

Скриншот 11:

1.4.12 2.12. Компиляция и запуск измененной программы

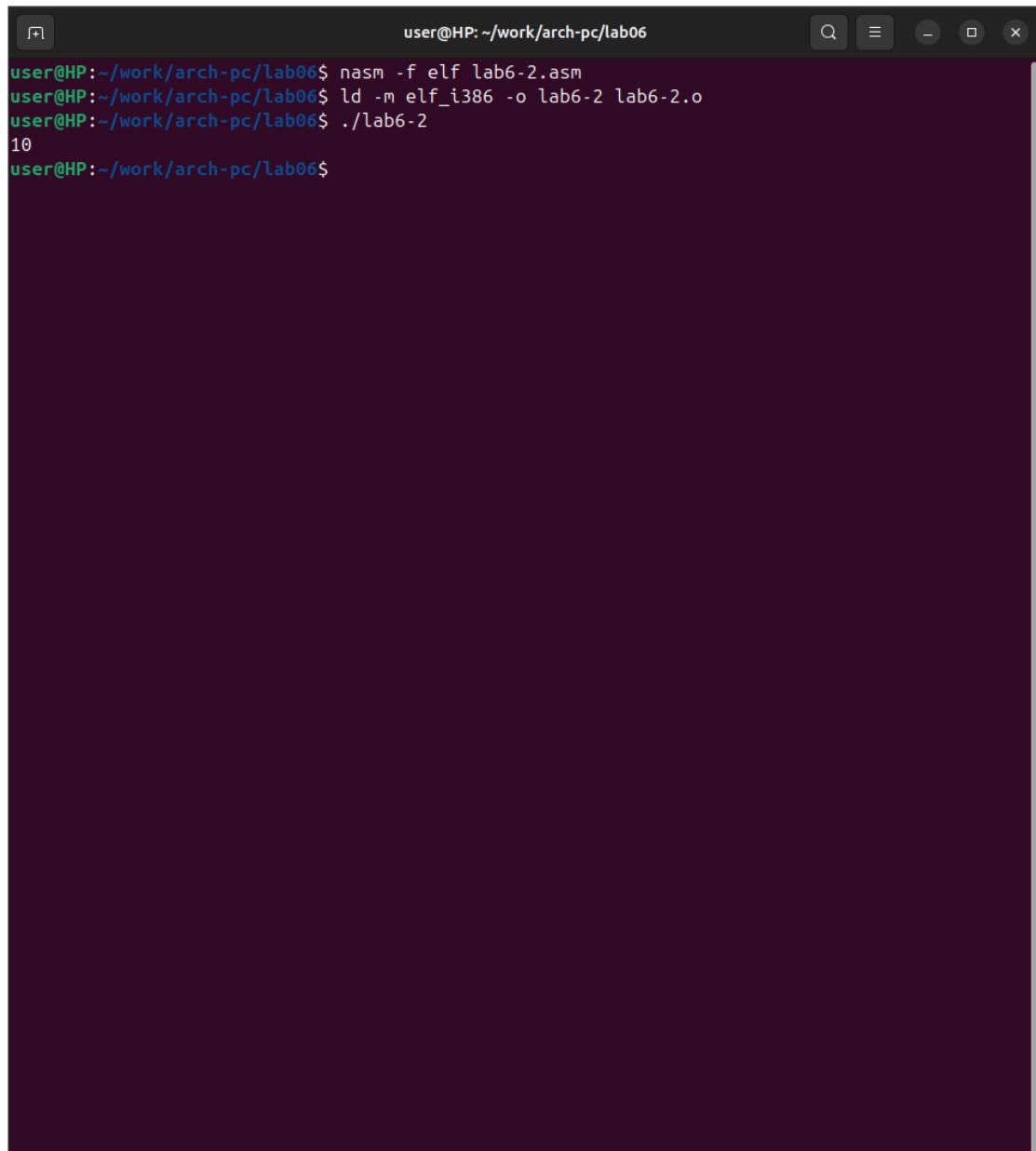
lab6-2.asm

Команды:

```
nasm -f elf lab6-2.asm
ld -m elf_i386 -o lab6-2 lab6-2.o
./lab6-2
```

Результат:

10

A screenshot of a terminal window with a dark purple background. The window title is "user@HP: ~/work/arch-pc/lab06". The terminal shows the following commands and output:

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

Скриншот 12:

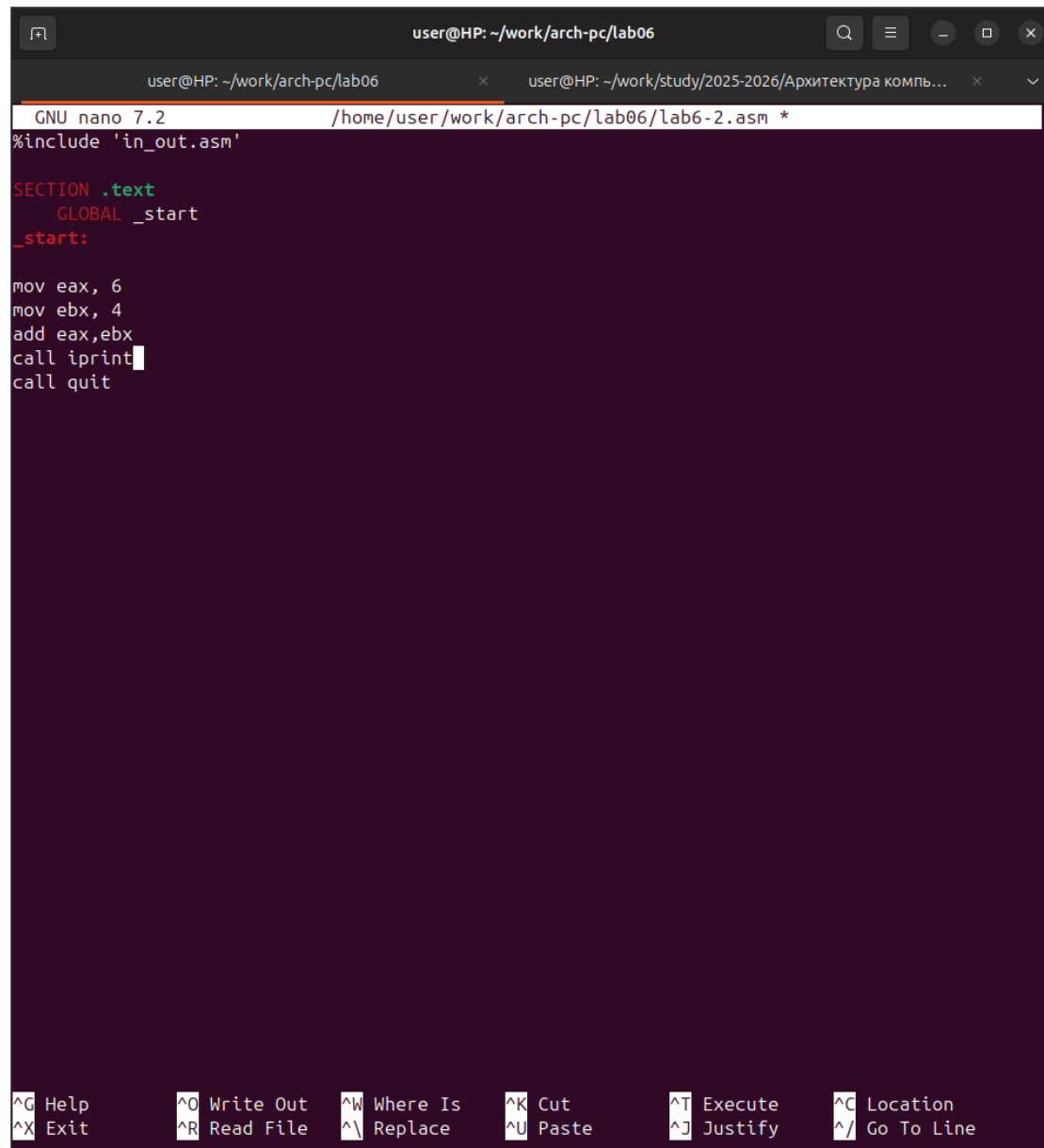
Комментарий: Теперь программа корректно выводит число 10, которое является результатом арифметического сложения чисел 6 и 4.

1.4.13 2.13. Замена `iprintLF` на `iprint`

Действия: - Открыл файл `lab6-2.asm` для редактирования - Заменяю строку `call iprintLF` на `call iprint`

Измененная строка:

```
call iprint
```



```
GNU nano 7.2 /home/user/work/arch-pc/lab06/lab6-2.asm *
#include 'in_out.asm'

SECTION .text
    GLOBAL _start
_start:

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

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^_ Go To Line

Скриншот 13:

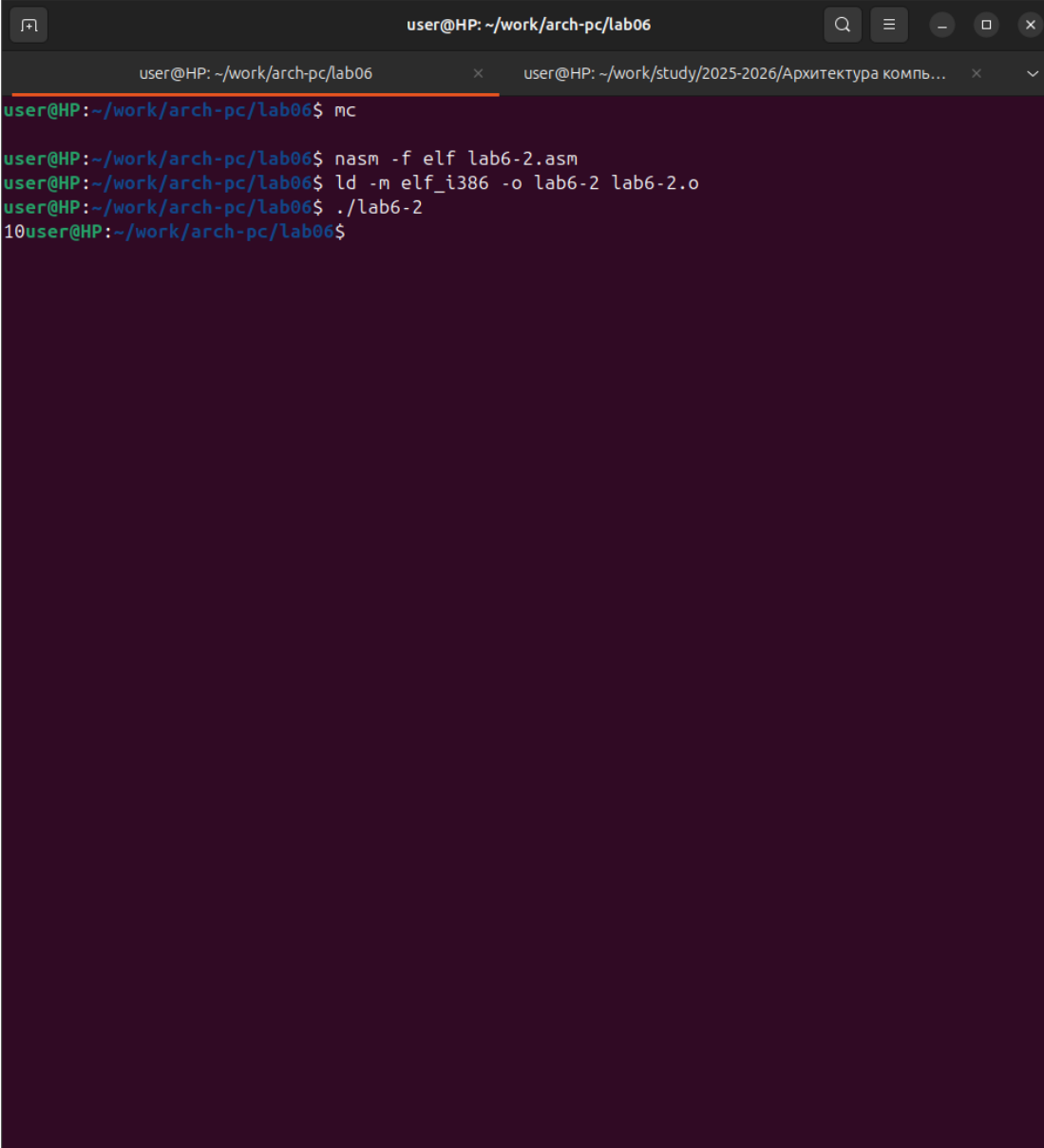
1.4.14 2.14. Компиляция и запуск с функцией iprint

Команды:

```
nasm -f elf lab6-2.asm
ld -m elf_i386 -o lab6-2 lab6-2.o
./lab6-2
```

Результат:

10user@HP:~/work/arch-pc/lab06\$

A screenshot of a terminal window with a dark background. The window title is "user@HP: ~/work/arch-pc/lab06". There are two tabs: "user@HP: ~/work/arch-pc/lab06" (active) and "user@HP: ~/work/study/2025-2026/Архитектура компь...". The terminal shows the following commands and output:

```
user@HP:~/work/arch-pc/lab06$ mc
user@HP:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
user@HP:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
user@HP:~/work/arch-pc/lab06$ ./lab6-2
10user@HP:~/work/arch-pc/lab06$
```

Скриншот 14:

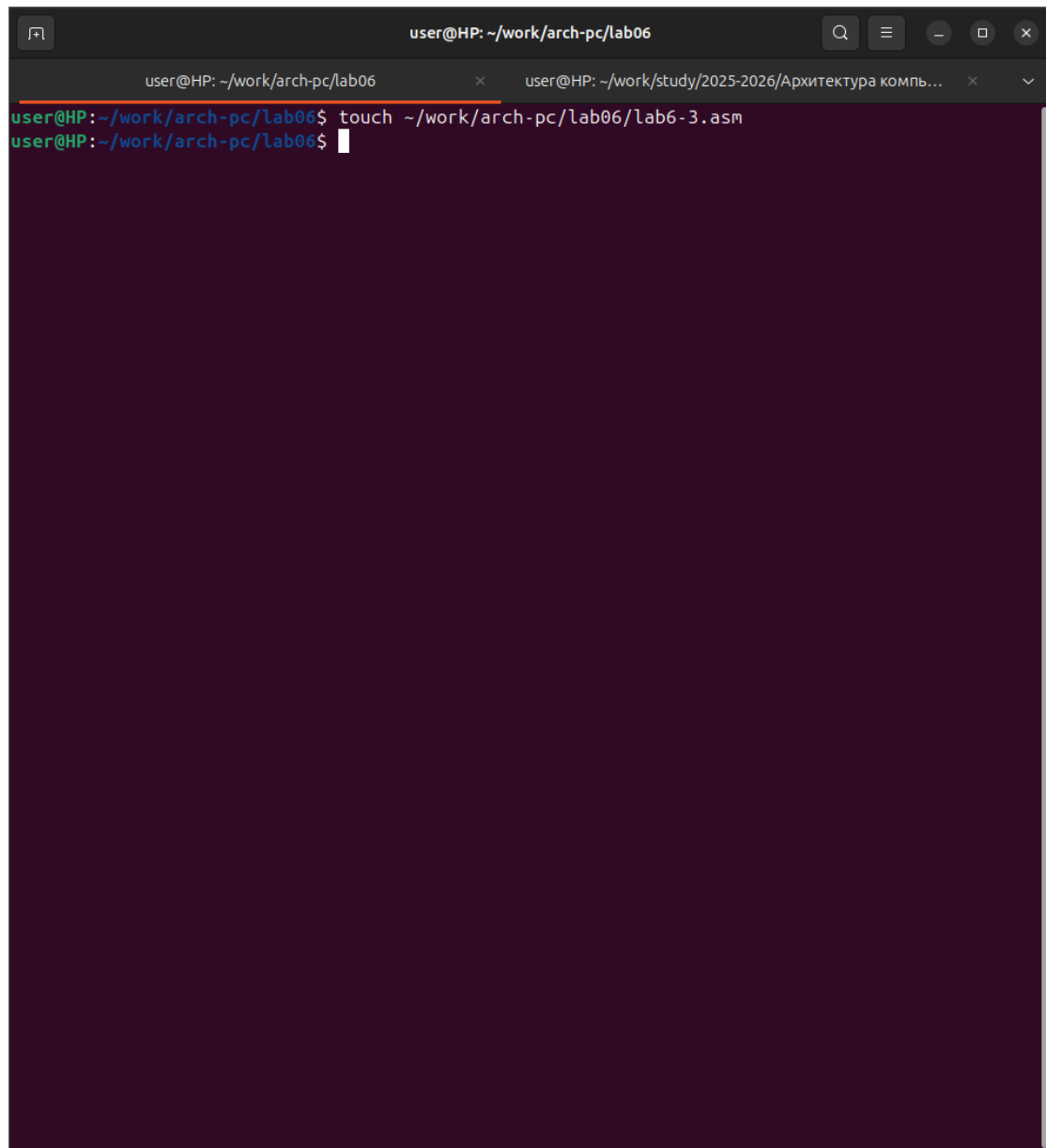
Комментарий:

Разница между `iprintLF` и `iprint`: - `iprintLF` - выводит число и добавляет символ перевода строки (курсор переходит на новую строку) - `iprint` - выводит только число без перевода строки (курсор остается на той же строке)

При использовании `iprint` приглашение командной строки появляется сразу после числа на той же строке.

1.4.15 2.15. Создание файла lab6-3.asm**Команда:**

```
touch ~/work/arch-pc/lab06/lab6-3.asm
```

A terminal window with a dark background and light green text. The window title is 'user@HP: ~/work/arch-pc/lab06'. The terminal shows two lines of input: 'user@HP:~/work/arch-pc/lab06\$ touch ~/work/arch-pc/lab06/lab6-3.asm' followed by a new prompt 'user@HP:~/work/arch-pc/lab06\$' with a cursor. The terminal is open within a browser window, with another tab visible in the background titled 'user@HP: ~/work/study/2025-2026/Архитектура компь...'.

```
user@HP: ~/work/arch-pc/lab06
user@HP:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-3.asm
user@HP:~/work/arch-pc/lab06$
```

Скриншот 15:

1.4.16 2.16. Редактирование файла lab6-3.asm

Листинг 6.3. Программа вычисления выражения $f(x) = (5 * 2 + 3) / 3$:

```

;-----
; Программа вычисления выражения
;-----

%include 'in_out.asm'

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
```

```
user@HP: ~/work/arch-pc/lab06
GNU nano 7.2 /home/user/work/arch-pc/lab06/lab6-3.asm *
;-----
; Программа вычисления выражения
;-----

%include 'in_out.asm'

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

^G Help      ^O Write Out ^W Where Is   ^K Cut        ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace    ^U Paste      ^J Justify   ^_ Go To Line
```

Скриншот 16:

Комментарий: Программа вычисляет выражение $(5 * 2 + 3) / 3 = 13 / 3 = 4$ (остаток 1).

1.4.17 2.17. Компиляция и запуск программы lab6-3.asm

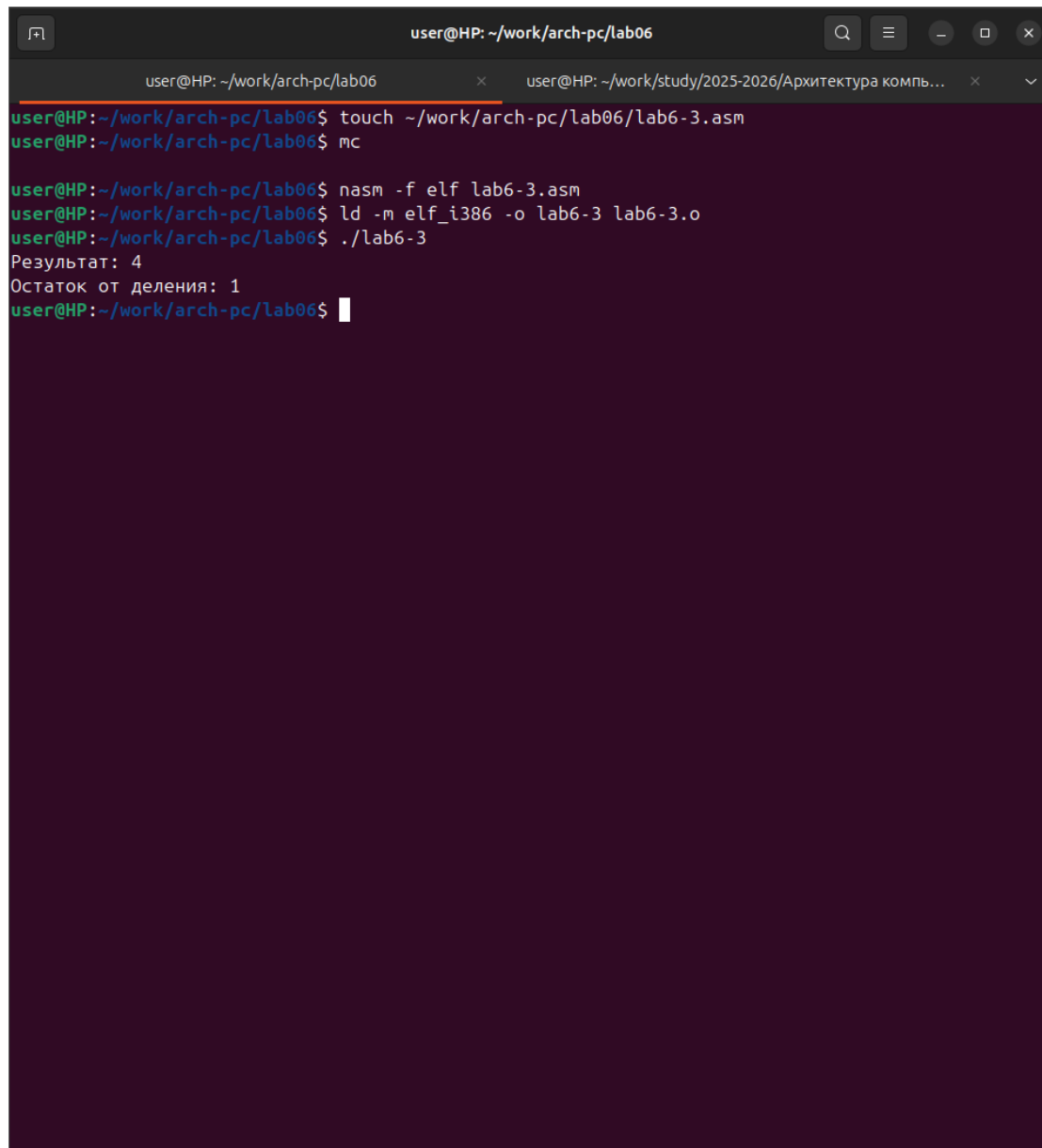
Команды:

```
nasm -f elf lab6-3.asm  
ld -m elf_i386 -o lab6-3 lab6-3.o  
./lab6-3
```

Результат:

Результат: 4

Остаток от деления: 1



```
user@HP: ~/work/arch-pc/lab06
user@HP: ~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-3.asm
user@HP: ~/work/arch-pc/lab06$ mc

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

Скриншот 17:

Комментарий: Программа корректно вычислила выражение: $- 5 * 2 = 10 - 10 + 3 = 13 - 13 / 3 = 4$ (остаток 1)

1.4.18 2.18. Изменение программы для вычисления $f(x) = (4 * 6 + 2) / 5$

Действия: - Открыл файл lab6-3.asm для редактирования - Изменил начальные значения и делитель

Измененный блок вычисления:

```
; ---- Вычисление выражения
mov eax,4          ; EAX=4
mov ebx,6          ; EBX=6
mul ebx            ; EAX=EAX*EBX (4*6=24)
add eax,2          ; EAX=EAX+2 (24+2=26)
xor edx,edx        ; обнуляем EDX для корректной работы div
mov ebx,5          ; EBX=5
div ebx            ; EAX=EAX/5 (26/5=5, остаток 1)
mov edi,eax        ; запись результата вычисления в 'edi'
```

```
user@HP: ~/work/arch-pc/lab06
GNU nano 7.2 /home/user/work/arch-pc/lab06/lab6-3.asm *
;-----
; Программа вычисления выражения
;-----

#include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0

SECTION .text
GLOBAL _start
_start:

; ---- Вычисление выражения
mov eax,4      ; EAX=4
mov ebx,6      ; EBX=6
mul ebx        ; EAX=EAX*EBX (4*6=24)
add eax,2      ; EAX=EAX+2 (24+2=26)
xor edx,edx    ; обнуляем EDX для корректной работы div
mov ebx,5      ; EBX=5
div ebx        ; EAX=EAX/5 (26/5=5, остаток 1)
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

```

Footer:

^G Help	^O Write Out	^W Where Is	^K Cut	^T Execute	^C Location
^X Exit	^R Read File	^_ Replace	^U Paste	^J Justify	^/ Go To Line

Скриншот 18:

1.4.19 2.19. Компиляция и запуск измененной программы

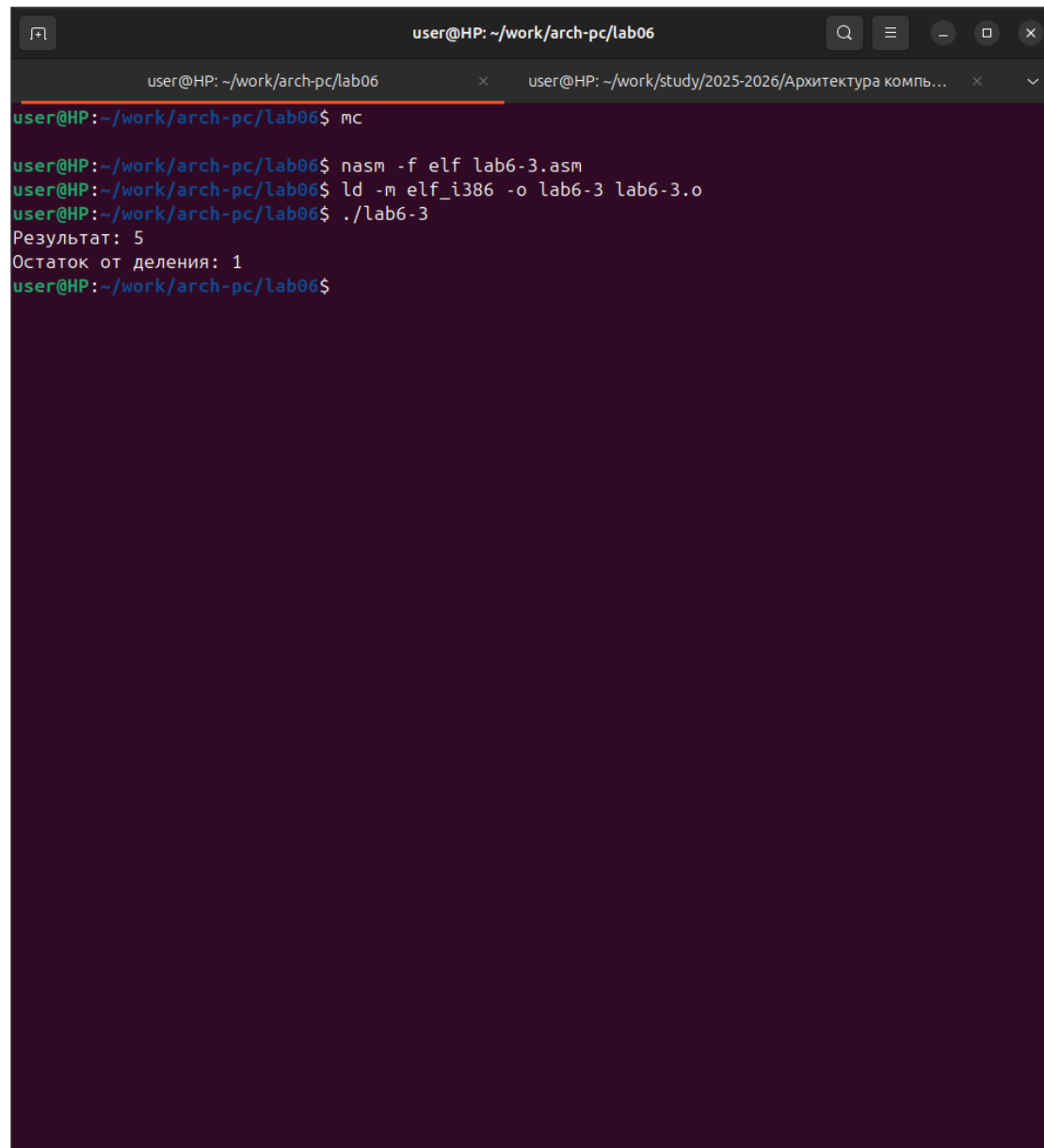
Команды:

```
nasm -f elf lab6-3.asm  
ld -m elf_i386 -o lab6-3 lab6-3.o  
./lab6-3
```

Результат:

Результат: 5

Остаток от деления: 1



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

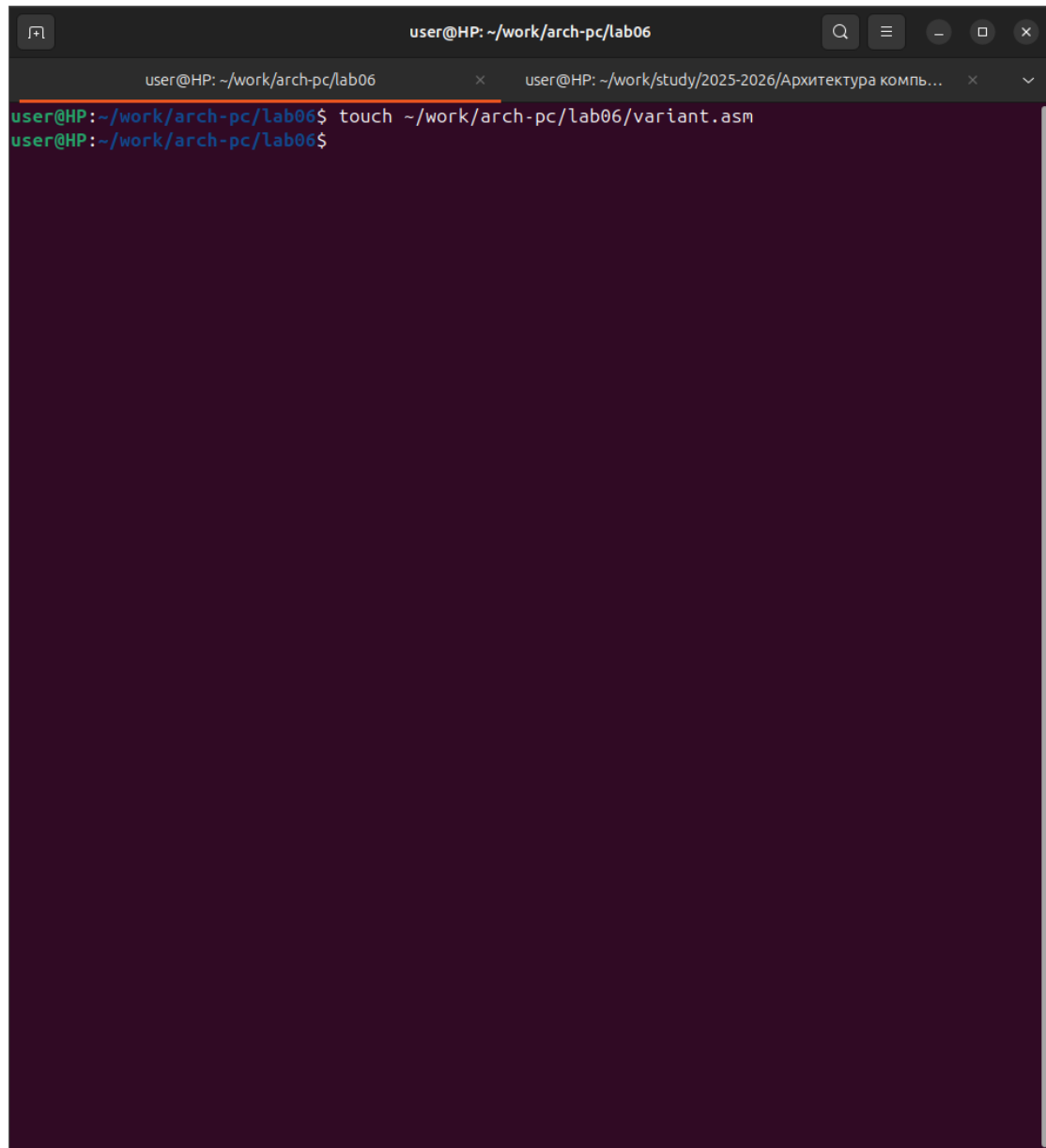
Скриншот 19:

Комментарий: Программа корректно вычислила новое выражение: $-4 * 6 = 24 - 24 + 2 = 26 - 26 / 5 = 5$ (остаток 1)

1.4.20 2.20. Создание файла variant.asm

Команда:

```
touch ~/work/arch-pc/lab06/variant.asm
```



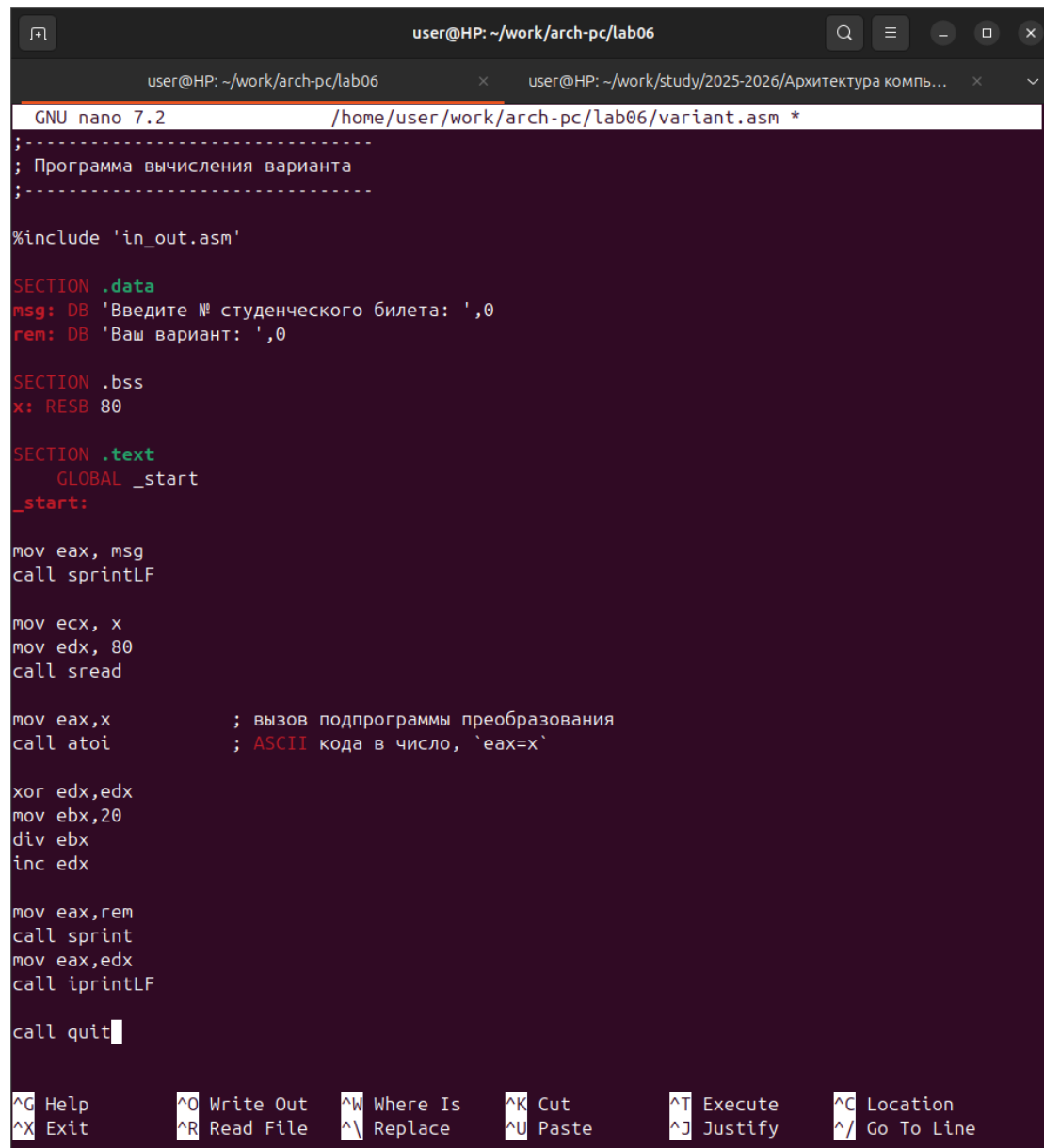
Скриншот 20:

1.4.21 2.21. Редактирование файла variant.asm

Листинг 6.4. Программа вычисления варианта задания по номеру студенческого билета:

```
;-----  
; 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
```



```
GNU nano 7.2 /home/user/work/arch-pc/lab06/variant.asm *
;-----
; 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 sprintf
mov eax, edx
call iprintLF

call quit
```

Скриншот 21:

Комментарий: Программа вычисляет номер варианта по формуле: $(S_n \bmod 20) + 1$, где S_n - номер студенческого билета.

1.4.22 2.22. Компиляция и запуск программы variant.asm

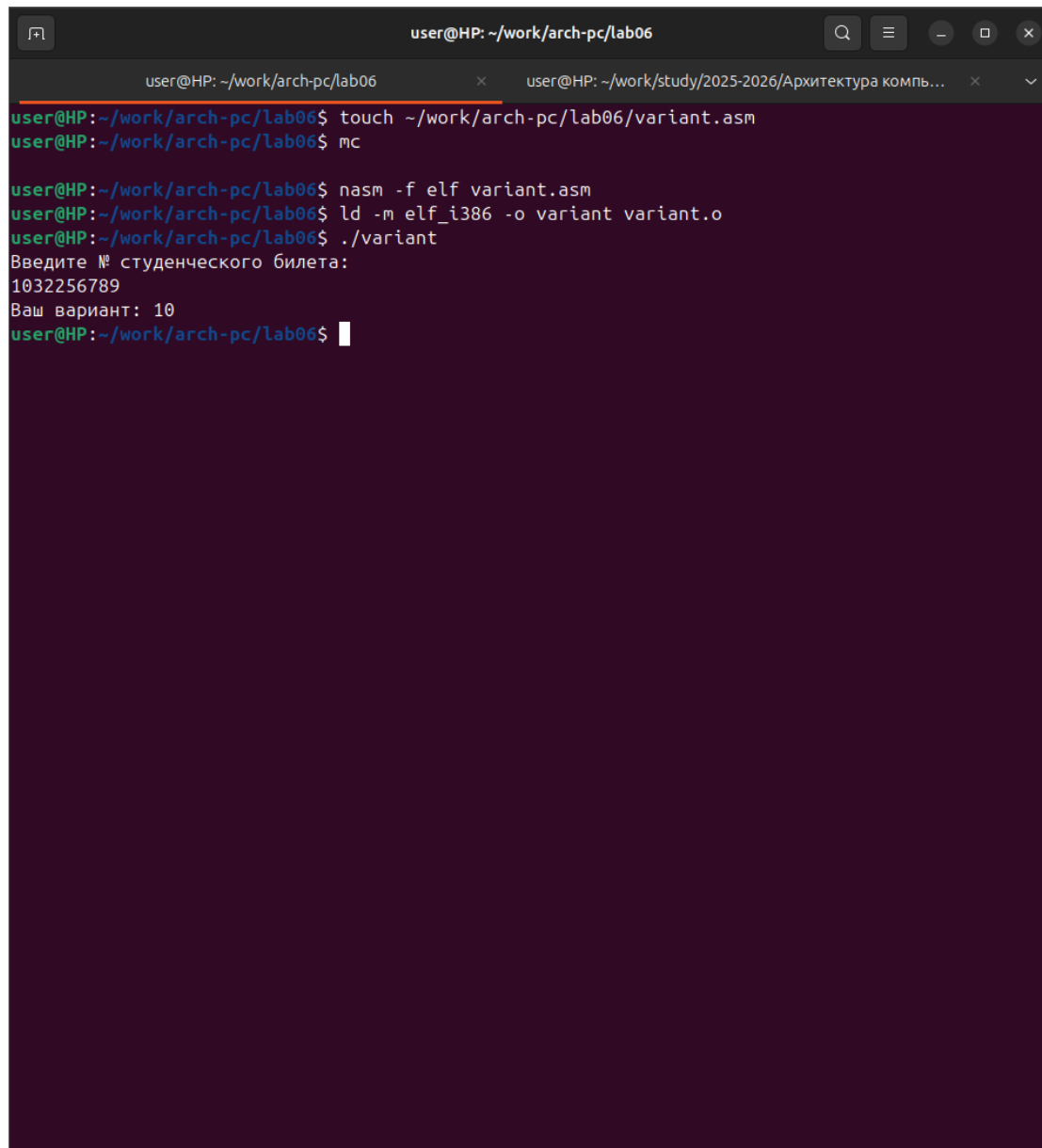
Команды:

```
nasm -f elf variant.asm  
ld -m elf_i386 -o variant variant.o  
./variant
```

Пример выполнения:

Введите № студенческого билета: 1032256789

Ваш вариант: 10



```
user@HP: ~/work/arch-pc/lab06
user@HP: ~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/variant.asm
user@HP: ~/work/arch-pc/lab06$ mc

user@HP: ~/work/arch-pc/lab06$ nasm -f elf variant.asm
user@HP: ~/work/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
user@HP: ~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1032256789
Ваш вариант: 10
user@HP: ~/work/arch-pc/lab06$
```

Скриншот 22:

Комментарий: Проверка вычислений: - Номер студенческого билета:
 $1032256789 - 1032256789 \bmod 20 = 9 - 9 + 1 = 10$ - **Ваш вариант: 10**

1.4.23 2.23. Ответы на вопросы по листингу 6.4

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения „Ваш вариант:“?

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

Эти строки загружают адрес строки „Ваш вариант:“ в регистр `eax` и вызывают функцию `sprint` для вывода сообщения на экран.

2. Для чего используются следующие инструкции?

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

Эти инструкции используются для ввода данных с клавиатуры: - `mov ecx, x` - загрузка адреса буфера `x` в регистр `ecx` (куда будет сохранена введенная строка) - `mov edx, 80` - загрузка максимальной длины вводимой строки (80 байт) в регистр `edx` - `call sread` - вызов подпрограммы чтения строки с клавиатуры

3. Для чего используется инструкция «call atoi»?

Инструкция `call atoi` используется для преобразования ASCII-кода символа (строки) в целое число. Перед вызовом в регистр `eax` должен быть записан адрес строки, содержащей число в символьном виде. После выполнения в регистре `eax` будет находиться числовое значение.

4. Какие строки листинга 6.4 отвечают за вычисления варианта?

```
xor edx,edx      ; обнуление регистра edx
mov ebx,20       ; загрузка делителя 20 в ebx
div ebx         ; деление eax на 20, результат в eax, остаток в edx
inc edx         ; увеличение остатка на 1
```

Эти строки реализуют формулу: $(S_n \bmod 20) + 1$

5. В какой регистр записывается остаток от деления при выполнении инструкции «div ebx»?

Остаток от деления записывается в регистр **EDX**.

При выполнении инструкции `div ebx`: - Частное записывается в регистр EAX - Остаток записывается в регистр EDX

6. Для чего используется инструкция «inc edx»?

Инструкция `inc edx` увеличивает значение регистра `edx` на 1.

В данной программе это необходимо, потому что: - После деления в `edx` находится остаток от деления (от 0 до 19) - Номера вариантов должны быть от 1 до 20 - Поэтому к остатку добавляется 1: `inc edx`

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

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

Эти строки выводят: - Сообщение «Ваш вариант:» (sprint выводит строку из rem) - Значение варианта из регистра edx (iprintLF выводит число с переводом строки)

1.5 3. Задание для самостоятельной работы

1.5.1 3.1. Определение варианта задания

Согласно результатам выполнения программы variant.asm, мой вариант: **10**

Из таблицы 6.3 для варианта 10:

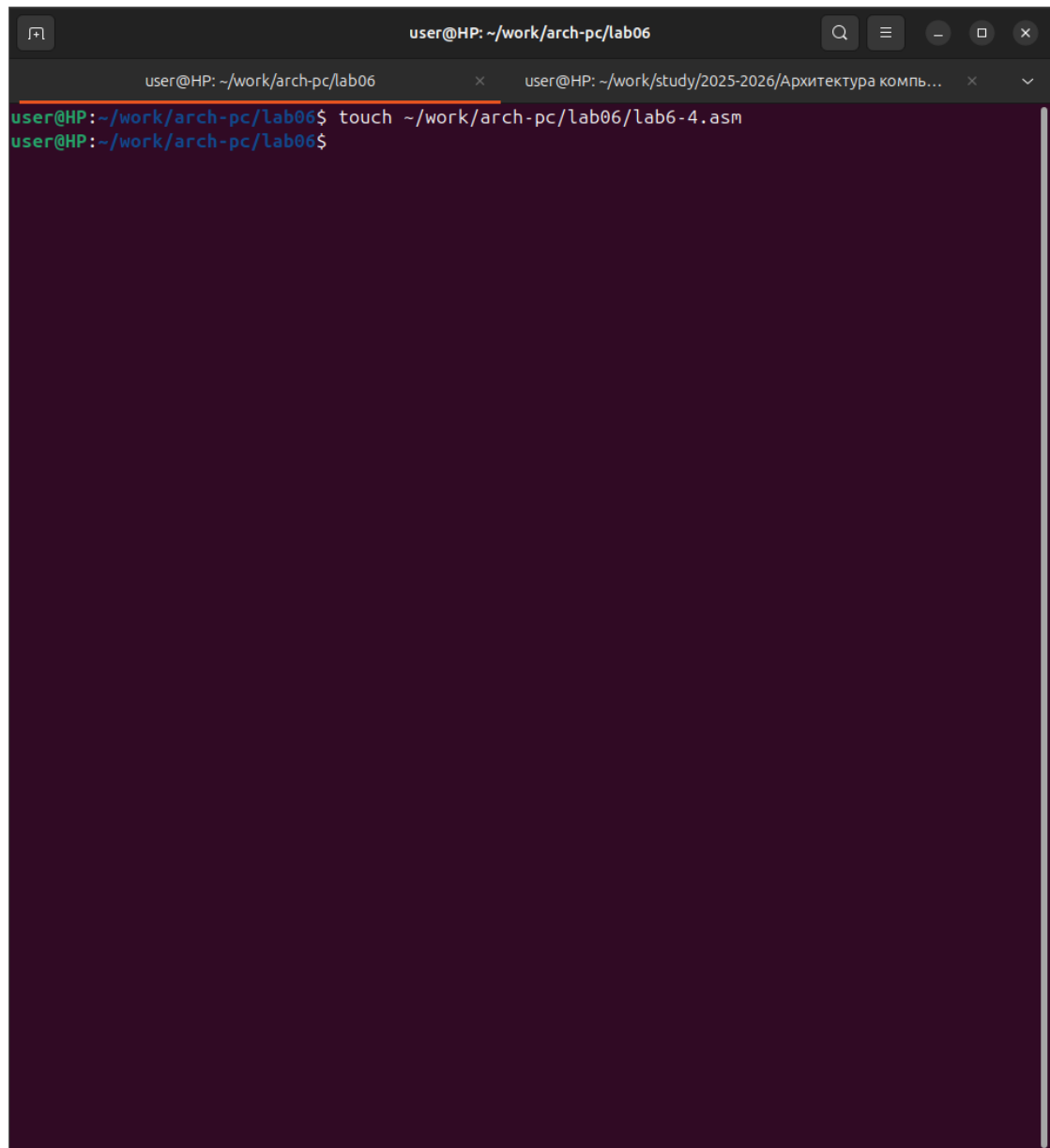
Выражение: $f(x) = 5(x + 18) - 28$

Тестовые значения: - $x_1 = 2$ - $x_2 = 3$

1.5.2 3.2. Создание программы для вычисления выражения

Команда:

```
touch ~/work/arch-pc/lab06/lab6-4.asm
```

A terminal window with a dark background. The title bar shows 'user@HP: ~/work/arch-pc/lab06'. There are two tabs: the active one is 'user@HP: ~/work/arch-pc/lab06' and the other is 'user@HP: ~/work/study/2025-2026/Архитектура компь...'. The terminal shows the command 'touch ~/work/arch-pc/lab06/lab6-4.asm' being entered and executed. The prompt 'user@HP:~/work/arch-pc/lab06\$' is visible before and after the command.

```
user@HP:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-4.asm
user@HP:~/work/arch-pc/lab06$
```

Скриншот 23:

1.5.3 3.3. Написание программы lab6-4.asm

Листинг программы lab6-4.asm для варианта 10:

```

;-----
; Программа вычисления выражения
;  $f(x) = 5(x + 18) - 28$ 
;-----

%include 'in_out.asm'

SECTION .data
msg_expr: DB 'Выражение:  $f(x) = 5(x + 18) - 28$ ',0
msg_input: DB 'Введите значение x: ',0
msg_result: DB 'Результат: ',0

SECTION .bss
x: RESB 80
result: RESB 80

SECTION .text
    GLOBAL _start
_start:

; ---- Вывод выражения
mov eax, msg_expr
call sprintf

; ---- Ввод значения x
mov eax, msg_input
call sprintf

```

```

mov ecx, x
mov edx, 80
call sread

; ---- Преобразование x из ASCII в число
mov eax, x
call atoi
mov ebx, eax          ; сохраняем x в ebx

; ---- Вычисление  $f(x) = 5(x + 18) - 28$ 
; Шаг 1:  $x + 18$ 
add eax, 18           ;  $eax = x + 18$ 

; Шаг 2:  $5 * (x + 18)$ 
mov ecx, 5
mul ecx               ;  $eax = 5 * (x + 18)$ 

; Шаг 3: результат - 28
sub eax, 28           ;  $eax = 5(x + 18) - 28$ 

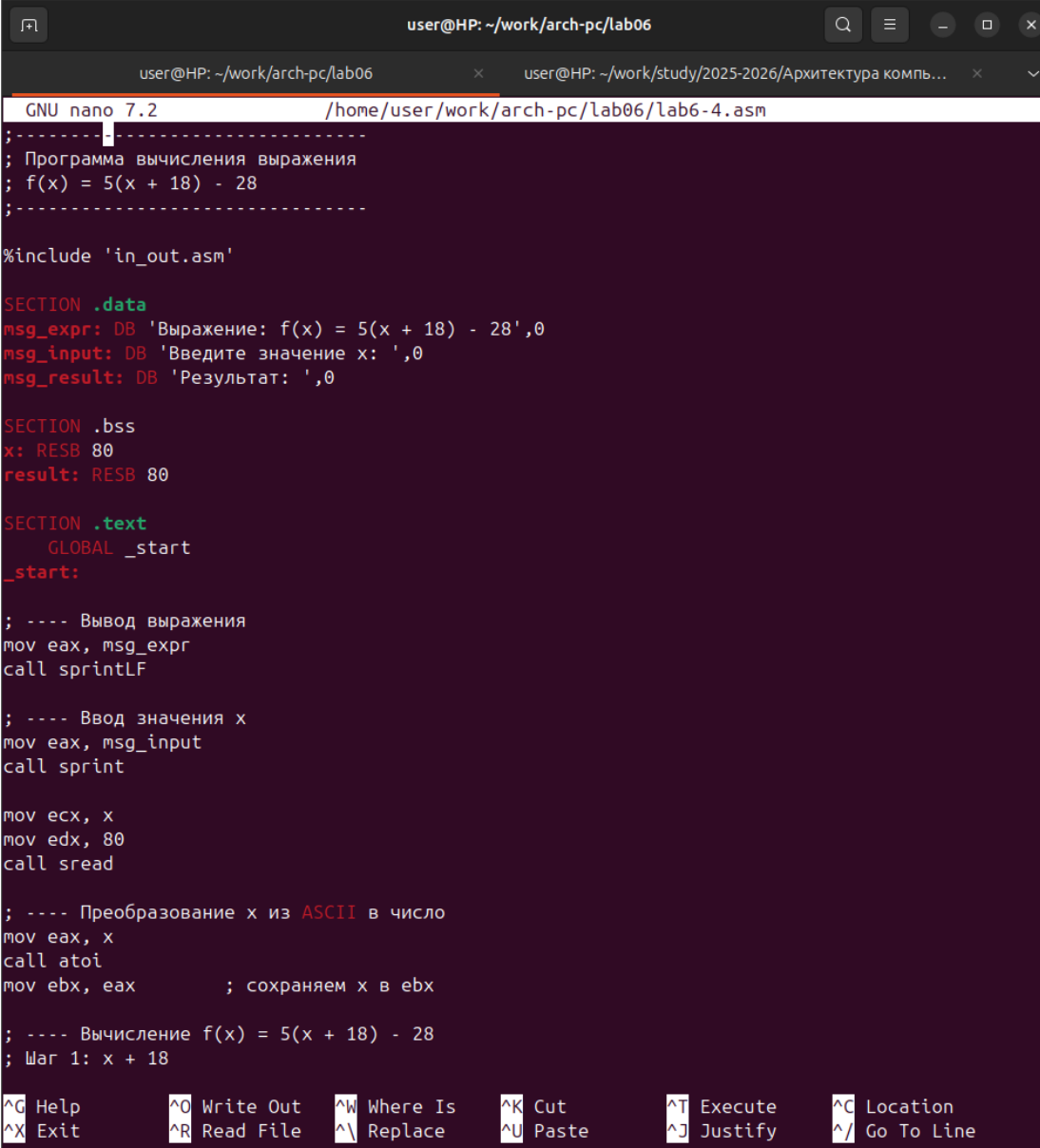
; ---- Вывод результата
mov edi, eax          ; сохраняем результат в edi

mov eax, msg_result
call sprint

mov eax, edi
call iprintLF

```

`call quit`



```
GNU nano 7.2 /home/user/work/arch-pc/lab06/lab6-4.asm
;-----
; Программа вычисления выражения
; f(x) = 5(x + 18) - 28
;-----

#include 'in_out.asm'

SECTION .data
msg_expr: DB 'Выражение: f(x) = 5(x + 18) - 28',0
msg_input: DB 'Введите значение x: ',0
msg_result: DB 'Результат: ',0

SECTION .bss
x: RESB 80
result: RESB 80

SECTION .text
GLOBAL _start
_start:

; ---- Вывод выражения
mov eax, msg_expr
call sprintLF

; ---- Ввод значения x
mov eax, msg_input
call sprint

mov ecx, x
mov edx, 80
call sread

; ---- Преобразование x из ASCII в число
mov eax, x
call atoi
mov ebx, eax      ; сохраняем x в ebx

; ---- Вычисление f(x) = 5(x + 18) - 28
; Шаг 1: x + 18

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

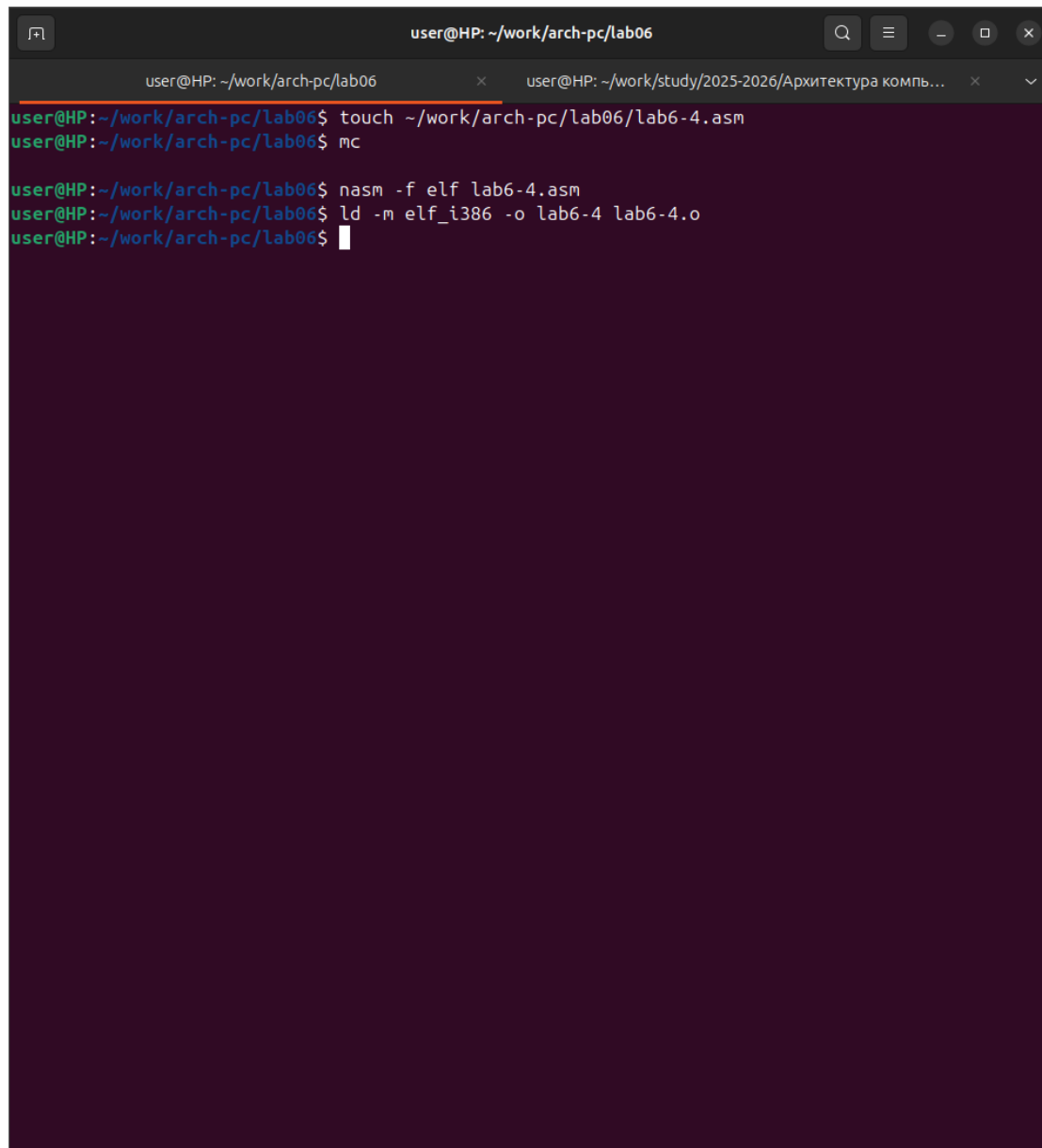
Скриншот 24:

Комментарий: Программа реализует вычисление выражения $f(x) = 5(x + 18) - 28$ по следующему алгоритму: 1. Вывод формулы на экран 2. Запрос значения x 3. Преобразование введенного значения в число 4. Вычисление: $x + 18$ 5. Умножение результата на 5 6. Вычитание 28 7. Вывод результата

1.5.4 3.4. Компиляция программы lab6-4.asm

Команды:

```
nasm -f elf lab6-4.asm  
ld -m elf_i386 -o lab6-4 lab6-4.o
```

A terminal window with a dark purple background. The title bar shows 'user@HP: ~/work/arch-pc/lab06'. There are two tabs: 'user@HP: ~/work/arch-pc/lab06' (active) and 'user@HP: ~/work/study/2025-2026/Архитектура компь...'. The terminal shows the following commands and their outputs:

```
user@HP:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-4.asm
user@HP:~/work/arch-pc/lab06$ mc

user@HP:~/work/arch-pc/lab06$ nasm -f elf lab6-4.asm
user@HP:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-4 lab6-4.o
user@HP:~/work/arch-pc/lab06$
```

Скриншот 25:

1.5.5 3.5. Тестирование программы с $x_1 = 2$

Команда:

./lab6-4

Ввод:

2

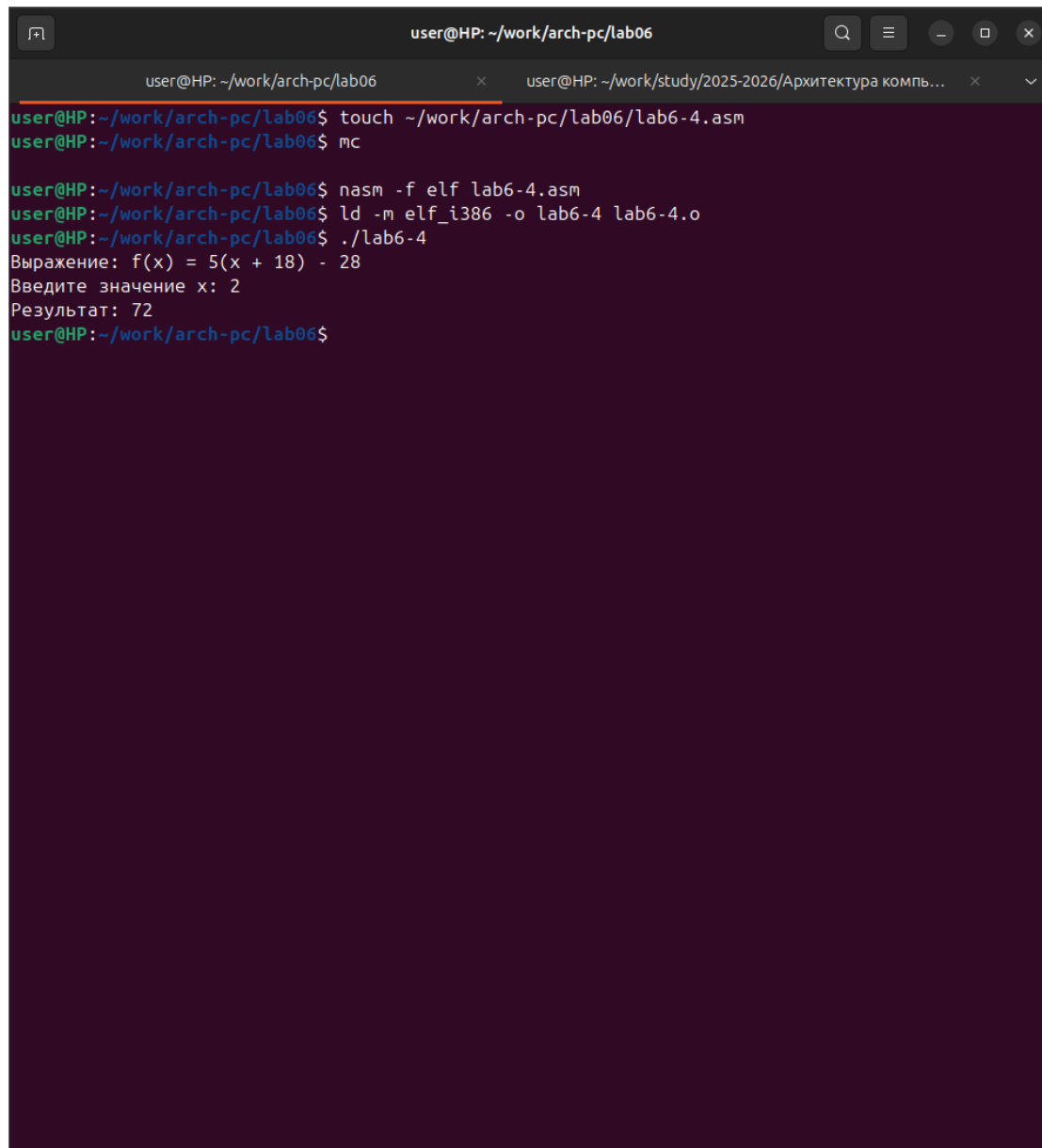
Результат:

Выражение: $f(x) = 5(x + 18) - 28$

Введите значение x : 2

Результат: 72

Проверка вычислений: - $f(2) = 5(2 + 18) - 28$ - $f(2) = 5 \times 20 - 28$ - $f(2) = 100 - 28$
- $f(2) = 72$ ✓



```
user@HP: ~/work/arch-pc/lab06
user@HP: ~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-4.asm
user@HP: ~/work/arch-pc/lab06$ mc

user@HP: ~/work/arch-pc/lab06$ nasm -f elf lab6-4.asm
user@HP: ~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-4 lab6-4.o
user@HP: ~/work/arch-pc/lab06$ ./lab6-4
Выражение: f(x) = 5(x + 18) - 28
Введите значение x: 2
Результат: 72
user@HP: ~/work/arch-pc/lab06$
```

Скриншот 26:

Комментарий: Результат совпадает с ожидаемым значением.

1.5.6 3.6. Тестирование программы с $x_2 = 3$

Команда:

./lab6-4

Ввод:

3

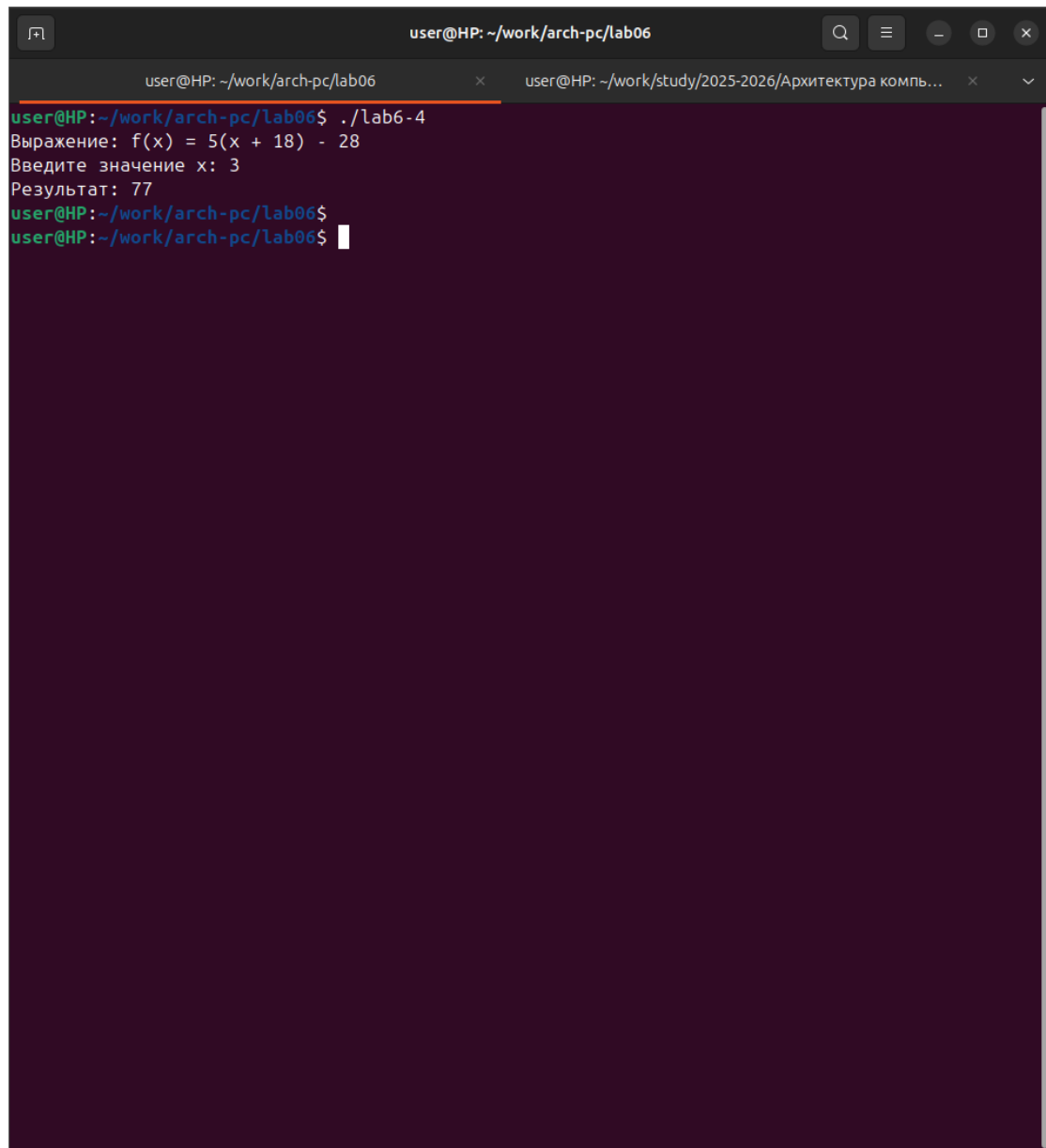
Результат:

Выражение: $f(x) = 5(x + 18) - 28$

Введите значение x : 3

Результат: 77

Проверка вычислений: - $f(3) = 5(3 + 18) - 28$ - $f(3) = 5 \times 21 - 28$ - $f(3) = 105 - 28$
- $f(3) = 77 \checkmark$



```
user@HP: ~/work/arch-pc/lab06
user@HP: ~/work/arch-pc/lab06$ ./lab6-4
Выражение: f(x) = 5(x + 18) - 28
Введите значение x: 3
Результат: 77
user@HP: ~/work/arch-pc/lab06$
user@HP: ~/work/arch-pc/lab06$
```

Скриншот 27:

Комментарий: Результат совпадает с ожидаемым значением. Программа работает корректно.

1.5.7 3.7. Загрузка файлов на GitHub

Команды:

```
cp -r ~/work/arch-pc/lab06/ ~/work/study/2025-2026/Архитектура\ компьютера/arch-pc/la
cd ~/work/study/2025-2026/Архитектура\ компьютера/arch-pc/labs/lab06
git add .
git commit -m "feat(lab06): add lab6 programs and report"
git push
```

Комментарий: Все файлы лабораторной работы успешно загружены в репозиторий.

1.6 4. Ответы на контрольные вопросы

1. Какой синтаксис команды сложения чисел?

Синтаксис команды сложения:

```
add <операнд_1>, <операнд_2>
```

Команда `add` выполняет сложение двух операндов и записывает результат по адресу первого операнда.

Допустимые сочетания операндов: - Регистр, регистр: `add eax, ebx` ($EAX = EAX + EBX$) - Регистр, память: `add eax, [var]` ($EAX = EAX + \text{значение из памяти}$) - Память, регистр: `add [var], eax` (значение в памяти = значение в памяти + EAX) - Регистр, непосредственное значение: `add eax, 5` ($EAX = EAX + 5$) - Память, непосредственное значение: `add [var], 10`

Недопустимо: `add [var1], [var2]` (нельзя использовать два операнда в памяти)

Примеры:

```
add ax, 5      ; AX = AX + 5
add dx, cx     ; DX = DX + CX
add eax, ebx   ; EAX = EAX + EBX
```

Важно: Размеры операндов должны совпадать (нельзя: `add eax, cl`).

2. Какая команда выполняет умножение без знака?

Команда **mul** (от англ. multiply – умножение) выполняет беззнаковое умножение.

Синтаксис:

```
mul <операнд>
```

Особенности работы команды mul:

Операнд указывает только один из множителей. Второй множитель неявно находится в регистре AL, AX или EAX (в зависимости от размера операнда). Результат помещается в специальные регистры:

Размер операнда	Неявный множитель	Результат умножения
1 байт	AL	AX
2 байта	AX	DX:AX
4 байта	EAX	EDX:EAX

Примеры:

```
mov al, 5
mov bl, 3
mul bl      ; AX = AL * BL = 5 * 3 = 15
```

```

mov ax, 100
mov bx, 200
mul bx           ; DX:AX = AX * BX = 100 * 200 = 20000

mov eax, 1000
mov ebx, 5000
mul ebx         ; EDX:EAX = EAX * EBX = 1000 * 5000 = 5000000

```

Для знакового умножения используется команда **imul**.

3. Какой синтаксис команды деление чисел без знака?

Команда **div** (от англ. divide - деление) выполняет беззнаковое деление.

Синтаксис:

```
div <делитель>
```

Особенности работы команды **div**:

В команде указывается только делитель (может быть регистром или ячейкой памяти, но не может быть непосредственным операндом). Делимое и результат зависят от размера делителя:

Размер делителя	Делимое	Частное	Остаток
1 байт	AX	AL	AH
2 байта	DX:AX	AX	DX
4 байта	EDX:EAX	EAX	EDX

Примеры:

Деление 1-байтовым делителем:

```
mov ax, 31      ; делимое в AX
mov dl, 15      ; делитель
div dl          ; AL = 31/15 = 2, AH = 31%15 = 1
```

Деление 2-байтовым делителем:

```
mov ax, 2       ; младшая часть делимого
mov dx, 1       ; старшая часть делимого (DX:AX = 0x10002)
mov bx, 16      ; делитель
div bx          ; AX = частное (0x1000), DX = остаток (2)
```

Деление 4-байтовым делителем:

```
mov eax, 100    ; младшая часть делимого
xor edx, edx     ; обнуляем старшую часть (EDX:EAX = 100)
mov ebx, 20     ; делитель
div ebx         ; EAX = 5, EDX = 0
```

Важно: Перед делением необходимо правильно инициализировать регистры DX или EDX (обычно обнуляются через `xor edx, edx`), иначе результат будет неверным.

Для знакового деления используется команда `idiv`.

4. Куда помещается результат при умножении двухбайтовых операндов?

При умножении двухбайтовых операндов (2 байта = 16 бит = слово) результат помещается в пару регистров **DX:AX**.

Подробнее: - Старшие 16 бит результата → регистр **DX** - Младшие 16 бит результата → регистр **AX** - Вместе DX:AX образуют 32-битное значение

Пример:

```

mov ax, 1000    ; AX = 1000
mov bx, 500     ; BX = 500
mul bx          ; DX:AX = 1000 * 500 = 500000

; После выполнения:
; AX = 0x86A0 (младшие 16 бит числа 500000)
; DX = 0x0007 (старшие 16 бит числа 500000)
; DX:AX = 0x0007_86A0 = 500000 в десятичной системе

```

Объяснение: - 500000 в шестнадцатеричной системе = 0x7A120 - Но при 16-битном умножении: 500000 = 0x0007_86A0 - DX = 0x0007 - AX = 0x86A0

Если результат умножения помещается в 16 бит (меньше 65536), то DX будет равен 0.

5. Перечислите арифметические команды с целочисленными операндами и дайте их назначение.

Команды сложения и вычитания:

1. **add** (addition) - сложение

```
add dst, src    ; dst = dst + src
```

Складывает два операнда, результат в первом операнде.

2. **sub** (subtraction) - вычитание

```
sub dst, src    ; dst = dst - src
```

Вычитает второй операнд из первого, результат в первом операнде.

3. **inc** (increment) - инкремент (увеличение на 1)

```
inc operand      ; operand = operand + 1
```

Увеличивает операнд на 1. Занимает меньше места, чем `add operand, 1`.

4. **dec** (decrement) - декремент (уменьшение на 1)

```
dec operand      ; operand = operand - 1
```

Уменьшает операнд на 1. Занимает меньше места, чем `sub operand, 1`.

5. **neg** (negate) - изменение знака

```
neg operand      ; operand = -operand
```

Меняет знак операнда на противоположный (для чисел со знаком).

Команды умножения:

6. **mul** (multiply) - беззнаковое умножение

```
mul operand      ; AX = AL * operand (1 байт)  
                  ; DX:AX = AX * operand (2 байта)  
                  ; EDX:EAX = EAX * operand (4 байта)
```

Умножает неявный операнд на явный, результат в специальных регистрах.

7. **imul** (integer multiply) - знаковое умножение

```
imul operand ; Аналогично mul, но для чисел со знаком
```

Команды деления:

8. **div** (divide) - беззнаковое деление

```
div operand ; AL = AX / operand, AH = остаток (1 байт)  
; AX = DX:AX / operand, DX = остаток (2 байта)  
; EAX = EDX:EAX / operand, EDX = остаток (4 байта)
```

Делит неявное делимое на операнд, частное и остаток в специальных регистрах.

9. **idiv** (integer divide) - знаковое деление

```
idiv operand ; Аналогично div, но для чисел со знаком
```

6. Где находится делимое при целочисленном делении операндов?

Местоположение делимого зависит от размера делителя:

Для 1-байтового делителя: - Делимое находится в регистре **AX** (16 бит) -

Пример: `nasm mov ax, 100 ; делимое в AX mov bl, 7 ; делитель
div bl ; AL = 100/7 = 14, AH = 100%7 = 2`

Для 2-байтового делителя: - Делимое находится в паре регистров **DX:AX** (32 бита) - DX содержит старшие 16 бит - AX содержит младшие 16 бит - Пример: `nasm mov dx, 0 ; старшие 16 бит делимого mov ax, 1000 ;
младшие 16 бит делимого (DX:AX = 1000) mov bx, 25 ; делитель div
bx ; AX = 1000/25 = 40, DX = 0`

Для 4-байтового делителя: - Делимое находится в паре регистров **EDX:EAX** (64 бита) - EDX содержит старшие 32 бита - EAX содержит младшие 32 бита -

Пример: `mov eax, 10000 ; младшие 32 бита делимого xor edx, edx`
`; обнуляем старшие 32 бита (EDX:EAX = 10000) mov ebx, 100 ; делитель`
`div ebx ; EAX = 10000/100 = 100, EDX = 0`

Важно: Перед делением необходимо правильно инициализировать регистр DX (или EDX). Обычно его обнуляют командой `xor edx, edx` или `xor dx, dx`, если делимое положительное и помещается в младшем регистре.

7. Куда помещаются неполное частное и остаток при делении целочисленных операндов?

Местоположение частного и остатка зависит от размера делителя:

Для 1-байтового делителя: - **Частное** → регистр **AL** (младшая часть AX) - **Остаток** → регистр **AH** (старшая часть AX) - Пример: `mov ax, 23 ; делимое`
`mov bl, 5 ; делитель div bl ; AL = 23/5 = 4, AH = 23%5 = 3`

Для 2-байтового делителя: - **Частное** → регистр **AX** - **Остаток** → регистр **DX** - Пример: `mov dx, 0 ; старшая часть делимого mov ax, 100`
`; младшая часть делимого mov bx, 7 ; делитель div bx ; AX = 100/7 = 14, DX = 100%7 = 2`

Для 4-байтового делителя: - **Частное** → регистр **EAX** - **Остаток** → регистр **EDX** - Пример: `mov eax, 1000 ; младшая часть делимого xor edx, edx`
`; обнуление старшей части mov ebx, 20 ; делитель div ebx ; EAX = 1000/20 = 50, EDX = 0`

Таблица для справки:

Размер делителя	Делимое	Частное	Остаток
1 байт	AX	AL	AH
2 байта	DX:AX	AX	DX

Размер делителя	Делимое	Частное	Остаток
4 байта	EDX:EAX	EAX	EDX

Важное замечание: После деления в регистре частного всегда находится целая часть от деления (без дробной части), а в регистре остатка - остаток от деления.

1.7 5. Выводы

В ходе выполнения лабораторной работы №6 были освоены арифметические инструкции языка ассемблера NASM и приобретены следующие знания и навыки:

1. Изучены способы адресации в NASM:

- Регистровая адресация (операнды в регистрах)
- Непосредственная адресация (константы в командах)
- Адресация памяти (работа с переменными)

2. Освоены основные арифметические команды:

- Команды сложения (add) и вычитания (sub)
- Команды инкремента (inc) и декремента (dec)
- Команда изменения знака (neg)
- Команды умножения (mul для беззнаковых, imul для знаковых)
- Команды деления (div для беззнаковых, idiv для знаковых)

3. Понято различие между символьными и численными данными:

- Символы хранятся в виде ASCII-кодов

- Арифметические операции над символами выполняются над их кодами
- Необходимость преобразования между символьным и числовым представлением

4. Изучены функции преобразования из файла in_out.asm:

- `iPrintLF` и `iPrint` - вывод чисел в формате ASCII
- `atoi` - преобразование ASCII-строки в целое число
- Понята разница между функциями с переводом строки и без

5. Получен практический опыт:

- Написаны и отлажены программы для вычисления арифметических выражений
- Реализована программа вычисления варианта по формуле $(S_n \bmod 20) + 1$
- Создана программа для вычисления индивидуального задания: $f(x) = 5(x + 18) - 28$

6. Проверена корректность работы программ:

- Программа для варианта 10 протестирована с $x_1 = 2$ (результат: 72)
✓
- Программа для варианта 10 протестирована с $x_2 = 3$ (результат: 77)
✓
- Все результаты совпали с аналитическими вычислениями

7. Освоены особенности работы с операндами разных размеров:

- Понято размещение результатов умножения в парах регистров (DX:AX, EDX:EAX)
- Изучено размещение делимого, частного и остатка при делении
- Понята важность обнуления регистров EDX/DX перед делением

Полученные навыки работы с арифметическими инструкциями являются фундаментальными для программирования на языке ассемблера и позволяют реализовывать сложные вычислительные алгоритмы на низком уровне.

1.8 6. Список файлов работы

Программы: - lab6-1.asm - программа вывода значения регистра eax (символы и числа) - lab6-2.asm - программа с использованием iprintf - lab6-3.asm - программа вычисления выражения $f(x) = (5 * 2 + 3) / 3$ и $f(x) = (4 * 6 + 2) / 5$ - variant.asm - программа вычисления варианта задания - lab6-4.asm - программа вычисления индивидуального задания (вариант 10) - in_out.asm - файл с подпрограммами ввода-вывода

Отчет: - report.md - отчет в формате Markdown - report.pdf - отчет в формате PDF - report.docx - отчет в формате DOCX

Ссылка на репозиторий GitHub:

https://github.com/hatimnhari/study_2025-2026_arch-pc

Конец отчета