

Отчет по лабораторной работе №7

Дисциплина: Архитектура компьютера

Мустафина Аделя Юрисовна

Содержание

| | | |
|----------|--|-----------|
| 1 | Цель работы | 5 |
| 2 | Задание | 6 |
| 3 | Теоретическое введение | 7 |
| 3.1 | 7.2.1. Команды безусловного перехода | 7 |
| 3.2 | 7.2.1. Команды безусловного перехода | 7 |
| 3.3 | 7.2.2.1. Регистр флагов | 8 |
| 3.4 | 7.2.2.2. Описание инструкции str | 8 |
| 3.5 | 7.2.2.3. Описание команд условного перехода. | 8 |
| 3.6 | 7.2.3. Файл листинга и его структура | 9 |
| 4 | Выполнение лабораторной работы | 10 |
| 5 | Изучение структуры файла листинга | 18 |
| 6 | Задание для самостоятельной работы | 21 |
| 7 | Выводы | 26 |
| 8 | Список литературы | 27 |

Список иллюстраций

| | | |
|-----|--|----|
| 4.1 | Создание файла | 10 |
| 4.2 | Листинг 7.1 | 10 |
| 4.3 | Запуск | 11 |
| 4.4 | Измененный листинг 7.2 | 13 |
| 4.5 | Запуск 7.2 | 14 |
| 4.6 | Листинг 7.3 | 15 |
| 4.7 | Листинг 7.3 запуск | 17 |
| 5.1 | Содержимое lab7-2.lst | 18 |
| 5.2 | Удаление операнда | 19 |
| 5.3 | операнд | 19 |
| 5.4 | листинг | 20 |
| 6.1 | Задание 1 для самостоятельной работы | 21 |
| 6.2 | Задание 2 для самостоятельной работы | 23 |

Список таблиц

1 Цель работы

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

2 Задание

1. Выполнение лабораторной работы
2. Выполнение заданий для самостоятельной работы

3 Теоретическое введение

Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Можно выделить 2 типа переходов: • условный переход – выполнение или не выполнение перехода в определенную точку программы в зависимости от проверки условия. • безусловный переход – выполнение передачи управления в определенную точку программы без каких-либо условий.

3.1 7.2.1. Команды безусловного перехода

Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление:

`jmp` Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре.

3.2 7.2.1. Команды безусловного перехода

Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление: `jmp` Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. Кроме того, в качестве операнда

можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре.

3.3 7.2.2.1. Регистр флагов

Флаг – это бит, принимающий значение 1 («флаг установлен»), если выполнено некоторое условие, и значение 0 («флаг сброшен») в противном случае. Флаги работают независимо друг от друга, и лишь для удобства они помещены в единый регистр – регистр флагов, отражающий текущее состояние процессора. В следующей таблице указано положение битовых флагов в регистре флагов.

3.4 7.2.2.2. Описание инструкции `cmp`

Инструкция `cmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `cmp` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания:

`cmp r1, r2`, Команда `cmp`, так же как и команда вычитания, выполняет вычитание `r2` из `r1`, но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов.

3.5 7.2.2.3. Описание команд условного перехода.

Команда условного перехода имеет вид `j label` Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. В их мнемокодах указывается тот результат сравнения, при котором надо делать переход. Мнемоники, идентичные по своему действию, написаны в таблице через дробь (например, `ja` и `jnb`). Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы.

3.6 7.2.3. Файл листинга и его структура

Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию. Ниже приведён фрагмент файла листинга.

```
10 00000000 B804000000 mov eax,4 11 00000005 BB01000000 mov ebx,1 12
0000000A B9[00000000] mov ecx,hello 13 0000000F BA0D000000 mov edx,helloLen
14 15 00000014 CD80 int 80h
```

Все ошибки и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не создаётся. Итак, структура листинга:

- номер строки — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы);
- адрес — это смещение машинного кода от начала текущего сегмента;
- машинный код представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция `int 80h` начинается по смещению `00000020` в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция `int 80h` ассемблируется в `CD80` (в шестнадцатеричном представлении); `CD80` — это инструкция на машинном языке, вызывающая прерывание ядра);
- исходный текст программы — это просто строка исходной программы вместе с комментариями (некоторые строки на языке ассемблера, например, строки, содержащие только комментарии, не генерируют никакого машинного кода, и поля «смещение» и «исходный текст программы» в таких строках отсутствуют, однако номер строки им присваивается).

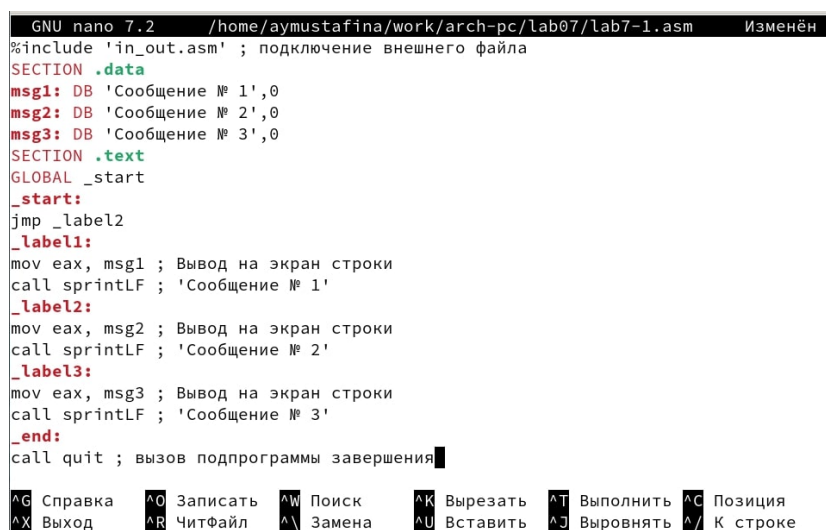
4 Выполнение лабораторной работы

Создаю каталог для программ лабораторной работы №7 и создаю там файл(рис. 4.1).

```
aymustafina@vbox:~$ mkdir ~/work/arch-pc/lab07
aymustafina@vbox:~$ cd ~/work/arch-pc/lab07
aymustafina@vbox:~/work/arch-pc/lab07$ touch lab7-1.asm
aymustafina@vbox:~/work/arch-pc/lab07$
```

Рис. 4.1: Создание файла

Заполняю файл с помощью листинга 7.1 с использованием инструкции jmp(рис. 4.2).



```
GNU nano 7.2 /home/aymustafina/work/arch-pc/lab07/lab7-1.asm Изменён
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выводить ^/_ К строке

Рис. 4.2: Листинг 7.1

Листинг 7.1. Программа с использованием инструкции jmp:

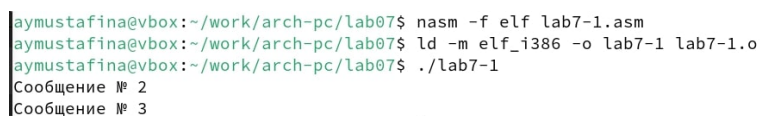
```
%include 'in_out.asm' ; подключение внешнего файла
```

```

SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения

```

Создаю исполняемый файл и запускаю его(рис. 4.3).



```

aymustafina@vbox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
aymustafina@vbox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
aymustafina@vbox:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3

```

Рис. 4.3: Запуск

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения. Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. Изменим программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала

работу. Для этого в текст программы после вывода сообщения № 2 добавим инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавим инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`). Измените текст программы в соответствии с листингом 7.2.

Листинг 7.2. Программа с использованием инструкции `jmp`:

```
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

Меняю код листинга 7.2, чтобы получить вывод сообщений в обратном порядке

(рис. 4.4).

```
GNU nano 7.2 /home/aymustafina/work/arch-pc/lab07/lab7-1.asm
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label3
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершени
```

Рис. 4.4: Измененный листинг 7.2

Измененный листинг 7.2

```
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label3
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
```

```

jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершени

```

Создаю исполняемый файл и запускаю его (рис. 4.5).

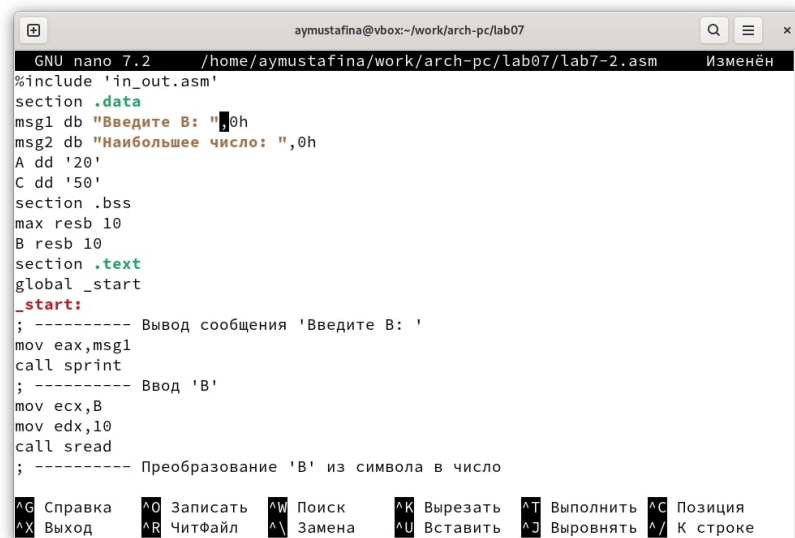
```

aymustafina@vbox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
aymustafina@vbox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
aymustafina@vbox:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1

```

Рис. 4.5: Запуск 7.2

Использование инструкции `jmp` приводит к переходу в любом случае. Однако, часто при написании программ необходимо использовать условные переходы, т.е. переход должен происходить если выполнено какое-либо условие. В качестве примера рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А, В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры. Создаю файл `lab7-2.asm` в каталоге `~/work/arch-pc/lab07`. И ввожу текст их листинга 7.3 (рис. 4.6).



```
GNU nano 7.2 /home/aymustafina/work/arch-pc/lab07/lab7-2.asm
%include 'in_out.asm'
section .data
msg1 db "Введите B: ",0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'
section .bss
max resb 10
B resb 10
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите B: '
mov eax,msg1
call sprint
; ----- Ввод 'B'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'B' из символа в число

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выровнять ^_ К строке
```

Рис. 4.6: Листинг 7.3

Листинг 7.3. Программа, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: A, B и C.

```
%include 'in_out.asm'
section .data
msg1 db 'Введите B: ',0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'
section .bss
max resb 10
B resb 10
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите B: '
mov eax,msg1
call sprint
```

```

; ----- Ввод 'B'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'B' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'ecx = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx = C'
mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
call atoi ; Вызов подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:

```



```

mov eax, msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintLF ; Вывод 'max(A,B,C)'
call quit ; Выход

```

Запускаю файл и проверяю его работу для разных значений В (рис. ??).

```

aymustafina@vbox:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
aymustafina@vbox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
aymustafina@vbox:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 4
Наибольшее число: 50
aymustafina@vbox:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 10
Наибольшее число: 50
aymustafina@vbox:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 100
Наибольшее число: 100
aymustafina@vbox:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 70
Наибольшее число: 70
aymustafina@vbox:~/work/arch-pc/lab07$ ./lab7-2

```

Рис. 4.7: Листинг 7.3 запуск

В данном примере переменные А и С сравниваются как символы, а переменная В и максимум из А и С как числа (для этого используется функция atoi преобразования символа в число). Это сделано для демонстрации того, как сравниваются данные. Данную программу можно упростить и сравнивать все 3 переменные как символы (т.е. не использовать функцию atoi). Однако если переменные преобразовать из символов числа, над ними можно корректно проводить арифметические операции.

5 Изучение структуры файла листинга

Создаю файл листинга для программы из файла lab7-2.asm

nasm -f elf -l lab7-2.lst lab7-2.asm Открываю файл листинга lab7-2.lst с помощью любого текстового редактора, например mcedit:

mcedit lab7-2.lst Внимательно ознакомлюсь с его форматом и содержимым.

Объясняю содержимое трёх строк файла листинга (рис. 5.1).

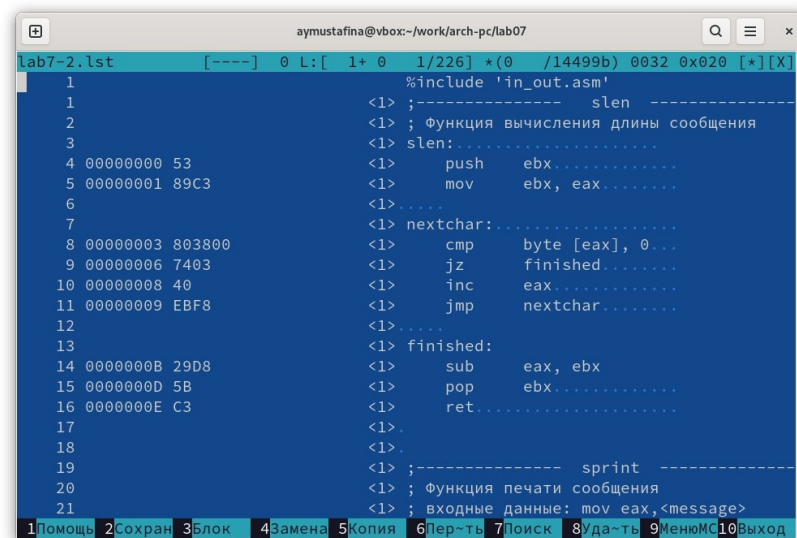


Рис. 5.1: Содержимое lab7-2.lst

Строка 5. С помощью этой строки мы добавляем значение как символ для переменной A

A dd '20'

Строка 7. Эти строки кода на ассемблере сравнивают байт по адресу [eax] с нулем. Если они равны, выполняется метка finished. Либо увеличивается значе-

ние регистра `eax` и выполняется метка `nextchar`. Это цикл который обрабатывает значения пока не встретится 0.

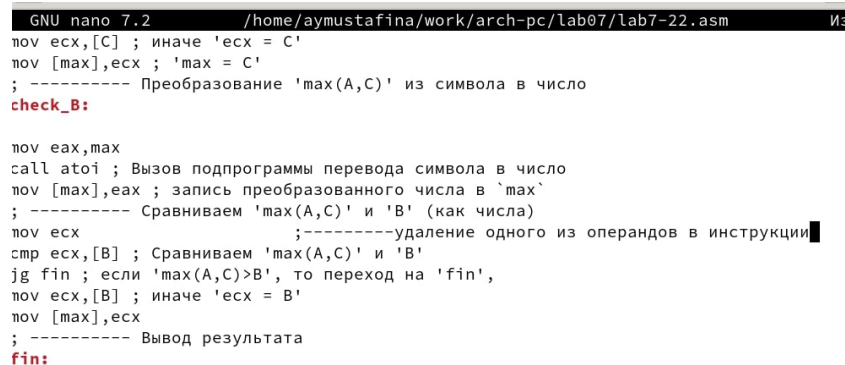
`nextchar:`

```
    cmp byte [eax], 0
    jz finished
    inc eax
    jmp nextchar
```

Строка 22. Функция `atoi` преобразует строку в целое число. Так, в данной программе мы сперва вводили для `B` значение в виде символа, а далее записывали значение в переменную.

`call atoi`

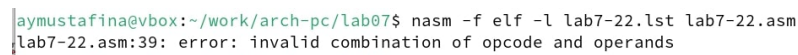
Копирую с программой `lab7-2.asm` и называю его `lab7-22.asm` и в инструкции с двумя операндами удаляю один операнд. Выполняю трансляцию с получением файла листинга: `nasm -f elf -l lab7-22.lst lab7-22.asm` (рис. 5.2).



```
GNU nano 7.2 /home/aymustafina/work/arch-pc/lab07/lab7-22.asm Ис
mov ecx,[C] ; иначе 'ecx = C'
mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
call atoi ; Вызов подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx ;-----удаление одного из операндов в инструкции
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:
```

Рис. 5.2: Удаление операнда

Удаляю из инструкции (рис. 5.3).



```
aymustafina@vbox:~/work/arch-pc/lab07$ nasm -f elf -l lab7-22.lst lab7-22.asm
lab7-22.asm:39: error: invalid combination of opcode and operands
```

Рис. 5.3: операнд

В листинге программы (рис. 5.4).

```
38 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
39 mov ecx ; -----удаление одного из операндов в инструкции
39 ***** error: invalid combination of opcode and operands
40 0000013F 3B0D[0A000000] cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
41 00000145 750F jz ffin ; если 'max(A,C)>B', то переходим из ffin!
```

Рис. 5.4: листинг

6 Задание для самостоятельной работы

Мой вариант 20. Программа нахождения наименьшей из 3 целочисленных переменных а, b и с (рис. 6.1).

```
aymustafina@vbox:~/work/arch-pc/lab07$ nasm -f elf lab-3.asm
nasm: fatal: unable to open input file `lab-3.asm' No such file or directory
aymustafina@vbox:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
aymustafina@vbox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
aymustafina@vbox:~/work/arch-pc/lab07$ ./lab7-3
Введите B: 2
Наименьшее число: 2
aymustafina@vbox:~/work/arch-pc/lab07$
```

Рис. 6.1: Задание 1 для самостоятельной работы

Листинг 7-3 самостоятельная работа

```
%include 'in_out.asm'

SECTION .data
msg1 db 'Введите B: ', 0h
msg2 db 'Наименьшее число: ', 0h
A dd '95'
C dd '61'

SECTION .bss
min resb 10
B resb 10
```

```

SECTION .text
GLOBAL _start
_start:
;----- вывод сообщения о вводе B
mov eax, msg1
call sprint
; ----- вводим B
mov ecx, B
mov edx, 10
call sread
; ----- преобразуем B из символа в чило
mov eax, B
call atoi
mov [B], eax

mov ecx, [A]
mov [min], ecx

cmp ecx, [C] ; сравниваем A и C
jg check_B
mov ecx, [C]
mov [min], ecx

check_B:
mov eax, min
call atoi
mov [min], eax

```

```

mov ecx, [min]
cmp ecx, [B] ;сравниваем max с B
jb fin
mov ecx, [B]
mov [min], ecx

fin:
mov eax, msg2
call sprint
mov eax, [min]
call iprintLF
call quit

```

Программа, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений. К сожалению, мой код не всегда работает корректно и иногда выводит “53” вместо “5”. Найти решение этой проблемы я не смогла (рис. 6.2).

```

aymustafina@vbox:~/work/arch-pc/lab07$ ./lab7-4
Введите значение переменной x: 1
Введите значение переменной a: 2
Результат: 5
aymustafina@vbox:~/work/arch-pc/lab07$ ./lab7-4
Введите значение переменной x: 2
Введите значение переменной a: 1
Результат: 1
aymustafina@vbox:~/work/arch-pc/lab07$ ./lab7-4
Введите значение переменной x: 9
Введите значение переменной a: 4
Результат: 5
aymustafina@vbox:~/work/arch-pc/lab07$ █

```

Рис. 6.2: Задание 2 для самостоятельной работы

Листинг 7-4 самостоятельная работа

```

#include 'in_out.asm'

SECTION .data
msg_x: DB 'Введите значение переменной x: ', 0
msg_a: DB 'Введите значение переменной a: ', 0

```

```
res: DB 'Результат: ', 0
```

```
SECTION .bss
```

```
x: RESB 80
```

```
a: RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov ebx, 5
```

```
mov eax, msg_x
```

```
call sprint
```

```
mov ecx, x
```

```
mov edx, 80
```

```
call sread
```

```
mov eax, x
```

```
call atoi
```

```
mov edi, eax
```

```
mov eax, msg_a
```

```
call sprint
```

```
mov ecx, a
```

```
mov edx, 80
```

```
call sread
```

```
mov eax, a
```

```
call atoi
```

```
mov esi, eax
```

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



```
cmp esi, edi ;сравниваем
```

```
jle sub_v
```

```
mov eax, ebx
```

```
jmp fin
```

```
sub_v:
```

```
mov eax, edi
```

```
sub eax, esi
```

```
fin:
```

```
mov edi, eax
```

```
mov eax, res
```

```
call sprint
```

```
mov eax, edi
```

```
call iprintLF
```

```
call quit
```

7 Выводы

Я изучила команды условного и безусловного перехода и приобрела навыки написания программ с использованием переходов. Ознакомилась с назначением и структурой файла листинга.

8 Список литературы

1. Лабораторная работа №7