Отчёт по лабораторной работе №7

Дисциплина: Архитектура компьютера

Долгаев Евгений Сергеевич НММбд-01-24

Содержание

Список иллюстраций

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
    1) Реализация переходов в NASM
    2) Изучение структуры файлы листинга
2) Задания 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.

3.0.1 Команда безусловного перехода

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

```
jmp <адрес_перехода>
```

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 (табл. 3.1).

Таблица 3.1: Типы операндов инструкции jmp

Тип операнда	Описание
jmp label	переход на метку label
jmp [label]	переход по адресу в памяти, помеченному меткой label
jmp eax	переход по адресу из регистра eax

В следующем примере рассмотрим использование инструкции `jmp`:

```
label:
... ;
... ; команды
... ;
jmp label
```

3.0.2 Команда условного перехода

Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов.

3.0.3 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 (рис. 3.1).

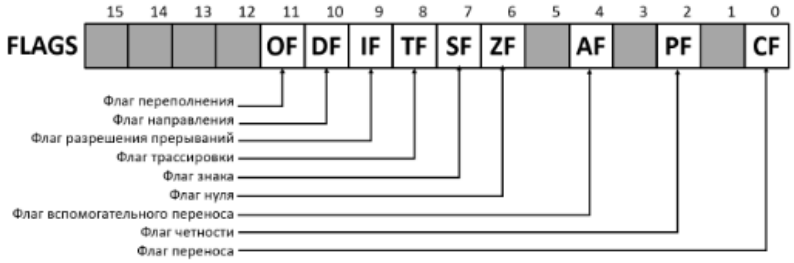

Рис. 3.1: Регистр флагов

Флаги состояния (биты 0, 2, 4, 6, 7 и 11) отражают результат выполнения арифметических инструкций, таких как `ADD`, `SUB`, `MUL`, `DIV`.

Таблица 3.2: Регистр флагов

Бит	Обозначение	Название	Описание
0	CF	Carry Flag - Флаг переноса	Устанавливается в 1, если при выполнении предыдущей операции произошёл перенос из старшего бита или если требуется заём (при вычитании). Иначе установлен в 0.

Бит	Обозначение	Название	Описание
2	PF	Parity Flag - Флаг чётности	Устанавливается в 1, если младший байт результата предыдущей операции содержит чётное количество битов, равных 1.
4	AF	Auxiliary Carry Flag - Вспомогательный флаг переноса	Устанавливается в 1, если в результате предыдущей операции произошёл перенос (или заём) из третьего бита в четвёртый.
6	ZF	Zero Flag - Флаг нуля	Устанавливается 1, если результат предыдущей команды равен 0.

Бит	Обозначение	Название	Описание
7	SF	Sign Flag - Флаг знака	Равен значению старшего значащего бита результата, который является знаковым битом в знаковой арифметике.
11	SF	Overflow Flag - Флаг переполнения	Устанавливается в 1, если целочисленный результат слишком длинный для размещения в целевом операнде (регистре или ячейке памяти).

3.0.4 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 <операнд_1>, <операнд_2>
```

Команда `cmp`, так же как и команда вычитания, выполняет вычитание

<операнд_2> - <операнд_1>, но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов.

Примеры.

```
cmp ax,'4' ; сравнение регистра ax с символом 4
cmp ax,4 ; сравнение регистра ax со значением 4
cmp al,cl ; сравнение регистров al и cl
cmp [buf],ax ; сравнение переменной buf с регистром ax
```

3.0.5 Описание команд условного перехода.

Команда условного перехода имеет вид

```
j<мнемоника перехода> label
```

Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов.

В табл. 3.3 представлены команды условного перехода, которые обычно ставятся после команды сравнения cmp. 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 jnbe). Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы.

Таблица 3.3: Инструкции условной передачи управления по результатам арифметического сравнения cmp a,b

Типы операндов	Мнемокод	Критерий условного перехода a (<,>,<=,>=,=) b	Комментарий
Любые	JE	a = b	ZF = 1

Типы операндов	Мнемокод	Критерий условного перехода a (<,>,<=,>=,=) b	Комментарий
Любые	JNE	`not(a = b)`	ZF = 0
Со знаком	Jl/JNGE	`a < b`	not(SF = OF)
Со знаком	JLE/JNG	`a <= b`	not(SF = OF) или ZF = 1
Со знаком	JG/JNLE	`a > b`	SF = OF и ZF = 0
Со знаком	JGE/JNL	`a >= b`	SF = OF
Без знака	JB/JNAE	`a < b`	CF = 1
Без знака	JBE/JNA	`a <= b`	CF = 1 или ZF = 1
Без знака	JA/JNBE	`a > b`	CF = 0 и ZF = 0
Без знака	JAE/JNB	`a >= b`	CF = 0

Примечание: термины «выше» («a» от англ. «above») и «ниже» («b» от англ. «below») применимы для сравнения беззнаковых величин (адресов), а термины «больше» («g» от англ. «greater») и «меньше» («l» от англ. «lower») используются при учёте знака числа. Таким образом, мнемонику инструкции JA/JNBE можно расшифровать как «*jump if above* (переход если выше) / *jump if not below equal* (переход если не меньше или равно)».

Помимо перечисленных команд условного перехода существуют те, которые которые можно использовать после любых команд, меняющих значения флагов (табл. 3.4).

Таблица 3.4: Инструкции условной передачи управления

Мнемокод	Значение флага для осуществления перехода	Мнемокод	Значение флага для осуществления переход
JZ	ZF = 1	JNZ	ZF = 0
JS	SF = 1	JNS	SF = 0
JC	CF = 1	JNC	CF = 0
JO	OF = 1	JNO	OF = 0
JP	PF = 1	JNP	PF = 0

В качестве примера рассмотрим фрагмент программы, которая выполняет умножение переменных a и b и если произведение превосходит размер байта, передает управление на метку `Error`.

```
mov al, a
mov bl, b
mul bl
jc Error
```

3.1 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
```

Строки в первой части листинга имеют следующую структуру (рис. 3.2).

Рис. 3.2: Структура листинга

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

Создим каталог для программ лабораторной работы № 7(рис. 4.1)

Рис. 4.1: Подготовка рабочего пространства

Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введём в файл `lab7-1.asm` текст программы(рис. 4.2)

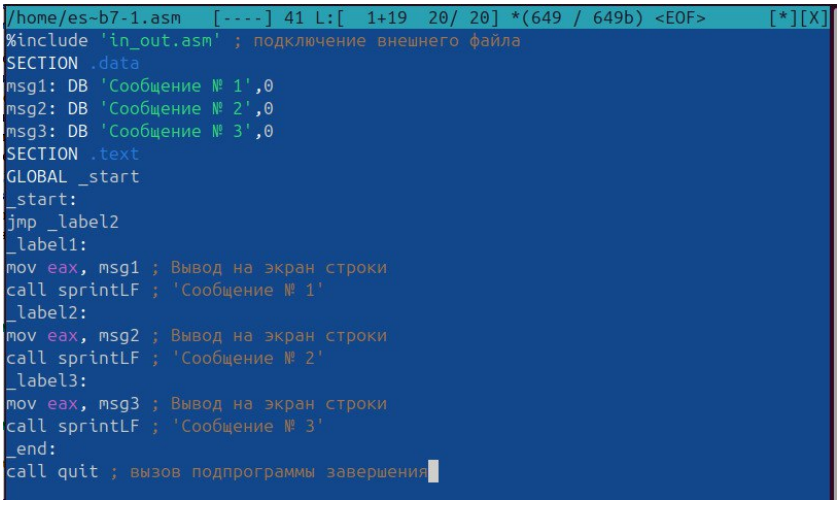

Рис. 4.2: Текст программы

Создим исполняемый файл и запустим его. Результат работы данной программы будет следующим(рис. 4.3):

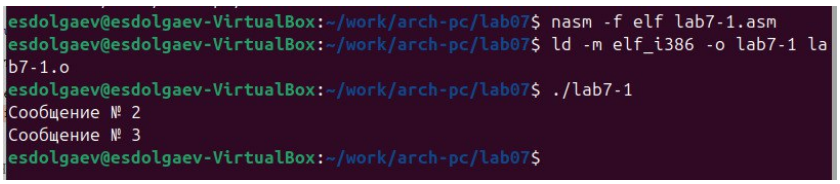

Рис. 4.3: Создание и работа исполняемого файла

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`)(рис. 4.4).

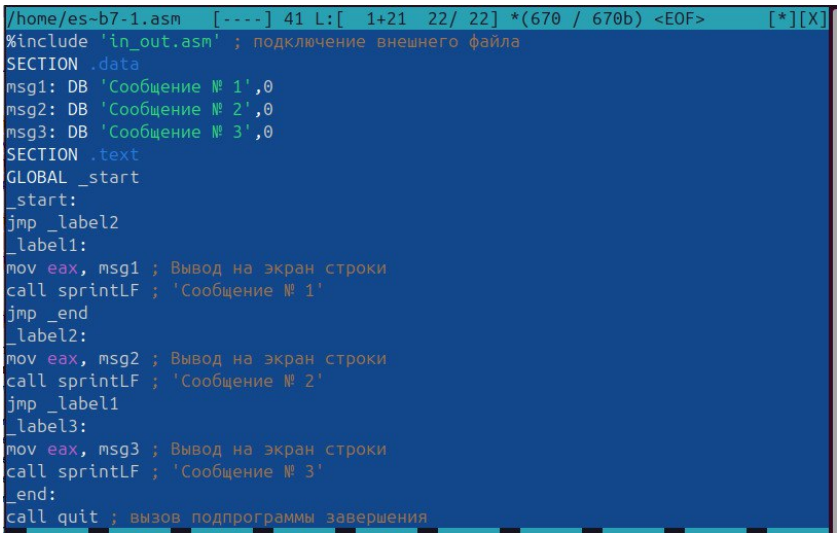

Рис. 4.4: Текст программы

Создим исполняемый файл и проверим его работу(рис. 4.5).

```
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 la
b7-1.o
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Рис. 4.5: Создание и работа исполняемого файла

Изменим текст программы добавив или изменив инструкции jmp, чтобы вывод програм- мы был следующим(рис. 4.6):

```
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

```
/home/es~b7-1.asm   [----] 11 L:[  1+20  21/ 23] *(607 / 682b) 0010 0x00A [*][X]
%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 sprintLF ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 3'
jmp _label2
_end:
```

Рис. 4.6: Текст программы

Создим исполняемый файл и проверим его работу(рис. 4.7).

```
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 la
b7-1.o
esdolgaev@esdolgaev-VirtualBox:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Рис. 4.7: Создание и работа исполняемого файла

Создим файл `lab7-2.asm` в каталоге `~/work/arch-pc/lab07` и введём в него текст программы(рис. 4.8, 4.9).

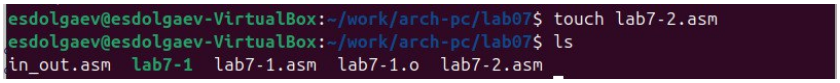

Рис. 4.8: Создание файла

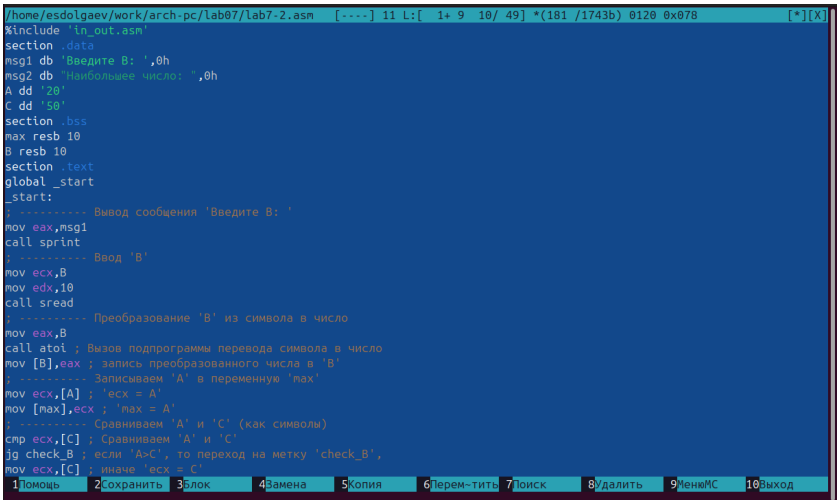

Рис. 4.9: Текст программы

Создим исполняемый файл и проверим его работу(рис. 4.10).

Рис. 4.10: Создание и работа исполняемого файла

Создим файл листинга для программы из файла `lab7-2.asm` и откроем его с помощью любого текстового редактора mcedit(рис. 4.11, 4.12).

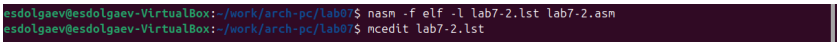

Рис. 4.11: Создание файла

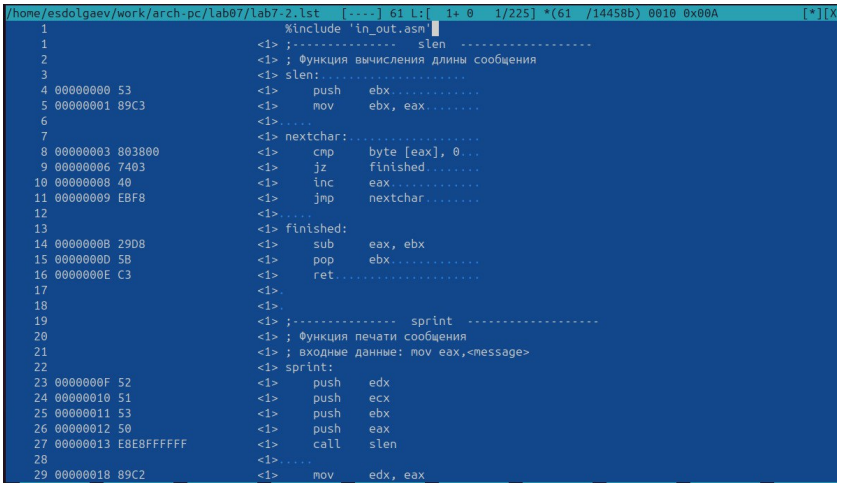

Рис. 4.12: Файл листинга

Откроем файл с программой `lab7-2.asm` и в любой инструкции с двумя операндами удалим один операнд. Выполним трансляцию с получением файла листинга(рис. 4.13, 4.14).

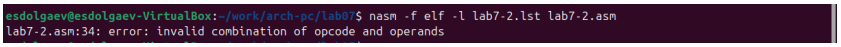

Рис. 4.13: Создание файла с ошибкой

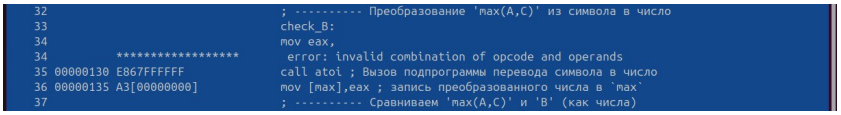

Рис. 4.14: Файл листинга

Файл создался тот же, но в нём появилась пометка об ошибке.

4.1 Задания для самостоятельной работы

Создадим копию файла `lab7-2.asm` и введём в него текст программы которая находит наименьшее из трёх чисел. Создадим исполняемый файл и проверим его работу(рис. 4.13, 4.14).

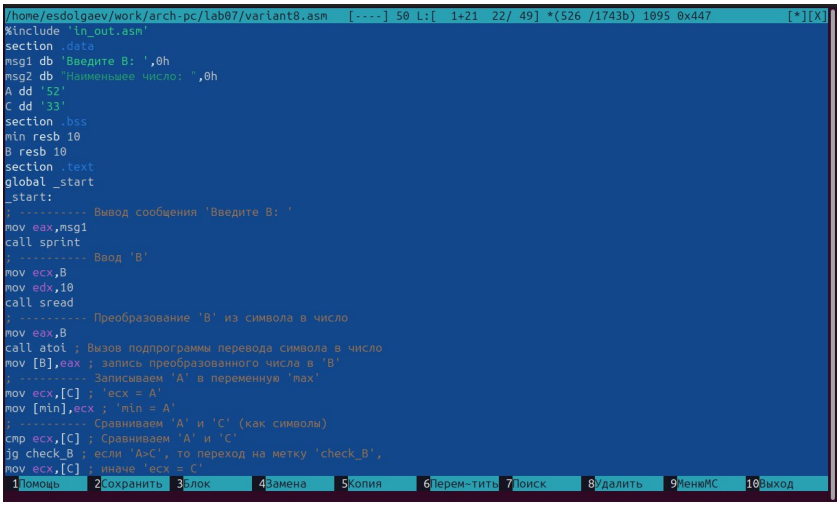

Рис. 4.15: Текст программы

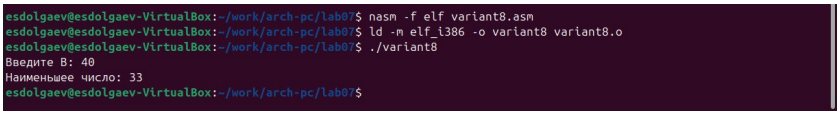

Рис. 4.16: Создание и работа исполняемого файла

Далее напишем программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции *f(x)* и выводит результат вычислений(рис. 4.17, 4.18).

Рис. 4.17: Текст программы

Рис. 4.18: Создание и работа исполняемого файла

5 Выводы

В ходе выполнения лабораторной работы, я изучил команды условного и безусловного переходов, приобрёл навыки написания программ с использованием переходов и познакомился с назначением и структурой файла листинга.

Список литературы