

Отчёта по лабораторной работе №7

**Команды безусловного и условного переходов в Nasm.
Программирование ветвлений**

Четвергова Мария Викторовна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	9
4	Выводы	19

Список иллюстраций

3.1	рис.1 Каталог для программ лабораторной работы No 7	9
3.2	рис.2 Пример программы с использованием инструкции jmp . . .	10
3.3	рис.3 Результат работы данной программы	10
3.4	рис.4 Проверка рабты исполняемого файла	11
3.5	рис.5 Проверка рабты исполняемого файла	11
3.6	рис.6 Листинг в программе	13
3.7	рис.7 Листинг в программе	13
3.8	рис.8 Создание файл листинга для программы из файла lab7-2.asm	14
3.9	рис.9 Листинг lab7-2.lst	14
3.10	рис.10 Трансляция с получением файла листинга	15
3.11	рис.11 программа нахождения наименьшей из 3 целочисленных переменных	16
3.12	рис.13 листинг программы для вычисления значений заданной функции	17
3.13	рис.14 вычисление значений заданной функции в консоли	18

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 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.

##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.

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

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

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

Команда условного перехода имеет вид:

`j label` Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. Представлены команды условного перехода, которые обычно ставятся после команды сравнения `cmp`. В их мнемониках 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.

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 можно использовать после любых команд, меняющих значения флагов

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.

Все ошибки и предупреждения, обнаруженные при ассемблировании, трансля-

тор 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)

3 Выполнение лабораторной работы

Создайте каталог для программ лабораторной работы No 7, перейдите в него и создайте файл lab7-1.asm:

```
mvchetvergova@dk8n73 ~ $  
mvchetvergova@dk8n73 ~ $ mkdir ~/work/arch-pc/lab07  
mvchetvergova@dk8n73 ~ $ cd ~/work/arch-pc/lab07  
mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ touch lab7-1.asm  
mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $
```

Рис. 3.1: рис.1 Каталог для программ лабораторной работы No 7

2. Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введите в файл `lab7-1.asm` текст программы из листинга 7.1.

Листинг 7.1. Программа с использованием инструкции `jmp`

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

```

lab7-1.asm          [-M--] 41 L: [ 1+19 20/ 20] *(643 / 643b) <EOF>
#include "lab_out.asm" ; подключение внешнего файла
SECTION .data
msg1: DB "Сообщение No 1",0
msg2: DB "Сообщение No 2",0
msg3: DB "Сообщение No 3",0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение No 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение No 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение No 3'
_end:
call quit ; вызов подпрограммы завершения

```

Рис. 3.2: рис.2 Пример программы с использованием инструкции jmp

Создайте исполняемый файл и запустите его. Результат работы данной программы будет следующим:

```

mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 3

```

Рис. 3.3: рис.3 Результат работы данной программы

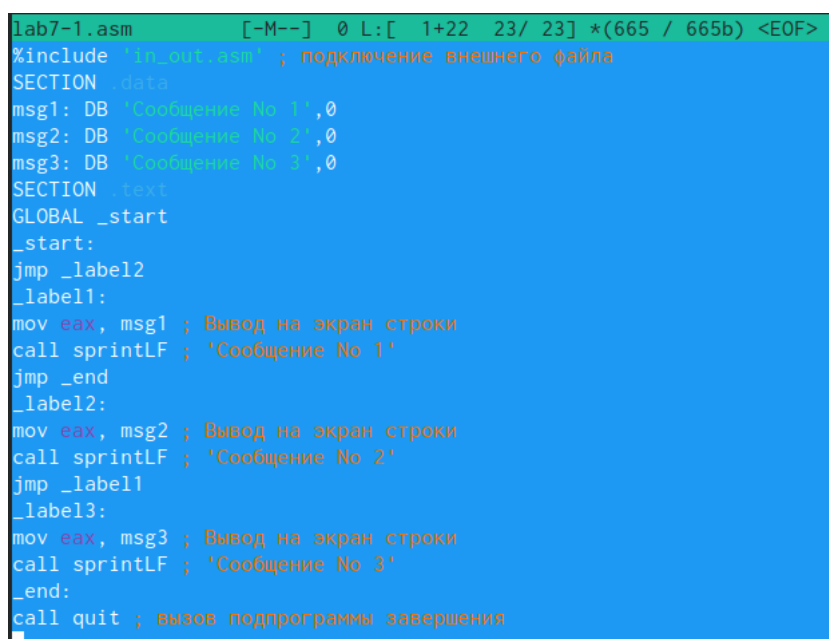
Таким образом, использование инструкции jmp _label2 меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки _label2, пропустив вывод первого сообщения.

Инструкция jmp позволяет осуществлять переходы не только вперед но и назад. Изменим программу таким образом, чтобы она выводила сначала 'Сообщение No 2', потом 'Сообщение No 1' и завершала работу. Для этого в текст программы после вывода сообщения No 2 добавим инструкцию jmp с меткой _label1 (т.е. переход к инструкциям вывода сообщения No 1) и после вывода сообщения No 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

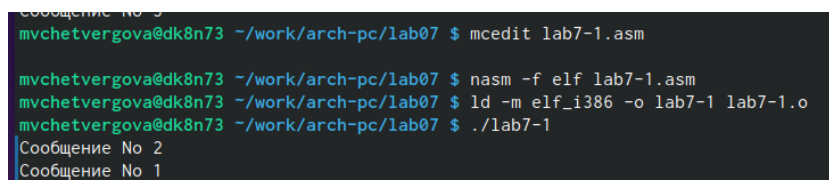
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0
SECTION .text
GLOBAL _start
_start: jmp _label2
_label1: mov eax, msg1 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 1'
jmp _end
_label2: mov eax,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

Создадим исполняемый файл и проверим его работу.



```
lab7-1.asm [-M--] 0 L:[ 1+22 23/ 23] *(665 / 665b) <EOF>
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 3'
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 3.4: рис.4 Проверка работы исполняемого файла



```
mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ mcedit lab7-1.asm
mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 1
```

Рис. 3.5: рис.5 Проверка работы исполняемого файла

Изменим текст программы добавив или изменив инструкции `jmp`, чтобы вывод программы был следующим.

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: A, B и C. Значения для A и C задаются в программе, значение B вводится с клавиатуры

Создайте файл `lab7-2.asm` в каталоге `~/work/arch-pc/lab07`. Внимательно изучите текст программы из листинга 7.3 и введите в `lab7-2.asm`.

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
```

```

lab7-1.asm [----] 11 L:[ 1+20 21/ 24] *(601 / 677b) 0010
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label3
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение No 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение No 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение No 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения

```

Рис. 3.6: рис.6 Листинг в программе

```

mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ mcedit lab7-1.asm
mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
mvchetvergova@dk8n73 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 3
Сообщение No 2
Сообщение No 1

```

Рис. 3.7: рис.7 Листинг в программе

Создайте исполняемый файл и проверьте его работу для разных значений В. Обратите внимание, 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.

##Изучение структуры файлы листинга Обычно nasm создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ -l и задав имя файла листинга в командной строке. Создадим файл листинга для программы из файла lab7-2.asm

Откроем файл листинга lab7-2.lst с помощью любого текстового редактора, например mcedit:

```
mvchetvergova@dk3n37 ~/work/arch-pc/lab07 $
mvchetvergova@dk3n37 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
mvchetvergova@dk3n37 ~/work/arch-pc/lab07 $ mcedit lab7-2.lst
```

Рис. 3.8: рис.8 Создание файл листинга для программы из файла lab7-2.asm

```
lab7-2.lst  [----]  0 L:[ 1+ 0 1/225] *(0 /14458b) 0032 0x020  [*][X]
1          %include 'in_out.asm'
2          <1> ;----- slen -----
3          <1> ; Функция вычисления длины сообщения
4          <1> slen:
5          00000000 53          <1> push    ebx
6          00000001 89C3       <1> mov     ebx, eax
7          <1>
8          00000003 803800     <1> cmp     byte [eax], 0
9          00000006 7403       <1> jz      finished
10         00000008 40          <1> inc     eax
11         00000009 EBF8       <1> jmp     nextchar
12         <1>
13         <1> finished:
14         0000000B 29D8       <1> sub     eax, ebx
15         0000000D 5B          <1> pop     ebx
16         0000000E C3          <1> ret
17         <1>
18         <1>
19         <1> ;----- sprintf -----
20         <1> ; Функция печати сообщения
21         <1> ; входные данные: mov eax,<message>
22         <1> sprintf:
23         0000000F 52          <1> push    edx
24         00000010 51          <1> push    ecx
25         00000011 53          <1> push    ebx
26         00000012 50          <1> push    eax
27         00000013 E8E8FFFF    <1> call    slen
28         <1>
29         00000018 89C2       <1> mov     edx, eax
30         0000001A 58          <1> pop     eax
31         <1>
32         0000001B 89C1       <1> mov     ecx, eax
33         0000001D BB01000000    <1> mov     ebx, 1
34         00000022 B804000000    <1> mov     eax, 4
35         00000027 CD80       <1> int     80h
36         <1>
37         00000029 5B          <1> pop     ebx
38         0000002A 59          <1> pop     ecx
```

Рис. 3.9: рис.9 Листинг lab7-2.lst

Внимательно ознакомимся с его форматом и содержимым. Откроем файл с программой lab7-2.asm и в любой инструкции с двумя операндами удалить один операнд. Выполните трансляцию с получением файла листинга:

```
mvchetvergova@dk3n37 ~/work/arch-pc/lab07 $  
mvchetvergova@dk3n37 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm  
mvchetvergova@dk3n37 ~/work/arch-pc/lab07 $ mcedit lab7-2.lst
```

Рис. 3.10: рис.10 Трансляция с получением файла листинга

Какие выходные файлы создаются в этом случае? Что добавляется в листинге?
Ответ: В листинг добавляются строки в первой части кода. Отображаются не только номер строки и исходный текст программы, но и адрес и машинный код. Таким образом, листинг - это выходной файл текстового типа, имеющий помимо самой программы дополнительную информацию. Адрес - это смещение машинного кода от начала текущего сегмента, а машинный код - это ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция `int 80h` начинается по смещению `00000020` в сегменте кода)

##Задание для самостоятельной работы 1. Напишите программу нахождения наименьшей из 3 целочисленных переменных `x`, `y` и `z`. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы No 7. Создайте исполняемый файл и проверьте его работу.

Вариант 7:

Листинг данной программы:


```

lab7-4.asm [----] 6 L:[ 1+37 38/ 50] *(627 / 769b)
#include 'in_out.asm'
SECTION .data
msg1: DB 'Введите x: ',0
msg2: DB 'Введите a: ',0
msg3 : DB 'f(x) = ',0
section .bss
x resb 10
a resb 10
f resb 10
SECTION .text
GLOBAL _start
_start:
mov eax, msg1
call sprint
mov ecx, x
mov edx, 10
call sread
mov eax, msg2
call sprint
mov ecx, a
mov edx, 10
call sread
;преобразование x из символа в число
mov eax, x
call atoi
mov [x], eax
;преобразование символа в число
mov eax, a
call atoi
mov [a], eax
;сравнение
mov ecx, [x]
cmp ecx, [a]
jne _Label ;если x ≠ a ;либо
jmp fun
_Label:
add ecx, [a]
mov [f], ecx
jmp _exit
fun:
mov eax, [a]
mov ecx, 6

```

Рис. 3.12: рис.13 листинг программы для вычисления значений заданной функции

Результат выполнения данной программы:

```
mvchetvergova@dk4n70 ~/work/arch-pc/lab07 $ mcedit lab7-4.asm

mvchetvergova@dk4n70 ~/work/arch-pc/lab07 $ nasm -f elf lab7-4.asm
mvchetvergova@dk4n70 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-4 lab7-4.o
mvchetvergova@dk4n70 ~/work/arch-pc/lab07 $ ./lab7-4
Введите x: 1
Введите a: 1
6
mvchetvergova@dk4n70 ~/work/arch-pc/lab07 $ nasm -f elf lab7-4.asm
mvchetvergova@dk4n70 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-4 lab7-4.o
mvchetvergova@dk4n70 ~/work/arch-pc/lab07 $ ./lab7-4
Введите x: 2
Введите a: 1
3
```

Рис. 3.13: рис.14 вычисление значений заданной функции в консоли

4 Выводы

В ходе выполнения лабораторной работы №7 мы изучили команды условного и безусловного переходов, а также приобрели навыки написания программ с использованием переходов и ознакомились с назначением и структурой файла листинга. Благодаря проделанной работе получены ценные навыки, позволяющие работать с условным и безусловным оператором на языке Nasm и программирование ветвлений.