Лабораторная работа №7

Команды безусловного и условного переходов в Nasm. Программирование ветвлений.

Сырцева Анастасия Романовна

Содержание

Список иллюстраций

1 Цель работы

Целью данной работы является изучение команд перезодов, а также приобритение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

2 Задание

- Работа с командами переходов;
- Знакомство с назначением и структурой файла листинга;

3 Теоретическое введение

3.1 Команды условного и безусловного перехода

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.

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

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. В следующей таблице указано положение битовых флагов в регистре флагов (рис. 3.1).

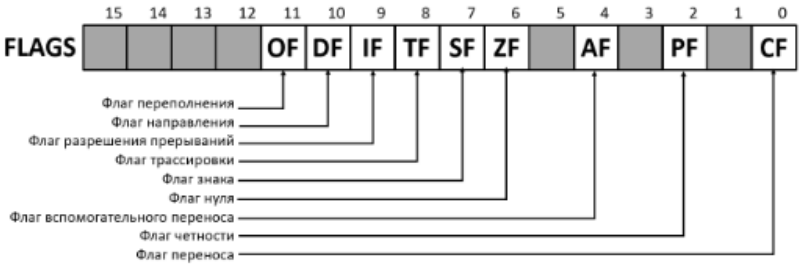

Рис. 3.1: Регистр флагов

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 вычитании.

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

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

В табл. 3.1 представлены команды условного перехода, которые обычно ставятся после команды сравнения cmp. В их мнемокодах указывается тот результат сравнения, при котором надо делать переход.

Таблица 3.1: Инструкции условной передачи управления по результатам арифметического сравнения cmp a,b

Типы операндов	Мнемокод	Критерий условного перехода	Комментарий
Любые	JE	a = b	Переход, если равно
Любые	JNE	a ≠ b	Переход, если не равно
Со знаком	JL/JNGE	a < b	Переход, если меньше

Типы операндов	Мнемокод	Критерий условного перехода	Комментарий
Со знаком	JLE/JNG	a ≤ b	Переход, если меньше или равно
Со знаком	JG/JNLE	a > b	Переход, если больше
Со знаком	JGE/JNL	a ≥ b	Переход, если больше или равно
Без знака	JB/JNAE	a < b	Переход, если ниже
Без знака	JBE/JNA	a ≤ b	Переход, если ниже или равно
Без знака	JA/JNBE	a > b	Переход, если выше
Без знака	JAE/JNB	a ≥ b	Переход, если выше или равно

3.2 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.

Строки листинга имеют следующую структуру:

- 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);

- Арес — это смещение машинного кода от начала текущего сегмента;
- Машинный код - ассемблированная исходная строка в виде шестнадцатеричной последовательности.
- 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).

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

4 Выполнение лабораторной работы

Создаю папку для данной лабораторной работы, переходу в неё и создаю файл lab7-1.asm (рис. 4.1).

```
arsihrceva@dk8n74 ~ $ mkdir ~/work/arch-pc/lab07
arsihrceva@dk8n74 ~ $ cd ~/work/arch-pc/lab07
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ touch lab7-1.asm
```

Рис. 4.1: Создание необходимого каталога и файла

Открываю файл и ввожу в файл текст программы с использованием новой инструкции, jmp(рис. 4.2).

```
%include 'in_out.asm'

        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
                        call sprintLF

                _label2:
                        mov eax, msg2
                        call sprintLF

                _label3:
                        mov eax, msg3
                        call sprintLF

                _end:
                        call quit
```

Рис. 4.2: Текст программы с использованием jmp

Копирую файл in_out.asm для корректной работы программы (рис. 4.3).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ cp ~/work/in_out.asm ~/work/arch-pc/lab07/in_out.asm
```

Рис. 4.3: Копирование in_out.asm в рабочий каталог

Создаю и запускаю исполняемый файл(рис. 4.4).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-1
```

Рис. 4.4: Запуск исполняемого файла

Результат работы программы совпадает с тем, что дано в условии лабораторной работы(рис. 4.5).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 3
```

Рис. 4.5: Результат работы программы

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

Изменяю программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу.(рис. 4.6).

```
%include 'in_out.asm'

        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
                        call sprintLF
                        jmp _end ; переход к call quit

                _label2:
                        mov eax, msg2
                        call sprintLF
                        jmp _label1 ; переход к label1

                _label3:
                        mov eax, msg3
                        call sprintLF

                _end:
                        call quit
```

Рис. 4.6: Изменённый текст программы

Создаю и запускаю исполняемый файл(рис. 4.7).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Рис. 4.7: Результат работы измененной программы

Результат удовлетворяет условию. Сначала выводится ‘Сообщение № 2’, потом ‘Сообщение № 1’, затем работа завершается.

Изменяю текст программы, используя jmp, таким образом, чтобы сообщения выводились в следующем порядке: ‘Сообщение №3’, ‘Сообщение № 2’, ‘Сообщение № 1’, (рис. 4.8).

```
%include 'in_out.asm'

        SECTION .data
        msg1: DB 'Сообщение № 1',0
        msg2: DB 'Сообщение № 2',0
        msg3: DB 'Сообщение № 3',0

        SECTION .text
        GLOBAL _start
                _start:

                jmp _label3 ; переход к _label3

                _label1:
                        mov eax, msg1
                        call sprintLF
                        jmp _end ; переход к call quit

                _label2:
                        mov eax, msg2
                        call sprintLF
                        jmp _label1 ; переход к label1

                _label3:
                        mov eax, msg3
                        call sprintLF
                        jmp _label2 ; переход к _label2

                _end:
                        call quit
```

Рис. 4.8: Изменений текст программы

Создаю исполняемый файл и запускаю его(рис. 4.9).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Рис. 4.9: Запуск файла и результат работы программы

Создаю файл lab7-2.asm и открываю его для редактирования(рис. 4.10).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ touch lab7-2.asm
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ gedit lab7-2.asm
```

Рис. 4.10: Создание файла lab7-2.asm

Ввожу текст программы, которая выводит наибольшее из трёх чисел(рис. 4.11).

```
%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:

        mov eax,msg1
        call sprint

        mov ecx,B
        mov edx,10
        call sread

        mov eax,B
        call atoi
        mov [B],eax
        mov ecx,[A]
        mov [max],ecx

        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'

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

        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

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

Рис. 4.11: Текст программы

Создаю и несколько раз запускаю исполняемый файл. На запрос программы ввожу различные числа(рис. 4.12).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 7
Наибольшее число: 50
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 13
Наибольшее число: 50
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 67
Наибольшее число: 67
```

Рис. 4.12: Результат работы программы при различных B

Создаю файл листинга для программы из файла lab7-2.asm(рис. 4.13).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
```

Рис. 4.13: Создание файла листинга

Открываю файл листинг в редакторе gedit(рис. 4.14).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ gedit lab7-2.lst
```

Рис. 4.14: Открытие файла листинга в редакторе

Внимательно изучаю текст программы.

Подробно разбираю строку под номером 197 (рис. 4.15).

- ‘22’ - номер данной строки в исходном файле ;
- ‘00000106’ - адрес строки;
- ‘E891FFFFFF’ - машинный код;
- ‘call atoi’ - вызов подпрограммы перевода символа в число из исходного файла

```
197     22 00000106 E891FFFFFF                  call atoi
```

Рис. 4.15: Строка 197 из файла листинга

Также разберу строку под номером 213(рис. 4.16).

- ‘38’ - номер данной строки в исходном файле ;
- ‘00000145’ - адрес строки;
- ‘3B0D[0A000000]’ - машинный код;
- ‘cmp ecx,[B] ; Сравниваем ’max(A,C)’ и ‘B’ ’ - строка из исходного файла, где ‘cmp ecx,[B]’ - команда для сравнения операнд ecx и [B], а ‘; Сравниваем ’max(A,C)’ и ‘B’ ’ - комментарий

```
213     38 00000145 3B0D[0A000000]              cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
```

Рис. 4.16: Строка 213 листинга

Делаю то же самое для строки 220(рис. 4.17).

- ‘45’ - номер данной строки в исходном файле ;
- ‘0000015E’ - адрес строки;
- ‘E8ACFEFFFF’ - машинный код;
- ‘call sprint’ ’ - строка из исходного файла, вывод сообщения ‘Наибольшее число:’

```
220     45 0000015E E8ACFEFFFF                  call sprint
```

Рис. 4.17: Строка 220 листинга

Открываю файл lab7-2.asm и в строке 35 удаляю операнд eax(рис. 4.18).

```
32 check_B:
33         mov eax,max
34         call atoi
35         mov [max] ; удален операнд eax
36
```

Рис. 4.18: Изменённая строка программы

Создаю файл листинга(рис. 4.19).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:35: error: invalid combination of opcode and operands
```

Рис. 4.19: Создание листинга

Выдаётся ошибка: invalid combination of opcode and operands.

Открываю файл lab7-2.lst(рис. 4.20).

```
210     35                                      mov [max] ; удален операнд eax
211     35          ******************      error: invalid combination of opcode and operands
```

Рис. 4.20: Файл изменённого листинга

В файле показано конкретное место, где есть ошибка(35 строка) и указано название ошибки(invalid combination of opcode and operands)

5 Самостоятельная работа

5.1 Первое задание

Создаю и открываю для редактирования файл lab7-3.asm(рис. 5.1).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ touch lab7-3.asm
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ gedit lab7-3.asm
```

Рис. 5.1: Создание файла для самостоятельной работы

Пишу текст программы для нахождения наименьшей из 3 целочисленных переменных a, b и c.(рис. 5.2).

```
%include 'in_out.asm'

section .data
        msg1 db 'Введите A: ',0h
        msg2 db 'Введите B: ',0h
        msg3 db 'Введите C: ',0h
        msg4 db "Наименьшее число: ",0h
section .bss
        min resb 10
        A resb 10
        B resb 10
        C resb 10
section .text
        global _start
_start:
; ---------- Вывод сообщения A
        mov eax,msg1
        call sprint
; ---------- Ввод A
        mov ecx,A
        mov edx,10
        call sread
; ---------- Перевод A в число
        mov eax, A
        call atoi
        mov [A],eax
; ---------- Вывод сообщения B
        mov eax,msg2
        call sprint
; ---------- Ввод B
        mov ecx,B
        mov edx,10
        call sread
; ---------- Перевод B в число
        mov eax, B
        call atoi
        mov [B],eax
; ---------- Записываем A в мин
        mov ecx,[A]
        mov [min],ecx ; A мин
        mov ecx,[min]
; ---------- Сравниваем A и B как символы
        cmp ecx,[B] ; сравниваем A и B
        jl check_C ; если A меньше B переходим к check_C
        mov ecx,[B] ; иначе B в ecx
        mov [min],ecx ; B мин

check_C:
; ---------- Вывод сообщения C
```

Рис. 5.2: Текст программы для нахождения наименьшего числа

Нахожу номер варианта, соответствующий варианту из лабораторной №6. A

именно первый вариант (рис. 5.3).

Номер варианта	Значения a, b, c
1	17,23,45

Рис. 5.3: Мой вариант

Создаю и запускаю исполняемый файл. На запросы программы ввожу числа из 1 варианта (рис. 5.4).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ nasm -f elf lab7-3.asm
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-3 lab7-3.o
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-3
Введите A: 17
Введите B: 23
Введите C: 45
Наименьшее число: 17
```

Рис. 5.4: Результат программы нахождения наименьшего числа

В ответе выводится число 17, наименьшее из трех введённых.

5.2 Второе задание

Создаю и открываю в текстовом редакторе файл lab7-4 (рис. 5.5).

```
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ touch lab7-4.asm
arsihrceva@dk8n74 ~/work/arch-pc/lab07 $ gedit lab7-4.asm
```

Рис. 5.5: Создание файла для второго задания

Нахожу 1 вариант для выполнения задания (рис. 5.6).

Номер варианта	Выражение для $f(x)$	(x_1, a_1)	(x_2, a_2)
1	$\begin{cases} 2a - x, & x < a \\ 8, & x \geq a \end{cases}$	(1;2)	(2;1)

Рис. 5.6: Уравнение для второго задания

Ввожу текст программы, которая будет вычислять значение функции f для введённых переменных a и x(рис. 5.7).

```
%include 'in_out.asm'

section .data
        msg1 db 'Введите значение переменной x: ',0h
        msg2 db 'Введите значение переменной a: ',0h
        msg3 db 'Ответ: ',0h

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,X
        call atoi
        mov [X],eax

        mov eax,msg2
        call sprint

        mov ecx,A
        mov edx,10
        call sread

        mov eax,A
        call atoi
        mov [A],eax

        mov ecx,[X]
        cmp ecx,[A]
        jl Uravnenie
        mov ecx,8
        mov [F],ecx
        jmp fin

        Uravnenie:
        mov eax,[A]
        mov ebx,2
```

Рис. 5.7: Текст программы для вычисления значения функции

Создаю и запускаю исполняемый файл. На запрос программы ввожу значения x=1, a=2(рис. 5.8).

```
arsihrceva@dk1n22 ~/work/arch-pc/lab07 $ nasm -f elf lab7-4.asm
arsihrceva@dk1n22 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-4 lab7-4.o
arsihrceva@dk1n22 ~/work/arch-pc/lab07 $ ./lab7-4
Введите значение переменной x: 1
Введите значение переменной a: 2
Ответ: 3
```

Рис. 5.8: Результат работы программы в первом случае

Запускаю файл ещё раз и ввожу новые значения переменных(рис. 5.9).

```
arsihrceva@dk1n22 ~/work/arch-pc/lab07 $ ./lab7-4
Введите значение переменной x: 2
Введите значение переменной a: 1
Ответ: 8
```

Рис. 5.9: Результат работы программы во втором случае

6 Выводы

Изучены команды условного и безусловного переходов. Получены навыки написания программ с использованием переходов. Также изучена структура и назначение файла листига.