Лабораторная работа ;7

Команды безусловного и условного переходов в Nasm. Программирование ветвлений.

Карпова Есения Алексеевна

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. Реализация перехода в NASM
2. Изучение структуры файлы листинга
3. 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

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

Команда условного перехода имеет вид j label

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 ,

Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, созда- ваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию. # Выполнение лабораторной работы

1. Реализация перехода в NASM

1) Создаю каталог для программы лабораторной работы №7, перейдя в него создаю файл lab7-1.asm (рис. 3.1).

```
eakarpova@dk5n60 ~ $ mkdir ~/work/arch-pc/lab07
eakarpova@dk5n60 ~ $ cd ~/work/arch-pc/lab07
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ touch lab7-1.asm
```

Рис. 3.1: Создание каталога и файла

2) Ввожу в файл lab7-1.asm текст программы из листинга 7.1 и создаю исполняемый файл (рис. 3.2).

```
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 3
```

Рис. 3.2: Создание исполняемого файла

На выходе получаю сообщения №2 и №3, так как с помощью инструкции jmp _label2 изменился порядок исполнения инструкций, и выполнение началось с метки _label2, пропустив вывод первого сообщения

3) Изменив текст программы согласно листингу 7.2 создаю исполняемый файл и проверяю его работу.(рис. 3.3).

```
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Рис. 3.3: Измененный вывод

Вывод изменился: теперь пользователь видит сообщения №2 и №1

4) Изменяю текст программы так, чтобы вывод программы был следующим: Сообщение №3, Сообщеие №2, Сообщеение №1 (рис. 3.4).

```
 GNU nano 6.4    /afs/.dk.sci.pfu.edu.ru/home/e/a/eakarpova/work/arch-pc/lab07/lab7-1.asm
%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:
call quit ; вызов подпрограммы завершения
```

Рис. 3.4: Измененние текста программы

5) Проверяю - действительно, сообщения выходят в указанном порядке (рис. 3.5).

```
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Рис. 3.5: Измененный вывод

6) С помощью утилиты touch создаю новый исполняемый файл lab7-2.asm и ввожу в него листинг 7. - программу, которая определяет и выводит на экран

наибольшую из 3 целочисленных переменныъ A, B, C. Создаю исполняемый файл и проверяю его работу на случайных значений B(рис. 3.6).

```
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 10
Наибольшее число: 50
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 100000000
Наибольшее число: 100000000
```

Рис. 3.6: Проверка работы файла

Убеждаюсь, что программа работает правильно

2. Изучение структуры файлы листинга

1) Создаю файл листинга lab7-2.lst с помощью текстового редактора mcedit (рис. 3.7).

```
eakarpova@dk4n62 ~/work/arch-pc/lab07 $ mcedit lab7-2.lst
```

Рис. 3.7: Создание каталога и файла

1.1) Объяснение трех строк на выбор: Строка 28: 00000117 - адрес, 3B0D[39000000] - машинный код, cmp ecx,[C] - код программы - происходит сравнение A и C с помощью инстирукции cmp Строка 29: 0000011D - адрес, 7F0C - машинный код, jg check_B - код программы, с помощью мнемокода jg происходит условный переход - если A > C, то выполнение программы переходит на check_B Строка 30: 0000011F - адрес, 8B0D[39000000] - машинный код, mov ecx,[C] - код программы - при невыполнении A > C, т.е. иначе C запишется в ecx (рис. 3.8).

```
  28 00000117 3B0D[39000000]          cmp ecx,[C]; Сравниваем 'A' и 'C'
  29 0000011D 7F0C                    jg check_B; если 'A>C', то переход на метку 'check_B',
  30 0000011F 8B0D[39000000]          mov ecx,[C]; иначе 'ecx = C'
```

Рис. 3.8: Объяснение файла листинга

2) Удаляю операнд msg1 (рис. 3.9).

```
_start:
; ---------- Вывод сообщения 'Введите B: '
mov eax
call sprint
; ---------- Ввод 'B'
```

Рис. 3.9: Удаление операнда

Выполняю трансляцию с получением файла листинга, но выходные файлы не созздаются, так как программа выдает ошибку (рис. 3.10).

```
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:18: error: invalid combination of opcode and operands
```

Рис. 3.10: Ошибка

В листинге в той же строке, где была намерено допущена ошибка, появляется сообщение о ней (рис. 3.11).

```
   9 0000000A <res Ah>              B resb 10
  10                                section        .text
  11                                global _start
  12                                _start:
  13                                ; ---------- Вывод сообщения 'Введите B: '
  14                                mov eax
  14                       ****************** error: invalid combination of opcode and operands
  15 000000E8 E822FFFFFF            call sprint
  16                                ; ---------- Ввод 'B'
```

Рис. 3.11: Изменение в листинге

3. Задания для самостоятельной работы

1) Написала программу нахождения наименьшей из 3 целочисленных переменных a, b и c (рис. 3.12).

```
 GNU nano 6.4
%include         'in_out.asm'
section         .data
msg1 db 'Введите B: ',0h
msg2 db "Наименьшее число: ",0h
A dd '24'
B dd '98'
C dd '15'
section .bss
min resb 10
section         .text
global _start
_start:
; ---------- Записываем 'A' в переменную 'min'
mov ecx, [A]
mov [min],ecx ; 'min = 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 [min],ecx ; 'min' = C'

check_B:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа в `min`
; ---------- Сравниваем 'min(A,C)' и 'B' (как числа)
mov ecx,[min]
cmp [B], ecx; Сравниваем 'min(A,C)' и 'B'
jg fin; если 'min(A,C)<B', то переход на 'fin',
mov ecx,[B]; иначе 'ecx = B'
mov[min],ecx
; ---------- Вывод результата
fin:
mov eax, msg2
call sprint
mov eax,[min]
call iprintLF; Вывод 'min(A,B,C)'
call quit; Выход
```

Рис. 3.12: Программа нахождения наименьшей из 3 целочисленных переменных

Моим вариант был №9 с числами 24, 98 и 15. Создаю исполняемый файл для проверки работы программы (рис. 3.13).

```
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ nasm -f elf lab7-3.asm
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-3 lab7-3.o
eakarpova@dk5n60 ~/work/arch-pc/lab07 $ ./lab7-3
Наименьшее число: 15
```

Рис. 3.13: Проверка работы программы

Программа работает корректно. 2) Написала программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции f(x) и выводит результат вычислений.(рис. **??**). (рис. 3.15).

```
%include        'in_out.asm'

SECTION .data
msg1: DB 'Введите значение переменной x: ', 0h
msg2: DB 'Введите значение переменной a: ', 0h
rem: DB 'Результат: ', 0h

SECTION .bss
x:      RESB 10
a:      RESB 10

SECTION .text
GLOBAL _start
_start:

; ---- Ввод  x
mov eax, msg1
call sprint
mov ecx, x
mov edx, 10
call sread

; ---- Преобразование x из символа в число
mov eax, x
call atoi
mov [x], eax

; ---- Ввод a
mov eax, msg2
call sprint
mov ecx, a
mov edx, 10
call sread

; ---- Преобразование a из символа в число
mov eax, a
call atoi
mov [a], eax
```

Рис. 3.14: Программа вычисления заданной функции

```
; ---- Сравниваем 'x' и 'a'
mov ebx, [x]
mov edx, [a]
cmp edx, ebx ; сравниваем x и a
jge check_X ; если x >= a
jmp _end ; иначе

; ---- Если x <= a
check_X:
add edx, ebx  ; a + x
jmp _end

; ---- Вывод результата на экран
_end:
mov eax, rem
call sprint
mov eax, edx
call iprintLF
call quit
```

Рис. 3.15: Программа вычисления заданной функции

Создаю исполняемый файл, чтобы убедиться в корректности работы программы (рис. **??**).

Создание исплолняемого файла

Рис. 3.16: Создание исплолняемого файла

4 Выводы

Я изучила команды условного и безусловного переходов. Приобрела навыки написания программ с использованием переходов. Познакомилась с назначением и структурой файла листинга.

Список литературы

Лабораторная работа №7. Команды безусловного и условного переходов в Nasm. Программирование ветвлений.- А.В. Демидова