Лабораторнрая работа №7

Команды безусловного и условного переходов в Nasm. Программирование ветвлений.

Соловьев Богдан Михайлович НКАбд-05-23

Содержание

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.

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

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 jnbe). Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы.

3 Выполнение лабораторной работы

Создаю файл lab7-1.asm (рис. 1).

```
[bmsolovjev@fedora ~]$ mkdir ~/work/arch-pc/lab07
[bmsolovjev@fedora ~]$ cd ~/wort/arch-pc/lab07
bash: cd: /home/bmsolovjev/wort/arch-pc/lab07: Нет такого файла или каталога
[bmsolovjev@fedora ~]$ cd ~/work/arch-pc/lab07
[bmsolovjev@fedora lab07]$ touch lab7-1.asm
[bmsolovjev@fedora lab07]$
```

Figure 1: Создание файла

Ввожу в файл lab7-1.asm код программы с использованием jmp (рис. 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 _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

```
[bmsolovjev@fedora lab07]$ nasm -f elf lab7-1.asm
[bmsolovjev@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[bmsolovjev@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 3
[bmsolovjev@fedora lab07]$
```

Figure 2: Работа программы с функцией jmp

Изменение файла lab7-1.asm, после которого программа выводит сначала сообщение 2, потом сообщение 1 (рис. 3).

```
%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 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'
_end:
call quit ; вызов подпрограммы завершения
```

```
[bmsolovjev@fedora lab07]$ nasm -f elf lab7-1.asm
[bmsolovjev@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[bmsolovjev@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 1
[bmsolovjev@fedora lab07]$
```

Figure 3: Другая программа с функцией jmp

Теперь изменяю файл lab7-1.asm таким образом, чтобы она выводил все сообщения (рис. 4).

```
%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
```

Figure 4: Код, выводящий все сообщения

Проверка работы моего кода (рис. 5).

```
[bmsolovjev@fedora lab07]$ nasm -f elf lab7-1.asm
[bmsolovjev@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[bmsolovjev@fedora lab07]$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
[bmsolovjev@fedora lab07]$
```

Figure 5: Проверка работы моего кода

Код программы, сравнивающей 3 числа (рис. 6).

```
_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)'
```

Figure 6: Код, сравнивающий 3 числа

Проверка работы кода, сравнивающего 3 числа (рис. 7).

```
[bmsolovjev@fedora lab07]$ ./lab7-2
Введите B: 6
Наибольшее число: 50
[bmsolovjev@fedora lab07]$ ./lab7-2
Введите B: 89
Наибольшее число: 89
[bmsolovjev@fedora lab07]$
```

Figure 7: Проверка кода

Создаю файл листинга и открываю его с помощью текстового редактора. 14 строка вычитает из значения, находящегося в регистре eax, значение, находящееся в регистре ebx. 15 строка берёт из стека значение, адрес которого находится в регистре ebx. 16 строка выполняет возврат из процедуры (рис. 7).

```
 7                                  <1> nextchar:
 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> ;---------------  sprint  -------------------
20                                  <1> ; Функция печати сообщения
21                                  <1> ; входные данные: mov eax,<message>
22                                  <1> sprint:
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 E8E8FFFFFF              <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
39 0000002B 5A                      <1>     pop     edx
40 0000002C C3                      <1>     ret
41                                  <1>
42                                  <1>
43                                  <1> ;----------------  sprintLF  ----------------
```

Figure 8: Файл листинга

Удаляю один операнд из инструкции mov и получаю ошибку (рис. 9).

```
%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] УДАЛИЛ ТУТ
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
```

```
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, ; УДАЛИЛ ТУТ
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)
```

Figure 9: Удаление операнда

4 Самостоятельная работа

4.1 1 задание

Мой код для программы, находящей наименьшую из 3 переменных 94б 5б 58 (рис. 10).

```
%include 'in_out.asm'
section .data
msg2 db "Наименьшее из трёх чисел:",0h
A dd '94'
B dd '5'
C dd '58'
section .bss
min resb 10
```

```
section .text
global _start
_start:
mov eax,C
call atoi ; Вызов подпрограммы перевода символа в число
mov [C],eax ; запись преобразованного числа в 'C'
; ---------- Записываем 'A' в переменную 'min'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'max = A'
; ---------- Сравниваем 'A' и 'B' (как символы)
cmp ecx,[B] ; Сравниваем 'A' и 'B'
jl check_C ; если 'A<B', то переход на метку 'check_C',
mov ecx,[B] ; иначе 'ecx = B'
mov [min],ecx ; 'min = B'
; ---------- Преобразование 'max(A,C)' из символа в число
check_C:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа в min
; ---------- Сравниваем 'min(A,B)' и 'C' (как числа)
mov ecx,[min]
cmp ecx,[C] ; Сравниваем 'min(A,B)' и 'C'
jl fin ; если 'min(A,B)<C', то переход на 'fin',
mov ecx,[C] ; иначе 'ecx = C'
mov [min],ecx
; ---------- Вывод результата
fin:
mov eax, msg2
call sprint ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход
```

```
%include 'in_out.asm'
section .data
msg2 db "Наименьшее из трёх чисел:",0h
A dd '94'
B dd '5'
C dd '58'
section .bss
min resb 10
section .text
global _start
_start:
mov eax,C
call atoi ; Вызов подпрограммы перевода символа в
mov [C],eax ; запись преобразованного числа в 'C'
; ---------- Записываем 'A' в переменную 'min'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'max = A'
; ---------- Сравниваем 'A' и 'B' (как символы)
cmp ecx,[B] ; Сравниваем 'A' и 'B'
jl check_C ; если 'A<B', то переход на метку 'chec
mov ecx,[B] ; иначе 'ecx = B'
mov [min],ecx ; 'min = B'
; ---------- Преобразование 'max(A,C)' из символа
check_C:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в
mov [min],eax ; запись преобразованного числа в mi
; ---------- Сравниваем 'min(A,B)' и 'C' (как числ
mov ecx,[min]
cmp ecx,[C] ; Сравниваем 'min(A,B)' и 'C'
jl fin ; если 'min(A,B)<C', то переход на 'fin',
```

Figure 10: Код программы, находящей min

Результат работы моего кода (рис. 11).

```
[bmsolovjev@fedora lab07]$ nasm -f elf zadanie_1.asm
[bmsolovjev@fedora lab07]$ ld -m elf_i386 -o zadanie_1  zadanie_1.o
[bmsolovjev@fedora lab07]$ ./zadanie_1
Наименьшее из трёх чисел:5
[bmsolovjev@fedora lab07]$
```

Figure 11: Проверка кода

4.2 2 задание

Мой вариант - 3, поэтому я написал код, который запрашивает 2 числа, и если первое не равно 3, то выводит второе число + 1, а если равно 3, то умножает первое число на 3(рис. 12).

```
%include 'in_out.asm'
section .data
msg1 db 'Введите x: ',0h
msg2 db 'Введите a: ',0h
section .bss
x resb 10
a resb 10
n resb 10
section .text
global _start
_start:
; ---------- Вывод сообщения 'Введите x: '
mov eax,msg1
call sprint
; ---------- Ввод 'x'
mov ecx,x
mov edx, 80
call sread
; ---------- Преобразование 'x' из символа в число
mov eax,x
call atoi ; Вызов подпрограммы перевода символа в число
mov [x],eax ; запись преобразованного числа в 'x'

; ---------- Вывод сообщения 'Введите a: '
mov eax,msg2
call sprint
; ---------- Ввод 'a'
```

```
mov ecx,a
call sread
; ---------- Преобразование 'a' из символа в число
mov eax,a
call atoi ; Вызов подпрограммы перевода символа в число
mov [a],eax ; запись преобразованного числа в 'a'

; --------- Функция
mov eax, 3
cmp eax, [x]
jne fin
jmp fin1

; ---------- Вывод результата
fin1:
mov eax, [a]
add eax, 1
mov [a],eax
mov eax, [a] ;
call iprintLF ;

call quit ; Выход

fin:
mov eax, [x]
mov ebx, 3
mul ebx
mov [x],eax
mov eax, [x] ;
call iprintLF ;

call quit ; Выход
```

```
; ---------- Вывод сообщения 'Введите a: '
mov eax,msg2
call sprint
; ---------- Ввод 'a'
mov ecx,a
call sread
; ---------- Преобразование 'a' из символа в число
mov eax,a
call atoi ; Вызов подпрограммы перевода символа в число
mov [a],eax ; запись преобразованного числа в 'a'

; --------- Функция
mov eax, 3
cmp eax, [x]
jne fin
jmp fin1

; ---------- Вывод результата
fin1:
mov eax, [a]
add eax, 1
mov [a],eax
mov eax, [a] ;
call iprintLF ;

call quit ; Выход

fin:
mov eax, [x]
mov ebx, 3
mul ebx
mov [x],eax
mov eax, [x] ;
call iprintLF ;
```

Figure 12: Код моей программы

Проверка работы кода (рис. 13).

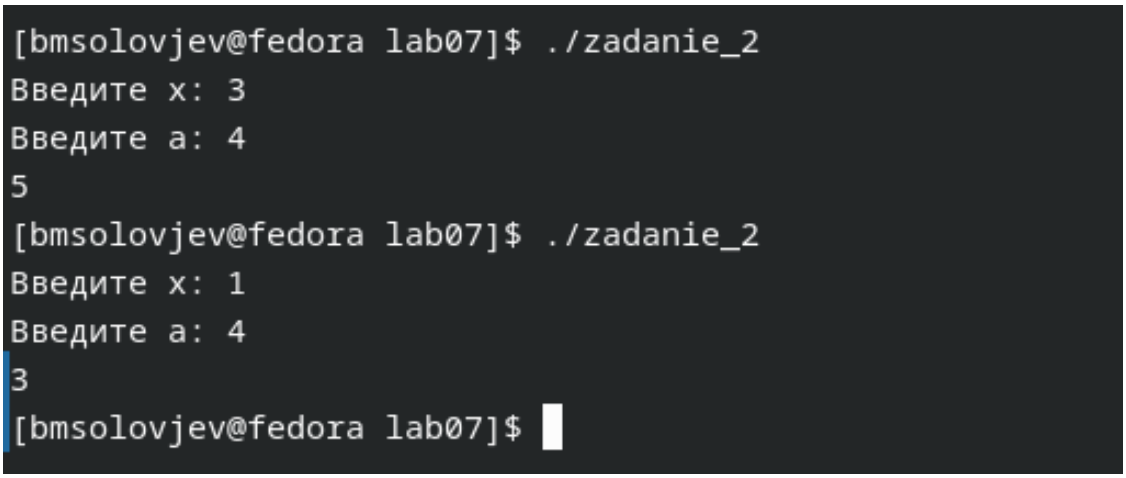

Figure 13: Проверка кода

5 Выводы

Выполнив данную лабораторную работу, я изучил команды условного и безусловного перехода. Приобрёл навыки написания программ с использованием переходов. Познакомился с назначением и структурой файла листинга.

Список литературы