РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

по лабораторной работе № 7

дисциплина: Архитектура компьютера

Студент: Бессонов Андрей Максимович

Группа: НКАбд - 01 - 25

МОСКВА

2025 г.

Содержание

Список иллюстраций

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.

2.1. 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>

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.

В следующем примере рассмотрим использование инструкции jmp:

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

2.2. Команды 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.

2.2.1. 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.

2.2.2. 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 - , но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов.

2.2.3. 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.

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.

3. Выполнение лабораторной работы

Создадим каталог для программам лабораторной работы № 7, перейдем в него и создадим файл lab7-1.asm.

```
ambessonov@fedora:~$ mkdir ~/work/arch-pc/lab07
ambessonov@fedora:~$ cd ~/work/arch-pc/lab07
ambessonov@fedora:~/work/arch-pc/lab07$ touch lab7-1.asm
```

Рис 3.1: 31

Введем в файл lab7-1.asm текст программы из листинга 7.1. Создадим исполняемый файл и запустим его.

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

Рис 3.2: 32

```
%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 3.3: 33

Создадим исполняемый файл и проверим его работу. Изменим текст программы добавив или изменив инструкции jmp, чтобы вывод программы был следующим:

Сообщение № 3

Сообщение № 2

Сообщение № 1

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

Рис 3.4: 34

```
%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
```

Рис 3.5: 35

Создадим файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. Внимательно изучим текст программы из листинга 7.3 и введем в lab7-2.asm. Создадим исполняемый файл и проверим его работу для разных значений B.

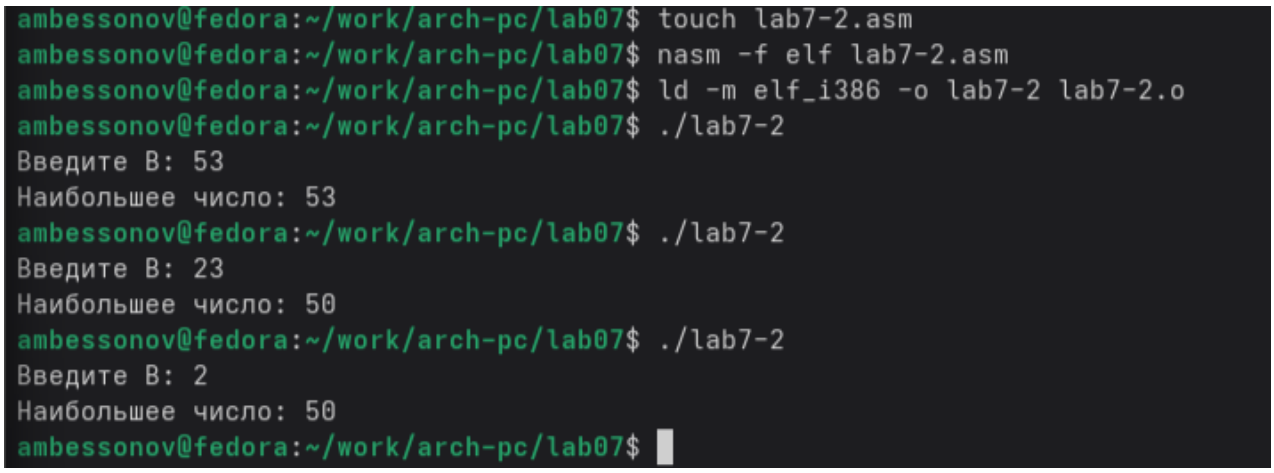

Рис 3.6: 36

Создадим файл листинга для программы из файла lab7-2.asm. Откроем файл листинга lab7-2.lst с помощью любого текстового редактора, например mcedit.

Внимательно ознакомимся с его форматом и содержимым. Подробно объясним содержимое трёх строк файла листинга по выбору:

Строка 4: *4 00000000 53 <l> push ebx*

Разбор: 4 - номер строки в исходном файле; 00000000 - смещение в текущей секции (адрес 0x00); 53 - машинный код инструкции (1 байт); 53 в шестнадцатеричной системе соответствует инструкции push ebx; <l> - метка, указывающая что это код из включенного файла; push ebx - исходная инструкция ассемблера.

Смысл: Сохраняет значение регистра EBX в стеке. Это стандартное начало функции для сохранения регистров.

Строка 8: *8 00000003 803800 <l> cmp byte [eax], 0*

Разбор: 8 - номер строки; 00000003 - смещение 0x03 (предыдущие инструкции заняли 3 байта); 803800 - машинный код инструкции (3 байта); 80 - код операции для операций с байтом; 38 - указывает операцию сравнения (cmp) с байтом в памяти по адресу в EAX**;** 00 - непосредственное значение 0 для сравнения**;** cmp byte [eax], 0 - исходная инструкция: сравнивает байт по адресу в EAX с нулем

Смысл: Проверяет, не является ли текущий символ строки нулевым (концом строки).

Строка 14: *14 00000008 2908 <l> sub eax, ebx*

Разбор: 14 - номер строки; 00000008 - смещение 0x08 (интересно, что совпадает со строкой 11); 2908 - машинный код инструкции (2 байта); 29 - код операции для вычитания (sub); 08 - modR/M байт, где: mod=00, reg=001 (ECX/CX/CL), r/m=000 (EAX/AX/AL)

Смысл: Вычитает из EAX значение EBX, вычисляя длину строки (разницу между конечным и начальным адресами).

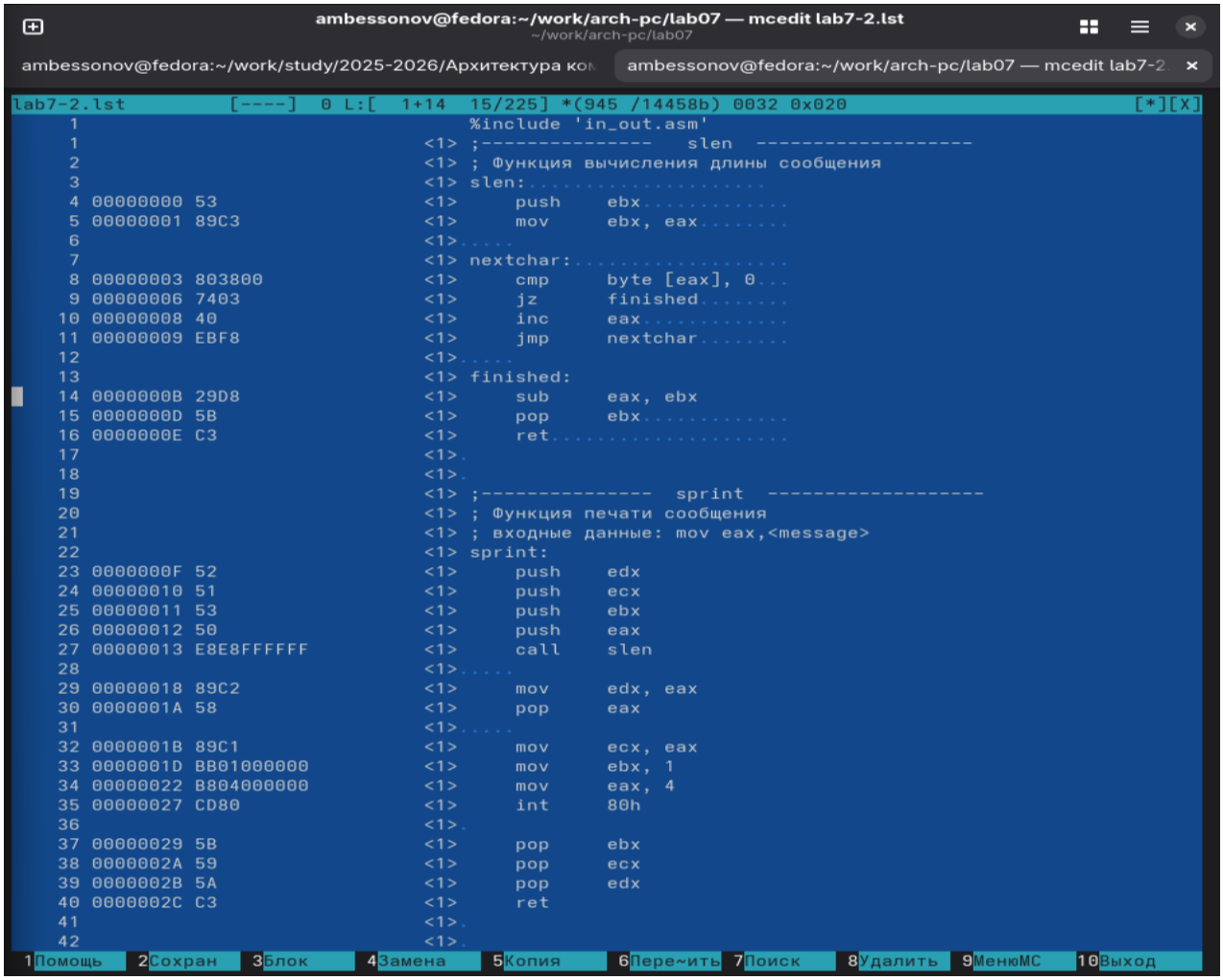

Рис 3.7: 37

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

Выходные файлы в этом случае создаются, в листинге добавляется строка: 25 ********** error: invalid combination of opcode and operands

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

ambessonov@fedora:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:25: error: invalid combination of opcode and operands
```

Рис 3.8: 38

```
20                                  ; ---------- Преобразование 'B' из символа в число
21 00000101 B8[0A000000]            mov eax,B
22 00000106 E891FFFFFF              call atoi ; Вызов подпрограммы перевода символа в число
23 0000010B A3[0A000000]            mov [B],eax ; запись преобразованного числа в 'B'
24                                  ; ---------- Записываем 'A' в переменную 'max'
25                                  mov ecx ; 'ecx = A'
25          ******************       error: invalid combination of opcode and operands
26 00000110 890D[00000000]          mov [max],ecx ; 'max = A'
27                                  ; ---------- Сравниваем 'A' и 'C' (как символы)
28 00000116 3B0D[39000000]          cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 0000011C 7F0C                    jg check_B ; если 'A>C', то переход на метку 'check_B',
30 0000011E 8B0D[39000000]          mov ecx,[C] ; иначе 'ecx = C'
31 00000124 890D[00000000]          mov [max],ecx ; 'max = C'
32                                  ; ---------- Преобразование 'max(A,C)' из символа в число
```

Рис 3.9: 39

Вывод: выполнили задания лабораторной работы.

4. Выполнение самостоятельной работы

4.1 Напишем программу нахождения наименьшей из 3 целочисленных переменных a, b, c. Создадим исполняемый файл и проверим его работу. (Значения a, b, c = 32,6,54)

Открыть

work.asm
~/work/arch-pc/lab07

```
%include 'in_out.asm'
section .data
msg_result db "Наименьшее число: ",0h
a dd 32
b dd 6
c dd 54
section .bss
min resb 10
section .text
global _start
_start:
; ---------- Преобразование a, b, c в числа (если нужно)
mov eax, [a]
mov [a], eax
mov eax, [b]
mov [b], eax
mov eax, [c]
mov [c], eax

; ---------- Находим наименьшее число
; Сначала сравниваем a и b, находим минимум
mov ecx, [a]    ; ecx = a
cmp ecx, [b]    ; Сравниваем a и b
jl compare_c    ; если a < b, переходим к сравнению с c
mov ecx, [b]    ; иначе ecx = b (b меньше)

compare_c:
; Теперь сравниваем найденный минимум (a или b) с c
cmp ecx, [c]    ; Сравниваем min(a,b) с c
jl save_result  ; если min(a,b) < c, сохраняем результат
mov ecx, [c]    ; иначе ecx = c (c меньше)

save_result:
mov [min], ecx  ; сохраняем наименьшее число

; ---------- Вывод результата
mov eax, msg_result
call sprint     ; Вывод сообщения 'Наименьшее число: '
mov eax, [min]
call iprintLF   ; Вывод наименьшего числа

call quit       ; Выход
```

Рис 4.1: 41

```
ambessonov@fedora:~/work/arch-pc/lab07$ touch work.asm
ambessonov@fedora:~/work/arch-pc/lab07$ nasm -f elf work.asm
ambessonov@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o work work.o
ambessonov@fedora:~/work/arch-pc/lab07$ ./work
Наименьшее число: 6
ambessonov@fedora:~/work/arch-pc/lab07$
```

Рис 4.2: 42

4.2 Напишем программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений. Создадим исполняемый файл и проверим его работу для значений x и a. (Значения (x1, a1) = (2, 3), (x2, a2) = (4, 2))

```
%include 'in_out.asm'
section .data
msg_x db 'x: ',0h
msg_a db 'a: ',0h
msg_result db "Результат: ",0h
x dd 0
a dd 0
result dd 0
section .bss
section .text
global _start
_start:
; ---------- Ввод x
mov eax, msg_x
call sprint
mov ecx, x
mov edx, 10
call sread
mov eax, x
call atoi
mov [x], eax

; ---------- Ввод a
mov eax, msg_a
call sprint
mov ecx, a
mov edx, 10
call sread
mov eax, a
call atoi
mov [a], eax

; ---------- Вычисление функции f(x)
; Сравниваем x и a
mov ebx, [x]
mov ecx, [a]
cmp ebx, ecx    ; Сравниваем x и a
jl less_than    ; если x < a, переходим на less_than

; x ≥ a: f(x) = x + 10
greater_or_equal:
add ebx, 10     ; ebx = x + 10
mov [result], ebx
jmp output

; x < a: f(x) = a + 10
less_than:
add ecx, 10     ; ecx = a + 10
mov [result], ecx

; ---------- Вывод результата
output:
mov eax, msg_result
call sprint     ; Вывод сообщения 'Результат: '
mov eax, [result]
call iprintLF   ; Вывод результата вычислений

call quit       ; Выход
```

Рис 4.3: 43

```
ambessonov@fedora:~/work/arch-pc/lab07$ touch work1.asm
ambessonov@fedora:~/work/arch-pc/lab07$ nasm -f elf work1.asm
ambessonov@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o work1 work1.o
ambessonov@fedora:~/work/arch-pc/lab07$ ./work1
x: 2
a: 3
Результат: 13
ambessonov@fedora:~/work/arch-pc/lab07$ ./work1
x: 4
a: 2
Результат: 14
ambessonov@fedora:~/work/arch-pc/lab07$
```

Рис 4.4: 44

Вывод: выполнили задания самостоятельной работы, отработали навыки, полученные в ходе лабораторной работы.

5. Выводы

Изучили команды условного и безусловного переходов. Приобрели навыки написания программ с использованием переходов. Ознакомились с назначением и структурой файла листинга.