РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

направление: *Компьютерные и информационные науки*

Лабораторная работа №7

дисциплина: *Архитектура компьютеров и операционные системы*

студент: Гробман Александр Евгеньевич

Группа: НКАбд-02-23

Цель работы

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

Задание

1. Реализация переходов в NASM
2. Изучение структуры файлы листинга.
3. Задания для самостоятельной работы

Теория

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. 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.
Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имееттекстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию.

Выполнение лабораторной работы

1. **Символьные и численные данные в NASM**

создаю файл lab7-1.asm и ввожу туда код из листинга

```
lab7-1.asm          [----]  0 L:[  1+ 0   1/ 21] *(0   / 650b) 0
%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
```

запускаем исполняемый файл

```
[aegrobman lab07]$ ./lab07-1
Сообщение № 2
Сообщение № 3
```

Таким образом, использование инструкции jmp _label2 меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки _label2, пропустив вывод первого сообщения.
Изменю программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2.

```
lab7-1.asm          [----]  5 L:[  1+14  15/ 23] *(382 / 671b)
%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
_label2:
mov eax, msg2
call sprintLF
jmp _label1
_label3:
mov eax, msg3
call sprintLF
_end:
call quit
```

Запускаем исполняемый файл

```
[aegrobman lab07]$ ./lab07-1
Сообщение № 2
Сообщение № 1
```

Затем изменяю текст программы, добавив в начале программы jmp _label3, jmp _label2 в конце метки jmp _label3, jmp _label1 добавляю в конце метки jmp _label2, и добавляю jmp _end в конце метки jmp _label1

```
lab7-1.asm          [----] 11 L:[  1+20  21/ 24] *(607 / 683b) 00
%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
_label1:
mov eax, msg1
call sprintLF
jmp _end
_label2:
mov eax, msg2
call sprintLF
jmp _label1
_label3:
mov eax, msg3
call sprintLF
jmp _label2
_end:
```

Запускаем исполняемый файл

```
[aegrobman lab07]$ ./lab07-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

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 вводиться с клавиатуры.

Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07.

```
lab7-2.asm          [----] 31 L:[  1+ 3   4/ 50] *(114 /1744b)
%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
```

Создаю исполняемый файл и проверьте его работу.

```
[aegrobman lab07]$ ./lab07-
Введите B: 70
Наибольшее число: 70
```

2. **Изучение структуры файлы листинга**

Создаю файл листинга для программы из файла lab7-2.asm

Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое.

В представленных трех строчках содержаться следующие данные:

```
 2                                  <1> ; Функция вычисления длины сообщения
 3                                  <1> slen:
 4 00000000 53                      <1>     push    ebx
```

“2” - номер строки кода, “; Функция вычисления длинны сообщения” - комментарий к коду, не имеет адреса и машинного кода.
“3” - номер строки кода, “slen” - название функции, не имеет адреса и машинного кода.
“4” - номер строки кода, “00000000” - адрес строки, “53” - машинный код, “push ebx” - исходный текст программы, инструкция “push” помещает операнд “ebx” в стек.

Возвращаемся в lab7-2.asm

Открываю файл с программой lab7-2.asm и в выбранной мной инструкции с двумя операндами удаляю выделенный операнд.

```
cmp ecx,[C]
```

На выходе я не получаю ни одного файла из-за ошибки:инструкция mov (единственная в коде содержит два операнда) не может работать, имея только один операнд, из-за чего нарушается работа кода.

3. **Задания для самостоятельной работы**

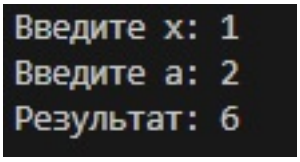
первый номер

```
task1.asm          [----] 15 L:[  1+11  12/ 38] *(190 /1301b) 10
%include 'in_out.asm'
section .data
msg db "Наименьшее число: ",0h
A dd '41'
B dd '62'
C dd '35'
section .bss
min resb 10
section .text
global _start
_start:

mov ecx,[A]
mov [min],ecx

cmp ecx,[C]
jg check_B
mov ecx,[C]
mov [min],ecx

check_B:
mov eax,min
```

```
[aegrobman lab07]$ ./lab07-2
Наименьшее число: 35
```

и второй номер

```
Введите x: 3
Результат: 1
```



5. **Вывод**

С помощью данной лабораторной работы я освоила арифметические инструкции языка ассемблер NASM, что пригодится мне при выполнении последующих лабораторных работ.

Отправляем файлы на гитхаб.

Ссылка на отчёт [https://github.com/DaOneme/AEGrobman_study_2023-2024_arhpc/tree/main/Labs/Lab06 (https://github.com/DaOneme/AEGrobman_study_2023-2024_arhpc/tree/main/Labs/Lab06)](https://github.com/DaOneme/AEGrobman_study_2023-2024_arhpc/tree/main/Labs/Lab06)