РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7.

дисциплина: *Архитектура компьютеров*

Студент: Подхалюзина Виолетта Михайловна

Группа: НКАбд-04-24

МОСКВА

2024 г.

Оглавление

1 Цель работы

Изучить команды условного и безусловного переходов. Приобрести навыки написания программ с использованием переходов. И познакомиться с назначением и структурой файла листинга.

2 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.

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.

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.

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

листинга:

• 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);

• адрес — это смещение машинного кода от начала текущего сегмента;

• машинный код представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция int 80h начинается по смещению 00000020 в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция int 80h ассемблируется в CD80 (в шестнадцатеричном представлении); CD80 — это инструкция на машинном языке, вызывающая прерывание ядра);

• 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).

3 Выполнение лабораторной работы

3.1 Начало работы

Я создаю каталог для программ лабораторной работы №7, перехожу в него и создаю файл lab7-1.asm:

```
podkhalyuzina_violetta_113224676@violetta-Mint:~$ mkdir ~/work/arch-pc/lab07
podkhalyuzina_violetta_113224676@violetta-Mint:~$ cd ~/work/arch-pc/lab07
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ touch lab7-1.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ls
lab7-1.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$
```

Копирую код из листинга в файл будущей программы:

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ls
in_out.asm  lab7-1.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$
```

```
 GNU nano 7.2
%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
```

Создаю исполняемый файл, запускаю его и убеждаюсь, что при запуске программы неусловный переход действительно изменяет порядок выполнения инструкций:

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$
```

Изменяю программу таким образом, чтобы поменялся порядок выполнения функций:

```
 GNU nano 7.2
%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'
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
```

Далее проверяю, работают ли изменения:

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$
```

После этого изменяю текст программы так, чтобы все три сообщения вывелись в обратном порядке:

```
 GNU nano 7.2
%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 _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
```

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$
```

Далее создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07, изучаю текст

программы из листинга 7.3 и ввожу в lab7-2.asm:

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ touch lab7-2.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$
```

```
 GNU nano 7.2
%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
```

Программа выводит значение переменной с максимальным значением, проверяю работу программы с разными входными данными:

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 1
Наибольшее число: 50
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 2
Наибольшее число: 50
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 100
Наибольшее число: 100
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 72
Наибольшее число: 72
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$
```

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

листинга lab7-2.lst:

Первое значение в файле листинга - номер строки, и он может вовсе не совпадать с номером строки изначального файла. Второе вхождение - адрес, смещение машинного кода относительно начала текущего сегмента, затем непосредственно идет сам машинный код, а заключает строку исходный текст прогарммы с комментариями. Удаляю один операнд из случайной инструкции, чтобы проверить поведение файла листинга в дальнейшем:

В новом файле листинга показывает ошибку, которая возникла при попытке трансляции файла. Никакие выходные файлы при этом помимо файла листинга не создаются:

3.2 Самостоятельная работа

Я написала программу нахождения наименьшей из 3 целочисленных переменных □,□ и . Значения переменных выбрала из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создала исполняемый файл и проверила его работу.

А затем, я написала программу, которая для введенных с клавиатуры значений □ и □ вычисляет значение заданной функции □(□) и выводит результат вычислений. Вид функции □(□) выбрала из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создала исполняемый файл и проверьте его работу для значений □ и □ из 7.6.

```
 GNU nano 7.2
%include 'in_out.asm'
SECTION .data
msg1 db 'Введите B: ', 0h
msg2 db 'Наименьшее число: ', 0h
A dd '82'
C dd '61'
SECTION .bss
min 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 [min], ecx
cmp ecx, [C]
jg check_B
mov ecx, [C]
mov [min], ecx
check_B:
mov eax, min
call atoi
mov [min], eax
mov ecx, [min]
cmp ecx, [B]
jb fin
mov ecx, [B]
mov [min], ecx
fin:
mov eax, msg2
call sprint
mov eax, [min]
call iprintLF
call quit
```

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ nasm -f elf sWork1.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ld -m elf_i386 -o sWork sWork1.o
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ./sWork
Введите B: 59
Наименьшее число: 59
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ touch sWork2.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2  lab7-2.asm  lab7-2.lst  lab7-2.o  sWork  sWork1.asm  sWork1.o  sWork2.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$
```

```
 GNU nano 7.2
%include 'in_out.asm'
SECTION .data
msg_x: DB 'Введите значение переменной x: ', 0
msg_a: DB 'Введите значение переменной a: ', 0
res: DB 'Результат: ', 0
SECTION .bss
x: RESB 80
a: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg_x
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
mov edi, eax
mov eax, msg_a
call sprint
mov ecx, a
mov edx, 80
call sread
mov eax, a
call atoi
mov esi, eax
cmp edi, esi
jl first
mov eax, edi
sub eax, 1
jmp print_result
first:
mov eax, esi
sub eax, 1
print_result:
mov edi, eax
mov eax, res
call sprint
mov eax, edi
call iprintLF
call quit
```

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ./sWork2
Введите значение переменной x: 5
Введите значение переменной a: 7
Результат: 6
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$ ./sWork2
Введите значение переменной x: 6
Введите значение переменной a: 4
Результат: 5
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab07$
```

Вывод: я изучила команды условного и безусловного переходов. Приобрела навыки написания программ с использованием переходов. И познакомилась с назначением и структурой файла листинга.

4 Контрольные вопросы для самопроверки

1. Файл листинга NASM нужен для отладки программы. В отличие от текста программы, он содержит дополнительную информацию: адреса команд, машинный код и их соответствие исходному коду, а также возможные сообщения об ошибках.

2. Формат файла листинга NASM текстовый. Он состоит из номера строки, адреса, машинного кода и исходного текста программы. Например, строка может включать номер строки в листинге, смещение команды от начала сегмента, её машинный код в шестнадцатеричном виде и строку исходного кода программы.

3. В программах на ассемблере ветвление реализуется с помощью команд передачи управления (переходов). Они делятся на условные и безусловные. Условные переходы зависят от значения флагов, безусловные всегда передают управление в указанную точку программы.

4. Безусловный переход выполняется с помощью команды jmp. Условные переходы, такие как je (если равно) или jne (если не равно), проверяют состояние флагов процессора после предыдущей команды.

5. Команда cmp сравнивает два операнда, выполняя их вычитание, но результат не сохраняется. Она лишь выставляет флаги, такие как Zero Flag (ZF) или Sign Flag (SF), для дальнейшего использования командами условного перехода.

6. Синтаксис команд условного перехода: j<мнемоника> <метка>. Например, je label выполнит переход на метку label, если результат сравнения равен нулю (ZF=1).

7. Пример: cmp ax, bx ; сравниваются регистры je equal ; переход на метку equal, если ax = bx ... equal: mov cx, ax ; выполняется, если ax = bx

8. Команды безусловного перехода не анализируют флаги. Они передают управление в указанную точку программы независимо от текущего состояния процессора.

5 Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.

2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.

3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.

4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 c. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.

6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 c. — ISBN 978-1491941591.

7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.

8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 c. — ISBN 9781784396879.

9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.

10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.

11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.

12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.

13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.

14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL:

http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).