Лабораторная работа №7

Команды безусловного и условного переходов в Nasm. Программирование ветвлений.

Хорошева Алёна Евгеньевна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Освоение команд переходов двух типов - условный и безусловный. Написание программ с их использованием. Изучение структуры файла листинга.

2 Задание

1. Напишите программу нахождения наименьшей из 3 целочисленных переменных a,b,c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте исполняемый файл и проверьте его работу.

2. Напишите программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции f(x) и выводит результат вычислений. Вид функции f(x) выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте исполняемый файл и проверьте его работу для значений x и a из 7.6.

3 Теоретическое введение

Для реализации ветвлений в ассемблере используются так называемые команды передачи упр

• 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.

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 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.

2. **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.

Флаги состояния (биты 0, 2, 4, 6, 7 и 11) отражают результат выполнения арифметических инструкций, таких как ADD, SUB, MUL, DIV.

3. **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 ,

   Команда cmp, так же как и команда вычитания, выполняет вычитание , но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов.

   Примеры:

   cmp ax,‘4’; сравнение регистра ax с символом 4

   cmp ax,4; сравнение регистра ax со значением 4

   cmp al,cl; сравнение регистров al и cl

   cmp [buf],ax; сравнение переменной buf с регистром ax

4. **Описание команд условного перехода**

   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 через дробь (например, ja и jnbe). Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы.

   Примечание: термины «выше» («a» от англ. «above») и «ниже» («b» от англ.

«below») применимы для сравнения беззнаковых величин (адресов), а термины «больше» («g» от англ. «greater») и «меньше» («l» от англ. «lower») используются при учёте знака числа. Таким образом, мнемонику инструкции JA/JNBE можно расшифровать как «jump if above (переход если выше) / jump if not below equal (переход если не меньше или равно)».

Помимо перечисленных команд условного перехода существуют те, которые которые можно использовать после любых команд, меняющих значения флагов.

В качестве примера рассмотрим фрагмент программы, которая выполняет умножение переменных и и если произведение превосходит размер байта, передает управление на метку Error.

mov al, a

mov bl, b

mul bl

jc Error

5. **Файл листинга и его структура**

   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.

   Ниже приведён фрагмент файла листинга.

   10 00000000 B804000000 mov eax,4

11 00000005 BB01000000 mov ebx,1

12 0000000A B9[00000000] mov ecx,hello

13 0000000F BA0D000000 mov edx,helloLen

14

15 00000014 CD80 int 80h

Все ошибки и предупреждения, обнаруженные при ассемблировании, трансля-

тор 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).

4 Выполнение лабораторной работы

1. Создаем файл lab-1.asm и вводим туда текст программы из листинга 7.1, где используется команда jmp для выполнения безусловного перехода. Запускаем исполняемый файл и получаем корректный ответ:

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

   Таким образом, использование инструкции jmp _label2 меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки _label2, пропустив вывод первого сообщения.

2. В этом же файле меняем текст программы, чтобы на выходе выводилось сначала сообщение №2, потом сообщение №3. Для этого в текст программы после вывода сообщения № 2 добавим инструкцию jmp с меткой _label1 (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавим инструкцию jmp с меткой _end (т.е. переход к инструкции call quit).

```
alyona@aekhorosheva:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
alyona@aekhorosheva:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
alyona@aekhorosheva:~/work/arch-pc/lab07$
```

3. Теперь изменяем код программы следующим образом:

```
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
```

Переходы начинаются с 3й метки, затем вторая и в конце первая, это позволяет нам вывести сообщения в обратном порядке.

4. Теперь при запуске мы получаем следующий результат:

```
alyona@aekhorosheva:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
alyona@aekhorosheva:~/work/arch-pc/lab07$
```

5. Создаем новый файл lab7-2.asm и вводим текст программы для определе-

ния наибольшего значения из трёх чисел(A, B, C) из листинга 7.3. В данном примере переменные A и C сравниваются как символы.

Запускаем исполняемый файл и проверяем работу программы для различных значений B, т.к. это число мы вводим с клавиатуры.

```
alyona@aekhorosheva:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
alyona@aekhorosheva:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 12
Наибольшее число: 50
alyona@aekhorosheva:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 56
Наибольшее число: 56
alyona@aekhorosheva:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 33
Наибольшее число: 50
```

6. С помощью команды *nasm -f elf -l lab7-2.lst lab7-2.asm* создаём файл листинга для нашей программы. Затем командой *mcedit lab7-2.lst* открываем листинг в текстовом редакторе.

```
/home/al~7-2.lst   [----]  0 L:[  1+ 0   1/235] *(0   /14874b) 0032 0x020 [*]
    1                                   %include 'in_out.asm'
    1                                <1> ;---------------   slen  ------------
    2                                <1> ; Функция вычисления длины сообщения
    3                                <1> slen:.....................
    4 00000000 53                    <1>     push    ebx............
    5 00000001 89C3                  <1>     mov     ebx, eax.......
    6                                <1>.....
    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>
```

Первые три строки после подключения внешнего файла пустые, поскольку в них нет команд - только комментарии. При этом номер строки присваива-

ется.

Следующие две строки - 4 и 5 - имеют адрес, т.к. начинается код, а значит есть и смещение машинного кода от начала сегмента(в нашем случае - это slen). После адреса идёт сам машинный код в виде шестнадцатеричной последовательности. Он представляет ассемблированную собой исходную строку.

7. Открываем lab7-2.asm и в функции fin в инструкции mov удаляем один из двух операндов.

```
; ---------- Вывод результата
fin:
mov eax
```

Создаём файл листинга (см.команду в п.6) и получаем ошибку, т.к. выполнение инструкции требует наличие обоих операндов. В самом тексте листинга она отображается следующим образом:

```
0000013F 8B0D[00000000]          mov ecx,[max]
                                 cmp [B] ; Сравниваем 'max(A,C)' и 'B'
         ******************       error: invalid combination of opcode an
00000145 7F0C                    jg fin ; если 'max(A,C)>B', то переход н
```

Делаем вывод, что адрес и машинный код больше не присваиваются этой строке, вместо них стоят “*” и после слова error идёт описание ошибки. Итак, файл листинга полезен для выявления ошибок в коде программы.

4.1 Задание для самостоятельной работы

1. Для написания программы lab7-3.asm нахождения наименьшего из трех чисел a/b/c используем листинг 7.3. и таблицу с мнемокодами для условного перехода по результатам сравнения чисел(cmp a,b).

   Начинаем код с подключения внешнего файла и задавания значений переменным: *1 вариант*: a = 17, b = 23, c = 45. Очевидно, что при корректном

тексте программы на выходе мы получим число 17.

```
%include ‘in_out.asm’
section .data
msg db “Наименьшее число:”,0h
A dd 17
B dd 23
C dd 45
```

Выделяем память для переменной, в которой будет храниться полученный результат(минимальное значение):

```
section .bss
min1 resb 10
```

Начинается основная часть, здесь: *check_a_le_b* - функция для сравнения a и b. Если a больше b, то переходим к сравнению a и c. Иначе программа идет дальше по порядку и переходит к сравнению b и c.

```
check_a_le_b:
cmp eax, ebx
jle check_a_le_c
jmp check_b_le_c
— check_a_le_c:
cmp eax, ecx
jle a_is_min
jmp c_is_min
— check_b_le_c:
cmp ebx, ecx
jle b_is_min
jmp c_is_min
```

— Далее выполняется одна из функций, которая записывает переменную(a/b/c) в min1 и переходит к функции fin для вывода результата на экран. a_is_min:

mov [min1], eax

jmp fin

— b_is_min:

mov [min1], ebx

jmp fin

— c_is_min:

mov [min1], ecx

jmp fin

— ; ———- Вывод результата

fin:

mov eax, msg

call sprint ; Вывод сообщения ‘Наименьшее число:’

mov eax,[min1]

call iprintLF ; Вывод ‘min(A,B,C)’

call quit ; Выход

— Запускаем программу для проверки корректности:

```
alyona@aekhorosheva:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
alyona@aekhorosheva:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.
alyona@aekhorosheva:~/work/arch-pc/lab07$ ./lab7-3
Наименьшее число: 17
```

2. Для написания программы *lab7-4.asm*, которая запрашивает на вход два значения(x и a) для вычисления значения функции(*вариант №1*) будем использовать навыки с лабораторной работы №6 и опыт с условными переходами для реализации системы функции f(x).

   Начинаем с подключения файла и секции с выводимым тектом:

   %include ‘in_out.asm’

   section .data

   msg1 db ‘Введите x и a:’,0h

   msg2 db ‘Результат:’,0h

   Теперь выделим место для трех значений: *x, a* - вводим с клавиатуры и *res* -

конечный результат программы(т.е. значение выражения при заданных x,a).

```
section .bss
x resb 10
a resb 10
res resb 10
```

Начинается основная часть: выводим сообщение, чтобы пользователь задал значения x и a, реализуем ввод с клавиатуры.

```
section .text
global _start
_start:
— ; ———- Вывод сообщения ‘Введите x и a:’
mov eax,msg1
call sprint
— ; ———- Ввод ‘x’
mov ecx,x
mov edx,10
call sread
— ; ———- Ввод ‘a’
mov ecx,a
mov edx,10
call sread
```

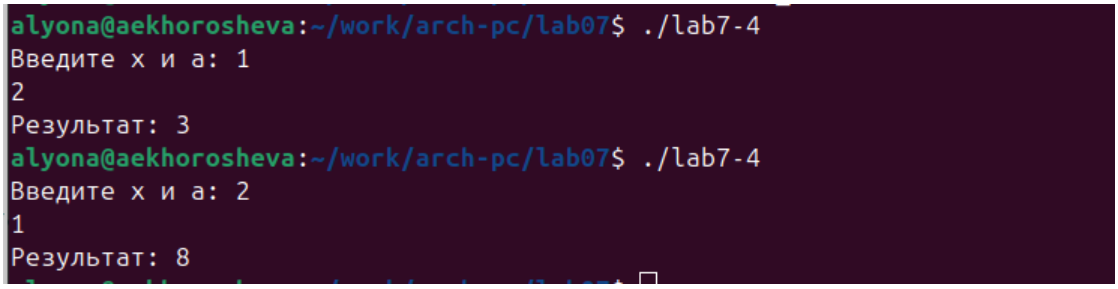
— Сравниваем x и a командой cmp. Если $x < a$, то переходим с помощью *jl to_calculate* к функции вычисления выражения $2a - x$.

Иначе, значение функции равно 8. Передаем это значение в res и безусловный переход к финальной функции.

```
; ———- Сравнение x и a
mov eax, [x]
mov ebx, [a]
```

```
cmp eax,ebx
jl to_calculate
mov eax, 8
mov [res],eax
jmp fin
—
; ———- Вычисление выражения при x < a
to_calculate:
mov eax,2
mov ebx,[a]
mul ebx
sub eax,[x]
mov [res],eax
—
; ———- Преобразуем результат из символа в число
mov eax,res
call atoi
mov [res], eax
—
fin:
mov eax, msg2
call sprint ; Вывод сообщения ‘Результат:’
mov eax, [res]
call iprintLF ; Вывод результата
call quit ; Выход
—
```

Проверяем корректность работы кода - запускаем исполнительный файл и сравниваем результат с самостоятельными вычислениями.



Таким образом, программа вычисления значения функции работает верно.

5 Выводы

Освоены команды условного и безусловного перехода, мнемоника для управления переходами

В результате выполнения лабораторной работы написаны программы для нахождения наименьш

Список литературы