8. Лабораторная работа №8. Команды безусловного и условного переходов в Nasm. Программирование ветвлений.

8.1. Цель работы

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

8.2. 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.

8.2.1. 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 (см. табл. 8.1).

Таблица 8.1. Типы операндов инструкции `jmp`

Тип операнда	Описание
`jmp label`	переход на метку `label`
`jmp [label]`	переход по адресу в памяти, помеченному меткой `label`
`jmp eax`	переход по адресу из регистра `eax`

В следующем примере рассмотрим использование инструкции `jmp`:

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

8.2.2. 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.

8.2.2.1. 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 (рис. 8.1).

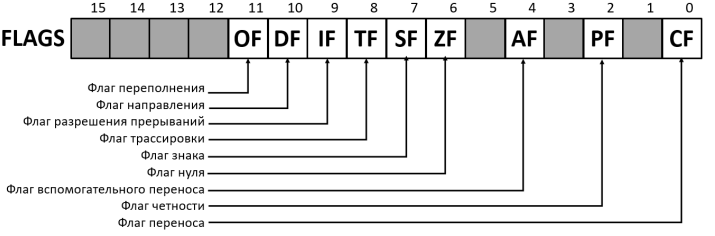

Рис. 8.1. Регистр флагов

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

Таблица 8.2. Регистр флагов

Бит	Обозначение	Название	Описание
0	CF	Carry Flag - Флаг переноса	Устанавливается в 1, если при выполнении предыдущей операции произошёл перенос из старшего бита или если требуется заём (при вычитании). Иначе установлен в 0.
2	PF	Parity Flag - Флаг чётности	Устанавливается в 1, если младший байт результата предыдущей операции содержит чётное количество битов, равных 1.
4	AF	Auxiliary Carry Flag - Вспомогательный флаг переноса	Устанавливается в 1, если в результате предыдущей операции произошёл перенос (или заём) из третьего бита в четвёртый.
6	ZF	Zero Flag - Флаг нуля	Устанавливается 1, если результат предыдущей команды равен 0.
7	SF	Sign Flag - Флаг знака	Равен значению старшего значащего бита результата, который является знаковым битом в знаковой арифметике.
11	SF	Overflow Flag - Флаг переполнения	Устанавливается в 1, если целочисленный результат слишком длинный для размещения в целевом операнде (регистре или ячейке памяти).

Более подробную информацию о регистре флагов см. [14; 15].

8.2.2.2. 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 <операнд_2> - <операнд_1>, 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
```

8.2.2.3. 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.

В табл. 8.3. 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.

Таблица 8.3. Инструкции условной передачи управления по результатам арифметического сравнения `cmp a,b`

Типы операндов	Мнемокод	Критенрий условного перехода $a \vee b$	Значения флагов	Комментарий
Любые	`JE`	$a = b$	ZF = 1	Переход если равно
Любые	`JNE`	$a \neq b$	ZF = 0	Переход если не равно
Со знаком	`JL/JNGE`	$a < b$	SF $\neq$ OF	Переход если меньше
Со знаком	`JLE/JNG`	$a \leq b$	SF $\neq$ OF или ZF = 1	Переход если меньше или равно
Со знаком	`JG/JNLE`	$a > b$	SF = OF и ZF = 0	Переход если больше
Со знаком	`JGE/JNL`	$a \geq b$	SF = OF	Переход если больше или равно
Без знака	`JB/JNAE`	$a < b$	CF = 1	Переход если ниже

Типы операндов	Мнемокод	Критенрий условного перехода $a \vee b$	Значения флагов	Комментарий
Без знака	`JBE/JNA`	$a \leq b$	CF = 1 или ZF = 1	Переход если ниже или равно
Без знака	`JA/JNBE`	$a > b$	CF = 0 и ZF = 0	Переход если выше
Без знака	`JAE/JNB`	$a \geq b$	CF = 0	Переход если выше или равно

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 (табл. 8.4).

Таблица 8.4. Инструкции условной передачи управления

Мнемокод	Значение флага для осуществления перехода	Мнемокод	Значение флага для осуществления перехода
JZ	ZF = 1	JNZ	ZF = 0

Мнемокод	Значение флага для осуществления перехода	Мнемокод	Значение флага для осуществления перехода
JS	SF = 1	JNS	SF = 0
JC	CF = 1	JNC	CF = 0
JO	OF = 1	JNO	OF = 0
JP	PF = 1	JNP	PF = 0

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

```
mov   al, a
mov   bl, b
mul   bl
jc    Error
```

8.2.3. 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
```

Строки в первой части листинга имеют следующую структуру (рис. 8.2).

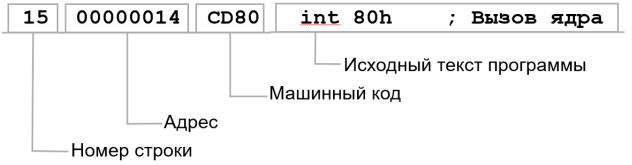

Рис. 8.2. Структура листинга

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).

8.3. Порядок выполнения лабораторной работы

8.3.1. Реализация переходов в NASM

1. Создайте каталог для программам лабораторной работы № 8, перейдите в него и создайте файл `lab8-1.asm`:

```
mkdir  ~/work/arch-pc/lab08
cd ~/work/arch-pc/lab08
touch lab8-1.asm
```

2. Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введите в файл `lab8-1.asm` текст программы из листинга 8.1.

Листинг 8.1. Программа с использованием инструкции `jmp`

```
%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
```

Создайте исполняемый файл и запустите его. Результат работы данной программы будет следующим:

```
user@dk4n31:~$ ./lab8-1
Сообщение № 2
Сообщение № 3
user@dk4n31:~$
```

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

Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. Изменим программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. 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`). Измените текст программы в соответствии с листингом 8.2.

Листинг 8.2. Программа с использованием инструкции `jmp`

```
%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
```

Создайте исполняемый файл и проверьте его работу.

Измените текст программы добавив или изменив инструкции jmp, чтобы вывод программы был следующим:

```
user@dk4n31:~$ ./lab8-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
user@dk4n31:~$
```

3. Использование инструкции jmp приводит к переходу в любом случае. Однако, часто при написании программ необходимо использовать условные переходы, т.е. переход должен происходить если выполнено какое-либо условие. В качестве примера 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.

Создайте файл lab8-2.asm в каталоге ~/work/arch-pc/lab08. Внимательно изучите текст программы из листинга 8.3 и введите в lab8-2.asm.

Листинг 8.3. Программа, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: A,B и C.

```
%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
```

Создайте исполняемый файл и проверьте его работу для разных значений `B`.

Обратите внимание, в данном примере переменные `A` и `C` сравниваются как символы, а переменная `B` и максимум из `A` и `C` как числа (для этого используется функция `atoi` преобразования символа в число). Это сделано для демонстрации того, как сравниваются данные. Данную программу можно упростить и сравнивать все 3 переменные как символы (т.е. не использовать функцию `atoi`). Однако если переменные преобразовать из символов числа, над ними можно корректно проводить арифметические операции.

8.3.2. Изучение структуры файлы листинга

4. Обычно `nasm` создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ `-l` и задав имя файла листинга в командной строке. Создайте файл листинга для программы из файла `lab8-2.asm`

```
nasm -f elf -l lab8-2.lst lab8-2.asm
```

Откройте файл листинга `lab8-2.lst` с помощью любого текстового редактора, например `mcedit`:

```
mcedit lab8-2.lst
```

Внимательно ознакомиться с его форматом и содержимым. Подробно объяснить содержимое трёх строк файла листинга по выбору.

Откройте файл с программой `lab8-2.asm` и в любой инструкции с двумя операндами удалить один операнд. Выполните трансляцию с получением файла листинга:

```
nasm -f elf -l lab8-2.lst lab8-2.asm
```

Какие выходные файлы создаются в этом случае? Что добавляется в листинге?

8.4. Задание для самостоятельной работы

1. Напишите программу нахождения наименьшей из 3 целочисленных переменных a,b и . Значения переменных выбрать из табл. 8.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. 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 8.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу для значений x и a из 8.6.

8.5. Варианты заданий

Таблица 8.5. Значения a, b, c для задания №1.

Номер варианта	Значения a, b, c	Номер варианта	Значения a, b, c
1	17,23,45	**11**	21,28,34
2	82,59,61	**12**	99,29,26

Номер варианта	Значения a, b, c	Номер варианта	Значения a, b, c
3	94,5,58	**13**	84,32,77
4	8,88,68	**14**	81,22,72
5	54,62,87	**15**	32,6,54
6	79,83,41	**16**	44,74,17
7	45,67,15	**17**	26,12,68
8	52,33,40	**18**	83,73,30
9	24,98,15	**19**	46,32,74
10	41,62,35	**20**	95,2,61

Таблица 8.6. Выражения для $f(x)$ для задания №2.

Номер варианта	Выражение для $f(x)$	(x_1, a_1)	(x_2, a_2)
1	$\begin{cases} 2a - x, & x < a \\ 8, & x \geq a \end{cases}$	(1;2)	(2;1)
2	$\begin{cases} a - 1, & x < a \\ x - 1, & x \geq a \end{cases}$	(5;7)	(6;4)
3	$\begin{cases} 3x, & x = 3 \\ a + 1, & x \neq 3 \end{cases}$	(3;4)	(1;4)
4	$\begin{cases} 2x + a, & a \neq 0 \\ 2x + 1, & a = 0 \end{cases}$	(3;0)	(3;2)
5	$\begin{cases} 2(x - a), & x > a \\ 15, & x \leq a \end{cases}$	(1;2)	(2;1)

Номер варианта	Выражение для $f(x)$	(x_1, a_1)	(x_2, a_2)
6	$\begin{cases} x + a, & x = a \\ 5x, & x \neq a \end{cases}$	(2;2)	(2;1)
7	$\begin{cases} 6a, & x = a \\ a + x, & x \neq a \end{cases}$	(1;1)	(2;1)
8	$\begin{cases} 3a, & a < 3 \\ x + 1, & a \geq 3 \end{cases}$	(1;4)	(1;2)
9	$\begin{cases} a + x, & x \leq a \\ a, & x > a \end{cases}$	(5;7)	(6;4)
10	$\begin{cases} x - 2, & x > 2 \\ 3a, & x \leq 2 \end{cases}$	(3;0)	(1;2)
11	$\begin{cases} 4a, & x = 0 \\ 4a + x, & x \neq 0 \end{cases}$	(0;3)	(1;2)
12	$\begin{cases} ax, & x < 5 \\ x - 5, & x \geq 5 \end{cases}$	(3;7)	(6;4)
13	$\begin{cases} a - 7, & a \geq 7 \\ ax, & a < 7 \end{cases}$	(3;9)	(6;4)
14	$\begin{cases} 3a + 1, & x < a \\ 3x + 1, & x \geq a \end{cases}$	(2;3)	(4;2)
15	$\begin{cases} a + 10, & x < a \\ x + 10, & x \geq a \end{cases}$	(2;3)	(4;2)
16	$\begin{cases} x + 4, & x < 4 \\ ax, & x \geq 4 \end{cases}$	(1;1)	(7;1)
17	$\begin{cases} a + 8, & a < 8 \\ ax, & a \geq 8 \end{cases}$	(3;4)	(2;9)

Номер варианта	Выражение для $f(x)$	(x_1, a_1)	(x_2, a_2)
18	$\begin{cases} a^2, & a \neq 1 \\ 10 + x, & a = 1 \end{cases}$	(1;2)	(2;1)
19	$\begin{cases} a + x, & x > a \\ x, & x \leq a \end{cases}$	(4;5)	(3;2)
20	$\begin{cases} x - a, & x \geq a \\ 5, & x < a \end{cases}$	(1;2)	(2;1)

8.6. Содержание отчёта

Отчёт должен включать:

- Титульный лист с указанием номера лабораторной работы и ФИО студента.
- Формулировка цели работы.
- Описание результатов выполнения лабораторной работы:
  - описание выполняемого задания;
  - скриншоты (снимки экрана), фиксирующие выполнение заданий лабораторной работы;
  - комментарии и выводы по результатам выполнения заданий.
- Описание результатов выполнения заданий для самостоятельной работы:
  - описание выполняемого задания;
  - скриншоты (снимки экрана), фиксирующие выполнение заданий;
  - комментарии и выводы по результатам выполнения заданий;
  - листинги написанных программ (текст программ).
- Выводы, согласованные с целью работы.

Отчёт по выполнению лабораторной работы оформляется в формате `Markdown`. В качестве отчёта необходимо предоставить отчёты в 3 форматах:

pdf, docx и md. А также файлы с исходными текстами написанных при выполнении лабораторной работы программ (файлы *.asm). Файлы необходимо загрузить на странице курса в ТУИС в задание к соответствующей лабораторной работе и загрузить на Github.

8.7. Вопросы для самопроверки

1. Для чего нужен файл листинга NASM? В чём его отличие от текста программы?
2. Каков формат файла листинга NASM? Из каких частей он состоит?
3. Как в программах на ассемблере можно выполнить ветвление?
4. Какие существуют команды безусловного и условных переходов в языке ассемблера?
5. Опишите работу команды сравнения cmp.
6. Каков синтаксис команд условного перехода?
7. Приведите пример использования команды сравнения и команд условного перехода.
8. Какие флаги анализируют команды безусловного перехода?