Лекция 9

2.4. Команды передачи управления

Команды передачи управления подразделяются на:

1. ***безусловные*** — в данной точке необходимо передать управление не той команде, которая идет следующей, а другой, которая находится на некотором удалении от текущей команды;
2. ***условные*** — решение о том, какая команда будет выполняться следующей, принимается на основе анализа некоторых условий или данных;
3. ***управления циклом***;
4. ***вызов процедур***.

2.4.1. Команда безусловного перехода JMP

JMP [модификатор] адрес_перехода - безусловный переход без сохранения информации о точке возврата.

Модификатор может принимать следующие значения:

- ***near ptr*** — прямой переход на метку внутри текущего сегмента кода. Модифицируется только регистр **eip/ip** (в зависимости от типа сегмента кода use16/32) на основе указанного в команде адреса (метки) или выражения;

```
JMP pt ;Переход на метку pt в пределах текущего сегмента
JMP near ptr pt ;то же самое
```

- ***far ptr*** — прямой переход на метку в другом сегменте кода. Адрес перехода задается в виде непосредственного операнда или адреса (метки) и состоит из 16-битного селектора **cs** и 16/32-битного смещения **ip/eip**;

```
JMP far ptr farpt ;Переход на метку farpt в другом программном
сегменте
JMP farpt ;Переход на метку farpt в другом
;программном сегменте, если farpt
;объявлена дальней меткой директивой farpt label far
```

- ***word ptr*** — косвенный переход на метку внутри текущего сегмента кода. Модифицируется (значением смещения из памяти по указанному в команде адресу, или из регистра) только **eip/ip**. Размер смещения 16 или 32 бит;

```
;В полях данных:
addr dw pt ;Ячейка с адресом точки перехода
;В программном сегменте:
jmp DS:addr ;Переход в точку pt
JMP word ptr addr ;То же самое
```

- ***dword ptr*** — косвенный переход на метку в другом сегменте кода. Модифицируются (значением из памяти, **из регистра нельзя**) оба регистра, **cs** и **eip/ip**. Первое слово/двойное слово этого адреса представляет смещение и загружается в **ip/eip**; второе/третье слово загружается в **cs**.

```
; В полях данных:
addr dd pt ;Поле с двухсловным
;адресом точки перехода ;В программном сегменте:
jmp DS:addr ;Переход в точку pt
JMP dword ptr addr ;То же самое
```

Кроме прямого перехода, можно осуществить непрямой переход. При этом в команде **JMP** отмечается имя слова, где хранится адрес перехода, или имя регистра, где записывается адрес перехода.

Примеры косвенных ближних переходов

Пример 1

```
mov BX,offset pt ;BX=адрес точки перехода
jmp BX ;Переход в точку pt
```

Пример 2

```
; В полях данных:
addr dw pt ;Ячейка с адресом точки перехода
;В программном сегменте:
mov DI,offset addr ;B1=адрес ячейки с адресом
;точки перехода
jmp [DI] ;Переход в точку pt
```

Пример 3

```
;В полях данных:
tbl dw ptl ;Ячейка с адресом 1
dw pt2 ;Ячейка с адресом 2
dw pt3 ;Ячейка с адресом 3
;В программном сегменте:
mov BX,offset tbl ;BX=aflpec таблицы адресов переходов
mov SI, 4 ;31=смещение к адресу pt3
call [BX][SI] ;Переход в точку pt3
```

2.4.2. Команды условного перехода

Всего предусматривается ***17 команд*** условного перехода. Но некоторые команды имеют несколько синонимов (мнемоник). Поэтому иногда говорят о ***31 команде***.

Эти команды позволяют проверить:

1. отношение между операндами со знаком (“больше — меньше”);
2. отношение между операндами без знака (“выше — ниже”);
3. состояния арифметических флагов **zf**, **sf**, **cf**, **of**, **pf** (но Не **af**).

Команды условного перехода имеют одинаковый синтаксис:
JCC метка_перехода

Как видно, мнемокод всех команд начинается с “**J**” — от слова jump (прыжок), **CC** — определяет конкретное условие, анализируемое командой.

Что касается операнда **метка_перехода**, то эта метка может находится *только в пределах текущего сегмента* кода, межсегментная передача управления в условных переходах не допускается.

Команда сравнения **CMP** имеет интересный принцип работы. Он абсолютно такой же, как и у команды вычитания.

CMP операнд_1, операнд_2 (compare) — сравнивает два операнда, вычитая **операнд_2** из **операнд_1**, не изменяя их и по результату устанавливает флаги.

Таблица

Значение аббревиатур в названии команды JCC

Мнемоническое обозначение	Английский	Русский	Тип операндов
E или e	equal	Равно	Любые
N или n	not	Не	Любые
G или g	greater	Больше	Числа со знаком
L или l	less	Меньше	Числа со знаком
A или a	above	Выше, в смысле “больше”	Числа без знака
B или b	below	Ниже, в смысле “меньше”	Числа без знака

Команды реакции на арифметические сравнения со знаком

Для таких сравнений используются слова “меньше” (less) и “больше” (greater). Ясно, что можно проверить 6 условий:

Типы операндов	Мнемокод команды условного перехода	Критерий условного перехода	Значения флагов для осуществления перехода
Любые	JE	операнд_1 = операнд_2	zf = 1
Любые	JNE	операнд_1<>операнд_2	zf = 0
Со знаком	JL / JNGE	операнд_1 < операнд_2	sf <> of
Со знаком	JLE / JNG	операнд_1 <= операнд_2	sf <> of or zf = 1
Со знаком	JG / JNLE	операнд_1 > операнд_2	sf = of and zf = 0
Со знаком	JGE / JNL	операнд_1 => операнд_2	sf = of

Например:

```
CMP AX, BX
JL LABEL2; переход, если AX<BX
```

Команды реакции на арифметические сравнения без знака

Типы операндов	Мнемокод команды условного перехода	Критерий условного перехода	Значения флагов для осуществления перехода
Любые	JE	операнд_1 = операнд_2	zf = 1
Любые	JNE	операнд_1<>операнд_2	zf = 0
Без знака	JB / JNAE	операнд_1 < операнд_2	cf = 1
Без знака	JBE / JNA	операнд_1 <= операнд_2	cf = 1 or zf=1
Без знака	JA / JNBE	операнд_1 > операнд_2	cf = 0 and zf = 0
Без знака	JAE / JNB	операнд_1 => операнд_2	cf = 0

Для таких сравнений используются слова “выше” (above) и ”ниже” (below), после сравнения (**CMP**) адресов:

Команды проверки отдельных флагов и регистров

Мнемоническое обозначение некоторых команд условного перехода отражает название флага, с которым они работают, и имеет следующую структуру: первым идет символ “**J**” (jump, переход), вторым — либо обозначение флага, либо символ отрицания “**N**”, после которого стоит название флага.

Название флага	№ бита в eflags/flag	Команда условного перехода	Значение флага для осуществления перехода
Флаг переноса cf	1	JC	cf = 1
Флаг четности pf	2	JP	pf = 1
Флаг нуля zf	6	JZ	zf = 1
Флаг знака sf	7	JS	sf = 1
Флаг переполнения of	11	JO	of = 1
Флаг переноса cf	1	JNC	cf = 0
Флаг четности pf	2	JNP	pf = 0
Флаг нуля zf	6	JNZ	zf = 0
Флаг знака sf	7	JNS	sf = 0
Флаг переполнения of	11	JNO	of = 0

Типы операндов	Команда условного перехода	Критерий условного перехода	Зн. регистров для перехода
Любые	JCXZ	Jump if **cx** is Zero	cx = 0
Любые	JECXZ	Jump Equal **ecx** Zero	ecx = 0

Например:

```
CMP AX, BX
JE cycl
JCXZ    m1  ; обойти цикл, если cx=0
cycl:;некоторый цикл
LOOP  cycl
m1:     ...
```

2.4.3. Команды управления циклами

Поскольку в программах часто придется реализовывать циклические процедуры, то для этого существует несколько команд.

LOOP метка - уменьшает регистр **CX** на 1 и осуществляет переход на определенную метку, если содержание **CX ≠ 0**, если **CX** = 0, осуществляется переход на следующую за **LOOP** команду.

Эти команды используют регистр **CX** как счетчик цикла.

Следовательно, **LOOP** должна завершать цикл. В начале цикла в **CX** необходимо занести количество повторений. То есть, фрагмент может иметь такой вид:

```
MOV CX, LOOP_COUNT
BEGIN_LOOP:
<тело цикла>
LOOP BEGIN_LOOP
```

Осуществится столько повторений, какое число занесли в регистр **CX**. Ясно, что в теле цикла регистр **CX** дополнительно изменять **НЕЛЬЗЯ**.

Поскольку при реализации циклов проверяется содержание регистра **CX**, то для этого существует специальная команда условного перехода **JCXZ** (if CX is zero), когда содержание **CX** равняется нулю.

Что будет, когда в регистр **CX** попадет не какое-то число, а ноль? В этом случае будет 65536 повторений. Чтобы этого не случилось, можно включить специальную проверку **CX** на ноль:

```
MOV CX, LOOP_COUNT
JCXZ END_LOOP
BEGIN LOOP: <тело>
LOOP BEGIN_LOOP
END_LOOP: <продолжение>
```

Существует две разновидности команды **LOOP**, которые учитывают дополнительные условия.

LOOPE/LOOPZ метка - команда продолжается до тех пор, пока ***CX ≠ 0*** или пока ***ZF=1.***

Таким образом, эту команду удобно применять для поиска первого ненулевого элемента в массиве без перебора всего массива до конца.

LOOPNE/LOOPNZ метка - команда продолжает цикл пока ***CX ≠ 0*** или пока ***ZF = 0***.

Эта команда удобная для поиска первого нулевого элемента массива.

Например:

```
; начальный адрес массива в BX
; конечный адрес массива в DI
SUB DI, BX; разница адресов
INC DI; количество байт в массиве
MOV CX, DI; счетчик цикла
DEC BX
NEXT: INC BX; к следующему элементу
        CMP BYTE PTR [BX], 0; сравнить с нулем
        LOOPE NEXT ; если 0 - продолжаем
        JNZ NZ_FOUND; найден ненулевой элемент
NZ_FOUND:
```

В приведенном фрагменте использован псевдооператор **PTR**. Его назначение - локально отменять типы **DB, DW, DD** или атрибуты дистанции **NEAR, FAR**. Употребляется с атрибутами **BYTE, WORD** и т.п. Здесь он используется для того, чтобы адрес **BX** толковать, как адрес байта или слова.

Отметим, что команды цикла, как и другие команды условного перехода, действуют в границах -128, 127 байт. Если объем команд больший, то тогда нужно иначе организовать фрагмент с использованием команды **JMP**.

```
model small
.stack 100h
.data
mas   db 1,0,9,8,0,7,8,0,2,0
          db 1,0,9,8,0,7,8,0,2,0
          db 1,0,9,8,0,7,8,?,2,0
          db 1,0,9,8,0,7,6,?,3,0
          db 1,0,9,8,0,7,8,0,2,0
. code
start:
 mov ax,@data
 mov ds.ax
 xor ax, ax
 lea bx, mas
 mov cx, 5
 cycl_l:
  push cx
  xor si , si
  mov cx, 10
  cycl_2:
    cmp byte ptr [bx+si], 0
    jne no_zero
    mov byte ptr [bx+si], 0ffh
    no_zero:
    inc si
  loop cycl_2
 pop cx
 add bx, 10
 loop cycl_1
exit:
mov ax,4c00h
int 21h
end start
```

2.4.4. Команды вызова процедур

CALL цель - вызов процедуры или задачи

```
                    TITLE   CALLPROC (EXE) Вызов процедур
0000                STACKSG SEGMENT PARA STACK 'Stack'
0000    20 [ ???? ]         DW      32 DUP(?)
0040                STACKG  ENDS

0000                CODESG  SEGMENT PARA 'Code'
0000                BEGIN   PROC    FAR
                            ASSUME  CS:CODESG,SS:STACKSG
0000  1E                     PUSH    DS
0001  2B C0                  SUB     AX,AX
0003  50                     PUSH    AX
0004  E8 0008 R              CALL    B10     ;Вызвать B10
                    ;       ...
0007  CB                     RET             ;Завершить программу
0008                BEGIN   ENDP
                            ;--------------------------------------
0008                B10     PROC
0008  E8 000C R              CALL    C10     ;Вызвать C10
                    ;       ...
000B  C3                     RET             ;Вернуться в
000C                B10     ENDP            ; вызывающую программу
                    ;---------------------------------------------
000C                C10     PROC
                    ;       ...
000C  C3                     RET             ;Вернуться в
000D                C10     ENDP            ; вызывающую программу
                    ;---------------------------------------------
000D                CODESG  ENDS
                            END     BEGIN
```