Лекция 4

1.3.5. Десятичные числа

Десятичные числа — специальный вид представления числовой информации, в основу которого положен принцип кодирования каждой десятичной цифры числа группой из четырех бит. При этом каждый байт числа содержит одну или две десятичные цифры в так называемом двоично-десятичном коде (BCD — Binary-Coded Decimal). Микропроцессор хранит BCD-числа в двух форматах (рис. Рис. 1.3.1):

упакованном формате — в этом формате каждый байт содержит две десятичные цифры. Десятичная цифра представляет собой двоичное значение в диапазоне от 0 до 9 размером 4 бита. При этом код старшей цифры числа занимает старшие 4 бита. Следовательно, диапазон представления десятичного упакованного числа в одном байте составляет от 00 до 99;

неупакованном формате — в этом формате каждый байт содержит одну десятичную цифру в четырех младших битах. Старшие четыре бита имеют нулевое значение. Это так называемая зона. Следовательно, диапазон представления десятичного неупакованного числа в одном байте составляет от 0 до 9.

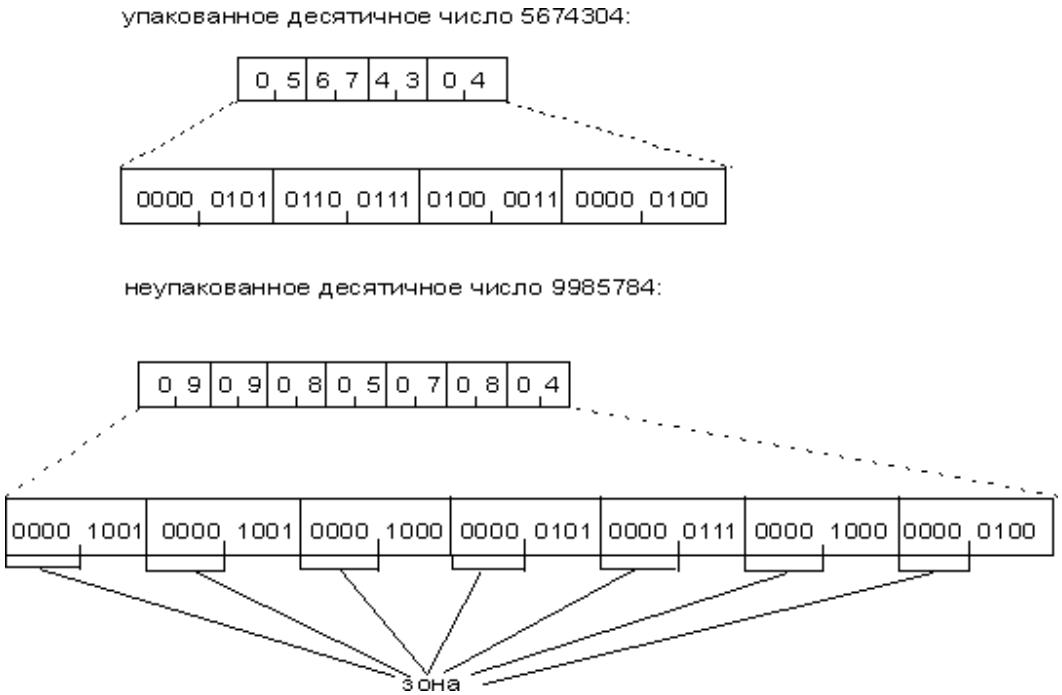


Рис. 1.3.1. Представление числовой информации в двоично-десятичном коде (BCD — Binary-Coded Decimal)

Для этого можно использовать только две директивы описания и инициализации данных — **db** и **dt**. Возможность применения только этих директив для описания BCD-чисел обусловлена тем, что к таким числам также применим принцип “младший байт по младшему адресу”, что, как мы увидим далее, очень удобно для их обработки. И вообще, при использовании такого типа данных как BCD-числа, порядок описания этих чисел в программе и алгоритм их обработки — это дело вкуса и личных пристрастий программиста. Это станет ясно после того, как мы ниже рассмотрим основы работы с BCD-числами. К

примеру, приведенная в сегменте данных листинга последовательность описаний BCD-чисел будет выглядеть в памяти так, как показано на рис. 1.3.2..

```
DSEG segment para PUBLIC "DATA"
SOURCE db 1,5,9,8
DESTpak Dt 232456791
DEST Dd 232456791
DSEG ENDS

; СЕГМЕНТ КОДУ

CSEG SEGMENT PARA PUBLIC "CODE"
MAIN PROC FAR
ASSUME CS:CSEG,DS:DSEG, SS:STSEG

; АДРЕСА ПОВЕРНЕННЯ

PUSH DS
XOR AX,AX ;AX=0
PUSH AX

;ІНІЦІАЛІЗАЦІЯ DS
```

```
cs:0000 1E              push    ds
cs:0001 33C0            xor     ax,ax
cs:0003 50              push    ax
cs:0004 B8D35B          mov     ax,5BD3
cs:0007 8ED8            mov     ds,ax
cs:0009 B8004C          mov     ax,4C00
cs:000C CD21            int     21
cs:000E 0000            add     [bx+si],al
cs:0010 FB              sti
cs:0011 52              push    dx
cs:0012 0003            add     [bp+di],al
cs:0014 0E              push    cs
cs:0015 0000            add     [bx+si],al
cs:0017 0002            add     [bp+si],al
cs:0019 0017            add     [bx],dl

ds:0000 01 05 09 08 91 67 45 32 ☺♣○◘CgE2
ds:0008 02 00 00 00 00 00 57 02 ☻      W☻
ds:0010 DB 0D 00 00 00 00 00 00 ■♪
ds:0018 00 00 00 00 00 00 00 00
ds:0020 1E 33 C0 50 B8 D3 5B 8E ▲3└P╕╙[O
```

```
DSEG segment para PUBLIC "DATA"
SOURCE db 1,5,9,8
DESTpak Dt 232456791
DEST Dd 232456791
DSEG ENDS

; СЕГМЕНТ КОДУ

CSEG SEGMENT PARA PUBLIC "CODE"
MAIN PROC FAR
ASSUME CS:CSEG,DS:DSEG, SS:STSEG

; АДРЕСА ПОВЕРНЕННЯ

PUSH DS
XOR AX,AX ;AX=0
PUSH AX

;ІНІЦІАЛІЗАЦІЯ DS
```

```
cs:0001 33C0            xor     ax,ax
cs:0003 50              push    ax
cs:0004 B8D35B          mov     ax,5BD3
cs:0007 8ED8            mov     ds,ax
cs:0009 B8004C          mov     ax,4C00
cs:000C CD21            int     21
cs:000E 0000            add     [bx+si],al
cs:0010 FB              sti
cs:0011 52              push    dx
cs:0012 0003            add     [bp+di],al
cs:0014 0E              push    cs
cs:0015 0000            add     [bx+si],al
cs:0017 0002            add     [bp+si],al
cs:0019 0017            add     [bx],dl

ds:0000 01 05 09 08 91 67 45 32 ☺♣○◘CgE2
ds:0008 02 00 00 00 00 00 57 02 ☻      W☻
ds:0010 DB 0D 00 00 00 00 00 00 ■♪
ds:0018 00 00 00 00 00 00 00 00
ds:0020 1E 33 C0 50 B8 D3 5B 8E ▲3└P╕╙[O
```

```
DSEG segment para PUBLIC "DATA"
SOURCE db 1,5,9,8
DESTpak Dt 232456791
DEST Dd 232456791
dfg db 03
DSEG ENDS

; СЕГМЕНТ КОДУ

CSEG SEGMENT PARA PUBLIC "CODE"
MAIN PROC FAR
ASSUME CS:CSEG,DS:DSEG, SS:STSEG

; АДРЕСА ПОВЕРНЕННЯ

PUSH DS
```

```
#lab1#26: PUSH AX
  cs:0003 50              push    ax
#lab1#30: MOV AX,DSEG
  cs:0004 B8D35B          mov     ax,5BD3
#lab1#31: MOV DS,AX
  cs:0007 8ED8            mov     ds,ax
#lab1#33: mov ax,4c00h
  cs:0009 B8004C          mov     ax,4C00
#lab1#34: int 21h
  cs:000C CD21            int     21
  cs:000E 0000            add     [bx+si],al

  ds:0000 01 05 09 08 91 67 45 32 ☺♣○◘CgE2
  ds:0008 02 00 00 00 00 00 57 02 ☻      W☻
  ds:0010 DB 0D 03 00 00 00 00 00 ■♪♥
  ds:0018 00 00 00 00 00 00 00 00
  ds:0020 1E 33 C0 50 B8 D3 5B 8E ▲3└P╕╙[O
```

Рис. 1.3.2. Листинги описаний BCD-чисел

1.4. Режимы адресации

При выполнении программы процессор обращается к памяти, где сохраняются команды и данные. В командах превращения данных определяются адреса, где сохраняется соответствующая информация, а в командах передачи управления определяется адрес команды, на которую нужно перейти, то есть, адреса переходов.

Способ или метод определения в команде адреса операнда или адреса перехода, называется *режимом адресации*, просто ли *адресацией*.

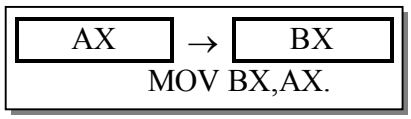
Можно в командах отмечать адреса операндов, то есть, использовать прямую адресацию. Но операнды сохраняются не только в ячейках памяти. Они могут находиться в регистрах общего назначения, сегментных регистрах. Кроме того, операндами могут быть константы, или операнды могут находиться в портах ввода/вывода.

В таких условиях использования только прямой адресации приведет к неэффективным программам. Поэтому в современных ЭВМ используют много других режимов адресации, что позволяет получать высокоэффективные программы для разных приложений.

Режимы адресации МП 8086 можно разделить на 7 групп:

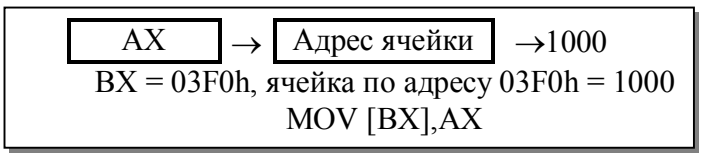
1. регистровая адресация;
2. непосредственная адресация;
3. прямая;
4. непрямая регистровая;
5. адресация за базой;
6. прямая адресация с индексированием;
7. адресация за базой с индексированием.

В последующем будем считать, что когда в команде отмечено имя регистра, то операндом будет его содержание.



Например:

Если же имя регистра заключено в прямоугольные скобки, то значат, что операндом является содержание ячейки, адрес которого сохраняется в регистре.



В первом случае - *прямая* адресация, а во втором – *непрямая(косвенная)*.

1.4.1. Регистровая адресация

Операнд (байт или слово) находится в регистре. Этот способ адресации применим ко всем программно-адресуемым регистрам процессора.

В этом случае операндом является содержание определенного регистра.

16-битовое слово, которое сохраняется в счетчике CX, переписывается в аккумулятор AX. CX останется неизменным, а AX изменится.

Аналогично для байтов MOV AL, BH.

То есть, сам операнд определяет свою длину - слово или байт. Этот метод не нуждается в обращении к памяти, сама команда занимает мало места. Потому выполняется очень быстро и является очень эффективной.

```
inc CH ;Плюс 1 к содержимому CH

push DS ;DS сохраняется в стеке

xchg BX,BP ;BX и BP обмениваются содержимым

mov ES, AX ;Содержимое AX пересылается в ES
```

1.4.2. Непосредственная адресация

В этом случае вместо операнда источника используется непосредственно константа:

```
MOV AX, 60
```

В аккумулятор заносится число 60. Эта константа размещается не в памяти, а в самой машинной команде, то есть, в очереди команд. Потому будет выполняться достаточно быстро. Можно:

```
MOV CL, -50
```

Константа может представляться *литералом*, а может быть и *именуемой*. Имя константе присваивается специальным оператором: EQU:

```
L EQU 256
……
MOV CX,L
```

1.4.3. Прямая адресация

Как отмечалось, в указателе команд **IP** сохраняется относительный адрес команды в сегменте, то есть, количество байт относительно его начала, или *исполнительный адрес*.

Для прямой адресации исполнительный адрес отмечается непосредственно в команде. Если это адрес данных, то МП добавляет ее к содержанию регистра данных **DS**, который сдвигается на 4 бита и получает 20-битовый абсолютный адрес.

Использовать конкретные числовые значения адресов неудобно. Поэтому адрес чаще задается меткой.

Чтобы разместить какое-то число по этому адресу, используются операторы ***DB, DW*** или ***DD*** (Define Byte, Define Word, Define Double Word).

```
TABLE DW 1560;          в ячейку TABLE записано 1560
INDEX DB -126;          в байт INDEX записано -126
```

Тогда можно записать:

```
MOV AX,TABLE;       переслать содержание TABLE в аккумулятор.
```

Отметим особенность такой пересылки. Когда в памяти было записано:

X	**TABLE**
Y	**TABLE+1**

то младший байт X будет переслан в младший байт регистра, а старший - в старший. И получим:

Y	**X**
AH	**AL**

То есть, байты будто поменялись местами (смещение записывается до команды).

1.4.4. Непрямая регистровая адресация

Этот способ адресации использует базовый регистр **BX**, указатель **BP** и индексные регистры SI, BI, где записан адрес операнда:

```
MOV AX,[BX]
```

Для выбора операнда-источника происходит обращение к регистру **BX**, где сохраняется его адрес. Если там записано 2000, то содержание ячейки 2000 пересылается в аккумулятор.

А как в регистр **BX** занести адрес ячейки, например, TABLE? Это можно с помощью операции **OFFSET** (смещение).

Например:

```
MOV BX, OFFSET TABLE
Сравните: MOV BX, TABLE.
```

1.4.5. Адресация за базой

Если необходимо получить доступ к одной ячейке, то ей нужно предоставить имя и использовать прямую адресацию:

```
MOV AX,TABLE
```

При работе с массивом данных помечать своим именем каждое слово нецелесообразно, а достаточно запомнить адрес начала массива, например, в регистре **BX** или **BP**.

Тогда адрес любого элемента массива определится как сумма базового адреса и целой константы ***[BP] + N***, где N -- количество байт от начала массива (смещение). Если начальный адрес массива записать в регистр **BP**, то второе слово можно переслать в аккумулятор так:

```
MOV AX, [BP] + 2
```

Исполнительный адрес будет определен как сумма содержания **BP** и определенного смещения. Такой метод адресации называется *адресацией за базой.*

Выше приведенная запись имеет и другие эквивалентные формы:

```
MOV AX, 2[BP]
MOV AX, [BP + 2]
```

Ясно, что смещение может быть и отрицательным.

То есть, выбрав необходимое смещение, можно произвольно адресовать элементы массива.

1.4.6. Прямая адресация с индексированием

Если зафиксировать базовый адрес элементов данных определенной меткой, тогда добраться до других элементов данных можно с помощью индексных регистров **SI, DI**.

Например:

```
MOV DI, 2
MOV AX, TABLE
```

Исполнительный адрес определится как сумма адреса TABLE и регистра **DI**. В данном случае в аккумулятор **AX** будет переслано второе слово после TABLE.

Это *прямая адресация с индексированием*. Этот метод адресации удобно использовать для регулярной обработки массивов. Если вторую команду разместить в цикле и там же добавлять 2 к **DI**, то можно обрабатывать все слова массива TABLE.

1.4.7. Адресация за базой с индексированием

Для обработки двумерных массивов удобно использовать адресацию за базой с индексированием, когда исполнительный адрес равняется сумме значений базового регистра, индексного регистра и сдвига.

```
MOV AX, VALUE [BX] [DI]
```

Здесь VALUE - именуемая константа, а не адрес ячейки. Вместо имени переменной можно задавать адресную константу. Например:

```
MOV AX, 2[BP] [SI]
```

Операнды в скобках можно записывать по-разному:

```
MOV AX, [BP + 2 + SI];
MOV AX, [SI + BP + 2];
MOV AX, [BP] [SI + 2].
```

Ясно, что сдвига может не быть. Такие основные методы адресации. Однако, это далеко не все методы адресации.