Глава X. Управление ресурсами памяти

> Мозг, хорошо устроенный, стоит больше, чем мозг, хорошо наполненный.
>
> *—Мишель Монтень*

X.1. Карта памяти (для компьютеров MSX 1)

Персональный MSX–компьютер имеет небольшой объем памяти — 96 Кбайт для MSX 1 и 242 Кбайта для MSX 2. Поэтому полезной для пользователя оказывается информация о распределении ресурсов памяти и сведения о наличии и объёме в ней свободных мест в любой момент времени.

Общий объем памяти у компьютеров серии MSX 1 равен 96 Кбайт. Здесь и далее мы будем рассматривать только 64 Кбайта, с которыми обычно и работает основная масса пользователей.

Взгляните на приведённый ниже рис. 1...

Вся память разбита на две основные части:

- ROM («Read Only Memory» — «Постоянное Запоминающее Устройство») и
- RAM («Random Access Memory» — «Оперативное Запоминающее Устройство»)

ROM содержит те программы и данные, которые «заложены» в компьютер при изготовлении. Вот почему он всегда выводит определённые сообщения при включении и способен «понимать» программу на языке MSX BASIC.

В ROM находится *интерпретатор* — программа на машинном языке, которая переводит один за другим операторы языка MSX BASIC в программу на машинном языке, т.е. на *единственном* языке, который понимает компьютер. С помощью этой программы компьютер проверяет синтаксис, выводит при необходимости сообщение об ошибке, переходит к следующему оператору или возвращается в командный режим и так далее.

Здесь же находятся подпрограммы управления клавиатурой и экраном, которые составляют *экранный редактор* MSX BASIC.

ROM в основном разделена на две части:

1. подпрограммы BIOS («Basic Input–Output System»);
2. другие подпрограммы. Так, например, при включении компьютера наступает небольшая пауза; в этот момент происходят различные инициализации экрана дисплея (установка определённого режима `SCREEN`, установка ширины экрана `WIDTH` и др.). Это происходит оттого,что «зашитые» в ROM подпрограммы инициализации «посылают» определённую информацию в рабочую область RAM, разговор о которой ещё пойдёт впереди.

Подпрограммы BIOS осуществляют переход к другим подпрограммам. Они напоминают последовательность операторов `GOSUB`, которую можно увидеть на первом уровне хорошо структурированной программы на MSX BASIC. Подпрограммы BIOS расположены по одним и тем же адресам ROM независимо от версии MSX BASIC и осуществляют переход к другим подпрограммам, положение которых может быть изменено.

В противоположность ROM, RAM не сохраняет информацию при выключении компьютера. Поговорим теперь о его структуре.

«Верхушка» памяти (она изображена в *нижней* части таблицы) занята *рабочей областью*, которая состоит из:

- таблицы системных переменных,
- таблицы ловушек («Hooks Table»).

«Нижняя» область памяти (она изображена в *верхней* части таблицы) занята:

1. текстом программы («Program Instruction Table», PIT);
2. таблицей переменных («Variable Table», VT). VT содержит все переменные, создаваемые в ходе выполнения программы;
3. таблицей массивов («Array Variable Table»).

Между «верхней» и «нижней» областью памяти располагаются:

- свободная область («Free Area»);
- *стек* («Stack Area»); стек содержит всю информацию, необходимую для выполнения программы. Например, именно здесь хранится адрес тех байт PIT, которые содержат сведения о следующем выполняемом операторе Вашей программы;
- *строковая* область («Character String Area»); по умолчанию для неё отводится 200 байт, однако размеры этой области можно менять оператором CLEAR (см. раздел X.7.);
- *блок управления* файлами («Files Control Block»).

Если в Вашей программе присутствует оператор MAXFILES= (напомним, что он задаёт максимальное количество одновременно открытых файлов), то для каждого файла автоматически резервируется 267–байтное пространство для осуществления обмена информацией с файловыми устройствами.

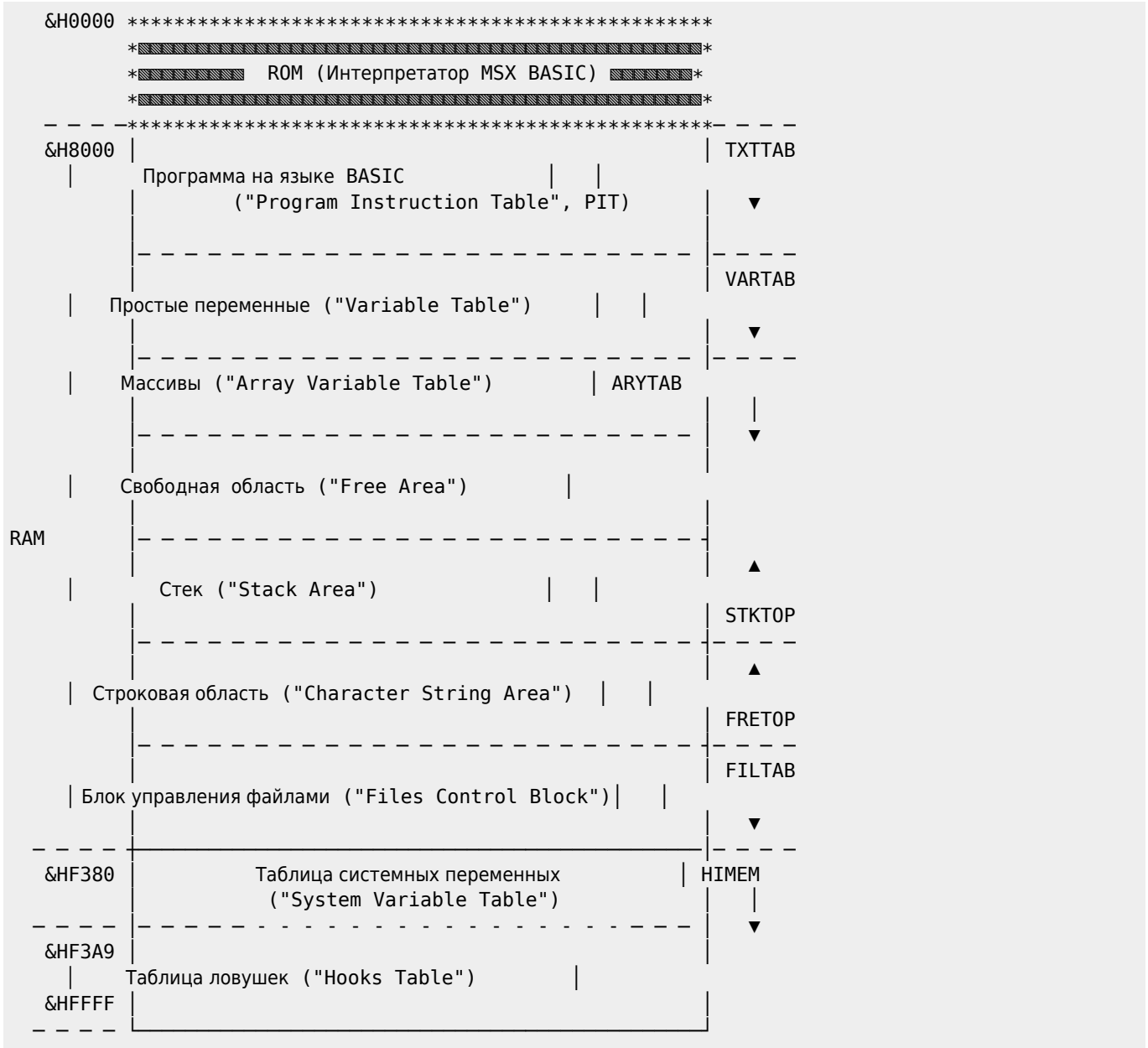


Рис. 1

Приведённая карта памяти справедлива и для компьютеров серии MSX 2. Но в отличие от компьютеров серии MSX 1 с объёмом ROM в 32 Кбайта и RAM в 64 Кбайта, компьютеры серии MSX 2 имеют гораздо больший объем памяти (108 Кбайт ROM и 134 Кбайта RAM). Спрашивается, где размещаются эта память?

Оказывается, вся память ПЭВМ разбита на блоки объёмом по 64 Кбайта, называемые *слотами* !

Однако рассмотрение этого вопроса потребует от читателя дополнительных знаний, и поэтому мы рассмотрим его позднее (см. раздел XI.1.9.).

Память разделена на ячейки (*байты*), каждая из которых имеет адрес, закодированный *двумя* байтами:

```
     Адрес ячейки (байта)
┌──────────────────────┬──────────────────────┐
│ 1 1 0 1 1 0 1 1 │ 1 0 1 1 1 1 0 1 │
└──────────▲───────────┴──────────▲───────────┘
     └──  Байты  ──┘
```

Поэтому максимально большой адрес байта равен

```
256·&B11111111+&B11111111+1 = 256·&hFF+&HFF+1 = 65535+1 = 65536 ,
```

а следовательно, и обратиться можно не более, чем к 65536 ячейкам памяти (подумайте, почему производится умножение именно на 256?).

Говорят, что «объём непосредственно адресуемой памяти — 65536 байта».

X.2. Функция PEEK и оператор POKE

Функция PEEK позволяет Вам «посмотреть» содержимое любой ячейки памяти в адресном пространстве MSX–компьютера. Её общий вид:

```
PEEK (адрес)
```
,

где:

- PEEK («to peek» — «заглядывать») — служебное слово;
- адрес — арифметическое выражение, значение которого находится в диапазоне от &h0 до &hFFFF.

Функция PEEK возвращает целое число в интервале от 0 до 255, содержащееся в проверяемой ячейке памяти.

Например:

1. ```
   10 WIDTH 7:? PEEK(&HF3B0)
   run
    7
   Ok
   ```

2. ```
   10 SCREEN 2:PSET(15,18):SCREEN0:PRINT PEEK(&HFCB3);PEEK(&HFCB5)
   run
    15  18
   Ok
   ```

В первом примере мы «попросили» компьютер вывести на экран содержимое ячейки с адресом &HF3B0 (в байте по этому адресу хранится значение систем ной переменной — длины дисплейной строки). Во втором примере мы использовали информацию из таблицы адресов системных переменных (см. Приложение 2[1)]).

Величину, возвращаемую функцией PEEK, можно интерпретировать как код символа, команду MSX BASIC, номер строки, число, «часть» числа, «хранящегося» в нескольких байтах, и т.д. В некоторых случаях правильную интерпретацию можно дать по контексту, однако, если должной уверенности нет, надо *анализировать* не только содержимое одной ячейки, но и содержимое ячеек, находящихся в её «окрестности»!

Пример 1.
102-01.bas
102-01.bas

```
10 X=&H8000'"Заглянем" в память,начиная с адреса &H8001!
```

```
20 X=X+1:Y=PEEK(X)
30 IF Y<32 THEN ?"|";:GOTO 20 ELSE ?CHR$(Y);" ";:GOTO20
```

Наличие условия Y<32 в строке 26 связано с тем, что существуют «непечатаемые» символы, имеющие код ASCII, меньший 32.

Поговорим теперь об очень полезном операторе POKE. Общий вид оператора:

```
POKE A, D
```
,

где:

- POKE («to poke» — «помещать») — служебное слово;
- A — арифметическое выражение, значение которого находится в диапазоне от &h8000 до &hFFFF;
- D — арифметическое выражение, значение которого принадлежит отрезку [0,255] (поскольку оно должно умещаться в один байт).

Оператор POKE вычисляет значения выражений A и D и сохраняет значение D (которое должно помещаться в одном байте!) по адресу A. Обратите внимание на то, что значение A может оказаться *отрицательным*!

Если значение A не удовлетворяет ограничениям, то компьютер сообщит об ошибке:

«Overflow» (*«Переполнение»*), а если значение D, то
«Illegal function call»
(*«Неправильный вызов функции»*).

Вы можете использовать оператор POKE для:

- модификации текста Вашей программы;
- изменения значений переменных;
- размещения в RAM программы, написанной на машинном языке (её запись производится *побайтно*).

Более того, этот оператор позволяет Вам экспериментировать с «подвалом» компьютера (рабочей областью). Но делайте так только в том случае, если Вы понимаете, что за этим последует!

Пример 2. Сравните результаты работы двух программ:

102-021.bas
102-021.bas

```
10 SCREEN 1:PRINT"A"
20 WIDTH 10
```

102-021.bas
102-021.bas

```
10 SCREEN 1:PRINT"A"
20 POKE &HF3B0,10
```

X.3. Таблица программных команд (PIT)

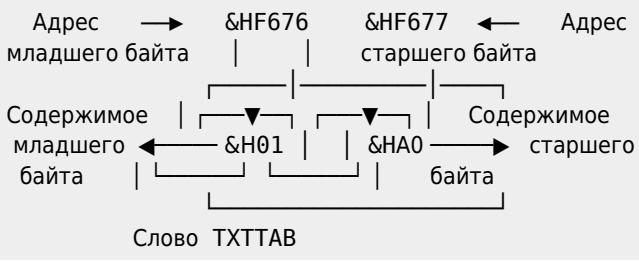
Таблица PIT обычно начинается по адресу &H8000. Однако её можно «сдвинуть», изменив значение системной переменной TXTTAB в таблице системных переменных.

Пример 1. Для помещения PIT с адреса &HA000,достаточно выполнить следующую программу:
103-01.bas
103-01.bas

```
NEW
Ok
5 'Адрес  &HA001,находящийся в двух ячейках с номерами, начиная с  &hF676  (слове  TXTTAB  (&HF676)),
6 'определяет место,с которого начнется текст программы
10  POKE  &HF676,&H01   'Заполнение младшего байта слова  TXTTAB
20  POKE  &HF677,&HA0   'Заполнение старшего байта слова  TXTTAB
30  POKE  &HA000,0      'Первый байт  PIT(&HA000)  должен быть нулевым!
```

```
40 NEW                    'Стираем данную программу!
```

Напомним Вам, что адрес &HA001 (как и любой другой!) размещается в двух байтах так:



Очевидно, что в результате этих «манипуляций» размер свободной области уменьшится на &H2000 байт (&HA000-&H8000=&H2000), и область, расположенная между &H8000 и началом PIT, будет защищена от «вторжения» программ на MSX BASIC, и следовательно, *«безопасна»* для программ на машинном языке.

Ясно, что величина PIT зависит от размера текста программы.

После выполнения данной программы нажмите кнопку сброса RESET.

А теперь мы расскажем Вам о том, как хранится программа, написанная на языке MSX BASIC в PIT.

Все строки программы на MSX BASIC начинаются с *двухбайтового* указателя. За этим указателем идут *два* байта, содержащие номер строки. Затем идёт текст строки с последующим нулевым байтом.

За последней строкой следуют *два* дополнительных нулевых байта адрес которых находится в указателе последней строки программы.

Цифры и зарезервированные служебные слова записываются во внутреннем коде (один или два байта на слово, цифру)

Для остального текста используется код ASCII.

Пример 2. Введём в память следующую короткую программу:

```
10 B=5
20 END
```

Теперь прочитаем, что же реально содержится в PIT, используя в непосредственном режиме команду

```
PRINT HEX$(PEEK(A))
```
,

где значение переменной A (адреса) изменяется от &H8000 до &H8010.

Вы обнаружите:

Значение A	HEX$(PEEK(A))	Комментарии
8000	0	Первый байт PIT всегда нулевой
8001	09	Указатель первой строки «говорит» нам, что указатель следующей строки находится по адресу &H8009
8002	80	
8003	A	Номер первой строки &H000A = 10
8004	0	
8005	42	Шестнадцатеричный код ASCII буквы «B»
8006	EF	Внутренний код знака равенства
8007	16	Внутренний код цифры 5
8008	0	Конец первой строки

Значение A	HEX$(PEEK(A))	Комментарии
8009	0F	Указатель второй строки показывает, что указатель следующей строки находится по адресу &H800F
800A	80	
800B	14	Номер второй строки &H0014 = 20
800C	00	
800D	81	Внутренний код оператора END
800E	0	Конец второй строки
800F	0	Конец программы
8010	0	

Теперь, надеемся, Вам стало ясно, как можно изменить программу с помощью оператора POKE.

Попробуйте выполнить следующее:

```
POKE &H8005,&H41   '41 - шестнадцатеричный код ASCII буквы "A"
POKE &H8007,&H17   '17 - внутренний код цифры "6"
```

А теперь наберите команду LIST, затем нажмите клавишу Ввод ⏎ и ... :

```
10 A=6
20 END
```

Пример 3.

Теперь Вам ясно, что «инструкции» PEEK и POKE таят в себе поистине безграничные возможности. По существу, они позволяют нам распоряжаться памятью компьютера по своему усмотрению.

Например, они позволяют нам при желании подшутить над компьютером: если известно, где хранится программа, то мы можем сделать так, что после одной из строк программы окажется строка с *меньшим* номером.

Пусть исходная программа имеет вид:

```
10 PRINT 4
20 PRINT 2
```

Вам, конечно, уже известно, что строки программы на языке MSX BASIC начинаются с двухбайтового указателя, за которым следуют два байта, содержащие номер строки. Поэтому вначале выполним команду:

```
PRINT HEX$(PEEK(&H8002));" ";HEX$(PEEK(&H8001))
80 9
Ok
```

Таким образом, указатель следующей (с номером 20) строки располагается в ячейках с адресами &H8009 и &H800A, а следовательно, номер второй строки находится в ячейках с адресами &H800B и &H800C . Проверим этот факт:

```
PRINT HEX$(PEEK(&H800C));" ";HEX$(PEEK(&H800B))        PRINT &H14
0 14    ──────────────────────────────────────────────►  20
Ok                                                     Ok

А теперь:
POKE &H800B,1
Ok
list
10 PRINT 4
1 PRINT 2
Ok
```

Программа действует, но строку с номером 1 нельзя ни стереть, ни исправить. Вы можете написать ещё одну 1–ю строку и даже новую 20–ю строку!

Пример 4.
Введём в память короткую программу:
103-04.bas
103-04.bas

```
10 FOR AB=-2.23227 TO 7 STEP 32.671533782376#
```

Теперь «просмотрим» содержимое PIT, используя в непосредственном режиме простейшие команды:

```
A=&H8000: PRINT HEX$(PEEK(A))
```
,

где значение переменной A (адреса) изменяется от &H8000 до &H8020.

Мы обнаружим массу интересных вещей:

Значение A	HEX$(PEEK(A))	Комментарии
8000	0	Первый байт PIT всегда нулевой
8001	21	Указатель первой строки «говорит» нам, что указатель следующей строки находится по адресу &H8021
8002	80	
8003	A	Номер первой строки &H000A = 10
8004	0	
8005	82	Код служебного слова FOR
8006	20	Внутренний код символа «пробел»
8007	41	Шестнадцатеричный код ASCII буквы «A»
8008	42	Шестнадцатеричный код ASCII буквы «B»
8009	EF	Внутренний код символа «=«
800A	F2	Внутренний код символа «-»
800B	1D	Указатель на тип одинарной точности (знак»!»)
800C	41	Вторая цифра числа указывает на порядок минимального значения параметра цикла
800D	22	1 и 2–я цифры мантиссы минимального значения параметра цикла
800E	32	3 и 4–я цифры мантиссы минимального значения параметра цикла
800F	27	5 и 6–я цифры мантиссы минимального значения параметра цикла
8010	20	Внутренний код символа «пробел»
8011	D9	Код служебного слова TO
8012	20	Внутренний код символа «пробел»
8013	18	Внутренний код символа «7»
8014	20	Внутренний код символа «пробел»
8015	DC	Код служебного слова STEP
8016	20	Внутренний код символа «пробел»
8017	1F	Указатель на тип двойная точность (знак #)
8018	42	Вторая цифра числа указывает на порядок шага
8019	32	1 и 2–я цифры мантиссы шага
801A	67	3 и 4–я цифры мантиссы шага
801B	15	5 и 6–я цифры мантиссы шага
801C	33	7 и 8–я цифры мантиссы шага
801D	78	9 и 10–я цифры мантиссы шага
801E	23	11 и 12–я цифры мантиссы шага

Значение A	HEX$(PEEK(A))	Комментарии
801F	76	13 и 14–я цифры мантиссы шага
8020	0	Конец строки
8021	0	Конец программы

Однако не пытайтесь изменять с помощью оператора POKE длину строки или путать указатели: результат будет *катастрофическим*!

Если Вы хотите защитить свою программу от «постороннего взгляда» (команды LIST), то примените в непосредственном режиме команду:

```
POKE &H8001,1
Ok
```

(разумеется, Ваша программа должна располагаться с адреса &H8000).

Ну а если Вы нечаянно нажали RESET, — не спешите отчаиваться! Вашу программу ещё можно спасти. Это очень легко сделать, набрав ту же команду

```
POKE &H8001,1

а затем
auto

На экране появятся строки:
10*
20*
и так далее ...
```

Строки с «*» — спасённые. Теперь достаточно «скомандовать»: LIST и ..., о, чудо! Но это ещё не все! Оказывается, спасены и все строки между теми, номера которых не делятся нацело на 10!

Если же Вы захотите защитить свою программу от запуска (команды RUN), то примените в непосредственном режиме команду:

```
POKE &H8000,1
Ok
```

(разумеется, Ваша программа должна располагаться с адреса &H8000).

X.4. Таблица переменных (VT)

Непосредственно следующая за PIT таблица VT начинается с адреса, указанного в слове VARTAB, хранящегося по адресу &HF6C2 в области системных переменных (см. Приложение 2[2)]). Её длина зависит от количества используемых переменных (скалярных и массивов) и их типов.

Отметим, что переменные и массивы хранятся в порядке их создания.

Будем говорить, что один программный объект «располагается в памяти *выше* другого», если адрес, с которого он расположен, больше.

Массивы хранятся *выше* переменных.

Это значит, что всякий раз, когда интерпретатор встречает новую скалярную переменную, все массивы сдвигаются *«вверх»*, чтобы высвободить пространство. Это может значительно замедлить выполнение программы!

Во избежание этого, описывайте все скалярные переменные и массивы в начале программы оператором DIM !

Теперь мы расскажем Вам о важной функции VARPTR, которая указывает адрес расположения данных в оперативной памяти. Её синтаксис:

```
VARPTR(γ)
```
,

где:

- VARPTR («VARiable PoinTeR» — «указатель переменной») — служебное слово;
- γ — идентификатор *числовой* переменной.

Функция VARPTR возвращает *адрес* X байта RAM, начиная с которого располагается значение переменной γ.

Если переменная не существует, то выдаётся сообщение:

«Illegal function call».

Пример. Будьте бдительны!
104-01.bas
104-01.bas

```
10 INPUT Z
20 PRINT VARPTR(Z)
run
? 0
-32743
Ok

run
?     ◄— Нажата клавиша "RETURN"
Illegal function call in 20
Ok
```

Функцию VARPTR часто используют совместно с функцией PEEK и оператором POKE соответственно для просмотра или изменения значения переменной.

X.4.1. Хранение простых переменных

Ты славно роешь землю, старый крот!
Годишься в рудокопы.

—В.Шекспир. Гамлет

Как уже неоднократно упоминалось, *целое* число кодируется в двух байтах. Меньший по адресу байт называется *старшим*, больший по адресу байт — *младшим*.

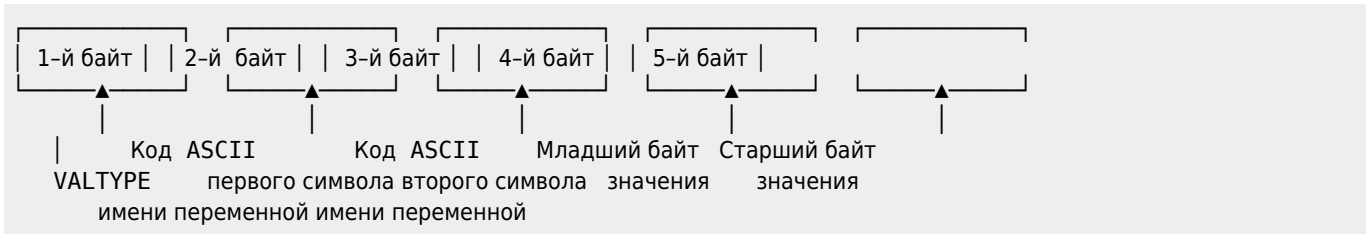
Однако, напомним Вам, что

процессор Z80 «хранит» младший байт «перед» старшим.

Когда *целочисленная* переменная получает значение, процессор записывает в оперативную память следующие *пять* байт информации:

1. число 2 («паспорт» VALTYPE), которое означает, что переменная является целочисленной (значение «паспорта» занимает два байта);
2. код ASCII первого символа имени переменной;
3. код ASCII второго символа имени (0, если имя состоит из одного символа);
4. младший байт значения;

5. старший байт значения.



Оказывается, что адрес четвёртого байта (младшего байта значения числовой переменной) возвращает как раз переменная VARPTR! Кроме того,напомним, что «содержимое» байта с известным адресом может быть «прочитано»функцией PEEK.

Перед тем как работать с нижеприведённым примером, во избежание расхождений в результатах не забудьте «почистить» память компьютера оператором CLEAR.

Пример 1

```
A%=356:PRINT HEX$(VARPTR(A%))
8006
Ok
```

Вы получили шестнадцатеричный адрес младшего байта значения переменной.

? PEEK(&H8003)	Этот адрес соответствует байту VALTYPE. Поэтому Вы должны получить 2.
? PEEK(&H8004)	Этот адрес соответствует первому символу имени переменной. Вы должны получить число 65, которое является кодом ASCII символа «A».
? PEEK(&H8005)	Получили число 0, поскольку имя переменной состоит лишь из одного символа.
? PEEK(&H8006)	Этот адрес, возвращённый функцией VARPTR, соответствует младшему байту значения. Должно быть получено число 100.
? PEEK(&H8007)	Этот адрес соответствует старшему байту значения. Вы, конечно же, получите число 1.

Перепишем два последних значения в двоичной системе счисления:

```
100 = 0110 0100₂
  1 = 0000 0001₂
```

А теперь примем во внимание инверсию порядка байт:

```
0000 0001 0110 0100  ⟶ 356₁₀
────▲──── ────▲────
    │         │
    1        100
```

И перед выполнением следующего примера не забудьте ввести в память компьютера оператор CLEAR !

Пример 2.

```
A%=-356:PRINT HEX$(VARPTR(A%))
8006
Ok
```

Полученный результат — это шестнадцатеричный адрес младшего байта.

? PEEK(&H8003)	Вы должны получить 2.
? PEEK(&H8004)	Этот адрес соответствует первому символу имени переменной. Вы должны получить 65 (код ASCII символа «A»).
? PEEK(&H8005)	0, т.к. имя переменной состоит лишь из одного символа.

? PEEK(&H8006)	Этот адрес, возвращаемый функцией VARPTR, соответствует инвертированному младшему байту значения. Разумеется, Вы получите число 156.
? PEEK(&H8007)	Этот адрес соответствует инвертированному старшему байту значения. Вы получите число 254.

Перепишем два последних значения в двоичной системе счисления:

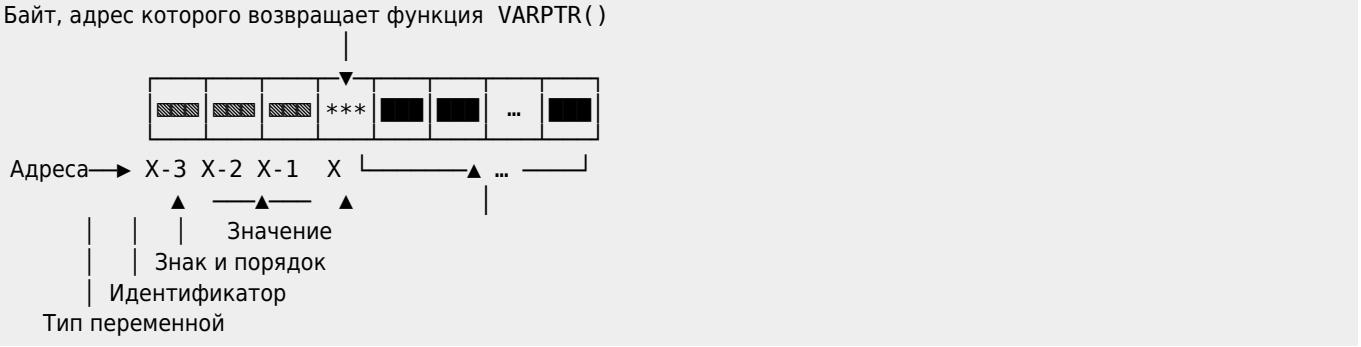
```
156 = 1001 1100₂
254 = 1111 1110₂
```

Принимая во внимание инверсию порядка байт, запишем их содержимое:

```
1111 1110 1001 1100
─────▲──── ─────▲────
     │           │
    254         156
```

А теперь учитывая, что двоичное число 1111111010011100₂ записано в дополнительном коде, получим 1111 1110 1001 1100₂ → 0000 0001 0110 0011₂ → 0000 0001 0110 0011+1 = 0000 0001 0110 0100₂ = 101100100₂ = 356. Ура!

Приведём схему расположения информации в памяти для простой числовой переменной *одинарной* и *двойной* точности:



Прежде чем проверить работу ниже приведённой командной строки,наберите команду CLEAR.

Пример 3.

```
AR!=22.3210E+4:PRINT HEX$(VARPTR(AR!))
8006
Ok
```

Если Вы прочитаете адреса с &H8006-&H3=&H8003 по &H8006+&H3=&H8009, то обнаружите:

&H8003	4	VALTYPE *переменной одинарной* точности (её значение занимает 4 байта)
&H8004	65	Код ASCII символа «A»
&H8005	82	Код ASCII символа «R»
&H8006	70 = &B 0100 0110	

Адрес &H8006 содержит порядок, который кодируется следующим образом:

Содержимое памяти по адресу &H8006		**Порядок**	**Примечание**
Двоичное значение	**Десятичное значение**		
01 000000	64	0	
01 000001	65	1	
01 000010	66	2	
…	…	…	
01 000110	70	6	←

Содержимое памяти по адресу &H8006		Порядок	Примечание
Двоичное значение	Десятичное значение		
…	…	…	
01 111110	126	62	
01 111111	127	63	
00 111111	63	- 1	
00 111110	62	- 2	
00 111101	61	- 3	
…	…	…	
00 000010	2	-62	
00 000001	1	-63	
00 000000	0	-64	Бит знака мантиссы

Величина порядка задаётся формулой:

Порядок = Двоичное значение 7 младших битов — 64.

Первый бит байта содержит *знак мантиссы*, причём 0 соответствует знаку «+», а 1 соответствует знаку «-». Продолжим наши исследования:

&H8007	34 = &B 0010 0010 = 22 в двоично–десятичной системе
&H8008	50 = &B 0011 0010 = 32 в двоично–десятичной системе
&H8009	16 = &B 0001 0000 = 10 в двоично–десятичной системе

В трёх последних байтах Вы, конечно же, «узнаете» число 225.

Итак, три последних адреса содержат мантиссу, записанную в двоично–десятичном виде (.223210).

Мантисса числа одинарной точности занимает 3 байта, которые позволяют закодировать 6 разрядов в двоично–десятичном виде.

Перед тем как выполнить предлагаемые в примере действия,наберите и введите в память компьютера оператор CLEAR.

Пример 4.

```
AR#=-22.321054981117E-4:PRINT HEX$(VARPTR(AR#))
8006
Ok
```

Если Вы прочитаете адреса с &H8006-&H3=&H8003 по &H8006+&H3=&H8009, то обнаружите:

&H8003	8	VALTYPE переменной *двойной* точности (её значение занимает 8 байт)
&H8004	65	Код ASCII символа «A»
&H8005	82	Код ASCII символа «R»
&H8006	190	&B 1011 1110 ▲ └— Знак мантиссы отрицательный!

Подсчитаем теперь величину порядка:

```
print &B0111110-64
-2
```

```
Ok
```

«Продолжим наши игры!»:

&H8007	34 = &B 0010 0010 = 22 в двоично–десятичной системе
&H8008	50 = &B 0011 0010 = 32 в двоично–десятичной системе
&H8009	16 = &B 0001 0000 = 10 в двоично–десятичной системе
&H800A	84 = &B 0101 0100 = 54 в двоично–десятичной системе
&H800B	152 = &B 1001 1000 = 98 в двоично–десятичной системе
&H800C	17 = &B 0001 0001 = 11 в двоично–десятичной системе
&H800D	23 = &B 0001 0111 = 17 в двоично–десятичной системе

В семи последних байтах «узнаётся» число 0.22321054981117.

Мантисса числа двойной точности занимает 7 байт, которые позволяют закодировать 14 разрядов в двоично-десятичном виде.

Подведём *итоги* всему сказанному о хранении числовых переменных.

Тип	Значение VALTYPE	Колич. байт	Номера байт	Значение (система счисления)	Примечание
Целая	2	5	-3	VALTYPE=2 (двоичная)	
			-2	Первый символ имени (код ASCII)	
			-1	Второй символ имени (код ASCII)	
			0	*Значение* (двоичная); для записи отрицательных чисел применяется двоичный дополнительный код;	«Содержимое» этого байта возвращает функция VARPTR()
			1		
Одинарной точности	4	7	-3	VALTYPE=4 (двоичная)	
			-2	Первый символ имени (код ASCII)	
			-1	Второй символ имени (код ASCII)	
			0	Порядок и знак (знак в первом бите) (двоичная) Величина порядка = двоичное значение семи последних битов — 64	«Содержимое» этого байта возвращает функция VARPTR()
			1÷3	*Мантисса* (двоично–десятичная)	
Двойной точности	8	11	-3	VALTYPE = 8 (двоичная)	
			-2	Первый символ имени (код ASCII)	
			-1	Второй символ имени (код ASCII)	
			0	Порядок и знак (знак в первом бите) (двоичная) Величина порядка = двоичное значение семи последних битов — 64	«Содержимое» этого байта возвращает функция VARPTR()
			1÷7	*Мантисса* (двоично–десятичная)	

Пример 5. Попробуйте самостоятельно в нем разобраться!
1041-05.bas
1041-05.bas

```
NEW
Ok
10 INPUT"Введите число";A:PRINT"Попробуем 'собрать' его из памяти"
30 B=VARPTR(A):K$=RIGHT$("00000000"+BIN$(PEEK(B)),8)
```

```
50 IF MID$(K$,1,1)="1" THEN Z$="-." ELSE Z$="+."
60 FOR T=1 TO 7:Z$=Z$+RIGHT$("00"+HEX$(PEEK(B+T)),2):NEXT
70 Z$=Z$+"E"
80 IF MID$(K$,2,1)="1" THEN Z$=Z$+"+" ELSE Z$=Z$+"-"
90 U=VAL("&b"+MID$(K$,2,7))-64
100 C$=MID$(STR$(U),2):Z$=Z$+RIGHT$("00"+C$,2)
120 PRINT"Вот Ваше число:";Z$

run
Введите число? 0
Попробуем 'собрать' его из памяти
Вот Ваше число:+.00000000000000E-64
Ok

run
Введите число? -23545e37
Попробуем 'собрать' его из памяти
Вот Ваше число:-.23545000000000E+42
Ok
```

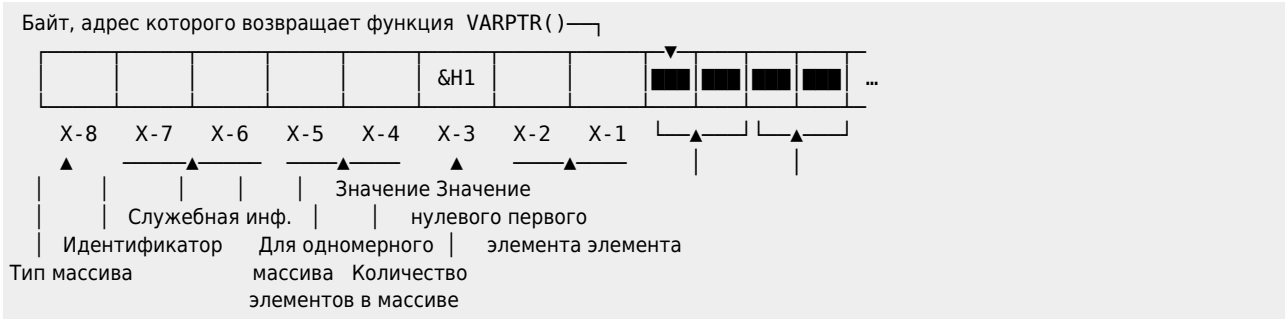
X.4.2. Хранение элементов числовых массивов

> Что имеем — не храним; потерявши — плачем.
>
> *—Козьма Прутков*

Вначале мы расскажем Вам о том, как хранится в памяти *целочисленный* массив.

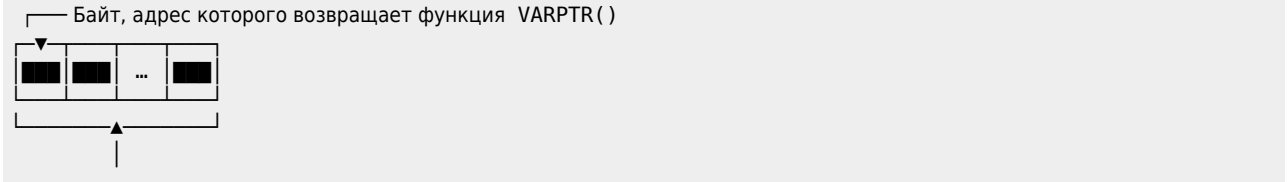
Приведём схемы расположения информации в памяти для целочисленных числовых массивов:

- *одномерный* массив.



- *двухмерный* массив.

- для *элементов* целочисленных числовых массивов.



```
Значение
```

Не забыли ли Вы набрать и ввести в память компьютера оператор CLEAR?

Пример.

```
DIM C5%(2):C5%(0)=32000:C5%(1)=13:C5%(2)=-4
Ok
print HEX$(VARPTR(C5%(0))-8)
8003
Ok
```

? PEEK(&H8003)	Этот адрес соответствует байту VALTYPE. Поэтому Вы должны получить 2.
? PEEK(&H8004)	Этот адрес соответствует первому символу имени массива. Вы получите число 67, которое является кодом ASCII символа «C».
? PEEK(&H8005)	Этот адрес соответствует второму символу имени массива Вы получите число 53, которое является кодом ASCII символа «5».
? PEEK(&H8006)	9 (служебная информация)
? PEEK(&H8007)	0 (служебная информация)
? PEEK(&H8008)	Массив C5% — одномерный, поэтому мы получили 1.
? PEEK(&H8009)	3 = 00000011_2
? PEEK(&H800A)	0 = 00000000_2

Теперь можно найти количество элементов в массиве: 00000000 00000011_2 =3

? PEEK(&H800B)	0 = 00000000_2
? PEEK(&H800C)	125 = 01111101_2

А тогда нулевой элемент массива равен: 01111101 00000000_2 = 32000

? PEEK(&H800D)	13 = 00001101_2
? PEEK(&H800E)	0

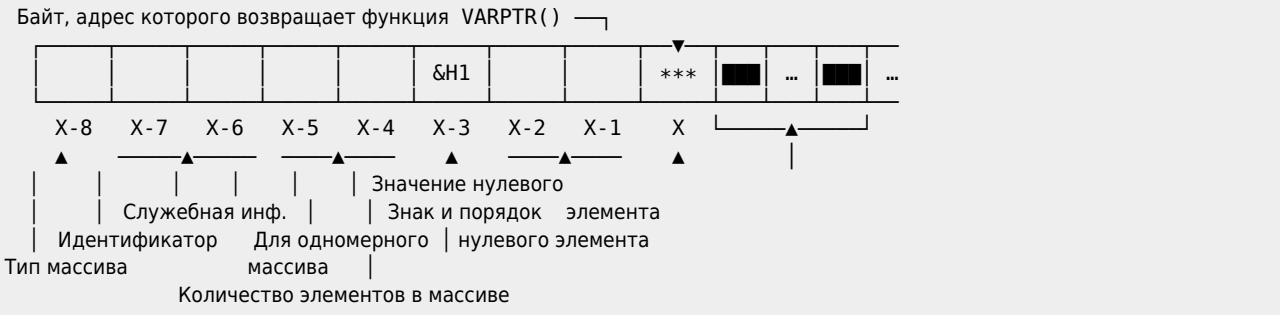
Добрались до первого элемента массива: 00000000 00001101_2 = 13

? PEEK(&H800F)	252 = 11111100_2
? PEEK(&H8010)	255 = 11111111_2

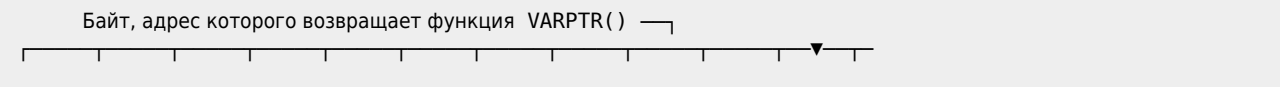
Далее 11111111 11111100_2 → 00000000 00000011_2 + 1 → 00000000 00000100_2 = 4

Приведём схемы расположения информации в памяти для нецелочисленных числовых массивов:

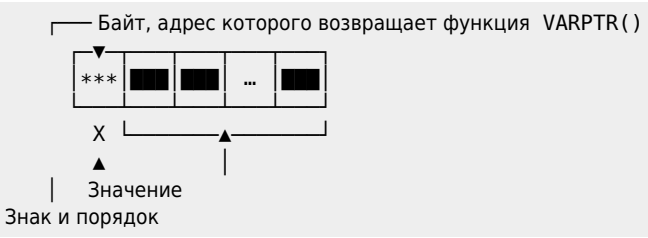
- *одномерный* массив.



- *двухмерный* массив.



- для *элементов* нецелочисленных числовых массивов.



Думаем, что теперь Вы в состоянии самостоятельно разобраться с вопросами, касающимися «хранения» в RAM многомерных (двухмерных, трёхмерных и т.д.) вещественных числовых массивов!

X.5. Стек

А люди все роптали и роптали,
А люди справедливости хотят:
— Мы в очереди первые стояли,
А те, кто сзади нас,— уже едят.

—В.Высоцкий

Стек (от англ. «stack» — «стог», «груда») — структура данных или устройство памяти для хранения наращиваемой и сокращаемой последовательности значений, в которой в любой момент доступен только последний член последовательности. Примером *стека* является стопка книг на столе, в которой брать и класть книги можно только сверху («Математический Энциклопедический Словарь»).

Стек используется как программой на MSX BASIC, так и подпрограммами на машинном языке.

«Вершина» стека указывается в слове `STKTOP(&HF674)`. Его позиция зависит от размеров строкового пространства и блоков управления файлами, а также от второго аргумента оператора `CLEAR` (если этот оператор был выполнен).

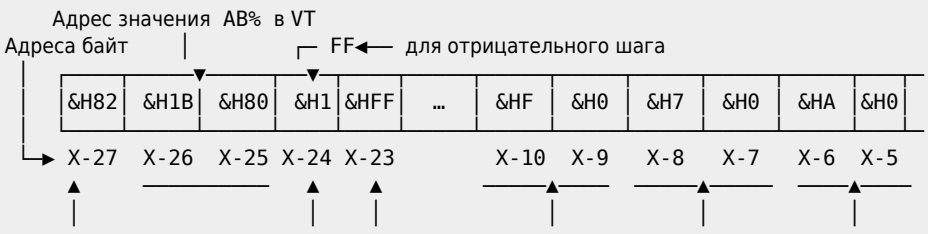
Если Вы хотите получить такие же результаты, как в последующем примере, воспользуйтесь командой `CLEAR`!

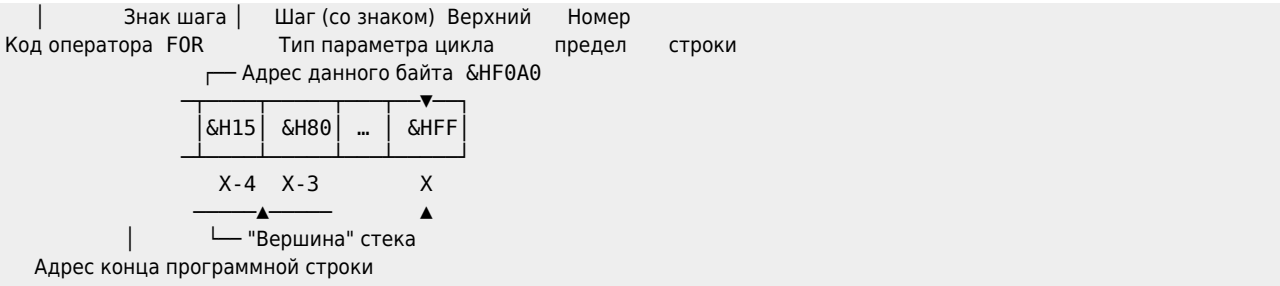
Пример 1.

Рассмотрим структуру расположения информации о цикле `FOR`.

- ```
  10 FOR AB%=2 TO 7 STEP 15 'Оператора NEXT быть не должно!
  run
  Ok

  PRINT HEX$(PEEK(&HF675))+HEX$(PEEK(&HF674))
  F0A0
  Ok
  ```


  ```



Перед выполнением следующего примера наберите команду CLEAR!

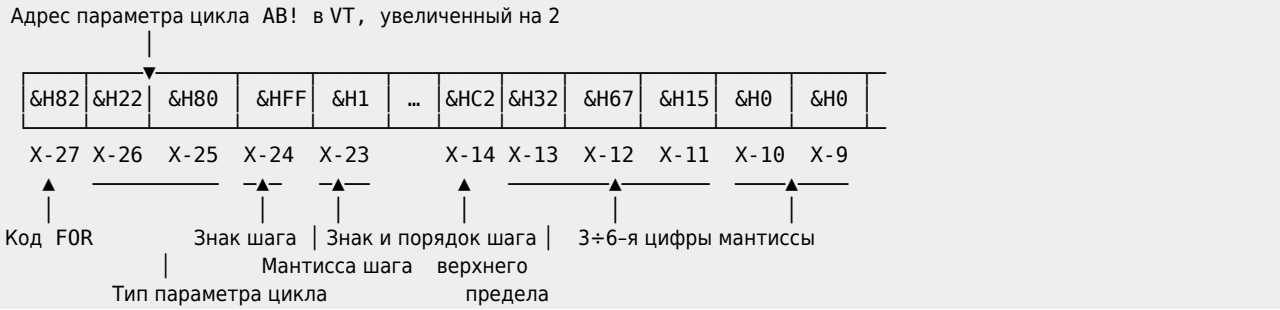
- ```
  10 FOR AB=2.7 TO 7 STEP-32.671533782376
  run
  Ok

  PRINT HEX$(PEEK(&HF675))+HEX$(PEEK(&HF674))
  F0A0
  Ok
  ```

Не забудьте о команде CLEAR!

- ```
  10 FOR AB!=2.7 TO 7 STEP-32.6715
  run
  Ok

  PRINT HEX$(PEEK(&HF675))+HEX$(PEEK(&HF674))
  F0A0
  Ok
  ```

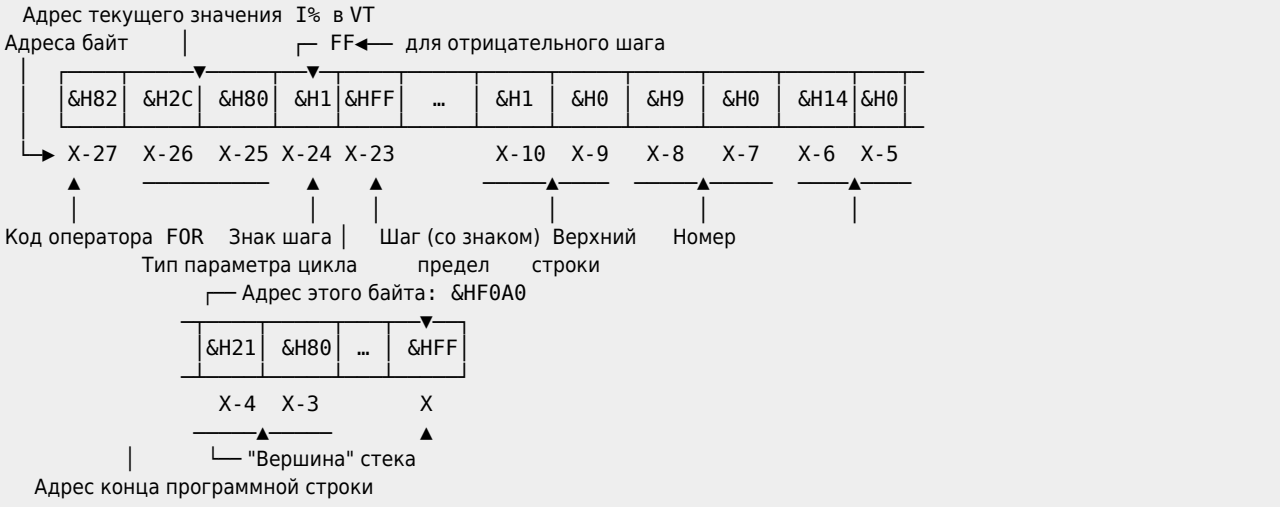


Хотите получить те же результаты — пользуйтесь оператором CLEAR!

- Пример под рубрикой: «Стек в действии!»

```
10 FOR AB%=2 TO 7:NEXT    ◄── Цикл закрыт!
20 FOR I%=3 TO 9    ◄── Цикл не закрыт!
run
Ok

PRINT HEX$(PEEK(&HF675))+HEX$(PEEK(&HF674))
F0A0
Ok
```



Отметим, что для версии MSX Disk BASIC с отключённым дисководом B при нулевой длине строковой области максимальное число вложенных циклов равно 576.

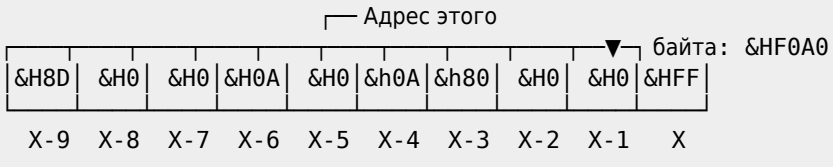
А теперь настала очередь оператора GOSUB...

Тем не менее, о команде CLEAR забывать не стоит!

Пример 2.

```
10 GOSUB 30:INPUT A
20 'Просто комментарий!
30 'Еще один комментарий!
run
Ok

PRINT HEX$(PEEK(&HF675))+HEX$(PEEK(&HF674))
F0A0
Ok
```



```
  ─▲─        ────▲────        ───▲───              ─▲─
 │       │          │            └─ Вершина стека
Код  GOSUB        │        Адрес следующего оператора
      Номер текущей строки
```

Пример 3.
Работу этих двух программ Вы должны проверить на ученическом дисплее.

1. 105-031.bas
   105-031.bas

```
Ok
10 GOSUB 30:PRINT 1:END
20 PRINT 2:END
30 'POKE (&HF0A0-4),&H10
40 RETURN
run
 1
Ok
```

2. 105-032.bas
   105-032.bas

```
Ok
10 GOSUB 30:PRINT 1:END
20 PRINT 2:END
30 POKE (&HF0A0-4),&H10'Адрес перехода
40 RETURN 'изменен с адреса &H800A на адрес &H8010
run
 2
Ok
```

X.6. Хранение строковых величин

Функция VARPTR указывает адрес расположения строковых данных в оперативной памяти. Она имеет следующий синтаксис:

```
VARPTR(γ)
```
,

где:

- VARPTR («VARiable PoinTeR» — «указатель переменной») — служебное слово;
- γ — идентификатор *строковой* переменной.

Если переменная не существует, то выдаётся сообщение:

«Illegal function call».

Функция VARPTR возвращает число X — *адрес* байта, находящегося на 3 позиции правее той, с которой располагается информация о переменной γ.

Пусть γ — простая строковая переменная. Изобразим «кусочек» памяти в окрестности байта с адресом X:

```
                  ┌── Байт, адрес которого возвращает функция
             ┌────┬────┬────┬───▼┬────┬────┐          VARPTR(γ)
             │▒▒▒▒│▒▒▒▒│▒▒▒▒│*** │▒▒▒▒│▒▒▒▒│
             └────┴────┴────┴────┴────┴────┘
Адреса──▶ X-3  X-2  X-1   X   X+1  X+2
             ▲   └───▲───┘  ▲  └───▲───┘
       │   │   │   └─ Ссылка на адрес в PIT или на адрес в строковой области
Тип переменной  │    └── Байт длины
```

```
    Идентификатор
```

Напомним Вам, что в программировании *ссылка* — содержимое ячейки памяти, воспринимаемое как адрес некоторой другой ячейки.

Указатели строковых переменных хранятся в VT. Они занимают 6 байт, причём:

- один байт содержит «паспорт» переменной VALTYPE (число 3);
- два байта содержат имя строковой переменной;
- один байт содержит длину строки, возвращаемую функцией VARPTR(γ)(таким образом, обе функции LEN(A$) и PEEK(VARPTR(A$)) возвращают одно и тоже значение). Приведём простой пример:

```
10 A$="карандаш":PRINT LEN(A$);PEEK(VARPTR(A$))
run
 8  8
Ok
```

- следующие два байта указывают адрес первого байта строки. Если символьная переменная создаётся явным присваиванием символьной константы, то указатель задаёт адрес этой константы в PIT. Лишь затем MSX BASIC *может* переслать значение этой символьной переменной в зарезервированное для неё строковое пространство.

Пример 1. «Сборка» значения строковой переменной A$ из памяти.
106-01.bas
106-01.bas

```
5 CLEAR:INPUT A$:A$=A$+"":A=PEEK(VARPTR(A$)):PRINT A
30 B$="&H"+HEX$(PEEK(VARPTR(A$)+2))+HEX$(PEEK(VARPTR(A$)+1)):PRINT B$
40 B=VAL(B$):BB=PEEK(B)
45 FOR I=0 TO A-1:PRINT CHR$(PEEK(B+I));:NEXT I:END
run
? MSX
 3
&HF163
MSX
Ok
```

Пример 2.
106-02.bas
106-02.bas

```
10 A$="ABCD"
run
Ok
PRINT HEX$(VARPTR(A$))
8014  ◀— Это адрес байта, содержащего длину  A$
Ok
```

Затем выполните команду PRINT PEEK(AD), где значение переменной AD изменяется от &H8011 до &H8016. Вы получите:

PRINT PEEK(&H8011)	3	VALTYPE
PRINT PEEK(&H8012)	65	Код ASCII символа «A»
PRINT PEEK(&H8013)	0	Второй символ отсутствует
PRINT PEEK(&H8014)	4	Длина значения A$
PRINT PEEK(&H8015)	9 = 0000 1001_2	Адрес (младший байт)
PRINT PEEK(&H8016)	128 = 1000 0000_2	Адрес (старший байт)

1000 0000 0000 1001_2 = &H8009

Итак, строка помещается в PIT по адресу &H8009.

Все остальное очевидно!

```
? PEEK(&H8009)  ? PEEK(&H800A)  ? PEEK(&H800B)  ? PEEK(&H800C)
 65       66       67       68 ◄── Код ASCII "D"
Ok              Ok              Ok              Ok
```

Пример 3.
106-03.bas
106-03.bas А теперь измените строку 10 и выполните программу.

```
10 A$="ABCD"+""
run
Ok
```

Повторите вышеуказанные шаги. Вы получите:

```
PRINT HEX$(VARPTR(A$))
8017
Ok
```

`PRINT PEEK(&H8014)`	3	VALTYPE
`PRINT PEEK(&H8015)`	65	Код ASCII для символа «A»
`PRINT PEEK(&H8016)`	0	В имени нет второго символа
`PRINT PEEK(&H8017)`	4	Длина строки
`PRINT PEEK(&H8018)`	101 = &H65	Новый адрес (младший байт)
`PRINT PEEK(&H8019)`	241 = &HF1	Новый адрес (старший байт)

Операция, выполненная над строкой, явилась причиной пересылки её по адресу &HF165 (без изменения). Вы можете убедиться в этом, используя «в окрестности» этого адреса функцию PEEK.

```
? PEEK(&HF165)   ? PEEK(&HF166)   ? PEEK(&HF167)   ? PEEK(&HF168)
 65               66               67               68
Ok               Ok               Ok               Ok
```

Забегая несколько вперёд, отметим, что функция `FRE("")` возвратила бы число 200 в первом случае, однако, во втором случае возвращает число 196.

Приведём схемы расположения информации в памяти для строковых массивов:

- *одномерный* строковый массив.

Пример 4. Обязательно разберите пример «с компьютером в руках»!
106-04.bas
106-04.bas

```
10 DIM SM$(2):SM$(0)="por"+"":SM$(1)="Inform"+"":SM$(2)="1989 г."+""
run
Ok
print HEX$(VARPTR(SM$(0))-8)
8047
```

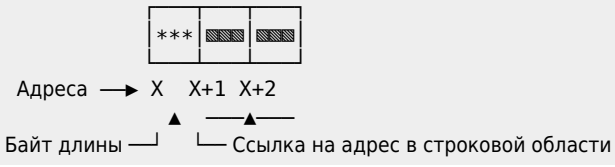
```
Ok
```

`? PEEK(&H8047)`	Этот адрес соответствует байту VALTYPE. Поэтому Вы должны получить 3.
`? PEEK(&H8048)`	Этот адрес соответствует первому символу имени строкового массива. Вы должны получить число 83 — код ASCII символа «S».
`? PEEK(&H8049)`	Этот адрес соответствует второму символу имени строкового массива. Вы должны получить число 77 — код ASCII символа «M».
`? PEEK(&H804A)`	12 Служебная информация
`? PEEK(&H804B)`	0 Служебная информация
`? PEEK(&H804C)`	Массив SM$ одномерный, поэтому мы и получили 1.
`? PEEK(&H804D)`	3 = 00000011_2
`? PEEK(&H804E)`	0 = 00000000_2

Теперь находим количество элементов в массиве: 00000000 00000011_2 = 3

```
? PEEK(&H800F)◄── 3 Длина значения элемента SM$(0).
```

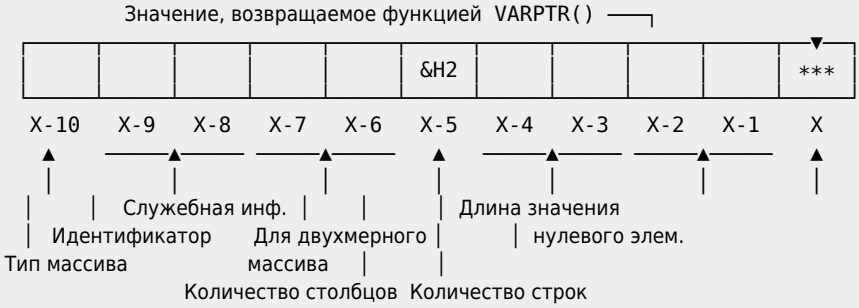
Приведём схему расположения информации в памяти для элемента строкового массива:



Продолжим наш пример:

```
PRINT HEX$(VARPTR(SM$(0)))
804F
Ok
? PEEK(&h804F)
3          ◄──── Длина значения 0-го элемента массива
Ok
? HEX$(PEEK(&h8050))
66         ◄──── Младший байт адреса 0-го элемента в строковой области
Ok
? HEX$(PEEK(&h8051))
F1                  ◄──── Старший байт адреса 0-го элемента в строковой области
Ok
? CHR$(PEEK(&hF166))
p          ◄┐
Ok          │
? CHR$(PEEK(&hF167))    │ Символы
o          ◄┤ значения
Ok          │ 0-го
? CHR$(PEEK(&hF168))    │ элемента
г          ◄┘
Ok
```

- *двухмерный* строковый массив.



X.7. Оператор CLEAR

> Чтобы вычистить одно, приходится выпачкать что-нибудь другое; но можно испачкать всё, что угодно, и ничего при этом не вычистить.
>
> *—Принцип Накопления Грязи по Питеру*

Оператор CLEAR в общем виде записывается так:

```
CLEAR [[n][,A]]
```

,

где:

- CLEAR («очистить») — служебное слово;
- n — арифметическое выражение, целая часть значения которого указывает количество байт, резервируемых под строковое пространство; значение параметра n меняется от нуля до размеров свободного пространства и равно 200 по умолчанию;
- A — арифметическое выражение, целая часть значения которого определяет адрес первого байта участка памяти, расположенного между блоком управления файлами и рабочей областью. Этот участок не будет «обработан» интерпретатором, поэтому он является как бы «резервной» памятью, где Вы можете хранить, например, подпрограммы на машинном языке и другие необходимые Вам данные.

Другими словами, значение A «переопределяет верхний предел пространства, используемого MSX BASIC».

Например, команда CLEAR 1000, &HF000 отводит 1000 байт для значений строковых констант в строковой области и RAM с адреса &HF000 для размещения машинной подпрограммы.

Максимальным значением выражения A, конечно же, является адрес, с которого начинается рабочая область (&HF380).

Минимальное значение выражения A может быть определено по формуле:

```
267·(MAXFIL+1)+VARTAB+145+n+2
```

,

где:

- MAXFIL — имя слова, расположенного в рабочей области по адресу &HF85F. Этот адрес занимает 1 байт памяти;
- VARTAB — имя слова, расположенного в рабочей области по адресу &HF6C2. Адрес занимает 2 байта памяти и отмечает конец PIT;
- n — размер строкового пространства в байтах.

Заметим, что *минимальная* величина стека равна 145 байтам (это пространство между концом PIT и вершиной стека). Отметим, что оба аргумента могут быть опущены; в этом случае оператор CLEAR производит «чистку» значений всех числовых и строковых переменных, элементов массивов и определённых пользователем функций DEFFN, а также «уничтожает» стек.

Сохранятся только текст программы на MSX BASIC и ранее зарезервированные машинные подпрограммы!

Приведём два простеньких примера:

- 1) 107-01.bas
  107-01.bas

```
10 DEF FNY(X)=X
11 CLEAR
12 PRINT FNY(1)
run
Undefined user function
```

```
Ok
```

- 2) 107-02.bas
  107-02.bas

```
10 M=9:T$=STRING$(5,"+"):?M;T$
20 CLEAR:?M;T$
run
 9 +++++
 0
Ok
```

Параметр n в операторе `CLEAR n` указывает размер строковой области, зарезервированной для хранения строковых значений, причём:

`0 ≤ n ≤ FRETOP-VARTAB-145-16` ,

где:

- `FRETOP` — имя слова, расположенного в рабочей области по адресу &HF69B. Этот адрес занимает 2 байта памяти;
- `VARTAB` — имя слова, расположенного в рабочей области по адресу &HF6C2. Этот адрес занимает 2 байта памяти.

Пример 3. Нажмите кнопку RESET Вашего компьютера.
107-03.bas
107-03.bas А теперь:

```
10 PRINT HEX$(PEEK(&HF69C));" ";HEX$(PEEK(&HF69B))
11 PRINT HEX$(PEEK(&HF6C3));" ";HEX$(PEEK(&HF6C2))
12 PRINT &HDC5F-&H8003-145-16
DC 5F  <-------  Вы узнали адрес FRETOP ?
80 3  <------- Вы узнали адрес VARTAB ?
 23483 <-------  0≤n≤23483
Ok
```

Пример 4.
107-041.bas
107-041.bas

```
10 CLEAR200:T$=SPACE$(198):B$=STRING$(2,"#"):?B$:X$=STRING$(1,"L"):?X$
run
##
Out of string space in 10
Ok
```

Получили сообщение об отсутствии места в строковой области, так как 200 байт, отведённых для неё по умолчанию, оказались уже исчерпанными.

Однако...
107-042.bas
107-042.bas

```
10 CLEAR 200:T$=SPACE$(198):B$=STRING$(2,"#"):?B$;:CLEAR1:X$=STRING$(1,"L"):? FRE("")
run
## 0
Ok
```

X.8. Функция FRE

> Garbage collection («чистка памяти», «сборка мусора») — действия системы динамического распределения памяти для обнаружения неиспользуемых программой блоков памяти и присоединения их к списку свободной памяти для повторного использования.
>
> *—Англо-русский словарь по программированию и информатике*

Информацию о размере свободной области («Free Area») в RAM можно получить с помощью функции FRE, обращение к которой имеет вид:

`FRE(A)`,

где:

- FRE («FREe» — «свободный») — служебное слово;
- A — арифметическое или строковое выражение, причём для интерпретатора важным является лишь тип выражения, а не его значение.

На практике применяется следующий синтаксис:

```
FRE(0)
```

или

`FRE("")`.

Функция `FRE(0)` возвращает количество байт, оставленных для расширения PIT, VT, стека, строковой области и блока управления файлами.

Пример 1.
108-01.bas
108-01.bas

```
10 ? FRE(0):X=451:? FRE(0):Z#=7.5:? FRE(0)
20 Y!=555:? FRE(0):W%=111:? FRE(0)
run
 28739
 28728 ◀— т.к. переменная X по умолчанию — двойной точности, а, следовательно, занимает в памяти 11 байт;
 28717 ◀— т.к. переменная  Z  —  двойной точности (занимает в памяти также 11 байт);
 28710 ◀— т.к. переменная  Y  —  одинарной точности (занимает в памяти 7 байт);
 28705 ◀— т.к. переменная  W  —  целого типа (занимает в памяти 5 байт).
Ok
```

Функция `FRE(0)` выдаёт сообщение:

«Out of memory» («*Не хватает памяти*»)

при достижении значения, меньшего 145 байт, минимально допустимого для стека системы MSX BASIC. Посмотрите (предварительно нажав кнопку RESET):

```
CLEAR 28868:PRINT FRE(0)
 147
Ok
```

```
CLEAR 28869:PRINT FRE(0)
Out of memory
```

```
Ok
```

Заметим,что слово VARTAB отличается от слова TXTTAB на 2 байта (при отсутствии программы!), поэтому, добавив эти 2 байта к 145 байтам, необходимым для работы стека, получаем число 147!

Функция FRE("") возвращает количество свободных байт в строковом пространстве. Например:

```
print FRE("")
 200
Ok
```

```
X$="2²"+"3²":print FRE("")
 196
Ok
```

Кроме того, функция FRE("") выполняет важное дополнительное действие. Оно связано с наличием в MSX BASIC строк переменной длины, обработка которых может привести к явлению «фрагментации памяти» (внутри строковой области появляются участки,содержащие неиспользуемую информацию — *«мусор»*). Поэтому, если в качестве аргумента функции FRE задано выражение строкового типа, перед вычислением объёма свободной памяти функция выполняет *«сборку мусора»*, т.е. удаление всех неиспользуемых данных и освобождение занимаемых ими областей.

Пример 2. Оказывается,что если у Вас в начале программы встречается оператор A$=«ABCD«+»EF», а затем — оператор A$=«X«+»Y», то Вы сразу же создадите 6-байтовое пространство, заполненное «мусором»!
Покажем это:
108-02.bas
108-02.bas
Fix Me!

```
print HEX$(PEEK(&HF69C)); HEX$(PEEK(&HF69B));
F168                                  ────▲──
Ok                                        └── Адрес "верхушки" строкового
a$="ABCD"+"EF"                                пространства
Ok

for t=0 to 5:print chr$(peek(&HF168-t));:next
FEDCBA
Ok
a$="X"+"Y"
Ok

for t=0 to 7:print chr$(peek(&hF168-t));:next
FEDCBAYX
└──▲─┘
Ok └─── "мусор"

print fre("")'Избавимся от "мусора"!
198
Ok

for t=0 to 7:print chr$(peek(&hF168-t));:next
YXXCBAYX
  └──▲─┘
Ok   └─── "мусор"
```

Из примера следует, что строки хранятся в строковом пространстве в том порядке, в каком они были определены.

Таким образом, функция FRE("") изменила положение значения строковой переменной (это и называется *«сборкой мусора»*).

Если под строки зарезервирован большой объем строкового пространства и определено много символьных переменных, время «сборки мусора» может составить несколько минут. При выполнении этой операции компьютер полностью «застывает». Посмотрите...

Пример 3.

108-03.bas
108-03.bas

```
10 CLEAR 5000 'Объявлен размер строковой области - 5000 байт
15 DEFINT A-Z:DIM A$(1500):FOR I=1 TO 1500:A$(I)="2"+"":NEXT
30 'Размещение данных в строковой области, "мусора" нет!
40 TIME=0:PRINT FRE(""),TIME/60/60"мин"
run
·3500·········3.61638888888 мин
Ok           (для MSX 1)

run
·3500·········3.3716666666667 мин
Ok           (для MSX 2)
```

Интересно, что при изменении в строке 10 оператора `CLEAR 5000` на оператор `CLEAR 1600`, результат получается почти тот же (≈3.607 мин. для компьютера MSX 1 и ≈3.38 мин. для компьютера MSX 2)!

Единственный способ уменьшить время «сборки мусора» — это использовать минимальное количество строк и особенно строковых массивов!

Следует заметить, что некоторые строки хранятся в тексте самой программы и, таким образом, не занимают места в строковой области.

Пример 4.
108-04.bas
108-04.bas

```
10 ? FRE("");:U$="fywapro":D$="K":? FRE("");:DIM E$(150):? FRE("")
20 FOR K=1 TO 150:E$(K)=CHR$(K):NEXT:? FRE("")
30 E$(1)="APR":? FRE(""):E$(1)=" "+E$(1):? FRE("")
run
200 200 200    Далее пауза для "сборки мусора"...
50
51 ◄── Произошла "сборка мусора" (свободное место в строковой области
47     увеличилось, т.к. значение элемента массива E$(1) уже хранится
Ok          в тексте программы)...
```

Таким образом, строковая область является областью памяти, резервируемой для хранения строковых данных. Если Вы хотите зарезервировать в строковом пространстве место для хранения 10 строк, содержащих каждая максимум 5 символов, то воспользуйтесь, например, оператором цикла:

```
FOR I=1 TO 10:A$(I)=SPACE$(5):NEXT
```

Во избежание «сборки мусора»:

1. определяйте все переменные в начале программы;
2. используйте строковые функции `MID$`, `LEFT$`, `RIGHT$`. Перед работой со следующим примером выключите, а затем снова включите Ваш компьютер.

   Пример 5.
   108-05.bas
   108-05.bas
   Fix Me!

```
A$="полет"
Ok
for t=0 to 10:print chr$(peek(&HF168-t));:next
телопп█
Ok   └▲─┘
   └──────── "м у с о р"
A$="налет"
Ok
for t=0 to 10:print chr$(peek(&HF168-t));:next
телоптеланн
```

```
└─▲─┘      ▲
 └────────└──── "м у с о р"
Ok
print fre("")
 195
Ok
for t=0 to 10:print chr$(peek(&HF168-t));:next
теланнеланн
      └─▲─┘
    └──────── "м у с о р"
Ok
mid$(A$,1,2)="по"
Ok
for t=0 to 10:print chr$(peek(&HF168-t));:next
телоппеланн
      └──▲─┘
    └── "м у с о р" (он остался на прежнем месте!)
Ok
```

Отметим, что строковые функции `MID$`, `LEFT$`, `RIGHT$` не изменяют указатели на значения строковых переменных. Обычный же оператор присваивания, разумеется, указатели изменяет! Покажем это на примере (не забудьте о команде `CLEAR` !).

Пример 6.
108-06.bas
108-06.bas
Fix Me!

```
a$="полет"
Ok
print hex$(peek(varptr(a$)+2)); hex$(peek(varptr(a$)+1))
F164
Ok
a$="налет"
Ok
print hex$(peek(varptr(a$)+2)); hex$(peek(varptr(a$)+1))
F15F
Ok
```

а теперь...

```
mid$(a$,1,2)="по"
Ok
print hex$(peek(varptr(a$)+2)); hex$(peek(varptr(a$)+1))
F15F    ←── Обратите внимание, это значение совпадает с предыдущим!
Ok
```

Как видим, значение указателя в последнем случае не изменилось!

3. при необходимости используйте оператор `SWAP A$,B$` , который не меняет расположение значений переменных, а лишь меняет местами указатели на эти значения. Проиллюстрируем этот факт на примере...

Пример 7.
108-07.bas
108-07.bas
Fix Me!

```
clear
Ok
print HEX$(PEEK(&HF69C)); HEX$(PEEK(&HF69B))
F168
Ok
A$="Зачет":B$="Автомат"
Ok
for t=0 to len(a$)+len(b$):print chr$(peek(&hF168-t));:next
течаЗтамотвАА
Ok
swap A$,B$
Ok
for t=0 to len(a$)+len(b$):print chr$(peek(&hF168-t));:next
течаЗтамотвАА
```

```
Ok
```

И наконец, функция FRE() может помочь Вам также в *защите* Вашей программы. Например, в «укромном» месте программы, работающей со строковой информацией, поместите оператор X$=SPACE$(FRE("")) — конечно, Вы должны учесть, что целая часть значения аргумента функции SPACE$ должна принадлежать отрезку [0,255]!. Это удержит «любознательных» от модификации значений переменных Вашей программы (разумеется, в данном случае строковых)!

Посмотрите:

```
10 X$=SPACE$(FRE("")):Y$="2"+Y$
run
Out of string space in 10
Ok
```

X.9. Рабочая область

В рабочей области содержатся системные подпрограммы, системные переменные и «ловушки», используемые интерпретатором во время выполнения операторов Вашей программы. В рабочей области хранятся данные о позиции курсора, цвете текста, состоянии функциональных клавиш и другая полезная информация, инициализируемая при включении компьютера.

Адрес, отмечающий *начало* рабочей области, указан в самой этой области в слове HIMEM, содержимое которого занимает 2 байта, расположенных с адреса &HFC4A .

Ещё раз напомним Вам, что адреса, занимающие два байта, всегда записываются так: вначале записывается содержимое младшего байта, а затем — содержимое старшего байта!

Отметим, что значением выражения HEX$(PEEK(&HFC4A)+256*PEEK(&HFC4B)) является *адрес начала рабочей* области.

Поскольку рабочая область расположена в RAM, её переменные могут изменяться операторами POKE. Но это следует делать только в том случае, если Вы *знаете*, что за этим последует!

X.9.1. Матрица клавиатуры

Матрицей клавиатуры для MSX–компьютеров назовём таблицу вида:

		0-й бит	1-й бит	2-й бит	3-й бит	4-й бит	5-й бит	6-й бит	7-й бит
	Адреса байт	**254**	**253**	**251**	**247**	**239**	**223**	**191**	**127**
0-я строка	**FBE5**	9	;	1	2	3	4	5	6
1-я строка	**FBE6**	7	8	0	=	-	H	:	V
2-я строка	**FBE7**	\	.	B	@	,	/	F	I
3-я строка	**FBE8**	S	W	U	A	P	R	[	O
4-я строка	**FBE9**	L	D	X	T	]	Z	J	K
5-я строка	**FBEA**	Y	E	G	M	C	~	N	Q
6-я строка	**FBEB**	SHIFT	CTRL	GRAPH	CAPS	РУС	F1	F2	F3
7-я строка	**FBEC**	F4	F5	ESC	TAB	STOP	BS	SELECT	RETURN
8-я строка	**FBED**	SPACE	HOME	INS	DEL	←	↑	↓	→
9-я строка	**FBEE**	RET	+	*	0	1	2	3	4
10-я строка	**FBEF**	5	6	7	8	9	-	,	.

Последние две строки соответствуют цифровой (правой) зоне клавиатуры учительского компьютера серии MSX 2. Более подробная информация по этой теме здесь[3)].

Ответим теперь на Ваш очевидный вопрос:

Как воспользоваться этой таблицей?

Пример 1. Ниже приведены программа, останавливаемая нажатием клавиши GRAPH:

```
10 Z=PEEK(&HFBEB):IF Z<>251 THEN 10
```

и программа, останавливаемая нажатием клавиш SHIFT+CTRL:

```
10 Z=PEEK(&HFBEB):IF Z<>(254 AND 253) THEN 10
```

А теперь ответ на Ваш следующий вопрос:

А как получить матрицу клавиатуры ?

Для «чтения» нажатой клавиши достаточно «прочесть» слово NEWKEY (11 байт) по адресу &HFBE5 из таблицы системных переменных.

Пример 2. Программа «пробегает» все клавиши и возвращает позицию нажатой клавиши (X,Y) матрицы клавиатуры. 11 значений, записанных в слове NEWKEY, соответствуют 11 строкам матрицы клавиатуры. Если не нажата ни одна клавиша, содержанием каждого из 8 байт, соответствующих строке матрицы является 1. Это фиксируется двоичным числом &B11111111=255. Когда же клавиша нажата, считанное на этой строке значение отличается от 255: бит соответствующей колонки «сбрасывается» в 0. 1091-01.bas
1091-01.bas

```
10 FOR Y=0 TO 10:Z=PEEK(&HFBE5+Y)
30 IF Z=255 THEN 80 '⟶
40 PRINT"Y=";Y:Z$=RIGHT$("00000000"+BIN$(Z),8)
60 PRINT"X=";8-INSTR(Z$,"0"):PRINT
80 NEXT:GOTO 10 '⟶
```

X.9.2. Динамическая клавиатура [46]

[46]

> Промедление с лёгким делом превращает его в трудное, промедление же с трудным делом превращает его в невозможное.
>
> *—Д.Лоример*

Исследуем один подход к разработке учебных программ, работающих под управлением интерпретатора MSX BASIC. Существенная особенность этого подхода состоит в том, что программа в процессе выполнения модифицируется (происходит изменение отдельных строк BASIC–программы или добавление новых строк). Считается, что допущение самомодификации программы во время выполнения является признаком плохого стиля программирования, поэтому начнём с примера, который показывает, что предлагаемый подход является не только оправданным, но и в ряде случаев единственно возможным.

Пусть необходимо табулировать функцию y=f(x), конкретно x^2, то есть для каждого значения аргумента вычислить значение функции и результат записать в таблицу. Соответствующая программа выглядит следующим образом:

```
10 'Программа табулирования функции.
20 DIM X(200),Y(200)
30 INPUT XN,XK 'Задание границ изменения аргумента функции.
           …
100 GOSUB 1000 'Обращение к подпрограмме табулирования.
           …
999 END
```

```
1000 FOR I=1 TO 200:Y(I)=X(I)*X(I):NEXT I:RETURN
```

Части программы между строками 30 и 100, 100 и 999 содержат операторы, обеспечивающие масштабирование, заполнение таблицы, защиту от ошибочных действий пользователя и т.д. Простота программы обусловлена тем, что задача табулирования решается для *фиксированной* функции. Попытаемся теперь разработать программу, которая позволяет табулировать любую функцию одной переменной, аналитическое выражение которой вводится с клавиатуры!

Для реализации на ПЭВМ «YAMAHA» используем специальный механизм, введенный Дж.Баттерфилдом (J.Batterfield) и названный им принципом *«динамической клавиатуры»*.

В *командном* (!) режиме информация, набираемая пользователем на клавиатуре, аппаратно записывается в буфер (называемый в дальнейшем *буфером клавиатуры*, БК). БК размещается в рабочей области с адреса &HFBF0 по адрес &HFC17 и занимает 40 байт. При нажатии клавиши Ввод ⏎ содержимое БК считывается интерпретатором и выполняется соответствующая команда.

Имитация действий пользователя на основе принципа «динамической клавиатуры» осуществляется следующим образом:

1. с помощью оператора INPUT вводится текст запроса (в данном случае аналитическое выражение табулируемой функции) в символьную строку F$ (в приведённой ниже программе — строка 10);
2. символьная строка дополняется впереди номером, а в конце — кодом команды RETURN (строки 15 и 1890): `F$="номер строки 1"+F$+CHR$(13) ;`
3. строка побайтно переписывается в БК, начиная с адреса &HFBF0, при помощи оператора POKE и функции PEEK (подпрограмма, начинающаяся со строки 1880);
4. строка `S$="goto"+"номер строки 2"+CHR$(13)`, где *номер строк* и 2 — номер строки программы, куда после модификации необходимо передать управление, побайтно переписывается в БК (строка 25);
5. выполнение программы прекращается командой END, в результате происходит переход из программного режима в командный. Интерпретатор считывает содержимое БК до первого появления CHR$(13) и выполняет его как команду, то есть модифицирует строку с номером *номер строки* 1. Далее считывается остаток содержимого БК до второго появления CHR$(13), и он также выполняется интерпретатором, как команда, после чего происходит переход в программный режим с передачей управления в строку с номером *номер строки 2*.

Таким образом, указанный алгоритм решает задачу автоматической модификации программы в соответствии с текстом запроса, вводимого пользователем с клавиатуры, и запуска её с указанного номера строки.

Пример.
1092-01.bas
1092-01.bas

```
1 GOTO 10
2 GOTO 30
10 LINEINPUT"Введите аналитическую запись функции:";F$
11 INPUT"Укажите номер строки, содержащей оператор описания функции пользователя DEFFN (51< номер строки
<59)";SN:GOSUB 2410
13 GOSUB 1550 'Сохранение F$
15 F$=STR$(SN)+F$:F$=MID$(F$,2,LEN(F$)-1)
20 GOSUB 1880
25 F$="goto2":GOSUB 1880:END
30 GOSUB 1730 'Восстановление F$
50'** Программа табулирования функции Y(x) **
60 INPUT"Введите[A,B] и шаг табулирования H";A,B,H
65 FOR X=A TO B STEP H:PRINT X;FNY(X):NEXT
90 END
1550'***** Формирование F$ *****
1590 F$="deffny(x)="+F$:POKE &HF600,LEN(F$)
1620 FOR I=1 TO LEN(F$):POKE &HF601+I,ASC(MID$(F$,I,1)):NEXT
1720 RETURN'⟶
1730'***** Восстановление F$ *****
1740 LF=PEEK(&HF600):F$=""
1750 FOR I=1 TO LF:C=PEEK(&HF601+I):F$=F$+CHR$(C):NEXT
1780 RETURN'⟶
1880'***** Динамическая клавиатура *****
```

```
1890 F$=F$+CHR$(13)
1900 AD=PEEK(&HF3F9)*256+PEEK(&HF3F8)-65536!
1910 L1=&HFC17-AD+1
1920 IF LEN(F$)<=L1 THEN GOTO 1990
1930 L2=LEN(F$)-L1:N=0
1940 FOR I=AD TO &HFC17:N=N+1
1950 POKE I,ASC(MID$(F$,N,1)):NEXT
1960 FOR I=&HFBF0 TO &HFBF0+L2-1:N=N+1
1970 POKE I,ASC(MID$(F$,N,1)):NEXT
1980 AD=&HFBF0+L2+65536!:POKE&HF3F9,FIX(AD/256):POKE&HF3F8,AD-FIX(AD/256)*256:GOTO 2050
1990 N=0
2000 FOR I=AD TO AD+LEN(F$)-1:N=N+1
2010 POKE I,ASC(MID$(F$,N,1)):NEXT
2020 IF LEN(F$)<L1 THEN AD=AD+LEN(F$) ELSE AD=&HFBF0
2030 AD=AD+65536!
2040 POKE&HF3F9,FIX(AD/256):POKE&HF3F8,AD-FIX(AD/256)*256
2050 RETURN'⟶
2410 IF LEN(F$)>19 THEN CLS:LOCATE 1,10:PRINT"Эта программа имеет ограничение:":GOTO 2420 ELSE GOTO
2460
2420 PRINT:PRINT "Длина формулы не должна превосходить 19 символов!";LEN(F$)
2440 PRINT:LOCATE 1,23:PRINT"Для продолжения нажмите любую клавишу"
2450 W$=INKEY$:IF W$="" THEN 2450 ELSE GOTO 10
2460 RETURN'⟶
```

X.10. Порты ввода–вывода

> И я надеюсь, что наши потомки будут благодарны мне не только за то, что я здесь разъяснил, но и за то, что мною было добровольно опущено с целью предоставить им удовольствие самим найти это.
>
> *—Рене Декарт. Геометрия*

Порт ввода–вывода — многоразрядный вход или выход компьютера, через который процессор обменивается данными с внешними устройствами (клавиатурой, принтером, дисководом, видеопамятью и видеопроцессором, игровыми манипуляторами). Часто говорят, что порты представляют собой «интерфейсные схемы компьютера».

Порт ввода–вывода напоминает морской порт, через который ввозят и вывозят товары. В нашем случае через порты вводятся и выводятся данные. Порты принимают данные от периферийных устройств и направляют их в эти устройства. Используя прямой доступ к портам ввода–вывода,Вы более полно используете возможности компьютера.

Процессор «работает» с портами по адресам, которые не следует путать с адресами ROM или RAM:

1. порты с адресами &H00÷&H7F. Вы не можете *изменить* их содержимое (сравните с ROM!);
2. порты с адресами &H80÷&HFF. Их содержимое изменять можно (сравните с RAM!);
3. порты с адресами &H100÷&HFFFF зарезервированы(пока не используются).

Некоторые порты, их функции и адреса перечислены ниже:

Адрес	Чтение(Запись)	Назначение
Порты, отвечающие за работу с локальной сетью КУВТ YAMAHA MSX 1		
&h00	Чтение(Запись)	Посылаемые данные
&h01	Чтение	Статус
&H02	Чтение	Номер компьютера в локальной сети (только для компьютеров MSX 1)
Порты, отвечающие за работу с локальной сетью КУВТ YAMAHA MSX 2		
&H09		Командный порт (передача или приём)
&H0C		Порт состояния

Адрес	Чтение(Запись)	Назначение
&H0E		Порт данных
— — —	— — —	— — —
&H0A		Используются при инициализации сетевого ОЗУ
&H0B		
&H0D		
Принтер		
&H90	Чтение	Ввод сигнала занятости принтера
&H91	Запись	Код выводимого символа
&H92	Запись	?
Видеопроцессор (VDP)		
&H98	Чтение(Запись)	Обращение к видеопамяти
&H99	Чтение(Запись)	Чтение (запись) в регистр VDP
&H9A	Запись	Запись в регистр палитры
&H9B	Запись	Косвенная запись в регистры VDP
Звукогенератор (PSG)		
&HA0	Запись	Ввод в порт номера регистра
&HA1	Запись	Ввод в порт информации для установленного регистра
&HA2	Чтение	Последнее число, записанное в PSG
Программируемый *периферийный интерфейс* (PPI)		
&HA8	Чтение(Запись)	Запись (чтение) данных в порт A
&HA9	Чтение(Запись)	Запись (чтение) данных в порт B
&HAA	Чтение(Запись)	Запись (чтение) данных в порт C
&HAB	Чтение(Запись)	Запись (чтение) режимов PPI
Порты, отвечающие за работу с *дисководом*		
&HB4	Запись	?
&HB5	Чтение(Запись)	?
&HFC	Чтение(Запись)	Регистры распределения *слотов* (расширений памяти) для компьютеров серии MSX 2
&HFD	Чтение(Запись)	
&HFE	Чтение(Запись)	
&HFF	Чтение(Запись)	

Для работы с портами ввода-вывода используются: функция `INP` и операторы `OUT` и `WAIT`.

Формат оператора `OUT`:

```
OUT  адрес, данное
```

,

где:

- `OUT` («OUTput» — «вывод») — служебное слово;
- *адрес* — арифметическое выражение, целая часть значения которого принадлежит отрезку [128,255] (128=&H80, 255=&HFF);
- *данное* — арифметическое выражение, целая часть значения которого принадлежит отрезку [0,255].

Оператор OUT «посылает» заданное операндом *данное* значение в порт, номер которого задан значением параметра *адрес*.

Внимание!
На компьютерах серии MSX 2 прежде, чем использовать оператор OUT, необходимо в непосредственном режиме выполнить команду `CALL NETEND` (отключить Ваш компьютер от локальной сети)

Опишем синтаксис функции INP:

```
INP (адрес)
```
,

где:

- INP («INPut» — «ввод») — служебное слово;
- *адрес* — арифметическое выражение, целая часть значения которого принадлежит отрезку [0,255].

Функция INP возвращает целочисленное значение, «прочитанное» из порта, имеющего указанный адрес.

Видна ли Вам аналогия между операторами POKE и OUT , PEEK и INP ?!

При помощи функции INP Вы можете использовать в своих расчётах номер Вашего компьютера. Чтобы поместить в переменную A номер компьютера, на котором Вы работаете в локальной сети MSX 1, примените оператор:

```
A=INP(&H02) AND 15
```
.

Объясним роль логической операции AND. Значением, возвращаемым функцией INP(&H02), является двоичное число, записанное в одном байте. «Содержимое» четырёх старших битов байта нас не интересует. Заметим, что число 15 = &b00001111. Как Вы уже, наверное, догадались, логическая операция AND позволяет выделить нужные нам четыре младших бита.

X.10.1. Программируемый периферийный интерфейс (PPI)

Теперь мы перейдём к рассказу о работе с портами Программируемого Периферийного Интерфейса (PPI — «Parallel Programming Interface»).

Напомним Вам, что *интерфейс* (англ. «interface» — «сопряжение» — способ и средства установления и поддержания информационного обмена между исполнительными устройствами автоматической или человеко–машинной системы.

В *параллельном* интерфейсе порция двоичной информации, состоящая из n битов, передаётся одновременно по n каналам.

Порт A используется для выбора *слотов*, осуществляющих управление расширенной памятью компьютера.

Порты PPI B и C применяются для «работы» с матрицей клавиатуры, причём номер строки матрицы клавиатуры «посылается» в порт C, а номер столбца «читается» в порту B.

Пример 1. *Обнаружение* нажатия клавиши GRAPH.
1011-01.bas
1011-01.bas

Отметим, что клавиша GRAPH находится в строке 6 и столбце 2 матрицы клавиатуры (и строки, и столбцы матрицы нумеруются, начиная с 0). Тогда:

1. номер строки матрицы клавиатуры «посылаем» в порт C :
   ```
   OUT &HAA,6
   ```
2. «извлекаем» номер столбца из порта B:
   ```
   X=INP(&HA9)
   ```

До нажатия клавиш значением X является число 255 = &b11111111.В момент нажатия какой-либо клавиши соответствующий бит порта B (в нашем случае второй) на мгновение обнуляется.

Таким образом, нажатие клавиши GRAPH легко обнаружить, если выделить значение интересующего нас бита командой:

```
IF (X AND &b00000100)=0 THEN PRINT"GRAPH"
```

Программа, позволяющая обнаружить нажатие клавиши GRAPH, выглядит так:

```
10 OUT &HAA,6:X=INP(&HA9)
20 IF (X AND 4)=0 THEN PRINT "GRAPH":END
30 GOTO 10
```

Надеемся, что Вы обратили внимание на недостаток этой программы: после её запуска неожиданно включается индикатор CAPS (но это не означает,что Вам удалось смоделировать нажатие клавиши CAPS!).

Разберёмся, почему так происходит.

Взгляните на приведённую ниже таблицу, в которой описаны назначения битов порта C:

Биты 0÷3	Строка клавиатуры
Бит 4	Если 0, то запускается кассетная лента
Бит 5	Сигнал записи на ленту
Бит 6	Если 0, то включается индикатор «CAPS»
Бит 7	Управление звуковым сигналом

Все ясно! Индикатор «CAPS» включается потому, что в порт C записывается значение

```
6 = &b00000110, а значит, шестой бит порта C "опрокинулся" в нуль.
       ▲                    ──────┬──────
       └──────────────────────────┘
```

Фактически только четыре *младших* бита порта C определяют номер строки матрицы клавиатуры. Для «маскирования» (игнорирования) значений четырёх старших битов достаточно вместо команды `OUT &HAA,6` выполнить команду:

```
OUT &HAA,6 OR (INP(&HAA) AND  &HF0)
         │        │            │                ┌── !!!
         ▼        ▼            ▼                ▼
 &B00000110 OR (&B*1****** AND &B11110000) = &B*1** 0110
```

(символом «*» отмечены биты, состояние которых в данном случае роли не играет).

Пример 2. Включение индикатора «CAPS» (если он выключен!) можно осуществить следующей командой:
1011-02.bas
1011-02.bas

```
OUT &HAA, INP(&HAA) XOR &B01000000
           │               │         ┌─ !!!
           ▼               ▼         ▼
    &B*1******  XOR  &B01000000 = &B*0******
```

Пример 3.
1011-03.bas
1011-03.bas

```
10 OUT &HAA,6 OR (INP(&HAA) AND &HF0):B=INP(&HA9)
20 IF (B AND 1)=0 THEN PRINT "Нажата клавиша SHIFT"
30 IF (B AND 2)=0 THEN PRINT "Нажата клавиша CTRL"
40 IF (B AND 4)=0 THEN PRINT "Нажата клавиша GRAPH":GOTO 10 ELSE 10
```

В ходе работы программы индикатор «CAPS» сохранит состояние, в котором он находился до пуска программы!

Пример 4. Получим матрицу клавиатуры при помощи оператора OUT!
1011-04.bas
1011-04.bas

```
10 INPUT "Номер строки";N
20 INPUT "Номер столбца";T
30 OUT &HAA,INP(&HAA) AND &HF0 OR N
40 B=((INP(&HA9) AND 2^T)=0)
50 IF B THEN PRINT "Клавиша нажата":END
60 GOTO 30
```

Пример 5. Включение и выключение кассетной ленты. Операторы MOTOR ON и MOTOR OFF (подробнее о них здесь) могут быть имитированы с помощью команды:
1011-05.bas
1011-05.bas

```
OUT &HAA, INP(&HAA) XOR &B00010000
```

X.10.2. Программируемый звуковой генератор (PSG)

Вначале приведём два примера записи информации в PSG при помощи портов ввода–вывода.

Пример 1.

1012-011.bas
1012-011.bas

```
10 SOUND 7,8  'Шум из канала A
20 SOUND 8,15 'Громкость
30 SOUND 6,26 'Частота звука
40 END
```

1012-012.bas
1012-012.bas

```
10 OUT &HA0,7:OUT &HA1,8: A=INP(&HA2)
20 OUT &HA0,8:OUT &HA1,15:B=INP(&HA2)
30 OUT &HA0,6:OUT &HA1,26:C=INP(&HA2)
40 PRINT A;B;C
```

```
run
 8  15  26
Ok
```

Пример 2. Представьте, что Вы находитесь на берегу Чёрного моря в районе Ялты. Закройте глаза и ...
1012-02.bas
1012-02.bas

```
10 FOR I=0 TO 13:READ V
20 OUT &HA0,I:OUT &HA1,V 'Имитация действия оператора SOUND I,V
30 NEXT:END
100 DATA 0,0,0,0,0,0,30,&HB7,16,0,0,0,90,14
```

А теперь мы расскажем Вам, как можно «музицировать» при помощи *непосредственной* записи в регистры звукового генератора. Взгляните на следующую небольшую табличку:

Исполняемая нота	Содержимое нулевого регистра	Содержимое первого регистра
PLAY "O4C"	172	1
PLAY "O4C#"	148	1
PLAY "O4D"	125	1
PLAY "O4D#"	104	1
PLAY "O4E"	83	1

Исполняемая нота	Содержимое нулевого регистра	Содержимое первого регистра
`PLAY "O4F"`	64	1
`PLAY "O4F#"`	46	1
`PLAY "O4G"`	29	1
`PLAY "O4G#"`	13	1
`PLAY "O4A"`	254	0
`PLAY "O4A#"`	240	0
`PLAY "O4B"`	227	0
`PLAY "O5C"`	214	0

Пример 3. Гамма «до-мажор».
1012-03.bas
1012-03.bas

```
10 DATA 1,172,1,125,1,83,1,64,1,29,0,254,0,227,0,214
20 OUT &HA0,8:OUT &HA1,15 'Установим громкость канала A
30 FOR T=1 TO 8:READ I1,I2
40 OUT &HA0,1:OUT &HA1,I1:OUT &HA0,0:OUT &HA1,I2
45 FOR K=1 TO 100:NEXT 'Если Вам захочется "озвучить", например, "выстрел", достаточно убрать из программы эту строку
50 NEXT
60 OUT &HA0,8:OUT &HA1,0 'Сбросим громкость канала A
```

Пример 4. «В траве сидел кузнечик!»
1012-04.bas
1012-04.bas

```
10 'DEFINTA-Z:BEEP
20 OUT &HA0,8:OUT&HA1,15:OUT &HA0,1:OUT &HA1,0:OUT &HA0,0:OUT &HA1,254
30 RESTORE 240:S=S+1:IF S=1 THEN K=14 ELSE K=12
40 FOR I=1 TO K:READ S1,S0,T1
50 OUT &HA0,8:OUT &HA1,15
60 OUT &HA0,1:OUT &HA1,S1:OUT &HA0,0:OUT &HA1,S0
70 TIME=0
80 T=TIME:IF T<T1 THEN 80
90 SOUND 8,0             'OUT &HA0,8:OUT &HA1,0:FOR J=1 TO 10:NEXT
100 NEXT
110 IF S=1 THEN 30
120 RESTORE 260:S=0
130 S=S+1:IF S=1 THEN K=20 ELSE K=14
140 FOR I=1 TO K:READ S1,S0,T1
150 OUT &HA0,8:OUT &HA1,15
160 OUT &HA0,1:OUT &HA1,S1:OUT &HA0,0:OUT &HA1,S0
170 TIME=0
180 T=TIME:IFT<T1 THEN 180
190 OUT &HA0,0:OUT &HA1,0:OUT &HA0,1:OUT &HA0,0:FOR J=1 TO 10:NEXT:NEXT
200 IF S=1 THEN RESTORE 270:GOTO 130
210 OUT &HA0,8:OUT &HA1,15
220 OUT &HA0,1:OUT &HA1,0:OUT &HA0,0:OUT &HA1,254
230 RUN
240 DATA 0,254,15,1,83,15,0,254,15,1,83,15,0,254,15,1,13,15,1,13,30
250 DATA 1,13,15,1,83,15,1,13,15,1,83,15,1,13,15,0,254,15,0,254,30
260 DATA 0,254,15,0,0,15,0,254,30
270 DATA 0,227,15,0,227,7.5,0,227,7.5,0,227,15,0,227,15
280 DATA 0,215,15,0,215,7.5,0,215,7.5,0,215,15,0,215,15
290 DATA 0,215,15,0,227,15,0,254,15,1,13,15,0,254,15,0,254,30
300 DATA 0,254,15
```

X.10.3. Другие порты. Оператор WAIT

Приведём примеры использования «содержимого» других портов.

Пример 1. Использование портов с адресами &H90 и &H91 для вывода символа на принтер.

Вначале о «роли» первого бита порта с номером &H90:

```
 7–й бит  6–й бит  5–й бит 4–й бит  3–й бит 2–й бит 1–й бит 0–й бит
┌────────┬────────┬────────┬────────┬────────┬────────┬────────┬────────┐
│   *    │   *    │   *    │   *    │   *    │   *    │        │   *    │
└────────┴────────┴────────┴────────┴────────┴────────┴────▲───┴────────┘
Принтер в режиме   ON LINE  (подключен к ПЭВМ): 0 ──────┤
Принтер в режиме  OFF LINE  (отключен от ПЭВМ): 1 ──────┘
```

А теперь: включите принтер и вставьте бумагу…
1013-01.bas
1013-01.bas

```
5 CLEAR 300
10 INPUT"Слово";A$
15 WAIT &H90,2,255 'Вы включите, наконец, принтер или нет?
20 A$=A$+CHR$(13)+CHR$(10) 'CHR$(13)  -  код возврата каретки;
21 '                        CHR$(10)  -  код перевода строки
30 FOR I=1 TO LEN(A$)
35    OUT &H90,0:OUT &H90,137 'Инициализация принтера
40    B=ASC(MID$(A$,I,1))
50    OUT &H91,B
60 NEXT I
```

Пример 2. Считывание кода выведенного на экран символа:
1013-02.bas
1013-02.bas

```
10 K$=INKEY$:IF K$="" THEN 10 ELSE PRINT K$;
20 PRINT INP(&H98):GOTO 10
```

Оператор `WAIT` используется сравнительно редко. Его синтаксис:

```
WAIT P,M[,C]
```
,

где:

- `WAIT` («ожидать») — служебное слово;
- P — арифметическое выражение, целая часть значения которого задаёт адрес порта;
- M и C — арифметические выражения, целые части значений которых принадлежат отрезку [0,255].

Оператор `WAIT` «заставляет» компьютер «ожидать», пока результатом «опроса» порта с указанным адресом не окажется число 0 (данный порт «работает» в режиме *чтения*). Другими словами, этот оператор является «бесконечным циклом», который «ждёт», пока от порта ввода–вывода не придёт определённый сигнал.

Вы можете прервать затянувшееся «ожидание», нажав клавиши CTRL+STOP (при этом Вы вернётесь в командный режим).

Содержимое порта с указанным адресом заносится в некоторый регистр процессора Z–80, который мы назовём X. Далее содержимое регистра X комбинируется со значениями параметров M и C, по формуле:

```
X = (X XOR C) AND M
```

Если после этого содержимое регистра X окажется равным 0, то происходит «выход из оператора `WAIT`». В противном случае порт вновь «опрашивается», и процесс повторяется.

Приведём таблицу–«подсказку»:

X	0 0 0 0 1 1 1 1
C	0 0 1 1 0 0 1 1
X XOR C	0 0 1 1 1 1 0 0
M	0 1 0 1 0 1 0 1
(X XOR C) AND M	0 0 0 1 0 1 0 0

Вопрос к читателю: Какой вид будет иметь таблица–«подсказка» при отсутствии параметра C ?

Пример 3.
1013-03.bas
1013-03.bas

```
10 WAIT &HAA,64,255

(&B*1****** XOR &B11111111) AND &B01000000 = &B00000000 = 0 !
```

Эта программа закончит свою работу, если загорится индикатор «CAPS».

Пример 4.

- `WAIT &H90,2,255 'Ожидается включение принтера`
- `WAIT &H90,2,0   'Ожидается отключение принтера`

Дополнение. Работа с портом ввода–вывода с адресом &h0C

Предварительно кратко опишем структуру порта &h0C...

```
Старший ┌─┬─┬─┬─┬─┬─┬─┬─┐ Младший
 бит    │*│*│*│*│*│*│*│*│  бит
        └─▲─┴─▲─┴─▲─┴─▲─┴─▲─┴─▲─┴─▲─┴─▲─┘
      │ │ │ │ │ │ │ └── Общий бит готовности сети
      │ │ Не используются │     (0: сеть готова)
      │ │               └──────── Бит регистрации данных
      │ │                    (0: данные поступили)
      │ └── Бит направления поступления информации (1: от учителя)
      └──────── Бит направления поступления информации (1: от ученика)
```

А теперь два примера использования данного порта...

Пример 1. *Посылка* байта по сети

```
OUT_BYTE::                  ; На входе в регистре  A  -  данное
    DI                  ;
        PUSH DE             ;
        LD   D,A            ;
    LD   A,005H         ; OUT 9,5  -  это запись байта в сетевое ОЗУ
    OUT  (009H),A       ;
OUT_B:  IN   A,(0CH)        ;
    AND  041H           ;
    CP   040H           ;
    JR   NZ,OUT_B       ; Если сеть не готова, то ждем
    LD   A,D            ;
    OUT  (00EH),A       ;
        POP  DE             ;
        EI                  ;
    RET                 ;
```

Пример 2. *Приём* байта из сети

```
IN_BYTE::                   ; На выходе в регистре  A  -  данное
    DI                  ;
    LD   A,003H         ; OUT 9,3  -  считывание байта из сетевого ОЗУ
    OUT  (009H),A       ;
IN_B:   IN   A,(00CH)       ;
    AND  083H           ;
    CP   080H           ;
    JR   NZ,IN_B        ; Если сеть не готова, то ждем
    IN   A,(00EH)       ;
        EI                  ;
    RET                 ;
```

Диск с примерами

Загрузить образ диска

Открыть диск в WebMSX

---

[1] [2]
,
Пока не найдено, подробнее здесь
[3]
Примечание редактора

http://sysadminmosaic.ru/msx/basic_programming_guide/10?rev=1587639916

2020-04-23 14:05