0-ой уровень, он же "DOS.SYS"

Нижний уровень системы в iS-DOS (рестарты с #00 по #1F) работает с кэшем, блочными и символьными устройствами каналами и прерываниями. Этот уровень не знает ни о файлах, ни о командных строках, ни об оконной технологии. Всем этим занимаются более высокие уровни. Итак, более подробно о рестартах:

Первые 7 (с #00 по #06) рестарты уровня DOS" работают с так называемой "кэш-памятью", поэтому сперва несколько слов о том, что это собственно такое. Итак, кэш блочных устройств - это системный буфер, используемый исключительно для блоков (по 256 байт), из которых состоят все блочные устройства. Кэш создается рестартом $creat(#00). Система автоматически создает его при загрузке, а также при выходе в систему по рестарту $shel0(#80) или по RET'у со значениями #F6 или #F7 в регистре A процессора. Его пересоздают также некоторые командные файлы: например копировщики. Делают они это в основном, чтобы уменьшить размер кэша и таким образом выиграть несколько килобайт памяти под буферы, а также, чтобы вернуть после этого кэш в прежнее состояние. Как уже говорилось в самом начале книги, система iS-DOS занимает память ZX сверху вниз, и кэш, создаваемый в последнюю очередь, лежит под системой. Сверху он ограничен областью каналов. Верхний адрес области кэш-памяти можно достать из вектора конфигурации системы g_cnf, использовав рестарт $g_cnf(#10) со смещением +32:

```
LD C,#10      ;$g_cnf возвращает в
RST #10       ;HL' адрес вектора
EXX           ;адрес в HL
LD DE,32      ;прибавляем смещение
ADD HL,DE
LD E,(HL)     ;получаем в DE
INC HL        ;искомый верхний
LD D,(HL)     ;адрес кэша
```

Нижний адрес кэша, естественно, зависит от верхнего адреса. Если Вам катастрофически не хватает памяти, можно убрать лишние резиденты (в этом Вам поможет set.com с ключом /e). Можно также подсократить область каналов, что умеет делать channel.com. Это вызовет перемещение всей области кэша вверх и, следовательно, поднимет и нижнюю его границу, освободив память для Ваших целей. Однако, из Вашей программы проще всего сократить кэш. Размер кэша, устанавливаемый по умолчанию, лежит в векторе g_cnf со смещением +7

Адрес нижней границы кэша находится в том же векторе со смещением +5 и +6. При необходимости его можно рассчитать по формуле (1):

$$CACHE = DGCHN - CSIZE \times 260 \quad (1)$$

где CACHE - искомый адрес нижней границы кэша
DGCHN адрес начала области каналов
CSIZE размер кэша в блоках.

По этому адресу располагается каталог кэша в виде массива описателей блоков по 4 байта на каждый блок кэша, после которого располагаются сами блоки.

Структура кэша:

Смещение	Размер	Описание
0	EDSIZ*4	каталог кэша состоит из 4-хбайтовых описателей блоков
0	4	описатель 1-го блока
4	4	описатель 2-го бока
8	4	описатель 3-го блока
EDSIZ*4	256 256 256	1-й блок 2-й блок 3-й блок Это, собственно, тело кэша. Его адрес хранится в векторе g_cnf со смещением -5

где EDSIZ - размер кэша.

Описатель блока имеет следующую структуру:

Смещение	Размер	Описание
0	1	регистр состояния блока, совмещённый с номером устройства. Биты: 7 - блок модифицирован (1) 6 - блок защищён (1) 5 - блок существует (1) 4 - не используется 3..0 - номер устройства, которому принадлежит блок
1	1	счётчик обращений к блоку
2	2	номер блока на устройстве

Минимальный размер кэша равен 6 блокам. Оптимальный в iS-DOS Classic от 20 до 30. В CHIC около 40. Кэш необходим для работы файловой службы. Через него система обращается к устройствам, т.е. читает и пишет описатели файлов, каталогов, карту занятых блоков устройств, заголовки устройств и т.п. При нехватке места в кэше в жертву приносится самый старый считанный блок. Запись также осуществляется через кэш, причем система и не имеет других буферов, поэтому блок просто видоизменяется, то бишь модифицируется прямо в кэше, а в описателе кэша устанавливается в единицу 7-ой бит. По мере накопления модифицированных блоков их записывают на устройство, а бит модификации сбрасывается. Это действие называется "автоматический флаш" или "авто-флаш" flush.

И, наконец, рестарты:

#00 $creat

Создает и пересоздает кэш. Требуемый размер в блоках необходимо подавать в регистре A. Содержимое остальных регистров на работу рестарта не влияют. На выходе проверьте флаг C. В случае ошибки он будет установлен в "1". Единственная возможная ошибка: **Error 130** - нехватка памяти, т.е. входное значение регистра A так велико, что кэш не влезает над адресом Utop (User Top). Достать адрес Utop Вы можете, применив рестарт $g_cnf(#10) (смещение +3 в векторе g_cnf). Utop служит, в основном, предохранителем от опускания кэша так низко, что программа set.com в процессе работы убила бы им саму себя. Поэтому обычно Utop=26676. Рестарт $creat(#00) обнуляет блоки размером по 256 байт под нижней границей области каналов ($g_cnf(#10) смещение (+32) в количестве указанном в регистре A и создает каталог кэша (по 4 байта на каждый блок). Работать с кэшем размером менее 4 блоков небезопасно, программа на это просто не рассчитана. Рекомендуемые размеры от 6..10 блоков минимум, если Вы хотите выиграть побольше памяти под буферы для программ или данных до 30 или даже может 70 (в iS-DOS Chic разумеется). Хотя наращивание кэша после 20 блоков почти не дает заметного выигрыша в скорости работы программ. Для восстановления размера кэша по умолчанию советуем такой вот пример:

```
LD C,#10        ;$g_cnf
RST 16
EXX             ;в HL адрес вектора
LD BC,-6        ;смещение -6 здесь
ADD HL,BC       ;хранится размер кэша по умолчанию
LD A,(HL)       ;помещаем размер в A
LD C,#10        ;$creat
RST 16          ;пересоздаем кэш
RET C
```

При расчетах размера создаваемого кэша, если известны верхняя и нижняя границы его, Вы можете пользоваться формулой (2), логически вытекающей из формулы (1):

$$CSIZE = \frac{DGCHN - CACHE}{260} \qquad (2)$$

где CSIZE - искомый размер кэша
DGCHN - нижняя граница области каналов она же - верхняя граница кэша
CACHE нижняя граница кэша.

Пример:

```
        LD C,#10        ;$g_cnf
        RST 16
        EXX             ;в HL адрес вектора
        LD BC,32        ;смещение +32
        ADD HL,BC
        LD E,(HL)       ;в DE адрес верхней
        INC HL          ;границы кэша
        LD D,(HL)
        LD HL,CACHE     ;это требуемая нижняя граница кэша, ниже которой вся память может
                        ;быть использована Вашей программой
        EX DE,HL        ;вычтем CACHE
        SBC HL,DE       ;из DGCHN
        XOR A
        LD DE,260       ;делим результат
1$      SBC HL,DE       ;на 260
        INC A
        JR NC,1$
        DEC A           ;проверяем не стал
        CP 6            ;ли CSIZE меньше
        JR NC,2$        ;6 блоков
        LD A,130        ;Нет места! Выход с
```

```
        RET             ;ошибкой 130 код ошибки в регистре A
2$      LD C,#00        ;$creat
        RST 16          ;пересоздаем кэш
        RET C
```

#01 $clear

Очистка кэша от блоков текущего устройства. Эта операция необходима при смене дискеты в дисководе или при отказе выгрузки на диск блоков, модифицированных рестартами $modwr(#03), $modo(#2E), $wpart(#2A), когда авто-флаш ещё не случился, а так же при записи на диск мимо кэша блоков, считанных через кэш, дабы диск и кэш соответствовали друг другу. При работе рестарта вся информация, имеющаяся в кэше, теряется. Если в кэше есть модифицированные блоки, и Вы хотите их сохранить, то перед очисткой вызовите рестарт $flush(#02). Ошибок после вызова быть не должно. Входных параметров нет. Пример использования рестарта $clear взят из программы format.com:

Это начало процедуры format. Возможно, перед форматированием был сменен диск. Поэтому кэш должен быть очищен рестартом $clear(#01). Этот пример также будет полезен для лучшего понимания работы рестартов $g_blk(#13), $g_drv(#14), $binit(#0F), которые будут рассмотрены нами далее. Вначале с помощью $g_blk(#13) мы узнаем текущее блочное устройство и достаем адрес тела соответствующего ему драйвера. Далее проверяем является ли этот драйвер драйвером флоппи диска sys_driv.blk т.е. можно ли форматировать это устройство через TRDOS. Затем мы пытаемся открыть это устройство как блочное устройство IS-DOS рестартами $binit(#0F) и $open(#20), дабы предотвратить случайное переформатирование существующего диска:

```
        LD C,#13            ;$g_blk
        RST 16              ;получаем в A номер текущего блочного устройства
        INC C               ;$g_drv
        RST 16              ;узнаем адрес его
        RET C               ;драйвера
        EXX                 ;в HL адрес тела драйвера текущего блочного устройства. Здесь мы проверяем,
                            ;является ли этот драйвер драйвером дисковода
        LD BC,9
        ADD HL,BC
        LD (DRCSR+1),HL
        RES 7,(HL)          ;не вызывать $erdrv(#1F) при ошибках на диске
        LD C,#01            ;$clear
        RST 16              ;очищаем кэш
        LD C,#0F            ;$binit
        RST 16              ;инициализируем устройство
        JR C,FOR_1          ;переход на процедуру форматирования, если устройство не существует
        LD C,#20            ;$open
        RST 16              ;пытаемся открыть устройство
        JR C,FOR_1          ;переход на форматирование, если не открывается
        ...
FOR_1...                    ;здесь расположена программа форматирования диска
```

#02 $flush

Рестарт $flush(#02) необходим для фиксации на любом блочном устройстве: диске, винчестере или электронном диске изменений, произведённых в кэше после записи с помощью рестартов $modwr(#03), $modo(#2E), $wpart(#2A), $crfil(#23), $erfil(#24) и др. Все вышеперечисленные рестарты модифицируют блоки в кэше. Естественно, что эти изменения должны в конце концов оказаться на диске. Дело в том, что кэш блочных устройств был создан, во-первых: для ускорения работы нижних уровней системы, дабы дважды и трижды не читать с устройства одни и те же блоки. К тому же системе был нужен некий свой буфер для работы с каталогами и файлами. В связи с этим запись на устройства система в большинстве случаев осуществляет не напрямую и не лезет с каждым новым блоком на диск, а помечает блок в кэше, как "модифицированный", то есть устанавливает в 1 один из разрядов описателя блока в кэше. При достижении определенного количества модифицированных блоков (обычно оно равно размеру кэша, деленному на 2 или размеру кэша минус 3) происходит автоматический флаш. При этом в 1 устанавливается 1-ый бит 1-го байта вектора блочного устройства. Входных параметров у этого рестарта нет. Возможны различные ошибки на выходе, например **ошибки кэша**:

61 указанный блок не найден

62 реальное кол-во модифицированных блоков не соответствует значению системной переменной ($QNMOD)

63 сектор защищен

64 кэш защищен

65 нет места в кэше

66 чтение запрещено

67 запись запрещена

Ошибки блочного драйвера sys_driv.blk:

6 нет диска
7 ошибка чтения/записи
20 обращение к диску прервано

Если модифицированных блоков в кэше на текущем (!) устройстве нет, то рестарт ничего не делает. Советуем вызывать $flush(#02) в конце вашей программы перед выходом в оболочку, а так же перед запуском из Вашей программы некоторых системных утилит (copy25.com, например) и перед пересозданием кэша при высвобождении памяти. Если Ваша программа писала через кэш на несколько различных блочных устройств, например:

- текущее T:
- системное S:
- быстрое Q:

,то не забудьте сделать flush на каждом из этих устройств, переключаясь на них с помощью рестарта $swblk(#1C) или $g_sta(#37). Вот пример выхода из программы, модифицирующей блоки на нескольких устройствах, сразу скажем на Q S и T:

```
LD A,"Q"          ;для устройства "Q"
CALL FLUSH        ;вызов подпрограммы
RET C
LD A,"S"          ;для устройства "S"
CALL FLUSH        ;вызов подпрограммы
RET C
LD A,"T"          ;для устройства "T"
CALL FLUSH        ;вызов подпрограммы
RET C             ;здесь происходит восстановление среды, которая была на момент запуска
LD C,#41          ;$fmrst
RST #10           ;вызов рестарта
RET C             ;выход в оболочку с перепечаткой экрана стандартным способом
XOR A
LD A,#F4
RET

 ;подпрограмма FLUSH. Сначала преобразуем логическое имя ;устройства ("S","Q","T") в физический номер (0..7)

FLUSH LD C,#4A          ;$dvtrn
      RST #10
      LD B,A            ;номер устройства помещаем в рег B, переключаемся на соответствующее
                        ;устройство
      LD C,#1C          ;$swblk
      RST #10
      RET C

 ;вот теперь собственно flush

      LD C,#02          ;$flush
      RST #10
      RET               ;возврат из подпрограммы
```

#03 $modwr

Запись одного блока (256 байт) с адреса, поданного в HL на текущее блочное устройство под номером блока, поданным в DE. Запись происходит через кэш, т.е. сперва в кэше освобождается место, выкидывается самый старый считанный блок, и на его место переносятся заданные 256 байт.

Новый блок получает свой номер и номер устройства (от 0 до 7) и тут же помечается как модифицированный. Если переполнения модифицированных блоков при этом не произошло, то на этом всё и кончается, иначе происходит автоматический флаш, (т.е. физическая запись модифицированных блоков на текущее устройство посредством обращения к драйверу).

При работе рестарта возможны ошибки, так что после его вызова советуем проверять флаг C. Коды ошибок те же, что и у $flush(#02). Рестарт $modwr(#03) удобен при работе непосредственно с устройством (минуя файл). Не забудьте только вызвать $flush(#02) в конце программы!

Пример из жизни: программа arzt+.com, а точнее ее фрагмент, в котором осуществляется запись на устройство измененной таблицы занятых блоков:

```
LD HL,$BUFF       ;адрес рабочего буфера помещаем в HL
LD DE,1           ;номер 1-го блока в DE
LD A,($MLTRD+2)   ;число блоков
```

```
        LD B,A          ;помещаем в B это счетчик последующего цикла
        LD C,#03        ;$modwr
3$      RST #10
        RET C
        INC DE          ;вызов в цикле
        INC H           ;для всех блоков
        DJNZ 3$         ;по очереди
        RET
```

#04 $unird

Парный к предыдущему рестарт чтения одного блока устройства с кэша. В HL помещаем адрес, в DE - номер блока на устройстве. Если блока в кэше нет, процедура считывает его через драйвер. Флаг C на выходе сообщает о неудачно проведенной операции. Коды ошибок, как и в предыдущих двух случаях. В реальной жизни этот рестарт практически не применяется, поэтому в качестве примера прочитаем 0-й блок устройства в буфер BUFF, очистив перед этим на всякий случай (если была смена диска) кэш.

```
LD C,#01        ;$clear
RST #10         ;очистка кэша
LD DE,0         ;номер блока = 0
LD HL,BUFF      ;адрес буфера
LD C,#04        ;$unird
RST #10
RET C
```

#05 $mltrd

Почти тоже самое, но читается сразу несколько блоков. Число блоков задается в регистре B. Сперва эти блоки ищутся в кэше и копируются по означенному адресу, затем прямо туда, куда заказывали, считываются через драйвер недостающие блоки, которые уже затем копируются в кэш. Если число считываемых блоков превосходит размер кэша пополам, то считывание происходит мимо кэша, дабы не выбивать из него такие часто требующиеся блоки как блоки каталогов и заголовок устройства и таким образом не замедлять работу системы с каталогами. Напрямую считываться блоки будут также при наличии в кэше модифицированных блоков. Можно заставить всегда работать этот рестарт напрямую с драйвером, сбросив 7-ой бит 0-го байта вектора блочных устройств. Этим рестартом пользуется рестарт чтения части файла $rpart(#29), с помощью которого в свою очередь запускаются все com-файлы системы. На всякий случай анализируйте флаг C на выходе. Коды ошибок те же, что и у $flush(#02). Пример взят опять же из arzt+.com: считывание карты занятых блоков устройства в буфер $BUFF. Байт $MLTRD+2 заранее заполняется размером карты:

```
        LD C,#35        ;$bkfcb
        RST #10         ;считываем описатель
        EXX             ;диска и находим в
        LD BC,-6        ;нем размер карты
        ADD HL,BC

;Округлим его до целых блоков: и поместим по адресу $MLTRD+2

        LD A,(HL)
        INC HL
        OR A
        LD A,(HL)
        JR Z,$+3
        INC A
        LD ($MLTRD+2),A
        CALL $MLTRD ;вызов подпрограммы

;подпрограмма, использующая $mltrd(#05)

$MLTRD  LD BC,#05       ;$mltrd
        LD HL,$BUFF     ;адрес буфера в HL
        LD DE,1
        RST #10
        RET
$BUFF                   ;это буфер
```

#06 $quard

Почти тоже, что и $unird(#04), но блок считывается только в кэш (или просто находится в нем). Поэтому регистр HL на входе не используется, зато на выходе в HL' возвращается адрес блока в кэше, DE' - адрес заголовка блока в каталоге кэша (необходим для дальнейшего использования $modo(#2E)). На входе в DE подается номер блока устройства.

Возможны различные ошибки (те же, что и у рестарта $flush(#02)), посему проверяйте флаг C. Удобен для работы резидентов или программ, которым жаль буферов под блок. Пример взят из программы date.com:

```
;Считаем 0-ой блок устройства. Именно там хранится дата загрузки системы:

        LD DE,0         ;номер блока = 0
        LD C,#06        ;$quard
        RST #10
        RET C
        EXX

;Сохраним адрес считанного блока (HL) и адрес описателя этого блока в кэше (DE):

        LD (TMPHL+1),HL
        LD (TMPDE+1),DE

;печать и изменение даты (эти процедуры пропущены здесь за ненадобностью) выход из программы: в DE подаем новое значение даты. Сравним его с хранящимся в 0-ом блоке устройства и, если они разнятся, положим новую дату в 0-ой блок и сохраним его на устройстве рестартами $modo(#2E) и $flush(#02):

FTMPHL  LD HL,0         ;в этот пустой байт помещается вышеупомянутый адрес считанного блока
        LD BC,30
        ADD HL,BC
        LD C,(HL)
        INC HL
        LD B,(HL)

;в BC теперь старая дата, считанная с устройства

        EX DE,HL
        XOR A
        PUSH HL         ;новая дата
        SBC HL,BC       ;сравнение
        POP HL
        EX DE,HL
        JR Z,SVSTYL     ;даты равны, обход

;Если же даты не равны, то кладем новую поверх старой прямо в кэше:

        LD (HL),D
        DEC HL
        LD (HL),E       ;Промодифицируем блок в кэше:
FTMPDE  LD DE,0         ;в пустой байт после DE как Вы помните, мы ранее поместили адрес описателя блока в кэше, полученный из рестарта $quard(6) несколькими строками выше. Этот адрес необходим для $modo.
        LD C,#2E        ;$modo
        RST #10
        RET C
        LD C,#02        ;$flush
        RST #10
        RET C

SVSTYL...
```

#07 $key

Ввод символа с клавиатуры. Обращается прямо в 1-ую точку входа драйвера клавиатуры. Адрес этой точки входа лежит в драйвере со смещением 2. Если буфер драйвера пуст, тогда драйвер обращается к портам клавиатуры и ждет, пока клавиша не будет нажата.

Выходное значение возвращается в регистре A. Оно зависит от нажатой клавиши и от режима, в котором находится устройство ввода. Режим определяется нулевым байтом вектора символьного устройства ввода g_key, который можно получить обратясь к рестарту $g_key(#11).

0-ой байт вектора. Биты (0/1):
0 строчные/ПРОПИСНЫЕ
1 lat/русские
2 текст/псевдографика
3 не ждать отпускания клавиш при $kwait(#08). Применяется для макросов в редакторе, сбрасывается сам при опустошении буфера драйвера.

Флаг C роли не играет. В качестве примера приводится подпрограмма из текстового редактора edit.com, зажигающая и гасящая курсор в текущих координатах и опрашивающая клавиатуру с ожиданием или без (управляется 5-ым битом в edcsr):

;Переключим клавиатуру в режим маленьких латинских, чтобы получить код управляющей клавиши, не зависящий от текущего состояния. 3-ий бит (макро) маскируется восьмёркой, дабы сохранить его. Биты 0,1,2 обнуляются:

```
F$KEY   LD C,#11          ;$g_key
        RST #10
        EXX
        LD A,(HL)
        AND 8
        LD (HL),A ;lat
```

;Переключение на текущий драйвер вывода на экран (t42/t64):

```
        CALL $TYCUR
        RET C
```

;Позиционирование Пересчитаем координаты в тексте xc в позицию на экране и положим YX в HL:

```
        LD H,(IX+ys)
        LD A,(IX+ys-1)
        ADD A,(IX+xc)
        SUB (IX+X_skip)
        LD L,A
L_F     LD C,#0C          ;$typos
        RST #10           ;зажжем мигающий курсор в текущей позиции:
        LD C,#76          ;$y
        RST #10
        LD C,#08          ;$kwait
```

;Обход ожидания отпускания клавиши:

```
        BIT 5,(IX+edcsr)
        JR Z,$+3
        RST #10
        DEC C             ;7=$key ввод
        RST #10           ;клавиши
        PUSH AF           ;сохраним код
        LD C,#77          ;$n_
        RST #10           ;погасить курсор
        POP AF            ;восстановим код
        RET
```

#08 $kwait

Ожидание отпускания клавиш. Рестарт необходим в некоторых случаях, когда опасна случайно дважды нажатая (удерживаемая) клавиша, например в самом начале программы, вызываемой клавишей и также опрашивающей клавиатуру с помощью предыдущего рестарта. Это вторая точка входа в драйвер (адрес ее находится в драйвере со смещением 4). Используется в рестартах $smbgt(#6E), $edstr(#7F), $menu(#91). На выходе может быть всё, что угодно. Пример см. предыдущий рестарт. Эта процедура вызывается, скажем, блочным оверлеем ed0+.ovr с ожиданием отпуска клавиши перед рестартом $analy(#7E) после печати нижней строки функций оверлея и при вызове функции <Cs/D>, требующей подтверждения. Такой вызов гарантирует надежность. Однако при отметке блока та же процедура вызывается без ожидания отпускания, что позволяет отмечать большие блоки удерживая клавишу и используя автоповтор. Этот рестарт при вызове опустошает буфер драйвера клавиатуры, т.о. драйвер "забывает" все нажатые до вызова рестарта клавиши. Если бы эта функция драйвера не отключалась бы 3-им битом регистра состояния клавиатуры, макросы работали бы лишь до первого $kwait.

#09 $ktest

Этот рестарт - третья точка входа в драйвер клавиатуры. Он определяет, нажата хоть одна клавиша на клавиатуре или нет. **Выход** во флаге Z:

Z - не нажата,

NZ - нажата.

Как и $key(#07) в первую очередь опрашивает буфер драйвера, где накапливаются быстро нажатые клавиши, и который используется для макросов или эмуляции нажатой клавиши. Этот рестарт удобен для опроса клавиатуры без ожидания, например, для прерывания работы циклов. При нажатии клавиши флаг на выходе - NZ, а код клавиши будет в регистре A, а также в буфере драйвера, откуда его необходимо извлечь рестартом $key(#07), иначе на него наткнется еще кто-нибудь.

Пример:

;Эта подпрограмма вернется с флагом Z, если клавиш нажато не было, и с флагом NZ и кодом клавиши в регистре A в противном случае.

```
FKTEST  LD C,#09        ;$ktest
        RST #10
        RET Z
        LD C,#07        ;$key
        RST #10         ;извлекаем "лишний" код из буфера драйвера клавиатуры
        OR A
        RET
```

#0A (10) $type

Печать символа из регистра A на текущем устройстве символьного вывода (экран или принтер). Выбор устройства (экран или принтер) можно осуществить при помощи рестарта $svtyp(#1B).

Флаг на выходе при печати на экране роли не играет, обращаясь к принтеру можно схлопотать следующие **ошибки:**

150 ошибка или принтер не готов,

151 печать прервана.

Последняя версия программы print.com не пользуется этим рестартом. Печать на экране происходит в текущей позиции, которая хранится в байтах 1..4 вектора символьного устройства вывода. Вектор можно достать с помощью рестарта $g_typ(#12).

Байты:

1 Позиция печати в строке

2 Номер строки на экране

Эти 2 байта подаются в HL на входе рестарта $typos(#0C), который передает их в регистре BC в 3-ю точку входа в драйвер (смещение 6). 3,4 Адрес печати на экране. Это левый верхний байт буквы. Например #40 для 0-ой позиции в 0-ой строке).

Сюда эти байты кладут рестарты $typos(#0C) и $wtpos(#6B). Отсюда берет их для печати следующего символа рестарт $type(#0A). Сюда же он их и возвращает после печати. Рестарт обращается к 1-ой точке входа драйвера (смещение 2 от начала драйвера). При этом адрес печати на экране подается в регистре DE'. Этим рестартом пользуются рестарты более высоких уровней, такие как $str(#6C), $smbgt(#6E) и др.

#0B (11) $tycpl

Устанавливает инверсный/нормальный режимы печати на экране в соответствии с содержимым регистра A:

A<>0 инверсная печать

A=0 нормальный режим

У драйверов принтеров не используется. Флаг C на выходе значения не имеет. Рестарт используется новой версией текстового редактора для блочной отметки, а также программой demon.com. Обращается во 2-ю точку входа в драйвер (адрес по смещению 4).

#0C (12) $typos

Задание позиции печати на экране для последующей печати рестартами $type(#0A), $str(#6C), $lnstr(#6D), $smbgt(#6E), $edstr(#7F). **Координаты печати** задаются в регистре HL:

H - номер строки,

L - позиция в строке.

Для позиционирования относительно окна также можно пользоваться рестартом $wtpos(#6B). Флаг C на выходе значения не имеет. Этот рестарт использует 3-ю точку входа в драйвер (смещение 6). У старых

драйверов принтера эта точка не использовалась. У новых она отвечает за печать символа из регистра A без перекодировки.

Вот как использует рестарты $typos и $type программа exebat.com для печати командной строки из системного буфера:

;Проверка наличия 8-го канала (канал драйвера символьного вывода) т.к. программа должна работать даже тогда, когда печатать нечем:

```
FPRINT  LD A,8          ;ty42
        LD C,#16        ;$stchn
        RST #10
        RET C
```

;Позиционирование:

```
        SBC HL,HL       ;HL=0: верхняя строка
        LD C,#0C        ;$typos
        RST #10
        LD C,#45        ;$g_com
        RST 16
        EXX             ;HL=адрес системного буфера
        LD BC,#2A0A     ;$type B=#2A=42-столько букв помещается в одну строку на экране, #0A - $type
```

;Печатаем строку до символа #0D:

```
1$      LD A,(HL)
        CP #D
        JR Z,2$
        RST #10
        INC HL
        DJNZ 1$
        RET
```

;Допечатка пробелов:

```
2$      LD A," "
        RST 16          ;C=$type (см выше)
        DJNZ 2$
        RET
```

#0D (13) $read

Чтение блоков напрямую с драйвера текущего блочного устройства. На **входе**:
HL - адрес,
DE - номер блока устройства,
B - число блоков.

Флаг C на **выходе** - признак ошибки. Ошибки только те, что выдает драйвер. Так драйвер электронного диска обычно не выдает ошибок, хотя никто не мешает встроить в него проверку, скажем номера блока на слишком большой номер или число считываемых блоков на 0 или вставить проверку контрольной суммы блока. У драйвера sys_driv.blk возможные **ошибки** следующие:

6 нет диска
7 сектор не читается
20 чтение прервано.

Обычно драйвер флоппи-диска отрабатывает эти ситуации, обращаясь к рестарту $erdrv(#1F) (см ниже). Рестарт обращается к 1-ой точке входа в драйвер. **Пример** взят из жизни копировщиков: начало рабочего цикла программы abc.com. В байты INDSK+2 и OUTDSK+2 кладутся номера входного и выходного устройств.

;B=число блоков для чтения/записи

```
FCOPY   PUSH BC
INDSK   LD BC,#1C       ;$swblk
        RST #10
        POP BC
        RET C
        LD HL,BUFF      ;буфер копирования
        LD C,#0D        ;$read
        RST #10
        RET C
        PUSH BC
OUTDSK  LD BC,#021C     ;$swblk в C
        RST #10
```

```
        POP BC
        RET C
        LD C,#0E        ;$write
        RST #10
        RET C

    ...
```

#0E (14) $write

Запись B блоков с адреса HL на текущее блочное устройство (флоппи диск, электронный диск или винчестер) начиная с блока номер DE. В IS-DOS любое блочное устройство состоит из логических блоков размером 256 (#100) байт, что очень удобно для расчетов в ассемблере. Пересчётом номера блока в трек, сторону и сектор занимается драйвер. Драйвер флоппи диска настраивается на формат IS-DOS дискеты сам рестартом $binit(#0F). Это 2-ая точка входа в драйвер.

Поскольку это прямое обращение к устройству минуя кэш, то рекомендуется сразу же после вызова этого рестарта вызывать $clear(#01) или $creat(#00) во избежании путаницы блоков в кэше и блоков на устройстве!

Ситуация с возможными ошибками точно такая же, как и у $read(#0D). Пример также см. выше.

#0F (15) $binit

Автонастройка драйвера флоппи диска на формат IS-DOS диска. Считывает 0-й блок диска, проверяет наличие признака "DSK" со смещением 10 (в новых версиях системы по смещению +13 в связи с введением 11-символьного имени устройства). Если признака нет, то выходит с ошибкой 9. Иначе настраивается на параметры диска, лежащие со смещением:

22	число цилиндров
23	тип диска (число сторон)
24	размер сектора (1/2/4:256/512/1024)
25	число секторов на дорожке
64	номера секторов (до 16 штук)

Возможна также ошибка 10 - попытка работы с двусторонним диском на одностороннем дисководе или с 80-дорожечным на 40-трековом. Это 3-ий вход в драйвер. Обычно не используется у драйверов электронного диска и винчестера (там ставятся "заглушки" для совместимости). Этот рестарт полезно вызывать при первом обращении к новому блочному устройству для настройки драйвера на диск и чтоб убедиться в том, что это именно IS-DOS устройство. Кроме ошибок 9 и 10 может выдать ошибки чтения: 6 7 и 20. Драйвер можно настроить и на работу с форматами TR-DOS и MS-DOS, но рестарт здесь уже мало чем поможет. В таких случаях лучше найти драйвер, с помощью рестарта $g_drv(#14), убедиться, что это именно драйвер флоппи-диска и по смещению 34(dec) сменить размер сектора, количество секторов и таблицу секторов. В качестве примера рекомендуем отрывок из format.com, приведенный после описания рестарта $clear(#01).

#10 (16) $g_cnfg

Этот рестарт выдает на выходе в HL' адрес вектора конфигурации ядра (см. **ПРИЛОЖЕНИЕ 3**) и системное блочное устройство S: в регистре A. Входные регистры не влияют. Флаг C на выходе роли не играет. Для **примера** рассмотрим перехват рестарта опроса клавиатуры, что может быть полезно, например, чтобы подвесить попискивание на клавиши.

```
        LD C,$g_cnfg
        RST 16
        EXX
        LD BC,8
        ADD HL,BC           ;адрес 0-го уровня DOS
        LD E,(HL)
        INC HL
        LD D,(HL)
        LD HL,14            ;$key*2
        ADD HL,DE           ;KEY
        LD (FN_A+1),HL

        LD E,(HL)
        INC HL
        LD D,(HL)
        LD (FN_RST+1),DE
PUTADR  LD DE,KEY
```

```
FN_A    LD HL,0
        DI
        LD (HL),E
        INC HL
        LD (HL),D
        EI
        RET

  KEY

        …

  FN_RST CALL 0

        …
```

Теперь при каждом вызове рестарта $key управление будет передаваться Вашей программе **KEY**, где Вы можете делать всё что угодно до и после обращения к драйверу. Если вздумаете оформить программу в виде резидента, а это самое разумное, не забудьте, что резидента могут сдвинуть, при этом надо заново положить новый адрес в таблицу рестартов DOS'a. Не забудьте также восстановить все как было при снятии резидента.

#11 (17) $g_key

Рестарт выдает в HL' адрес вектора символьного устройства ввода (**ПРИЛОЖЕНИЕ 2**), а в A - канал текущего устройства ввода (обычно 8). Входные регистры не влияют. Флаг C на выходе роли не играет.

Пример: используем счетчик прерываний для получения псевдослучайного числа в интервале от 1 до 16:

```
        LD C,$g_key
        RST 16
        EXX
        LD BC,-20
        ADD HL,BC
        LD A,(HL)
        AND #F
        INC A
```

Счетчик уменьшается на 1 каждое прерывание. Вы можете использовать его для запуска некоей процедуры через определенное время. Для этого в счетчик положите интервал времени в 1/50 долях секунды, а со смещением -14 адрес процедуры. Через этот же вектор Вы можете получить доступ к цепочке процедур вызываемых по прерываниям. Цепочка устроена очень просто: перед каждой процедурой цепочки резервируются 2 пустых байта, являющихся признаком конца цепочки. При установке в цепочку следующего "звена" сюда кладется его адрес. Занимается этим рестарт $l_im2(#1E).

```
        LD C,$g_key
        RST 16
        EXX
        LD BC,-16
        ADD HL,BC
        LD E,(HL)
        INC HL
        LD D,(HL)
        EX DE,HL        ;HL=адрес 1-го звена цепи

;достанем следующий адрес:

1$      DEC HL
        LD D,(HL)
        DEC HL
        LD E,(HL)

;если 0, то это конец цепи:

        LD A,D
        OR E
        RET Z

;иначе работаем с ним, к примеру печатаем этот адрес на экране, или проверяем, не лежит ли этот адрес
;в интересующем нас драйвере или уровне...

        CALL IMTST      ;DE сохраняется
        EX DE ,HL
        JR 1$
```

#12 (18) $g_typ

Рестарт возвращает в HL' адрес символьного устройства вывода и номер текущего устройства в регистре A (от #10 до #17). Входные регистры не влияют. Флаг C на выходе не играет роли. Как **пример:** переключение на другой драйвер печати на экране, используемый в программе tv.com:

```
$swap   LD C,$g_typ
        RST 16

;Сохраним номер текущего устройства:

        LD (DEV0+1),A

;Следующее устройство:

NEXT    INC A

;должно быть в пределах от #08 до #0F:

        CP #10
        JR NZ,$+3

        RRA ; A=8

DEV0    CP 0
        RET Z                   ;Других устройств нет

        LD B,A                  ;сохраним номер

;Достанем драйвер:

        LD C,$g_drv
        RST 16
        LD A,B
        JR C,NEXT               ;устройства видимо нет

;Проверим 13-й байт драйвера: у .typ=0, у .lpr=1

        EXX
        LD BC,13
        ADD HL,BC
        LD A,(HL)
        EXX
        OR A
        LD A,B                  ;номер устройства
        JR NZ,NEXT

;Всё OK. Можно переключаться:

        LD C,$swtyp             ;#1B
        RST 16
        RET C

        LD C,$g_typ
        RST 16
        EXX
        LD A,(HL)

;A=4/6 - горизонтальный размер символа в точках

        ...
```

#13 (19) $g_blk

Рестарт возвращает в HL' адрес вектора текущего устройства (см. **ПРИЛОЖЕНИЕ 6**) и номер текущего устройства в регистре A (от 0 до 7). Входные регистры не влияют. Флаг C на выходе роли не играет. Что касается примера, смотрите отрывок из программы format.com приведенный при описании рестарта $clear(#01).

#14 (20) $g_drv

Этот рестарт по номеру устройства в регистре A ищет драйвер. Если всё OK, то на выходе флаг C сброшен, A=номер канала драйвера, HL'=адрес тела драйвера. Возможные **ошибки:**

124 - Ошибка в структуре области каналов
125 - нет канала

Устроен этот рестарт очень просто и в принципе делает вот что:

```
        LD C,$stchn         ;найти канал номер A
        RST 16
        RET C

        EXX
        LD A,(HL)
```

;номер канала драйвера (см. **ПРИЛОЖЕНИЕ 7**)

```
        EXX
        RST 16              ;канал драйвера
        RET C

        EXX
        INC HL
        INC HL
        INC HL
        INC HL
```

;адрес тела драйвера (см. **ПРИЛОЖЕНИЕ 8**)

```
        LD E,(HL)
        INC HL
        LD D,(HL)
        EX DE,HL            ;HL=адрес тела драйвера
```

Примеры применения самого рестарта $g_drv см. описания рестартов $g_typ(#12) и $clear(#01).

#16 (22) $stchn

Сей рестарт по номеру в регистре A ищет соответствующий канал. Если находит, то:

- флаг C сброшен
- A сохранится
- HL'-адрес тела канала
- BC'-длина канала
- DE'-указатель за последний канал (он же $g_cnf+36).

Возможные ошибки те же, что и у рестарта $g_drv(#14). Пример из предыдущего рестарта также будет полезен. Еще **пример** - добывание канала пути альтернативной (или текущей) панели:

```
        LD C,$gmpan
        RST 16
        EXX
        DEC BC
        LD A,(BC)           ;текущая панель 0/1
        INC BC
        OR A
        JR NZ,$+3           ;Z для текущей

        INC BC

        LD A,(BC)
        LD C,$stchn
        RST 16
        RET C

        EXX
        INC HL
        INC HL
        INC HL
        LD A,(HL)           ;канал пути панели
        EXX
        RST 16
        RET C

        EXX
```

;HL указывает на путь альтернативной панели.

#17 (23) $dlchn

Этот рестарт находит канал по номеру в регистре A и удаляет его. Возможные ошибки см. $g_drv(#14). В жизни этот рестарт, как и создание канала применяется не часто. Любят этим заниматься программы set.com, exebat.com, print.com. Каналы необходимы при установке резидентов и тем более драйверов (ведь устройства суть каналы). Каналы удобны там, где надо передать не очень большой объём информации от одной программы другой (print) или сохранить кое-что для себя, а памяти для буферов нет (exebat). Для примера напишем программку, удаляющую все лишние каналы пользователя. Они могут оставаться в системе при аварийном завершении работы программ их создававших. Каналы пользователя имеют номера с #18 по #D7, из них первые 5 обычно заняты под системные надобности. Например:

#18 - путь левой панели
#19 - "тропа" поиска командников (path)
#1A - левая панель
#1B - путь правой панели
#1C - правая панель

Поэтому начнём с канала #1D:

```
        LD BC,#1D00+$dlchn
1$      LD A,B
        CP #D8
        RET Z
        RST 16
        JR NC,2$

;Флаг C: ошибка

        CP 125          ;нет канала
        SCF
        RET NZ
2$      INC B
        JR 1$
```

#18 (24) $crchn

Этот рестарт создает канал. Номер канала задается на входе в регистре A. Размер канала в регистре DE. Возможные **ошибки:**

124 - ошибка в структуре области каналов
126 - канал с таким номером уже есть
127 - неверная длина канала (DE=0)
128 - нет места для канала

При нормальном завершении работы рестарта адрес канала выдается на выходе в HL'. Рестарт необходим для таких системных трюков, как создание устройств и установка драйверов и резидентов. Им пользуются программы set.com и dev.com, т.к. только этим рестартом можно создавать каналы с номерами от #00 до #17 и от #D8 до #FF. Для создания же каналов пользователя предназначен следующий рестарт:

#19 (25) $newch

Этот рестарт также как и предыдущий создает канал, но номер ему он дает сам, выбирая наименьший свободный в диапазоне от #18 до #D7. Длину канала, как и для $crchn подавайте в DE. Возможны те же **ошибки**, что и у предыдущего рестарта, за исключением, пожалуй, ошибки 126. Зато можно схлопотать ошибку: 129 - нет свободного номера, но для этого придется постараться. Если все OK, то флаг C сброшен, а в регистре A будет номер созданного канала. Регистр HL' как и в предыдущем случае укажет на адрес тела канала. **Пример:**

```
        ORG 24000

        LD DE,CHSIZ
        LD C,$newch
        RST 16
        RET C

        PUSH DE
        EXX
        POP BC
        LD DE,CHBUF
        EX DE,HL
        LDIR
```

```
        XOR A
        LD A,#F8
        RET

CHBUF   DEFM |Канал пользователя|
CHSIZ   EQU $-CHBUF
```

Следующие 3 рестарта переключают устройства всех трех исдосовских типов: символьного ввода и вывода и блочные. Номер входного устройства всем трём рестартам подается в регистре B, причем имеют значение лишь 3 младших разряда, т.к. все 3 рестарта смены устройств маскируют входной параметр. Возможны **ошибки:**

121 - нет устройства, т.е. нет канала с номером B.
122 - нет драйвера, обслуживающего данное устройство.
124 - Ошибка в структуре области каналов

#1A (26) $swkey

Переключение устройств символьного ввода (клавиатурных). В теперешнем исдосе не применяется. В старых версиях на драйвере ttyin.blk "висели" 2 устройства. Одно было для работы в редакторе, т.е. на него переключались рестарты $smbgt(#6E) и $edstr(#7F). Другим устройством пользовались все остальные, и работало оно обычно лишь в режиме маленьких латинских букв. Т.о. развязывались режимы работы в строковом редакторе и в оболочке и утилитах, пользовавшихся рестартом $key(#07) напрямую или через рестарт $menu(#91). Однако, в момент переключения устройств останавливались макросы. Развязку пришлось осуществить другим способом, а от 2 устройств отказаться. В принципе никто не мешает написать программу, переключающуюся на другой драйвер клавиатуры и работающий с ним или даже с двумя поочередно.

#1B (27) $swtyp

Переключение устройств вывода на экран или принтер. Используется программами tv.com, edit.com и некоторыми программами печатающими на принтере, например: PrintLux, Picasso. Вход и выход описаны выше. Пример см. к рестарту $g_typ(#12).

#1C (28) $swblk

Переключение блочных устройств. Интерфейс стандартный для всех трех $sw... рестартов, и добавить тут решительно нечего. Применяется программами: arzt+, abc, format, doctor, практически всеми копировщиками. См. пример к рестарту $read(#0D).

Перемещение информации о текущем состоянии при работе с драйверами и устройствами можно представить в виде следующей схемы:

set.com: драйвер -> устройство
$ldnew(#1D): устройство -> вектор драйвер -> вектор
sw...(#1A-#1C): вектор -> старое устройство новое устройство -> вектор

Итак, set.com, загрузив драйвер, создает устройства (т.е. каналы), и копирует в них 8 байт из драйверов со смещения 8. Рестарты смены устройств сохраняют эти же 8 байт системного вектора в канале отключаемого устройства и грузят на их место аналогичные 8 байт из подключаемого устройства.

#1D (29) $ldnew

Этот рестарт используется при подключении новых устройств. И используется он, похоже, только программой set.com. На входе ему подают номер устройства в регистре A и адрес системного вектора минус 9 в регистре HL. Каналы устройства и драйвера должны быть уже созданы и заполнены. Из канала устройства рестарт достает 8 байт данных (со смещением 2, см. **ПРИЛОЖЕНИЕ 7**, перед описанием $g_drv(#14)) и копирует их в вектор соответствующего устройства (**ПРИЛОЖЕНИЯ 2,5,6**, см. рестарты $g_***(#11…#13)). Со смещением 0 в канале устройства рестарт добывает номер канала драйвера, по нему находит заполненный канал, в котором со смещением 4 (см. **ПРИЛОЖЕНИЕ 8**, перед рестартом $g_drv(#14)) находится адрес драйвера. После чего три точки входа в драйвер (смещения 2,4,6) копируются в системный вектор в позиции -8,-5,-2 соответственно. Нечто подобное делается и при переключении на другое устройство рестартами $sw***(#1A…#1C). Вот что делает set.com после загрузки или удаления любого резидента или уровня:

```
        LD C,$g_key
        LD B,3
1$      PUSH BC
        RST 16
        EXX
        LD BC,-9
        ADD HL,BC
```

```
        LD C,$ldnew
        RST 16
        POP BC
        INC C
        DJNZ 1$
        RET
```

#1E (30) $l_im2

Рестарт работает с цепочкой функций, вызываемых по прерываниям от таймера. Как уже рассказывалось при описании работы рестарта $g_key(#11), в исдосе по прерываниям вызывается сперва драйвер клавиатуры, а затем программы пользователя, образующие цепь. Перед точкой входа в каждую программу цепи резервируется 2 байта, для ссылки на следующую программу цепи. 0 в этих байтах является признаком конца цепочки. Рестарту на вход подается адрес устанавливаемой/снимаемой процедуры в регистре HL. Регистр A сообщает рестарту, устанавливать (A<>0) или удалять из цепи данную процедуру. На время вызова рестарта обязательно запретите прерывания! Возможная **ошибка** только одна и проявляется она лишь при снятии фоновой задачи: 39 - нет такой фоновой задачи.

Начало цепочки фоновых задач можно достать из вектора рестарта $g_key(#11)(см. выше)

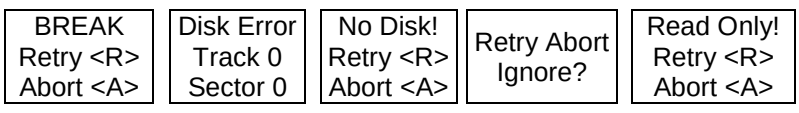
Пример:

```
        ORG 24000

        DEFS 4              ;2 точки входа резидента

        LD A,-1
        LD HL,IM_2
        LD C,$l_im2
        RST 16
        RET C

        XOR A
        LD A,#F8
        RET

        DEFW 0
IM_2    LD A,25
        DEC A
        JR NZ,$+4

        LD A,25

        LD (IM_2+1),A
        RET NZ

1$      LD A,0
        CPL
        LD (1$+1),A

        OUT (-2),A
        RET
```

Отлинкуйте эту программку с ключом /res, загрузите полученный резидент, установив на него файловый курсор и нажав <Enter>. Вызовите резидентную задачу командой @имя. В принципе этот же пример можно отлинковать и запустить как com-файл, но после подобного примерчика рекомендуем перезагрузиться, т.к. удаление из цепочки здесь не предусмотрено, а set.com при снятии резидента сам отключит его от цепочки IM 2.

#1F (31) $erdrv

Этот рестарт создан исключительно для драйвера sys_driv.blk, который вызывает его в случае ошибки, а рестарт всего лишь печатает ваши любимые окна и ждёт ответа.



Интерфейс следующий:
B=0: "Disk Error", D=трек, E=сектор
B=1: "Read Only"
B=2: "No Disk"
B=3: "Break"

На выходе в регистре A код нажатой клавиши в ВЕРХНЕМ регистре.

Хотя этот рестарт формально является рестартом нижнего уровня, в 0-ом уровне находится лишь маленькая процедурочка, опрашивающая 7-ой бит 1-го байта вектора блочных устройств ($g_blk(#13), см. выше) и, если бит сброшен, обращающейся по адресу ERDEV (вектор ядра системы, смещение +38, доступен через рестарт $g_cnf(#10), см. выше). Вы можете перехватить на себя данное обращение драйвера или запретить драйверу обращаться к этому рестарту, установив 7-ой бит 1-го байта вектора в 1. После этого по любой ошибке драйвера управление будет возвращаться сразу вашей программе с установленным флагом C и уже знакомыми кодами ошибок 6,7 и 20. Так поступают программы format, doctor и т.п. (см. пример к рестарту $clear(#01)).

Пример "тормозов" из новой версии драйвера ed128-b.blk:

```
TRANS

;1.8.96 BREAK:

        LD C,$ktest     ;нажата клавиша ?
        RST 16
        JR Z,1$         ;если нет, то иди на 1$

        LD C,$key       ;какая клавиша ?
        RST 16
        CP #16          ;<Cs/Space> ?
        JR NZ,1$

        PUSH BC
        LD BC,#300+$erdrv
        RST 16
        POP BC
        CP "A"          ;"A": "ABORT"
        LD A,20         ;отваливаем с ошибкой 20
        SCF
        RET Z
1$      …
```

ФАЙЛОВАЯ СЛУЖБА (DUD.SYS)

Для начала о структуре блочного устройства в iS-DOS'e. Любое блочное устройство (дискета, электронный диск или винчестер) с точки зрения iS-DOS'a представляет собой набор блоков размером в 256 байт и нумерующихся от 0 до 65535. Заголовок устройства содержится в 0-ом блоке. Подробно структура 0-го блока приводилась ранее при описании рестарта $binit(#0F), сейчас же повторим лишь следующее:

10(3)* "DSK": признак iS-DOS
18(2) Размер устройства в блоках
20(2) Номер первого блока главного каталога

Подробнее см. **ПРИЛОЖЕНИЕ 1**.

Размер устройства должен быть кратен 8. Связано это с представлением карты занятых и свободных блоков, в которой каждому блоку устройства соответствует 1 бит. Занятому блоку соответствует 1, свободному - 0. Карта начинается с 1-го блока диска и занимает столько блоков подряд, сколько необходимо для устройства соответствующего размера. Структура карты очень проста: по 1 биту на блок (8 блоков в байте, до 2048 блоков устройства на 1 блок карты), 0-му блоку устройства соответствует 7-ой бит 0-го байта карты, 1-му блоку - 6-ой бит и т.д. На обычном флоппи-диске карта занимает 2 блока. Максимально (на винчестере) карта может занимать 32 блока.

На устройстве должен находиться хотя бы один каталог, т.е. главный. Номер 0-го блока главного каталога указан в 0-ом блоке устройства со смещением 20.

Любой каталог в iS-DOS'e представляет собой файл, содержащий в себе 32-байтовые описатели файлов, первый из которых (0-ой) - это описатель самого каталога. Т.о. в iS-DOS'e у каталогов два описателя: внутренний и внешний. Исключение составляет лишь сам главный каталог, не имеющий внешнего описателя. У двух описателей должны быть одинаковые имена, основную же информацию о каталоге хранит внутренний описатель. Внешний же служит лишь для открывания каталога по имени и печати имени каталога в панели.

Любой файл (и каталог) в iS-DOS'e может быть непрерывным или сегментированным. Непрерывный файл (каталог) занимает на устройстве непрерывную последовательность блоков. Номер начального блока файла указывается в описателе файла со смещением 17. У сегментированных файлов здесь указывается **номер блока описателя сегментов**. Это специальный блок, имеющий следующую структуру:

0-й байт - число сегментов (от 0 до 85),
следующие 2 байта - номер начального блока 1-го сегмента,
3-й байт - размер сегмента в блоках (от 1 до 255).

Следующие 3 байта описывают следующий сегмент и т.д. (см. **ПРИЛОЖЕНИЕ 22**)

В iS-DOS'e существуют понятия текущего устройства (их может быть 8, они переключаются рестартом $swblk), текущего каталога (открывается рестартом $open1) и текущего файла.

#20 (32) $open

Этот рестарт открывает устройство и его главный каталог. Для этого считывается 0-й блок устройства и проверяется на наличие признака "DSK" со смещением 10 (13 для новой системы) в нём. Проверяется размер устройства на кратность 8 и неравенство 0. Затем рестарт пытается открыть главный каталог по номеру его начального блока, указанного в том же блоке со смещением 20.

Входных параметров нет. Выходных также. Возможные **ошибки:**

87 - нет признака "DSK" или неверный размер устройства
86 - каталог не открывается: ошибка в описателе каталога плюс все возможные ошибки драйвера

Пример: выход из программы fileshow. Ожидается нажатие клавиши. Если это <Cs/Ent>, то открывается новое устройство:

```
LD C,$kwait     ;ждем отпуска клавиш
RST 16
DEC C           ;ждем нажатия клавиши
RST 16

CP #17          ;Cs/Enter?
JR NZ,3$

LD C,$clear     ;очистить кэш
RST 16
```

; очистить буфер драйвера и настроить его на новую дискету:

```
LD C,$binit
RST 16
RET C

LD C,$open      ;открыть устройство
RST 16
RET C
```

```
;и, наконец, выход с перерисовкой экрана

3$      XOR A
        LD A,#F4
        RET
```

#21 (33) $open1

Открыть каталог. На входе в регистре DE подается номер начального блока каталога. Выходных параметров нет. Возможна ошибка с номером 86 (см. предыдущий рестарт). Считывается 0-й блок каталога и проверяется: байты 8..10 должны быть пробелами, в 11-ом байте 0-ой и 5-ый биты должны быть в 1, 19-й и 20-й байты должны содержать номер 0-го блока. При любом несоответствии получим ошибку 86.

Возможные **ошибки:**

81 - файл не найден
85 - ломаный блок описателя сегментов (текущего или искомого каталога)
86 - ломаный каталог

В качестве примера рассмотрим как программа coca.com открывает устройство и каталог альтернативной панели:

;достанем системный вектор оболочки (см. **ПРИЛОЖЕНИЕ 17**)

```
    LD C,$g_mpan        ;#87
    RST 16
    EXX
    DEC BC
    LD A,(BC)           ;номер текущей панели
    INC BC
    OR A
    JR NZ,$+3
    INC BC
    LD A,(BC)           ;канал альтернативной панели
    LD C,$stchn
    RST 16
    RET C
    EXX
```

;HL=адрес тела канала альтернативной панели (см. **ПРИЛОЖЕНИЕ 18**)

```
    LD B,(HL)           ;номер устройства
    INC HL
    LD E,(HL)
    INC HL
    LD D,(HL)
```

; DE= номер 0-го блока каталога

```
    LD C,$swblk
    RST 16
    RET C

    LD C,$open1
    RST 16
    RET C

    …
```

#23 (35) $crfil

Создать файл в текущем каталоге на текущем устройстве. В HL - адрес 32-байтового описателя файла. В нем необходимо указать имя, расширение файла, байт состояния (если непрерывный - 6-ой бит =1). Старший байт длины обнуляется, т.о. создать можно файл длиной не более 255 блоков (65280 байт). Если Вам требуется создать файл большей длины, пользуйтесь дополнительно рестартом $fadd(#2F) или $eadd(#31). После создания файл будет открыт. Следует помнить, что сей рестарт, равно как и большинство рестартов этого уровня работает через кэш. Поэтому:

1. Всегда следите за сохранностью кэша;
2. Так как результат действий рестарта будет также в кэше в виде модифицированных блоков, не забудьте по окончании работы программы сохранить эти модифицированные блоки на устройстве с помощью рестарта $flush(#02).

Возможные **ошибки:**

82 - файл с таким именем и типом уже существует
84 - переполнение каталога (128 файлов)
92 - на диске нет места
105 - каталог непрерывен и заполнен до краев.
А также любые ошибки кэша и драйвера.

Для примера предлагается кусок из программы move.com:

```
    …
;Открыть выходной каталог:
    CALL G_DEST
    RET C
    SBC HL,HL
    LD (FLEN),HL        ;Обнулить длину
    LD HL,FSTAT
    SET 6,(HL)          ;Непрерывный файл
    LD HL,FILE
    LD C,$crfil
    RST 16
    RET C
;Копируем описатель файла:
    EXX
    EX DE,HL
    LD HL,FILE+32
    LD BC,32
    LDIR
    LD C,$putf          ;и сохраняем его
    RST 16
    RET C
    …
FILE    EQU $
FSTAT   EQU FILE+11
FLEN    EQU FILE+14
```

#24 (36) $erfil

Найти файл по имени и уничтожить. Если файл не найден, не докладывать. В регистре HL подается адрес 11-байтового описателя: 8 байт имени и 3 байта расширения.
Возможные **ошибки:**

89 - файл защищен от удаления
85 - неверный блок описателя сегментов
93 - попытка освободить блоки за концом устройства

Пример из программы sv_image.com:

```
    ...
;текущая дата для создаваемого файла:
    LD C,$g_com         ;#45
    RST 16
    EXX
    DEC HL
    DEC HL
    LD D,(HL)
    DEC HL
    LD E,(HL)
    LD (DATE),DE
;в зависимости от ключа: удалять или нет:
    LD HL,FILE
OPTION LD A,0
    OR A
    LD C,$erfil
```

```
        CALL NZ,16
        RET C

        DEC C           ;$crfil
        RST 16
        RET C

        …
```

#25 (37) $fopen

Найти и открыть файл по имени и расширению. В HL на входе подается адрес 11-байтового описателя файла. Если найденный файл - каталог, то он открывается.

Выход (если без ошибок, т.е. флаг C сброшен), то в регистре A:
#00 - открыт файл;
#20 - открыт каталог;
HL' - адрес системного 32-байтового описателя файла (этот адрес можно получить с помощью рестарта $bkfcb(#35)).

Возможные **ошибки:**
81 - файл не найден
85 - ломаный блок описателя сегментов (текущего или искомого каталога)
86 - ломаный каталог

Рассмотрим для примера кусочек программы unicolor.com в той части, где программа ищет подкаталог HELP\ и в нем файл с расширением .hlp:

```
        …
;Текущий путь для печати его в окне:

        LD HL,PATH+8        ;буфер для пути
        PUSH HL
        LD B,pathSZ         ;длина буфера

;Заполним буфер пробелами

        LD (HL)," "
        INC HL
        DJNZ $-3
        POP HL

        XOR A
        LD DE,56
        LD C,$g_way         ;#47
        RST 16
        RET C

;Ищем конец пути, т.е. первый пробел:

1$      LD A,(HL)
        CP " "
        INC HL
        JR NZ,1$

        DEC HL
        PUSH HL

;В системном описателе - текущий файл:

        LD C,$bkfcb
        RST 16
        EXX
        POP DE

;Перекодируем имя текущего файла в буфер пути:

        LD C,$convr
        RST 16
        RET C

;Попытаемся найти \HELP\*.hlp

        XOR A
        LD (hlpCSR+1),A
```

```
;Копируем 8 байт имени текущего файла в буфер имени hlp-файла:

        LD DE,hlpNAM
        PUSH DE
        LD BC,8
        LDIR

; …и, наконец, пользуемся нашим рестартом $fopen. Сперва откроем каталог:

        LD HL,HELP
        LD C,$fopen
        RST 16
        POP HL
        JR C,GW_EX          ;уходим по ошибке

;а затем им же и файл:

        RST 16
        JR C,GW_EX

;не отходя далеко от кассы, достанем длину файла:

        EXX
        LD BC,14
        ADD HL,BC
        LD E,(HL)
        INC HL
        LD D,(HL)

        …

hlpNAM  DEFS 8
        DEFM "hlp"
HELP    DEFM "HELP"
```

#26 (38) $opnum

Открыть файл по номеру в регистре A.
На **выходе**, если все в порядке: A-сохраняется, HL' - адрес системного 32-байтового описателя файла.

Возможные **ошибки:**
80 - номер файла слишком велик (больше 21-го байта описателя каталога -1)

Пример из программы calc.com. С помощью рестарта $opnum программа открывает файл, на котором стоит курсор:

```
        ORG 24000

;пытаемся открыть параметр:

        LD C,$opcat
        RST 16
        RET C
        JR Z,1$

;параметра в строке нет, следовательно, открываем файл под курсором:

        LD C,$mwait         ;"Wait Please!"
        RST 16
        LD C,$g_cur         ;#8A
        RST 16

;A=номер файла, на котором стоит курсор. Открываем файл по его номеру в каталоге:

        LD C,$opnum
        JR 2$
1$      EXX

;открываем параметр по имени:

        LD C,$find
2$      RST 16
        RET C

        EXX

;каталоги не обслуживаем:
```

```
        LD DE,11
        ADD HL,DE
        BIT 5,(HL)
        RET NZ

        LD DE,26-11
        ADD HL,DE
```

;HL указывает на длину файла в его описателе

```
        …
```

#27 (39) $gname

Считать в кэш или найти в кэше соответствующий блок каталога и установить регистровую пару HL' на описатель искомого файла в этом блоке.

Вход: E - номер файла.

Файл **не открывается**, т.е. сохраняется текущий открытый файл.

Возможные **ошибки**: см. предыдущий рестарт.

Пример из программы univ.res. Цикл подбора файлов, подходящих под маску:

```
PODBOR
        LD DE,#100
1$      INC E           ;номер файла
        LD C,$gname
        RST 16
        JR C,2$

        PUSH DE
        EXX
```

;HL указывает на описатель файла в кэше

```
        LD DE,TRAF      ;трафарет (11 байт)
```

;сравнение 11 байт по адресам в HL и DE:

```
        LD C,$cpfil     ;#8F
        RST 16
        POP DE
        JR NZ,1$

        PUSH DE
        CALL WIBCN      ;работа с файлом
        POP DE
        RET C

        JR 1$
```

;Ошибка. В регистре A ее код:

```
2$      CP 80           ;каталог кончился
        RET Z
        SCF             ;иначе: что-то серьезное
        RET
```

#28 (40) $putf

Сохранить в текущем каталоге измененный вектор текущего файла (тот, что доступен через $bkfcb(#35)). Полезно для переименования файла и тому подобных операций. **Номер файла** лежит по адресу bkfcb-1.

Возможные **ошибки:** ошибки файловые, ошибки кэша и драйвера:

80 - номер файла слишком велик

85 - неверный блок описателя сегментов (ломаный каталог)

7 - ошибка чтения/записи (драйвер)

62 - несоответствие числа модифицированных блоков (кэш)

Вот так выглядит применение этого рестарта на выходе из программы calc:

```
…

;в HL - контрольная сумма открытого файла
;в (ADRKK+1) - адрес контрольной суммы в системном описателе файла (bkfcb+26)

ADRKK   LD (0),HL
```

```
LD C,$putf
RST 16
RET C

LD C,$flush
RST 16
RET C

XOR A
RET
```

#29 (41) $rpart

Прочитать DE байт из текущего открытого файла со смещением AHL байт от начала файла по адресу в IX.

Возможные **ошибки:**
100 - попытка чтения за концом файла (т.е. AHL+DE больше длины файла).
106 - файл не открыт
170 - чтение 0 байт (DE=0)
171 - файл защищен от чтения
7 - ошибка чтения/записи (драйвер)

Пример из программы tree.com. Она открывает главный каталог, находит там файл treecat.txt, проверяет, влезет ли он в буфер под кэш и считывает его:

```
…

LD C,$open          ;главный каталог
RST 16
RET C

LD HL,FILE
LD C,$find          ;найти по имени файл
RST 16
RET C

EXX
LD BC,#000F
ADD HL,BC
LD D,(HL)
DEC HL
LD E,(HL)           ;DE=длина файла
LD HL,BUFF

;BUFF - буфер за концом программы

ADD HL,DE

; адрес конца файла в буфере

LD C,$g_cnf         ;#10
RST 16
PUSH DE             ;длина файла
EXX
LD BC,5
ADD HL,BC
LD E,(HL)
INC HL
LD D,(HL)
PUSH DE             ;адрес кэша
EXX
POP DE
EX DE,HL
XOR A
SBC HL,DE
LD A,130
POP DE              ;длина файла
RET C               ;не хватает памяти

XOR A
LD L,A
LD H,A              ;AHL=0
```

```
        LD IX,BUFF
        LD C,$rpart
        RST 16
        RET C

        …
```

#2A (42) $wpart

Записать DE байт в текущей открытый файл со смещением AHL байт от начала файла с адреса в IX.

Возможные **ошибки:**
100 - попытка записи за концом файла (т.е. AHL+DE больше длины файла).
106 - файл не открыт
170 - запись 0 байт (DE=0)
172 - файл защищен от записи
7 - ошибка чтения/записи (драйвер)
62 - несоответствие числа модифицированных блоков (кэш)

Для **примера** рассмотрим сохранение параметров программой date.com на выходе из программы:

```
;сохранение текущей среды в буфере curID

        LD IX,curID
        XOR A
        LD C,$p_stat
        RST 16

;Достать среду последнего запущенного com-файла, т.е. date.com

        LD C,$g_com
        RST 16
        EXX
        LD BC,#FFF7
        ADD HL,BC
        PUSH HL
        POP IX

;открыть файл date.com:

        XOR A
        LD C,$g_stat
        RST 16
        RET C

;записать в него 1 байт со смещением 2 из WDCSR:

        LD IX,WDCSR
        LD DE,1
        XOR A
        LD HL,2
        LD C,$wpart
        RST 16
        RET C

;сосчитать контрольную сумму date.com

        LD HL,CALC
        LD C,$run
        RST 16
        JR NC,0$

;ошибка:

        CP 37           ;нет резидентной задачи?
        SCF
        RET NZ
0$      LD C,$flush
        RST 16
        RET C
```

```
        …
CALC    DEFM "@calc"
        DEFB 13
```

#2B (43) $rifle

Прочитать B блоков открытого файла начиная с DE-го по адресу в HL. Чтение осуществляется мимо кэша, однако следует помнить, что если файл сегментированный, то система сама подчитывает в кэш блок описатель сегментов файла. Т.е. следите, чтобы ваша программа не повредила кэш к тому моменту, когда Вы соберетесь воспользоваться этим рестартом, равно как и большинством рестартов этого уровня системы.

Возможные **ошибки:**
101 - попытка чтения за концом файла (т.е. DE+B больше длины файла в блоках).
106 - файл не открыт
7 - ошибка чтения/записи (драйвер)

Пример из жизни копировщиков: начало цикла копирования программы image.com:

```
COPY

;B=размер буфера копирования

        PUSH BC

;переключиться на входное устройство:

FROM    LD BC,$swblk
        RST 16
        POP BC
        RET C

;Считать B блоков в буфер BUFF из открытого файла:

        LD HL,BUFF
        LD C,$rifle
        RST 16
        RET C

 ;переключиться на выходное устройство:

        PUSH BC
DEST    LD BC,$swblk
        RST 16
        POP BC
        RET C

;Записать B блоков на текущее устройство

        LD C,$write
        RST 16
        RET C

        …
```

#2C (44) $wifle

Записать B блоков в открытый файл начиная с DE-го с адреса в HL. Запись осуществляется мимо кэша. Посему после использования данного рестарта может возникнуть ситуация несоответсвия блоков в кэше блокам на устройстве, что чревато боком. Исправить дело можно очистив принудительно кэш рестартом $clear(#01) или пересоздав его рестартом $creat(#00).

Возможные **ошибки:**
101 - попытка чтения за концом файла (т.е. DE+B больше длины файла в блоках).
106 - файл не открыт
7 - ошибка чтения/записи (драйвер)

В качестве **примера** - кусок программы ibm_is.com:

```
;Открыть выходной файл:

O_FILE  LD A,0
        CALL OPNUM
        JR C,0$
```

```
        EXX
        LD BC,15
        ADD HL,BC
        LD E,(HL)
        INC HL
        LD D,(HL)

;DE=длина файла в блоках

        LD A,(OBUFSZ+2)
        LD B,A

;A=B=размер выходного буфера в блоках.
;Добавить в конец файла A блоков:

        LD C,$fadd
        RST 16
        JR C,0$

;Записать B блоков в файл начиная с DE-го блока из буфера O_BUF:

        LD C,$wifle
        LD HL,O_BUF
        RST 16
0$      POP HL
        POP DE
        POP BC
        RET

;Подпрограмма открывает файл по номеру в регистре A.

OPNUM   CP 0
        RET Z                   ;файл уже и так открыт

        LD (OPNUM+1),A
        LD C,$opnum
        RST 16
        RET
```

#2D (45) $qrvbl

Прочитать один блок номер DE открытого файла в кэш.
Выход (если все O.K.):
HL' - адрес блока в кэше. (Если блок уже находился в кэше, то считывания с устройства не происходит, рестарт просто находит его и сообщает адрес. Тогда A=1, иначе A=0.)
DE'- адрес описателя блока в кэше. Необходим для рестарта $modo(#2E).

Возможные **ошибки:**
101 - попытка чтения за концом файла (т.е. AHL+DE больше длины файла).
106 - файл не открыт
170 - чтение 0 байт (DE=0)
171 - файл защищен от чтения
7 - ошибка чтения/записи (драйвер)

Пример: очистка каталога при его создании программой mkdir.com:

```
        LD HL,(FILE+14)

;HL=длина файла-каталога в байтах

        LD A,L
        OR A
        LD A,H
        JR NZ,$+3
        DEC A
        OR A
        RET Z

        LD B,A

;B=A=число блоков, которые нужно очистить

        LD DE,1         ;номер блока
1$      LD C,$qrvbl     ;прочитать блок
```

```
        RST 16
        RET C

        EXX

;HL=адрес блока в кэше

        XOR A
        LD B,A

;256 нулей:

        LD (HL),A
        INC HL
        DJNZ $-2

        PUSH DE         ;адрес описателя блока
        EXX
        POP HL
        EX DE,HL
        LD C,$modo
        RST 16
        RET C

        EX DE,HL
        INC DE          ;номер блока
        DJNZ 1$
        RET
```

#2E (46) $modo

Модифицирование блока в кэше, т.е. пометка блока как измененного и подлежащего выгрузке на устройство рестартом $flush(#02) или автофлашем, запускающимся при превышении числа модифицированных блоков некоего предельного значения.

Вход: DE - адрес описателя блока в кэше. Этот адрес выдается на выходе рестартами $quard(#06) и $qrvbl(#2D). Обычно рестарт $modo используется сразу после этих рестартов. Именно сразу, поскольку блок может сместиться или вообще быть вытеснен из кэша при вызове многих рестартов, считывающих блоки в кэш, например: $open, $open1, $fopen, $find, $rpart, $gname, $opnum, $crfil и др.

Возможные **ошибки:**
62 - несоответствие числа модифицированных блоков (кэш)
7 - ошибка чтения/записи (драйвер)

Вот что делает программа arzt+.com, когда сдвигает файл is_dos.sys:

```
        …

;Прочитать 0-ой блок устройства в кэш:

        LD DE,0
        LD C,$quard
        RST 16
        RET C

        EXX
        LD BC,32+17
        ADD HL,BC
        LD BC,(FR1ST+1)

;BC=номер блока, где после сдвига находится файл is_dos.sys:

        LD (HL),C
        INC HL
        LD (HL),B

;Сосчитаем новую контрольную сумму описателя файла is_dos.sys:

        LD BC,31-18
        ADD HL,BC
        LD B,#20
        XOR A

        XOR (HL)
        DEC HL
        DJNZ $-2
```

;Контрольная сумма также помещается в 0-ом блоке устройства:

```
DEC HL
DEC HL
DEC HL
DEC HL
LD (HL),A
```

;И, наконец, модифицируем блок в кэше:

```
LD C,$modo
RST 16
RET C

…
```

#2F (47) $fadd

Добавить A блоков к сегментированному файлу. Добавленные блоки вставляются перед блоком номер DE (первый добавленный блок становится DE-ым). Если DE слишком велик, то добавление все равно производится в конец файла.

Возможные **ошибки:**
62 - несоответствие числа модифицированных блоков (кэш)
92 - нет места на диске
94 - переполнение блока описателей сегментов (85 сегментов)
102 - непрерывный файл
103 - добавить/удалить 0 блоков (A=0)
106 - файл не открыт
7 - ошибка чтения/записи (драйвер)

Пример из программы wet.com. Добавление специального блока в начало текстового файла и заполнение его вектором:

```
EXIT    XOR A
        LD E,A
        LD D,A          ;DE=0: номер блока
        INC A           ;A=1: число блоков
        LD C,$fadd
        RST 16
        RET C

        LD HL,BUFF      ;буфер блока
```

;Очистка буфера:

```
        PUSH HL
        XOR A
        LD B,A
        LD (HL),A
        INC HL
        DJNZ $-2

        LD HL,VECTOR            ;вектор редактора
        LD C,VECSZ              ;размер вектора
        LD DE,BUFF+#80+edcsr    ;+смещение
        LDIR
        POP HL                  ;буфер
        LD D,B
        LD E,B                  ;DE=0
        LD C,$wvblk             ;записать 1 блок
        RST 16
        RET C

        …
```

#30 (48) $fcut

Удалить A блоков из сегментированного файла. Номер блока задается в DE.

Возможные **ошибки:**
62 - несоответствие числа модифицированных блоков (кэш)
94 - переполнение блока описателей сегментов (85 сегментов)
102 - непрерывный файл
103 - добавить/удалить 0 блоков (A=0)
106 - файл не открыт
7 - ошибка чтения/записи (драйвер)

Пример из программы dry.com. Удаление первого блока wet-файла для превращения его обратно в текстовый:

```
        …

        XOR A
        LD D,A
        LD E,A          ;DE=0: номер блока
        INC A           ;A=1: число блоков
        LD C,$fcut
        RST 16
        RET C

        …
```

#31 (49) $eadd

Добавить в конец файла DE байт. Если при этом произойдет увеличение длины файла в блоках, тогда файл должен быть сегментированным.

Возможные **ошибки:**
92 - нет места на диске
94 - переполнение блока описателей сегментов (85 сегментов)
102 - непрерывный файл
103 - добавить/удалить 0 байт (DE=0)
106 - файл не открыт
А также ошибки **7**,**20**,**62**

Пример из программы sv_image.com, копирующей устройство в файл. Если размер файла более 255 блоков, то после его создания приходится в цикле добавлять блоки. В данном случае по 128 блоков.

```
        …

        LD HL,FILE
        LD C,$crfil
        RST 16
        RET C

FADD    LD BC,$eadd     ;B=число циклов
        DEC B
        INC B
        JR Z,1$

        LD DE,#8000     ;128 блоков (D=128)
        RST 16
        RET C
        DJNZ $-2
1$

        …
```

#32 (50) $ecut

Удалить от конца файла DE байт. Файл может быть непрерывным.

Возможные **ошибки:**
103 - добавить/удалить 0 байт (DE=0)
104 - DE больше длины файла
106 - файл не открыт
А также ошибки **7**,**20**,**62**

Пример окончания программы is_ibm.com:

```
        …
        LD DE,(SHIFT+1)
;DE=смещение в выходном буфере=число байт, которые надо добавить в конец файла.
        LD A,E
        OR D
        JR Z,FLUSH
;Добавим и запишем весь буфер:
        LD HL,(OBUFSZ+1)
        DEC HL
        LD (SHIFT+1),HL
        CALL PRINT
        RET C
;Удалим всё лишнее: DE=(SHIFT+1)
        LD HL,(OBUFSZ+1)
        XOR A
        SBC HL,DE
        EX DE,HL
        LD C,$ecut          ;отрезать DE байт
        RST 16
        RET C
FLUSH   LD C,$flush         ;выгрузка на диск
        RST 16
        RET C
EXIT    XOR A
        LD A,#F4            ;перепечатка экрана
        RET                 ;выход в iS-DOS
```

#33 (51) $g_cat

Подать в регистрах:

A – номер текущего устройства (0-7);
HL'= номер 0-го блока текущего каталога (байты 19,20 описателя каталога или bkfcb+51,+52),
BC'=номер 0-го блока старшего каталога (байты 12,13 описателя каталога или bkfcb+44,+45),
D'= уровень вложенности текущего каталога (байт 16 описателя каталога или bkfcb+48),
E'= количество файлов в каталоге включая удаленные и сам каталог (байт 21 описателя каталога или bkfcb+53).
Флаг C сохраняется и признаком ошибки не является.

Вот так выглядит начало программы mkdir.com (mkdir.res). Поскольку панели не в состоянии правильно отобразить каталоги с уровнем вложенности более шести (точки не поместятся), в программу встроен следующий ограничитель:

```
        ORG 25000
        LD C,$g_cat
        RST 16
        EXX
        LD A,D          ;уровень вложенности
        CP 6
```

;Сбросить флаг Z, дабы выйти в iS-DOS без приключений:

```
        INC A           ;NZ
        RET NC          ;если >6, то выход

        …
```

#34 (52) $find

Открыть файл по имени. Почти то же, что и $fopen, но не открывает каталог, если файл оказывается таковым. **Вход:** HL - 11 байт имени и расширения файла.

Выход: HL' - 32-байтовый системный описатель файла,
A=номер найденного файла.

Возможные **ошибки:**
81 - файл не найден
85 - ломаный блок описателя сегментов (текущего или искомого каталога)
86 - ломаный каталог
7 - ошибка чтения/записи (драйвер)

См. **примеры** из программ calc.com ($opnum(#26)) и tree.com ($rpart(#29)).

#35 (53) $bkfcb

Установить регистр HL' на адрес системного описателя файла и каталога (см. **ПРИЛОЖЕНИЕ 9**). Флаг C не изменяется.

В качестве **примера** - начало резидента calc.res:

```
        DEFS 4          ;стандартное начало резидентов - адреса точек входа:
                        ;0. инициализации (вызывается после перемещения резидента программой set.com.
                        ;Если 0, то не вызывается.
                        ;2. Рабочая точка входа. Если 0, то с 4-го байта.
        LD C,$bkfcb
        RST 16
        EXX
        LD IX,-1        ;здесь будет сумма
        LD BC,14
        ADD HL,BC
        LD A,(HL)       ;DEA = длина файла
        INC HL
        LD E,(HL)
        INC HL
        LD D,(HL)
        LD C,10
        ADD HL,BC       ;адрес контрольной суммы
        LD B,A
        LD C,$qrvbl
        OR A
        JR NZ,$+3
L1      DEC DE          ;следующий (предущий) блок
        RST 16          ;читаем блок в кэш
        RET C

        …
```

См. также пример к рестарту $fopen(#25) (unicolor.com).

#36 (54) $p_stat

Сохраняет значения текущего устройства, каталога и файла в 4-х байтах. Если на входе A=0, то заполняются 4 байта по адресу в IX, иначе - в канале номер A.

Возможные **ошибки:** (если A<>0)
124 - испорчена область каналов
125 - нет канала

Пример с каналом из программы exebat.com. При запуске программа создает канал, в котором сохраняет среду (устройство, каталог, файл) рабочего batch-файла, а также указатель в нем и ссылку на предыдущий batch-канал.

```
…
LD DE,9          ;длина канала, байт
LD C,$newch      ;создать канал
RST 16
RET C
EXX
PUSH AF          ;номер канала
PUSH HL          ;адрес тела канала
POP IX
CALL KETTE       ;A=(g_com-1) - номер
LD (IX+4),A      ;bat-файла в цепочке
XOR A
LD (IX+chr),A    ;смещение в блоке
LD (IX+blk),A    ;номер блока
POP AF           ;номер канала
LD (HL),A
```

; и, наконец, сохраняем среду в канале:

```
LD C,$p_sta
RST 16
RET C

…
```

См. также пример к рестарту $w_part(#2A) (date.com)

#37 (55) $g_stat

Устанавливает среду по 4-байтовому вектору:
0 - устройство
1,2 - каталог
3 - файл
Если на входе A=0, то берутся 4 байта по адресу в IX, иначе - из канала номер A.

Возможные **ошибки:**
121 - нет устройства
86 - каталог не открывается
80 - номер файла слишком велик
124 - испорчена область каналов
125 - нет канала
7 - ошибка чтения/записи (драйвер)

Пример из программы date.com. Восстановление среды на выходе из программы. Причем не только текущей, но и среды для рестарта $fmrst(#41), т.к. с его помощью восстанавливает среду exebat.com:

```
…
LD C,$flush      ;выгрузка на диск
RST 16
RET C

LD IX,curID      ;4 байта, в которых мы сохранили текущую среду в начале работы
XOR A
LD C,$g_stat
RST 16
RET C
```

;копируем среду curID в системный вектор:

```
LD C,$g_com      ;#45
RST 16
EXX
LD HL,curID
LD BC,4
LDIR
RET
```

См. также пример к рестарту $w_part(#2A) date.com

#38 (56) $distr

Найти A свободных блоков подряд на текущем устройстве и занять их.
Выход: BC'= номер начального блока найденной области.

Возможные **ошибки:**
92 - нет места на диске
162 - занять 0 блоков (A=0)
А также ошибки кэша и драйвера.

Пример преобразования непрерывного файла в сегментированный из программы rename.com. Для этого надо занять на диске 1 блок, и сделать в нем описатель сегментов файла.

```
$FLIP   LD C,$bkfcb         ;#35
        RST 16
        EXX
        LD BC,#B
        ADD HL,BC
        BIT 6,(HL)
        JR Z,S_C            ;обратное преобразование

        LD A,1              ;число блоков
        LD C,$distr
        RST 16
        RET C
```

;У каталогов старший байт длины обнулять

```
        BIT 5,(HL)          ;NZ-каталог, Z-файл
        INC HL
        INC HL
        INC HL
        LD A,(HL)
        INC HL
        LD E,(HL)
        INC HL
        LD D,(HL)
        JR Z,$+4
        LD D,0              ;каталог
```

;Округление длины файла до блока:

```
        OR A
        JR Z,$+3
        INC DE

        LD (FLENG+1),DE
        INC HL
        LD E,(HL)
        INC HL
        LD D,(HL)
        LD (FSDBN+1),DE     ;начало файла
```

;Прочитаем этот блок в кэш:

```
        EXX
        PUSH BC             ;номер блока
        LD D,B
        LD E,C ; DE=BC
        LD C,$quard
        RST 16
        POP IX
        RET C

        ...
```

#39 (57) $rtran

Единственный рестарт в этом уровне, не имеющий никакого отношения к файловой службе. Просто в свое время он не поместился в 0-ой уровень DOS. Рестарт перенастраивает резидент (драйвер) на новый адрес после перемещения последнего. Пользуется этим рестартом вроде бы только программа set.com при установке и снятии резидентов и драйверов.

Вход: BC=длина,
HL=старый адрес,
DE=новый адрес.

#3B (59) $crf__

Создание файла. То же что и $crfil(#23), но без проверки на существование файла с такими же именем и расширение, что позволяет ускорить процесс копирования файлов и не проверять наличие двойников файла дважды в копировщиках copy.com, filecopy, from_trd и т.п.

Возможные **ошибки:**
84 - переполнение каталога (128 файлов)
85 - ломаный блок описателя сегмента каталога
92 - на диске нет места
105 - каталог непрерывен и заполнен до краев.
А также любые ошибки кэша и драйвера.

Пример из программы treecat.com. Старый файл treecat.txt в главном каталоге удаляется, а по сему незачем еще раз искать его при создании:

```
        …

        LD HL,FILE
        LD C,$erfil         ;удалить старый файл
        RST 16
        RET C

        PUSH HL
        LD HL,(PTR+1)
        LD (HL),3           ;код 3 в конец файла
        INC HL
        LD DE,BUFF          ;буфер с текстом
        PUSH DE
        POP IX
        SBC HL,DE
        LD (FLENG),HL       ;длина файла
        EX DE,HL
        POP HL ; FILE
        LD C,$crf__         ;создать новый файл
        RST 16
        RET C
```

;записать в него текст из буфера (файл открыт)

```
        XOR A
        SBC HL,HL
        LD C,$wpart
        RST 16
        RET C

        …
FILE    DEFM "treecat txt"
        DEFB #41
        DEFW 0
FLENG   DEFW 0
        DEFS 14
DATE    DEFW 0

        …
```

#3C (60) $erf__

Удаление открытого файла. Иногда бывает удобно. Действительно, зачем искать по имени файл, чтобы удалить его рестартом $erfil(#24), если файл уже и так открыт?

Возможные **ошибки:**
89 - файл защищен от удаления
85 - неверный блок описателя сегментов
93 - попытка освободить блоки за концом устройства
А также ошибки кэша и драйвера.

Вот как выглядит упрощенный вариант начала программы delete.com в той части, которая удаляет файл, указанный как параметр:

```
        ;Обработка командной строки:

START   LD C,$opcat     ;#43
        RST 16
        JR NZ,OPEX      ;конец строки

        OR A
        JR NZ,START     ;ключ игнорируем

;A=0:Файл. Командный режим:

        EXX             ;в HL - описатель файла
        LD C,$find      ;ищем этот файл
        RST 16
        JR NC,ER__F

;ошибка поиска:

        CP 81           ;нет файла
        SCF
        RET NZ          ;другая ошибка

        JR FILEX        ;нормальный выход

;удаляем найденый файл:

ER__F   LD C,$erf__
        RST 16
        RET C

        LD C,$flush       ;выгрузка на диск
        RST 16
        RET C

;восстановить среду до вызова $opcat:

FILEX   LD C,$fmrst       ;#41
        RST 16
        XOR A
        LD A,#F4          ;перерисовать экран
        RET               ;выход

OPEX
;Интерактивный режим

        …
```

#3D (61) $rvblk

Чтение одного блока открытого файла в буфер по адресу в HL. В DE - номер блока. Блок читается через кэш. Рестарт используется текстовым редактором, вьюером и программой печати текстовых файлов.

Возможные **ошибки:**
101 - попытка чтения за концом файла (т.е. DE больше длины файла в блоках-1).
106 - файл не открыт
7 - ошибка чтения/записи (драйвер)

#3E (62) $wvblk

Запись одного блока открытого файла из буфера по адресу в HL. В DE - номер блока. Блок пишется через кэш. Рестарт используется текстовым редактором, вьюером и программой печати текстовых файлов.

Возможные **ошибки:**
101 - попытка записи за концом файла (т.е. DE больше длины файла в блоках-1).
106 - файл не открыт
А также ошибки кэша и драйвера.

#3F (63) $free

Освободить A блоков на текущем устройстве, начиная с DE-го. Парный к рестарту $distr(#38).

Возможные **ошибки:**
93 - освободить блоки за концом устройства (DE+A больше размера диска)
163 - освободить 0 блоков (A=0)
А также ошибки кэша и драйвера.

Пример из программы is_gens.com. Под конец работы программа пытается сделать выходной файл непрерывным. Заметим, что он однозначно сегментированный ибо создается таковым. Поэтому здесь нет проверки на сегментированность.

```
CONTIG
;Прочитаем блок описатель сегментов файла:
        LD C,$bkfcb
        RST 16
        EXX
        LD BC,17
        ADD HL,BC
        LD E,(HL)
        INC HL
        LD D,(HL)
        LD C,$quard
        RST 16
        RET C

        EXX             ;DE'=(FSDBN), HL'=FSDBN+1
        LD A,(HL)       ;число сегментов
        INC HL
        LD E,(HL)
        INC HL
        LD D,(HL)
        DEC A           ;A=число сегментов=1?
        JR NZ,1$
;1 сегмент: освобождаем блок-описатель:
        PUSH DE         ;первый блок файла
        EXX             ;DE=(FSDBN)
        LD A,1          ;число блоков
        LD C,$free
        RST 16
        POP DE          ;первый блок файла
        RET C

        LD (HL),D
        DEC HL
        LD (HL),E
;поднимем бит непрерывности:
        LD BC,-6
        ADD HL,BC
        SET 6,(HL)
1$
        ...
```

Рестарты 2-го уровня системы (с #40-го по #54) занимаются в основном обработкой текстовых командных строк, из которых состоят batch-файлы, которые подаются в строковом мониторе mon.res (mon.com) и которыми запускаются любые командные файлы из оболочки при отработке нажатых клавиши <Enter> (строки пишутся в файле Q:extent.txt), клавиш <3> и <H> (extview.txt и extprint.txt в каталоге Q:SHELL\ соответственно) и, наконец, всех остальных клавиш, строки для которых описаны в файле Q:SHELL\extkey.txt. Обрабатывается командная строка в специальном системном буфере уровня. Размер буфера 128 байт. Адрес буфера можно получить с помощью рестарта $g_com(#45). Признаком конца строки служит символ #0D. Обрабатывается строка слева направо. Указатель в буфере двигается до первого пробела или конца строки (#0D). После запуска командного файла рестартом $run(#48) или $exbat(#44) указатель в строке указывает на первый пробел после имени и расширения файла или символ #0D.

Как и для рестартов 0-го и 1-го уровня флаг C на выходе большинства рестартов является признаком ошибки. Своих кодов ошибок у уровня COM совсем немного. Это:

код ошибки	рестарт	комментарии
31	$fncor	Запрещенные символы в имени файла
37	$run $exebat $fndev $opres $exeres	Нет резидентной задачи с таким именем или номером
79	$run $exbat	Неверное имя устройства в командах L_S, L_Q, L_T
130	$run $exbat	Запускаемая программа не влезает под кэш
138	$rcdel	Неверный номер канала в рекурсивной цепочке
139	$p_com	Длина командной строки превышает 128 символов
250	$run $exbat	Неверная контрольная сумма запускаемого com'a

Т.о. все остальные ошибки, которые Вы можете здесь заполучить - это ошибки двух нижних уровней, т.е. 0-го и 1-го. Например: 62, 80, 81, 85, 86, 87, 121, 125. При вызове рестартов $run, $exbat, $exres, $swrun, т.е. рестартов обращающихся к командным файлам или к резидентным задачам, Вы можете получить на выходе практически все что угодно в зависимости от конкретной запускаемой задачи.

Для рестартов $oparm, $comst, $opcat выходной флаг NZ при сброшеном флаге C сигнализирует о конце командной строки (или о пустой командной строке).

Для всех рестартов, кроме $exbat, $run, $exres и $swrun сохраняются все прямые регистровые пары кроме AF. Эти же 4 рестарта возвращаются в программу напрямую из вызываемых задач и передают все выходные регистры задачи.

Путь может быть задан как в символьном виде, так и в упакованном формате, т.е.:

Смещение	Длина	Примечание
0	1	код 8
1	1	номер устройства (диск)
2	2	каталог (см. g_stat(#37))
4	11	файл (имя и тип либо шаблон (см. trans(#46))

#40 (64) $oparm

Открыть файл-параметр. Рестарт удобен тем, что за один прием открывает сразу и путь к файлу, и сам файл. Однако данный рестарт не умеет распознавать ключи, т.е. символьные последовательности начинающиеся с косой черты "/" и заканчивающиеся пробелом или символом #0D. Ключ, нечаянно затесавшийся в строку, обрабатываемую рестартом $oparm будет воспринят, как имя файла. В этом случае, скорее всего, Вы получите ошибку 81 и бредовую надпись типа:

"No tv.+ф1"

Признак нормального завершения - сброшенный флаг C и установленный Z.

Возможные **ошибки** - ошибки нижних уровней:

81 - Нет файла или каталога
85 - Неверный блок описателя сегментов каталога
86 - Каталог не открывается
87 - Нет устройства на диске
121 - Устройство с данным номером не установлено

6 - Нет диска в дисководе
7 - Ошибка чтения
20 - Чтение прервано

Пример из начала программы tv.com. Рестартом $oparm программа пытается открыть файл-параметр. Все возможные ключи задаются лишь после параметра и обрабатываются программой "вручную" через рестарт $g_com(45)! Если параметр не задан (флаг NZ), файл открывается по курсору.

```
        LD (STACK+1),SP
;Сперва сохраняется текущая среда:
        LD IX,strtID
        LD C,$p_sta
        XOR A
        RST 16
        ...
        LD C,$oparm
        RST 16
        RET C
;+8 byte:
        JR Z,VIEW_1
;откроем файл под курсором:
G_CURS  LD SP,(STACK+1)
        LD C,$g_cur
        RST 16
;A=номер файла под курсором
        LD C,$opnum
        RST 16
        RET C
VIEW_1
        ...
```

#41 (65) $fmrst

Восстановление среды до последнего вызова какого-либо рестарта данного уровня, меняющего среду, например $oparm, $comst, $opcat, $nwcom, $nwcat. Среда (4 байта) хранится в системном векторе уровня по адресу, подаваемом в DE' рестартом $g_com. Особенность выходных флагов этого рестарта в том, что при отсутствии собственных ошибок он сохраняет входное значение AF.

Возможные **ошибки** те же, что и у рестарта $g_sta:
86 - каталог не открывается
80 - номер файла слишком велик
121 - нет устройства
124 - испорчена область каналов
125 - нет канала
7 - ошибка чтения/записи (драйвер)

Пример: главный цикл программы exebat.com:

```
RSTRT
;Открыть bat-файл и прочитать следующую командную строку в буфер COM-уровня:
        CALL G\LIN
        RET C
;Восстановить текущую среду:
        LD C,$fmrst
        RST 16
        RET C
;Напечатать командную строку на экране:
        CALL PRINT
;Запустить командную строку
```

```
        XOR A
        LD C,$exbat
        RST 16
        RET C

;И так далее...

        JR RSTRT
```

#42 (66) $comst

Открывает устройство и каталог по пути, указанному в текстовой строке или снимает ключ (часть строки, начинающаяся с косой черты и заканчивающаяся пробелом или символом #0D). Вход: HL - адрес командной строки. **Выход** в зависимости от флага:

C - ошибка. Восстанавливается прежняя среда

NC,Z - A=0: ПУТЬ\[файл]:

В этом случае в HL' будет 11-байтовый описатель или шаблон файла. Шаблоном называется описатель частично или полностью заполненный символами #FF, что означает "любой символ". Такое может случиться, если на вход преобразующей процедуре подать имя файла, содержащее символы "?" и/или "*". Тот же эффект будет при отсутствии имени файла в командной строке.

NC, Z, A≠0 - Обнаружен **ключ**. HL' - адрес символа "/", A - символ после знака "/"

NC, NZ - Или ключа нет, или пути нет, или синтаксическая ошибка с командной строке.

Возможные **ошибки:**

81 - Нет каталога с таким именем
86 - каталог не открывается
121 - нет устройства
7 - ошибка чтения/записи (драйвер)

Пример из программы tree.com:

```
;Программа вывалилась из рестарта $panel с кодом клавиши в регистре A:

        CP #D           ;Enter?
        JR NZ,FCMON     ;обратно в $panel

        LD E,(IX-34)
        LD D,(IX-33)

;DE=номер записи панели (номер строки)

        CALL G_PATH

;HL=путь, соответствующий строке
;Вернуть панель iS-DOS'y:

        CALL RESTOR

;Открыть выбранный каталог:

        LD C,$comst
        RST 16
        RET C

;Закрепить выбранный каталог в среде для $fmrst. Для этого на вход подается пус-
;тая строка (по адресу в HL: #0D). Это позволит вызывать tree.com из bat-файла:

        LD HL,ENTER
        LD C,$comst
        RST 16
        RET C

;Выход в iS-DOS с перепечаткой панели:

        XOR A
        LD A,#F1
        RET
```

#43 (67) $opcat

В принципе то же, что и предыдущий рестарт, но только работающий с внутренним буфером командной строки уровня COM. Т.о. на вход данному рестарту ничего подавать не требуется. Этот рестарт чрезвычайно удобен для снятия входных параметров из командной строки. Он используется во многих com-файлах,

работающих с файлом-параметром и/или ключами. Например: cache.com, flush.com, delete.com, filecopy.com, sort.com и во многих других.

Возможные **ошибки** те же, что и у предыдущего рестарта.

Пример из программы sort.com:

```
OPCAT   LD C,$opcat
        RST 16
        RET C
        JR NZ,OPEX

;A=ключ, т.е. первый байт после "/":

        CP "-"
        JR NZ,1$

;"-": инверсия

        LD A,#D4            ;CALL NC вместо CALL C
        LD (INVER),A
        JR OPCAT

1$      CP "c"
        JR Z,__C

        CP "e"
        JR Z,__E

        CP "d"
        JR NZ,OPCAT

;D:date - дата

        LD A,1
        LD (DAT+1),A
        JR OPCAT

;Extention - расширение

__E     LD A,2
        LD (EXT+1),A
        JR OPCAT

;Catalogue - каталоги

__C     LD A,1
        LD (CATS+1),A
        JR OPCAT

OPEX
```

Смотрите также пример к рестарту $opnum(#26) из программы calc.com и пример к рестарту $erf__(#3C) из программы delete.com.

#44 (68) $exbat

Как и предыдущие 3 рестарта ($oparm, $comst и $opcat), сей рестарт также работает с командной строкой, но в отличие от них не только открывает путь к файлу и файл, но и запускает этот файл на исполнение. Причем, пользоваться этим рестартом можно по-разному в зависимости от входных данных:

1. Если системный буфер не пуст, то отрабатывается командная строка в нем, а регистры HL, DE и A не имеют значения. Командная строка исполняется до первого пробела или, за отсутствием таковых, до конца строки, т.е. до кода #0D. Формат строки тот самый, что и у bat-файлов (см. любое описание системы для пользователей):

[ПУТЬ\]файл [параметры]
здесь: файл - имя командного файла или
@резидент [параметры]
при запуске резидентной программы
Параметры рестарт не обрабатывает, путь открывается, но перед самым запуском командного файла восстанавливается среда запуска.

2. Если системный буфер строки пуст, т.е. первый байт в буфере #0D, то по адресу в HL этому рестарту надо подать первые 12 байт описателя файла. Тогда, если регистр A=0, рестарт откроет файл Q:extent.txt, найдет в нем строку с расширением файла по адресу в HL, сделает из всего этого новую командную строку и отработает ее так же, как и в первом варианте. Поскольку в регистре HL подается описатель файла, то для нормальной работы в этом режиме, файл должен находится в текущем каталоге.

Такой способ использует рестарт $exbat система при нажатии клавиши <Enter> на файле. Первым же способом пользуется программа exebat.com для отработки командных строк bat-файлов. Именно этот рестарт удобен для данной программы, т.к. у программы exebat.com нет места для буфера командной строки. Для всех прочих же случаев обычно удобнее рестарт $run(#48). При A>0 во втором случае в регистре DE подается адрес строки с путем файла формата extent.txt.

Адрес системного буфера командной строки доступен с помощью рестарта $g_com(#45):

```
LD C,$g_com
RST 16
EXX
```

;HL=адрес командного буфера

Размер буфера: 128 байт. Заполнить буфер можно с помощью рестрарта $p_com(#49).

Возможные **ошибки:**

81 - нет каталога или файла с таким именем
86 - каталог не открывается
121 - нет устройства
7 - ошибка чтения/записи (драйвер)
130 - запускаемый файл не влезает под кэш
250 - неверная контрольная сумма запускаемого com-файла
+ все возможные ошибки, которые может получить и передать дальше запущенный com-файл.

Пример из программы exebat.com. Главный цикл программы:

```
RSTRT
;Заполнить командный буфер новой строкой:
        CALL G\LIN
        RET C
;Восстановить текущую среду:
        LD C,$fmrst
        RST 16
        RET C
;Напечатать строку на экране:
        CALL PRINT
;Отработать строку:
        XOR A
        LD C,$exbat
        RST 16
        RET C
        JR RSTRT
```

Пример из 4-го уровня системы:

```
RUN
;Отработка клавиши <Enter>:
        LD A,($FLCUR+11)        ;CSR файла
        AND #20                 ;каталог?
        JR Z,CNRN0
;иначе открытие каталога...
VIEW
;Отработка клавиши <3>:
        LD HL,$IMAVW
;HL указывает на "Q:SHELL\extview.txt"
        JR $EXRUN
PRNT
;Отработка клавиши <h>:
        LD HL,$PRNT
;HL указывает на "Q:SHELL\extprint.txt"
$EXRUN
```

```
;Заполнить буфер соответствующей строкой:

        CALL DCRUN
        EX DE,HL
        PUSH DE

;Зажечь "VIEW" на экране:

        LD A,"3"
        CALL $BLICK
        POP DE
        LD A,-1
CNRN0   LD HL,$FLCUR        ;имя и CSR файла
        LD B,2

$CNrun  LD C,$exbat
OUTRST  RST 16
        RET C
        RET Z
ORFF    OR -1
        RET
```

#45 (69) $g_com

Этот рестарт выдает в регистрах HL' и DE' адреса системных векторов уровня COM. **HL:**

Смещение	Длина	Описание значения
-11	1	проверять контрольную сумму com-файлов при запуске
-10	1	не используется
-9	4	среда последнего $exebat'a ($run'a). Позволяет запущенному com-файлу найти самого себя
-5	2	указатель в командной строке. Перемещается от начала к концу рестартами $oparm(#40) и $opcat(#43). Устанавливается рестартами $exbat(#44) и $run(#48)
-3	2	Текущая дата (см. **ПРИЛОЖЕНИЕ 23**)
-1	1	Номер канала bat-файла в рекурсивной цепочке. Используется программой exebat.com и рестартом $rcdel(#4D). Позволяет вызывать один bat-файл из другого подобно подпрограмме
0	128	Буфер командной строки

Системный вектор в **DE:**

Смещение	Длина	Описание значения
-3	1	Номер канала "PATH", содержащего пути для дополнительного поиска командных файлов, запускаемых рестартами $exbat и $run. Используется, если путь не указан. Если 0, значит канала "PATH" нет. Устанавливается и снимается программой path.com
0	4	Среда для $fmrst(#41). Активно используется практически всеми рестартами уровня COM, которые открывают устройства и каталоги

Регистровая пара AF сохраняется.

Пример использования рестарта $g_com из программы date.com. Данная процедура открывает файл date.com и сохраняет в нем параметры текущего режима работы:

```
curID   DEFS 4

;Сперва сохраним текущую среду в curID:

SVST_1  LD IX,curID
        XOR A
        LD C,$p_stat
        RST 16

;Откроем файл date.com:

        LD C,$g_com
        RST 16
        EXX
        LD BC,-9
        ADD HL,BC
        PUSH HL
        POP IX
        XOR A
```

```
        LD C,$g_stat
        RST 16
        RET C
```

;Запишем в него 1 байт со смещением 2:

```
        LD IX,WDCSR
        LD DE,1
        XOR A
        LD HL,2
        LD C,$wpart
        RST 16
        RET C
```

;Обязательно сосчитаем новую контрольную сумму!

```
        LD HL,CALC
        LD C,$run
        RST 16
        JR NC,1$

        CP 37
        SCF
        RET NZ
```

;Зафиксируем все на дискете:

```
1$      LD C,$flush
        RST 16
        RET C
```

;Восстановим текущую среду:

```
        LD IX,curID
        XOR A
        LD C,$g_stat
        RST 16
        RET C
```

;Вот ещё 1 **способ** восстановить среду для exebat.com. Предыдущий способ - пустой $comst или $opcat использовался в примере из программы tree.com к рестарту $comst(#42). Если этого не сделать, то после вызова из bat-файла текущим каталогом может стать каталог, в котором находится date.com.

```
        LD C,$g_com
        RST 16
        EXX
        LD HL,curID
        LD BC,4
        LDIR
        RET

CALC    DEFM /@calc/
        DEFB 13
```

#46 (70) $trans

Этот рестарт преобразует имя и расширение файла, написанные через точку, как они пишутся в командной строке, в 11-байтовый описатель файла, по которому этот файл могут найти или создать рестарты 1-го уровня. Адрес входного буфера подается на входе в регистре HL, выходного - в DE. Входной буфер заканчивается символом #0D или пробелом. Символы "*" и "?" на входе преобразуются в коды #FF на выходе. При этом устанавливается флаг C на выходе, сообщающий, что на входе был подан шаблон, требующий специальной обработки. Такая обработка осуществляется теми программами, которым это необходимо, например: copy.com, find.com, scan.res, mark.com.

Пример из программы qu.com:

```
TRANS
;Преобразуем в 11-байтовый формат:

        LD HL,FILE
        LD DE,F_11
        LD C,$trans
        RST 16
```

;Если встретились символы "?" или "*", то изменить в редакторе:

```
        JR C,SMBGT
;Проверим на правильность:
        EX DE,HL
        LD C,$fncor
        RST 16
;Если всё OK, то отправимся на поиски:
        JR NC,FIND
;Иначе попросим скорректировать имя:
SMBGT
;Спозиционируемся в окне для вывода строки редактора:
        LD HL,#202
        LD C,$wtpos
        RST 16
;Отредактируем строку из 12 символов:
        LD HL,FILE
        LD DE,#100
        LD BC,#8100+$smbgt
        LD A,12
        RST 16
        JP NZ,NEXT          ;<Ss/A> - отказ
        JR TRANS            ;Ещё раз
FIND
;Поиск файла:
        LD C,$find
        RST 16
;Если нашли, то поменять имя:
        JR NC,SMBGT
;Ошибка 81 - нет файла.
        CP 81
        SCF
        RET NZ
        …
F_11    DEFS 11
FILE    DEFS 13
```

#47 (71) $g_way

Данный рестарт заполняет буфер либо канал текстовой строкой с путем текущего подкаталога. Неиспользованное место заполняется пробелами.

Входные регистры: A>0 - номер канала, в который записать путь. Не рекомендуется пользоваться системными каналами, особенно с номерами менее #18 и более #D8. И вообще, этим способом пользуется система, а для домашнего употребления удобнее **второй метод:**

A=0, тогда в HL - адрес буфера, в DE - размер буфера. Разумный размер буфера: 6x9+2=56 байт.

Среда после вызова сохраняется почти полностью, за исключением текущего файла.

Возможные **ошибки:**

6 - нет дискеты в дисководе
7 - диск не читается
86 - каталог не открывается
125 - нет канала

Вот как используется этот рестарт в программе unicolor.com:

```
G_WAY
;Сохраним текущую среду, т.к. впоследствии будем пытаться открыть каталог HELP\
;Рестарт $g_way переоткрывает каталог, посему сохранить номер текущего файла также будет не вредно...
        XOR A
        LD IX,tmpID
```

```
        LD C,$p_sta
        RST 16
```

;адрес буфера + его размер:

```
        LD HL,PATH+8+pathSZ
```

;Очистим его:

```
        LD B,pathSZ

        DEC HL
        LD (HL)," "
        DJNZ $-3
```

;Положим в него путь:

```
        XOR A
        LD DE,56
        LD C,$g_way
        RST 16
        RET C
```

;Сразу за путем поместим имя файла: ищем первый пробел от начала:

```
1$      LD A,(HL)
        CP " "
        INC HL
        JR NZ,1$

        DEC HL
```

;Достанем опмсатель текущего файла:

```
        LD C,$bkfcb
        RST 16
        PUSH HL
        EXX
        POP DE
```

;Положим его имя в буфер:

```
        LD C,$convr
        RST 16
        RET C

        …
```

#48 (72) $run

Этот рестарт предназначен исключительно для запуска командных файлов или резидентных задач из командной строки, т.е. он выполняет одну из функций рестарта $exbat(44), но проще в использовании, т.к. командная строка может находиться где угодно. Адрес командной строки подается на входе в регистре HL. Рестарт сам сперва копирует эту строку в системный буфер уровня, поэтому размер строки не должен превышать 128 символов. Завершаться строка должна символом #0D. Поэтому к возможным ошибкам рестарта $exbat добавьте 139-ую. Рестарт передает вызываемой программе регистры IX и B (резиденту можно передать также регистр DE). Возврат происходит из программы напрямую, т.о. все регистры содержат на выходе все, что угодно, в зависимости от вызываемой программы.

Пример из программы coca.com:

```
LENWT
        LD HL,#5FF          ;координаты окна
        LD (WIND+1),HL
        LD HL,LIN_1         ;печать строки
        LD C,$lenwt
        RST 16

        XOR A
        LD IX,curID
        LD C,$p_sta         ;сохранить среду
        RST 16
```

;Запустить com-файл mkdir или copy25:

```
        LD HL,LINE
```

```
        LD C,$run
        RST 16
        RET C

        XOR A
        LD IX,curID
        LD C,$g_sta         ;Восстановить среду:
        RST 16
        RET

WIND    DEFB 0,0,3,32,0,-1,0,42
copy    DEFM "copy25 *.*"
copLN   EQU $-copy
LINE    DEFM "Q:SHELL\"
LIN_1   DEFM "mkdir /c /s00 /f"

PATH    EQU $
```

Рекомендуем также все примеры, запуска резидента calc.res для подсчета новой контрольной суммы (рестарт $g_com(45), программа date.com).

#49 (73) $p_com

Данный рестарт копирует командную строку в системный буфер. В HL на входе подается адрес строки. Строка должна заканчиваться кодом #0D. Единственная возможная ошибка с номером 139 может возникнуть при длине строки более 128 символов.

На практике рестарт почти не применяется. Единственная программа из просмотренных мной, которая могла бы им воспользоваться, без него таки обошлась. Это процедура сохранения системы sv.com:

```
        LD C,$g_com
        RST 16
        EXX
        PUSH HL
        LD DE,LINE
        LD BC,LINESZ
        EX DE,HL
        LDIR
        POP HL

        …

LINE    DEFM "Q:SHELL\exebat Q:autoexec.bat"
        DEFB 13
LINESZ  EQU $-LINE
```

#4A (74) $dvtrn

Этот рестарт работает исключительно с регистром A на входе и на выходе. Он преобразует т.н. логическое имя устройства в физическое, т.е. заданное символьно в виде буквы от "A" до "H" или букв "S", "Q" или "T" (обязательно латинских!) в номера устройств им соответствующих, т.е. числа от 0 до 7, означающие номера каналов устройств. При отсутствии данного логического устройства (на входе вместо символов A..H,Q,S,T подано что-то другое) на выходе флаг Z будет сброшен.

Пример из программы image.com:

```
        JP START

        …

;Обработка ключа:

OPTION  CP "-"
        JR NZ,OPCAT

        INC HL
        INC HL
        LD A,(HL)           ;логическое имя устройства
        LD C,$dvtrn
        RST 16
        JR NZ,OPCAT

;A=физическое имя устройства. Сравним с текущим

        LD HL,FROM+2
```

```
        CP (HL)
        JR Z,OPCAT      ;Защита от дурака

        LD (Dest+1),A   ;Выходное устр-во
        ADD A,"A"
        LD (destin+11),A ;для печати
        JR OPCAT

START

        …

        LD C,$g_blk
        RST 16
        LD (FROM+2),A   ;Текущее устр-во

;Обработка параметров командной строки:

OPCAT   LD C,$opcat     ;следующий параметр
        RST 16
        RET C

        EXX
        JR NZ,OPEX      ;конец ком. строки

        OR A
        JR NZ,OPTION    ;ключ

;иначе - файл:

        LD DE,FILE
        LD BC,11
        LDIR
        JR OPCAT

OPEX

…
```

#4B (75) $newcom

Этот рестарт полностью аналогичен рестарту $comst(#42), т.е. он открывает устройство и каталог по пути, указанном в текстовой строке, но при обращении к устройству вызывает рестарт $binit(#0F), что желательно при обращении к новому диску, т.к. в принципе iS-DOS поддерживает разные форматы дисков, хотя на практике используется обычно лишь один формат.

Возможные **ошибки**, соответственно все те же, что и у $comst плюс две ошибки рестарта $binit:
9 - не исдос-устройство
10 - конфликт диска и дисковода

Рестарт применяется, похоже, только в копировщиках. Вот **пример** из filecopy.com:

```
PROVER
;Открыть новое устройство и путь:

        LD HL,$PATH
        LD C,$newco
        RST 16
        JR NC,PRO_OK    ;Если O.K.

;иначе - ошибка:

        CP 81           ;нет каталога?
        JR Z,PRO_81

        CP 9            ;не iS-DOS-формат?
        SCF
        RET NZ

;Не iS-DOS диск:

        CALL NOTISD     ;Печать: "Not iS-DOS"
        JR PROCRE

PRO_81  CALL NOPATH     ;Печать: "No path!"

;Пересоздать кэш блочных устройств:
```

```
PROCRE
        LD A,($BLOK)
        LD C,$creat
        RST 16
        XOR A
        INC A                   ;Выход с NZ
        RET

PRO_OK

        JR NZ,PRO_81
        EXX                     ;имя выходного файла
        LD DE,TRAF
        LD BC,11
        LDIR
        LD C,$g_cat
        RST 16
        LD ($DIS0+1),A          ;текущий диск
        EXX
        LD (OPISK2+1),HL        ;тек. каталог
        XOR A
        RET                     ;Выход с Z
```

#4C (76) $newcat

Аналог $opcat(#43) для нового диска. Как и предыдущий рестарт, дергает $binit сразу после переключения на новое устройство. На практике почти не применяется, разве что в программе copy.com, отрывок из которой очень напоминает предыдущий пример:

```
TRAF    DEFS 11

NEWCAT
        LD C,$newcat
        RST 16
        RET C
        RET NZ
PROV1   EXX
        LD DE,TRAF
        LD BC,11
        LDIR
        LD C,$g_cat
        RST 16
        RET C
        LD (DIS1+2),A
        EXX
        LD (OPISK2+1),HL
        LD B,A
        XOR A
        RET
```

#4D (77) $rcdel

Данный рестарт удаляет рекурсивную цепочку каналов, созданную программой exebat.com или, теоретически любую другую цепочку. В регистре A подается на входе номер последнего рестарта. Каждый канал цепочки в своем 4-ом байте должен содержать номер предыдущего канала. У первого канала 4-ый байт должен быть равен 0.

Возможные **ошибки:**

124 - неверная структура области каналов
125 - нет канала
138 - ошибка в рекурсивной цепочке. Возникает, если номер канала меньше #18 или больше #D8.

На практике используется, кажется, только рестартом $shexe(93):

```
SHEXE
        LD SP,$ADRSP        ;Перехват стека

;Уже знакомый Вам рестарт:

        LD C,$g_com
```

```
        RST 16
        EXX
        DEC HL
        LD A,(HL)           ;канал последнего bata
        OR A
        JR Z,Exbat
```

;А вот и он...

```
        LD C,$rcdel
        RST 16
        JR C,Shout
Exbat   XOR A
```

;Отработка командной строки:

```
        LD C,$exbat
        RST 16
        LD HL,$ERCSR
```

;Если все O.K., то выполняем $shel(80):

```
        JR NC,SHEL00
```

;Одноразовый обход ошибки "нет файла". Чтобы не ругалась при загрузке, когда нет autoexec.bat:

```
        BIT 0,(HL)
        SET 0,(HL)
        JR NZ,Shout

        CP 81
        JR Z,SHEL00

        SCF
Shout   JP $SHOUT
```

#4E (78) $cp_ir

Этот рестарт ищет в таблице слово.

Вход: **DE** - адрес таблицы слов, разделенных символом #0D. Длина слов от 1 до 7 байт. Первые 2 байта таблицы - размер самой таблицы (без 2 байт длины).
В **HL** - адрес искомого слова в строке или тексте. Слово должно заканчиваться символом менее #41 (буква A латинская).

Выход: если поиск увенчался успехом, то флаг Z будет установлен, и в регистре A - номер слова в таблице (нумерация от 0). В HL' - продолжение текста после найденной модели, в DE' - продолжение таблицы. Если модель не найдена - флаг Z сброшен.

Пример из программы unicolor.com. Здесь с помощью рестарта $cp_ir анализируется расширение файла, создаваемого пользователем для хранения в нем цветов. Таблица содержит список запрещенных расширений.

;Копируем расширение файла в буфер, чтобы в оно оканчивалось на 0-ой байт:

```
        LD HL,FILE+8        ;расширение файла
        LD DE,TPBUF
        LD BC,3
        PUSH DE
        LDIR
        POP HL
        LD DE,EXTAB
        LD C,$cp_ir
        RST 16
        JR NZ,FIND
```

;Печать окна: "Неверное расширение файла"

```
        LD IX,`WRNG1
        LD A,(color+5)
        CALL WRNG_1
        JP INNAME
```

;Ищем файл:

```
FIND    LD HL,FILE
        LD C,$find
        RST 16
```

```
        RET
`WRNG1 DEFB 9,11,3,13,7,1,13,15
        DEFM "Wrong extention"
TPBUF   DEFS 4
d       EQU #D
EXTAB   DEFW TABLEN
        DEFB "b","a","t",d
        DEFB "b","l","k",d
        DEFB "c","o","m",d
        DEFB "d","a","t",d
        DEFB "d","o","c",d
        DEFB "d","p","r",d
        DEFB "h","l","p",d
        DEFB "k","e","y",d
        DEFB "l","p","r",d
        DEFB "n","m","i",d
        DEFB "o","v","r",d
        DEFB "p","a","c",d
        DEFB "r","e","s",d
        DEFB "s","y","s",d
        DEFB "t","x","t",d
        DEFB "t","y","p",d
        DEFB "w","e","t",d
TABLEN EQU $-EXTAB-2
```

#4F (79) $convr

Этот рестарт совершает преобразование обратное тому, что делает рестарт $trans(#46), т.е. он получает на входе первые 11 байт с именем и расширением файла, и формирует на выходе текстовую строку с именем и расширением файла, разделенными точкой и символом #0D в конце. Адрес входного буфера подается в HL, выходного - в DE. На выходе DE' будет указывать на символ #0D в конце строки. Флаг на выходе смысла не имеет.

Пример из программы image.com:

```
;Ищем файл, поданный в командной строке:

OPEX    LD HL,FILE
        LD C,$find
        RST 16
        RET C

;Положим его имя и расширение в текстовый буфер окна:

        EXX
        LD DE,NAME
        LD C,$convr
        RST 16
        EXX

;Забьем пробелом ненужную "ВКашку":

        LD A," "
        LD (DE),A
        EXX

;Размер файла в блоках (2 байта):

        LD BC,15
        ADD HL,BC
        LD DE,DSIZE+1
        LDI
        LDI

        …

;Вектор окна:

`PROM   DEFB 5,8,7,22,0,0,8,27
        DEFM "Copy file"
NAME    DEFM " "
        DEFB 13
```

```
destin  DEFM "To device: A"
out     DEFB 13,13
        DEFM "<Enter> to continue"
        DEFB 13
        DEFM "<SS/A> to quit"
        DEFB 3
```

См. также пример из программы unicolor.com к рестарту $g_way(#47).

#50 (80) $fncor

Сей рестарт проверяет на правильность и корректирует прямо во входном буфере имя и расширение файла или каталога. В регистре HL на входе подается адрес 11-байтового буфера с именем и расширением файла. Признак каталога - установленный 5-й бит регистра A на входе, т.о. в регистр A Вы можете положить регистр состояния файла (11-ый байт описателя). Рестарт проверяет 11 байт на наличие среди них запрещенных, а запрещены все кроме тех, которые разрешены. Это: латинские и русские символы, цифры и еще 8 символов: #$&+-=_` (коды: #23,#24,#26,#2B,#2D,#3D,#5F,#60).

С 21.IX.1999 к разрешённым добавлены символы: !@%'(){|}~ для упрощения копирования файлов из MS-DOS. При наличии запрещенных символов в имени выдается **ошибка 31**. В случае каталога все маленькие буквы заменяются большими.

Пример из программы mkdir.com(.res):

;Позиционирование в окне для ввода имени каталога:

```
WTPOS  LD HL,#301
       LD C,$wtpos
       RST 16
```

;редактирование имени:

```
       LD A,8              ;длина строки
       LD DE,#A01          ;ПРОПИСНЫЕ
       LD HL,FILE          ;текстовый буфер
       LD BC,#100+$smbgt
       RST 16
       JR NZ,1$            ;отказ
       OR A                ;длина введенной строки
       JR Z,1$             ;пустая строка
```

;Проверка на корректность:

```
       LD A,#20            ;каталог
       LD C,$fncor
       RST 16
       JR C,WTPOS          ;повторить ввод

       JR MAKE             ;создать файл
```

;выход по отказу с перепечаткой текущей панели:

```
1$     XOR A
       LD A,#F8
       RET
```

#51 (81) $fndev

Этот рестарт ищет установленный драйвер или резидент по имени. 8-буквенное имя резидента или драйвера хранится в канале драйвера (резидента), со смещением 10. 18-байтовый канал создается при установке резидента (драйвера) программой set.com и удаляется ей же при его снятии.

На **входе** в рестарт подается адрес буфера с именем в регистре HL и устанавливаются границы поиска: A=номер канала, с которого начинать поиск, B=число каналов перебора. Рекомендуемые значения регистров A и B приведены в следующей таблице:

	A	B
res	#D8	10
key	#E8	8
typ	#F0	8
blk	#F8	8

Выход: A=номер канала в указанном диапазоне. HL'= адрес тела канала. Подробно он описывался в **ПРИЛОЖЕНИИ 8**. Таблица приводилась в описании рестартов #14..#1D, работающих с каналами. Напомним лишь, что со смещением 4 лежат 2 байта адреса резидента в памяти, 6(2) - длина.

Возможные **ошибки:**

37 - Нет резидента с таким именем

124 - Ошибка в структуре области каналов

Этим рестартом пользуются такие программы как: set.com, dev.com и новейшая программа antipod.com, откуда и приведем **пример:**

```
_mon    DEFM "mon"
_mon_   DEFM "mon+"

;Ищем резидента mon.res:

        LD HL,_mon
        LD A,#D8
        LD BC,#1000+$fndev
        RST 16
        JR NC,1$                ;нашли

        CP 37                   ;нет такого?
        SCF
        RET NZ                  ;другая ошибка

;Ищем резидента mon+.res:

        LD A,#D8
        LD HL,_mon_
        RST 16
        JR NC,1$

        CP 37
        SCF
        RET NZ

;Печать сообщения "mon.res not installed"

        LD DE,mon
        CALL TY_NO
        JR 2$

1$      EXX

;в HL - канал резидента:

        INC HL
        INC HL
        INC HL
        INC HL
        LD E,(HL)
        INC HL
        LD D,(HL)

;DE=адрес резидента в памяти

        EX DE,HL
        LD DE,Mon
        LD A,MonSZ              ;размер модели

;поиск по модели:

        CALL Search
        JR NZ,2$

        ...
```

#52 (82) $opres

Этот рестарт подобно рестарту $opcat(#43) обрабатывает командную строку, находящуюся в системном буфере, однако предназначен для поиска не файлов, а резидентов и драйверов. В отличие от $opcat ключей снимать не умеет. По имени и типу (после точки) ищет установленного резидента или драйвер. "@" понимает как "bat.res". **Выход** без флага C полностью аналогичен предыдущему рестарту, т.е. A=номер канала резидента, HL'=тело канала. Возможные **ошибки** тоже те же. В старых версиях системы (до 25.II.1997) ошибка 37 на выходе (флаг C, A=37) подменялась сброшенными флагами C и Z, т.е. признаком нормального завершения был установленный флаг Z при сброшенном флаге C.

Пример из программы scan.res:

```
        DEFS 4          ;начало резидента

        XOR A
        LD (RNCSR+1),A
        LD C,$opres
        RST 16
        RET C
        LD (RSRN+1),A   ;номер канала

;Старый стандарт. Восстановим Status Quo:

        LD A,37
        SCF
        RET NZ
```

#53 (83) $exres

Запуск резидентной задачи по номеру ее канала в регистре A. Передает резиденту все регистры кроме AF. Возврат напрямую из резидента, также как в случае с рестартом $run. Используется резидентами univ и scan для запуска других резидентов.

Возможные **ошибки:**
124 - Ошибка в структуре области каналов
125 - Нет канала
Плюс все ошибки, возвращаемые данным резидентом.

Еще один кусок резидента scan.res в качестве **примера** (см. также рестарт $opres):

```
;Процедура, вызываемая после открытия каждого следующего файла:

_WORK   LD C,$bkfcb
        RST 16
        EXX

;Сравнение имени и расширения с шаблоном:

        LD DE,TRAF
        LD C,$cpfil
        RST 16
        RET NZ          ;не подходит

        PUSH HL
        LD BC,11
        ADD HL,BC
        LD A,(HL)       ;CSR файла
        POP HL

RNCSR   XOR 0           ;маска каталогов
        AND #20         ;каталог ?
        RET NZ

RSRN    LD A,-1

;Восстановим указатель в командной строке, чтобы вызываемый резидент мог снова
;снять те же самые параметры (ключи):

pathBC  LD BC,0
pathHL  LD (0),BC
        LD DE,TRAF      ;шаблон для ren.res
        LD BC,#7700+$exres
        RST 16
        RET

TRAF    DEFS 11
```

#54 (84) $swrun

Запуск программы (.com-файла или резидента) аналогично рестарту $run(#48) с сохранением части памяти в файле Q:swap0.swp.

Вход: HL - адрес командной строки для run'a,
IX - адрес начала сохраняемой области памяти,
DE - размер сохраняемой области.

После возвращения из вызванной этим рестартом программы востанавливается область памяти из swap-файла. Применяется в случае острой нехватки памяти, например в базах данных подобным образом вызывается

программа print.com. Ранняя версия программы vicomm.com (X-MODEM) с помощью этого рестарта вызывала tv.com для просмотра файлов не выходя из программы. Затем автор отказался от этой тормозной и, в данном случае, не самой необходимой функции.

Возможные **ошибки** - это ошибки рестартов $swblk(#1C), $crfil(#23), $run(#48).

Специально написанный **тестовый пример:**

```
        ORG 26000
START   XOR A
        LD IX,`OKHO
        LD C,$wt
        RST 16
        LD C,$adrwt
        RST 16

        LD C,$kwait
        RST 16
        DEC C
        RST 16

        CP #10
        LD A,#F4
        RET Z

        LD IX,START
        LD DE,SIZE
        LD HL,LINE
        LD C,$swrun
        RST 16
        RET C

        JR START

`OKHO   DEFB 4,8,7,24,#38,1,7,29
        DEFM "Пример использования рестарта"
        DEFM "$swrun(#54). Нажмите любую"
        DEFM "клавишу для вызова tv.com по-"
        DEFM "верх тестовой программы или"
        DEFM "<Ss/A> для выхода."
        DEFB #D
LINE    DEFM "tv S:extent.txt"
        DEFB #D
SIZE    EQU $-START

        …
```

"Оконная система IS-DOS"

Уровень WIND.SYS содержит 23 рестарта, обеспечивающих работу с окнами, строками символов, курсором и т.п. При помощи этих рестартов можно организовать вывод информации на дисплей, оформление рабочего экрана, а также организовать диалог с пользователем при помощи строкового редактора.

Создание окон.

Работа любого оконного интерфейса начинается, естественно, с процедуры открытия окна. Окно в системе IS-DOS имеет следующую структуру (см. рис 1):

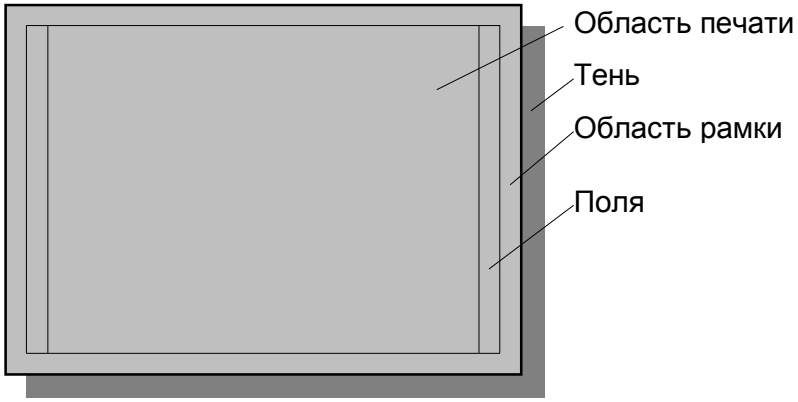


Рис. 1: Структура окна в IS-DOS

Область окна определяется значениями XS (X-size) и YS (Y-size), и измеряется в знакоместах 8*8 пикселей, область рамки имеет ширину в 8 пикселей (1 знакоместо 8*8) по всему периметру окна, следовательно, размеры рабочей области окна равны XS-2 и YS-2. Размер и расположение в окне поля печати, а также ширина левого и правого полей определяются prnt pos. и print size и измеряются в знакоместах 6*8. Верхний и нижний край поля печати ограничивается только областью рамки.

Область тени представляет собой окантовку нижней и правой сторон окна толщиной в 1 знакоместо 8*8 и с отступом в 1 знакоместо от левой и верхней границ области окна.

#61 (97) $wt

В системе IS-DOS для открытия окна используется рестарт wt (#61). Он осуществляет формирование на экране окна с параметрами, заданными в специальной таблице, называемой "вектором окна". Адрес вектора окна на момент вызова рестарта должен находиться в регистровой паре IX, при этом содержимое регистра A определяет внешний вид окна следующим образом:

7-й бит, установленный в 1 делает открываемое окно "прозрачным", т. е. окно открывается только изменением атрибутов, а пиксели внутри него не сбрасываются.

Остальные биты определяют вид рамки окна:

000000 - одинарная рамка

000010 - двойная рамка

Все остальные значения приводят к открытию окна без рамки.

Таким образом, формат задания внешнего вида окна можно представить в виде таблицы:

окно	одинарная рамка	двойная рамка	без рамки
прозрачное	A=#80	A=#82	A=#81 A=#83..#FF
непрозрачное	A=#00	A=#02	A=#01 A=#03..#7F

Итак, процедура открытия окна в системе IS-DOS выглядит следующим образом:

```
LD IX,WIND   ;загрузка в IX адреса вектора окна
LD C,#61     ;загрузка в C кода рестарта wt
LD A,n       ;загрузка в A значения, определяющего вид окна
RST #10      ;вызов рестарта
```

Структура вектора окна:

```
WIND   DEFB #00      ;X-координата верхнего левого угла окна в знакоместах 8*8 пикселей
       DEFB #00      ;Y-координата верхнего левого угла окна в знакоместах 8*8 пикселей
                     ;значения координат отсчитываются от левого верхнего угла экрана
       DEFB #0F      ;высота окна в знакоместах 8*8
       DEFB #0F      ширина окна в знакоместах 8*8
       DEFB %00000111   ;цвета окна
                        ;цвета кодируются стандартным образом:
                        ;бит 7 - flash
                        ;бит 6 - bright
                        ;биты 5...3 - paper
                        ;биты 2...0 - ink
       DEFB %00000001    ;цвета тени окна
                         ;Примечательно, что в цветах тени окна можно задавать как цвет paper, так и
                         ;цвет ink, что позволяет сделать видимой часть информации, на которую
                         падает ;тень от окна и обеспечивает максимальную естественность
                         восприятия. Если ;байт цветов тени равен #FF, то тень не выводится.
       DEFB #01          ;X-координата для процедуры печати текста в окне
       DEFB #12          ;ширина поля печати в окне
```

Последние два параметра выражаются в знакоместах размером 6*8, а не 8*8 как остальные, это сделано потому, что такую систему отчёта используют все рестарты печати в iS-DOS. Значение X-координаты печати абсолютное и отсчитывается не от левой границы окна, а от левой границы экрана. Перевод координат окна в координаты печати и обратно можно осуществить по формулам:

$$n=N*8/6 \quad (1)$$

$$N=n*6/8, \quad (2)$$

где n - значение в знакоместах 6*8,
N - значение в знакоместах 8*8.

Y-координата печати не задается, так как большинство рестартов печати самостоятельно отсчитывает ее от верхней строки окна.

На экране может быть открыто несколько окон, при этом текущим будет считаться то окно, адрес вектора которого находится в регистровой паре IX. В целях более рационального использования объема ОЗУ, информация, закрываемая окном в момент его открытия не сохраняется, поэтому, если Вам вдруг понадобилось вернуться в предыдущее окно, то придется его открывать заново.

Рассмотрим простейший пример. Задача - создать в точке с координатами X=5 и Y=3 окно размером 28*16 знакомест с двойной рамкой, синей бумагой, белыми чернилами и с черно-синей тенью, поле печати текста определить с отступом в 1 знакоместо от левой и в 5 знакомест от правой границы окна.

Для создания этой несложной программы кроме рестарта открытия окна wt(#61) нам понадобятся еще три подпрограммы - очистки экрана, ожидания нажатия клавиши и возврата в IS-DOS.

Итак, рассмотрим листинг 1:

Листинг 1 Пример создания окна

```
;Исходные данные:

;    X-coord (X) = 5
;  Y-coord (Y) = 3
;  X-size (XS) = 20
;  Y-size (YS) = 16
;  W.Colors(WC) = %00001111
;  S.Colors(SC) = %00000001
;  L.Margin(LM) = 1
;  R.Margin(RM) = 5

        ORG #5D64

;основная программа

START   CALL CLS        ;вызов процедуры очистки экрана
        LD IX,WIND      ;адрес вектора окна
        LD C,#61        ;код рестарта wt
        LD A,2          ;код двойной рамки
        RST #10         ;вызов рестарта
```

```
        CALL WAIT       ;вызов процедуры ожидания клавиши
        JP EXIT         ;переход на процедуру возврата в IS-DOS

    ;вектор окна

WIND    DEFB 5
        DEFB 3
        DEFB 16
        DEFB 20
        DEFB %00001111
        DEFB %00000001
        DEFB 8
        DEFB 20
```

;Данные для print position (печать с отступом в 1 знакоместо 6*8 от левой ;границы окна) получены путем расчета по формуле (3):

$$PP=X*8/6+LM, \qquad (3)$$

;где PP - print pos. в знакоместах 6*8
; X - X-коорд. окна в знакоместах 8*8
; LM - левое поле в знакоместах 6*8

;Данные для print size получены путем расчета по формуле (4):

$$PS=XS*8/6-LM-RM, \qquad (4)$$

;где PS - print size в знакоместах 6*8
; XS - размер окна в знакоместах 8*8
; LM - левое поле в знакоместах 6*8
; RM - правое поле в знакоместах 6*8

дополнительные процедуры

```
CLS     LD C,#73        ;рестарт cls
        RST #10           ;для очистки экрана
        RET

WAIT    LD C,#07        ;рестарт ttyin
        RST #10           ;для ожидания
        RET               ;нажатия клавиши

EXIT    XOR A             ;стандартный выход
        LD A,#F4        ;в IS-DOS
        RET
```

Результат работы программы
Листинг 1

#62 (98) $box

Следующий рестарт box(#62) позволяет нарисовать или стереть в области рамки окна любую возможную рамку, или любую из ее сторон в любом сочетании. Поскольку ширина области рамки в окне IS-DOS равна 8 пикселям, всего возможно 8 различных одинарных рамок. Вызывая рестарт box несколько раз можно рисовать двойные, тройные и даже счетверенные рамки, а также рамки различной толщины - всего 256 комбинаций.

В качестве входных параметров рестарт box(#62) использует вектор окна, адрес которого задается в регистре IX, а также значения отступа от края окна по Y и по X в пикселях, задаваемые в регистрах D и E соответственно, и содержимое регистра A, определяющее, что именно требуется нарисовать или стереть.

Содержимое регистра A определяется следующим образом:

бит 7=0 - рисовать, 1 - стирать
биты 6...4 - не используются

остальные биты показывают, с какими из сторон рамки производится операция:

бит 3=0 - левая сторона рамки
бит 2=0 - правая сторона рамки
бит 1=0 - верхняя сторона рамки
бит 0=0 - нижняя сторона рамки

Для битов 3...0 возможны любые сочетания.

Общая форма процедуры рисования/стирания рамки выглядит так:

```
        LD IX,WIND      ;адрес вектора окна
        LD C,#62        ;код рестарта box
        LD D,N1         ;отступ по Y
        LD E,N2         ;отступ по X
        LD A,n          ;что делать
        RST #10           ;вызов рестарта
```

Для задания значений отступа рамки предпочтительнее использовать числа от 0 до 7, причем при загрузке в регистры D и E значения 0 рамка будет выведена по наружному краю окна, а при загрузке значения 7 - по внутреннему краю области рамки. Значение 8 даст тот же эффект, что и 0, 9 -тот же, что и 1 и т.д.

Рассмотрим пример. В качестве основы возьмем Листинг 1, но изобразим сначала окно без рамки, а потом выведем двойную рамку так, чтобы ее наружная линия была вдвое шире внутренней, после чего сделаем левую и правую стороны наружной рамки одинарной толщины.

Листинг 2 Пример рисования рамки окна

```
        ORG #5D64

    ;открытие окна

START   CALL CLS        ;очистка экрана
        LD IX,WIND
        LD C,#61
        LD A,1          ;окно без рамки
        RST #10
        CALL WAIT

    ;рисование наружной рамки двойной толщины производится в два приема

        LD IX,WIND      ;адрес вектора окна
        LD C,#62        ;код рестарта box
        LD D,0          ;отступ по Y
        LD E,0          ;отступ по X
        LD A,%00000000

    ;все биты равны 0 - рисовать всю рамку

        RST #10
        LD IX,WIND
        LD C,#62
        LD D,1
        LD E,1
        LD A,%00000000
        RST #10
        CALL WAIT

    ;рисование внутренней рамки

        LD IX,WIND
        LD C,#62
        LD D,3
        LD E,3
        LD A,%00000000
        RST #10
        CALL WAIT

    ;стирание боковых сторон

        LD IX,WIND
        LD C,#62
        LD D,2
        LD E,1
        LD A,%10000011
```

;7-й бит установлен - команда стирания установленные 0-й и 1-й биты обозначают не затрагиваемые в этой операции верхнюю и нижнюю стороны рамки

```
        RST #10
        CALL WAIT
        JR EXIT

    ;вектор окна

WIND    DEFB 5
        DEFB 3
        DEFB 16
        DEFB 20
        DEFB %00001111
        DEFB %00000001
        DEFB 8
        DEFB 20
```

При необходимости организации рамок, рисования вертикальных и горизонтальных линий внутри окна (за пределами области рамки) также можно пользоваться услугами рестарта box(#62), однако, для этого придется создать фиктивный вектор окна таким образом, чтобы требуемая рамка вписывалась в область рамки этого воображаемого окна.

Поясним на примере:

Пусть необходимо создать внутри окна (см. Листинг 1) рамку с отступом в 2 знакоместа от верхней границы окна, в 4 - от нижней и в 3 от левой и правой сторон.

Для этого создадим фиктивный вектор окна WIND1, а для расчета его параметров воспользуемся следующими формулами:

$$Y1=Y+UM, \qquad (5)$$

где Y1 - Y-координата рамки,
Y - Y-координата окна,
UM - отступ сверху

$$X1=X+LM, \qquad (6)$$

где X1 - X-координата рамки,
X - X-координата окна,
LM - отступ слева

$$YS1=YS-UM-DM, \qquad (7)$$

где YS1 - размер рамки по Y,
YS - размер окна по Y,
UM - отступ сверху,
DM - отступ снизу

$$XS1=XS-LM-RM, \qquad (8)$$

где XS1 - размер рамки по X,
XS - размер окна по X,
LM - отступ слева,
RM - отступ справа

Все величины измеряются в знакоместах 8*8. Значения цветов выберем такие же, как и в основном окне, но без тени (S.Colors=#FF), значения Print pos. и Print size могут быть любыми.

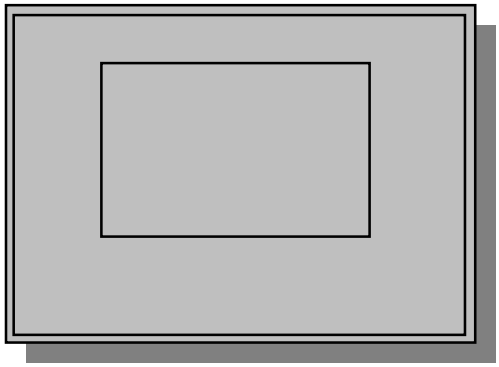
Итак,

Листинг 3 Пример рисования внутренних рамок при помощи box(#62)

```
        ;Исходные данные:
        ;       Up Margin (UM) = 2
        ;       Down Margin (DM) = 4
        ;       Left Margin (LM) = 3
        ;       Right Margin (RM) = 3

                ORG #5D64

        ;открытие окна

START   CALL CLS        ;очистка экрана
        LD IX,WIND
        LD C,#61
        LD A,2
        RST #10
        CALL WAIT
```

;рисовать второе окно не требуется, достаточно просто поместить адрес его вектора в регистр IX для передачи параметров рестарту box

```
        ;рисование рамки внутри окна

                LD IX,WIND1     ;вектор рамки
                LD C,#62
                LD D,0
                LD E,0
                LD A,%00000000
                RST #10
                CALL WAIT
                JR EXIT

        ;вектор окна

WIND    DEFB 5
        DEFB 3
        DEFB 16
        DEFB 20
        DEFB %00001111
        DEFB %00000001
        DEFB 8
        DEFB 20

        ;вектор окна для рамки

WIND1   DEFB 8
        DEFB 5
        DEFB 10
        DEFB 14
        DEFB %00001111
        DEFB #FF
        DEFB #00
        DEFB #00
```



Результат работы программы
Листинг 3

Подобным же образом выполняются вертикальные и горизонтальные линии - они представляются как стороны воображаемой рамки и задаются фиктивным вектором окна. Подбирая содержимое регистров D и E нетрудно добиться позиционирования рамки или линии с точностью до 1 пикселя.

#63 (99) $awt, #64 (100) $awtc

Еще несколько служебных рестартов могут пригодиться при работе с окнами в IS-DOS. Два из них позволяют подсветить требуемым цветом выбранную строку окна. Различаются они только тем, что рестарт awt(#63) действует на все окно, включая и область рамки, а действие рестарта awtc(#64) распространяется только на рабочую область окна.

Входными параметрами для этих рестартов служат следующие значения:

IX - адрес вектора окна

A - номер строки (начиная с 1-й)

B - цвета в стандартном формате

Выход за нижнюю границу окна не проверяется и подсветка происходит за его пределами, при этом сохраняется правильная ширина подсветки и положение ее по оси X.

Пример работы awt и awtc:

Листинг 10 Пример использования awt(#63) и awtc(#64)

```
        ORG #5D64

;открытие окна

        CALL CLS
        LD IX,WIND
        LD C,#61
        LD A,2
        RST #10
        CALL WAIT

;печать строк текста из буфера в окне через подфункцию 3 рестарта $prstr

        LD IX,WIND      ;вектор окна
        LD C,#68        ;код рестарта
        LD E,3          ;код подфункции
        LD HL,TEXT      ;адрес начала текста
        LD A,%11000010

;печать с центровкой и отступ сверху - 2 строки

        LD B,1          ;номер строки в тексте
        RST #10         ;вызов рестарта
        LD IX,WIND      ;вектор окна
        LD C,#68        ;код рестарта
        LD E,3          ; код подфункции
        LD A,%11000101
        LD B,2          ;печать следующей
        RST #10         ;строки в окне

;подсветка строки вместе с рамкой

        LD IX,WIND
        LD A,2          ;2-я строка
        LD B,%00110001
        LD C,#63
        RST #10

;подсветка строки в рабочей области окна

        LD IX,WIND
        LD A,5          ;5-я строка
        LD B,%00110001
        LD C,#64
        RST #10
        CALL WAIT
        JP EXIT

;вектор окна

WIND    DEFB 1
        DEFB 1
        DEFB 8
        DEFB 30
        DEFB %00001111
        DEFB %00000001
```

```
        DEFB 3
        DEFB 38

    ;текстовый буфер

TEXT    DEFM "подсветка строки с рамкой"
        DEFB #0D
        DEFM "подсветка строки без рамки"
        DEFB #0D
        DEFB #03
```

#65 (101) $lwt

Позволяет распечатать в окне заданное количество строк из текстового буфера с заданным отступом по Y от верхнего края окна.

Входные параметры: C - код рестарта (#65)
IX - адрес вектора окна
HL - адрес текстового буфера
A - величина отступа по Y от верхней границы окна
B - количество печатаемых строк

Формат текстового буфера для lwt является стандартным для большинства рестартов печати в IS-DOS: символы располагаются последовательно, начиная от стартового адреса в сторону увеличения адресов, строки отделяются друг от друга символом с кодом #0D, маркер окончания текста - символ с кодом #03.

При работе lwt маркер #03 не обязателен, т. к. печатаемый текст ограничивается заданным количеством строк. Однако, если маркер #03 встретится в тексте до того, как будет напечатано требуемое количество строк, печать прекращается. Символы перевода строки тоже не обязательны, так как при достижении правого края окна печать продолжается с начала следующей строки, причем счетчик строк учитывает это, так что независимо от наличия символов #0D в окне будет распечатано заданное количество строк. Остальная часть текста обрезается.

Если заданное количество строк превышает размер окна, текст продолжает выводиться за его пределами.

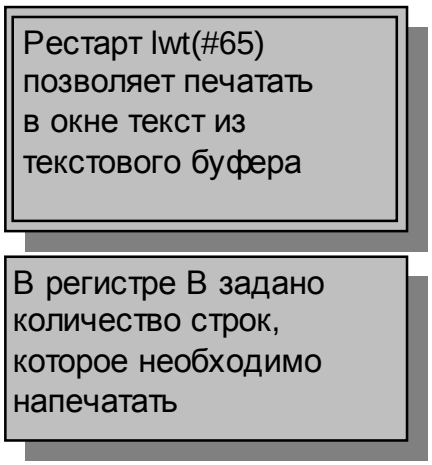
При работе рестарта lwt сохраняется неизменным содержимое регистра IX и регистровой пары HL.

Рассмотрим пример:

Листинг 4 Пример использования lwt(#65)

```
        ORG #5D64

    ;открытие окон

        CALL CLS
        LD IX,WIND1
        LD C,#61
        LD A,2
        RST #10
        LD IX,WIND2
        LD C,#61
        LD A,0
        RST #10
        CALL WAIT

    ;печать строк текста из буфера в окне 1

        LD IX,WIND1     ;адрес вектора окна
        LD HL,TEXT1     ;адрес текста
        LD C,#65        ;код рестарта
        LD A,1
        LD B,4
        RST #10
        CALL WAIT
        JP EXIT
```

```
        ;векторы окон

WIND1   DEFB 5
        DEFB 2
        DEFB 8
        DEFB 20
        DEFB %00001111
        DEFB %00000001
        DEFB 8
        DEFB 20

WIND2   DEFB 5
        DEFB 12
        DEFB 8
        DEFB 20
        DEFB %00001111
        DEFB %00000001
        DEFB 8
        DEFB 20

        ;текстовые буферы

TEXT1   DEFM "Рестарт lwt(65)"
        DEFB 13
        DEFM "позволяет печатать"
        DEFB 13
        DEFM "в окне текст из"
        DEFB 13
        DEFM "текстового буфера."
        DEFB 13

TEXT2   DEFM "В регистре B задано"
        DEFB 13
        DEFM "количество строк,"
        DEFB 13
        DEFM "которое требуется"
        DEFB 13
        DEFM "напечатать"
        DEFB 13
```



Результат работы программы
Листинг 4

#66 (102) $adrwt

Предназначен для распечатки в окне текста, расположенного непосредственно после вектора окна.
Входные параметры: C - код рестарта (#66)
IX - адрес вектора окна

Рестарт adrwt очень удобен для печати окон с текстом, жестко привязанным к конкретному окну. Главное его достоинство - всего один входной параметр (не считая самого кода рестарта в регистре C).

Для вывода текста на экран adrwt использует те же процедуры, что и lwt, поэтому все, что было сказано относительно формата текста для lwt, справедливо и для adrwt. При достижении нижнего края окна печать прекращается, а оставшаяся часть текста игнорируется.

При работе рестарта adrwt содержимое регистра IX сохраняется.

Пример:

Листинг 5 Печать при помощи $adrwt(#66)

```
  ORG #5D64

;открытие окон

  CALL CLS
  LD IX,WIND1
  LD C,#61
  LD A,2
  RST #10
```

```
        LD IX,WIND2
        LD C,#61
        LD A,0
        RST #10
        CALL WAIT

      ;печать строк текста в окне 1

        LD IX,WIND1     ;адрес вектора окна
        LD C,#66        ;код рестарта
        RST #10
        CALL WAIT

      ;печать строк текста в окне 2

        LD IX,WIND2
        LD C,#66
        RST #10
        CALL WAIT
        JP EXIT

      ;векторы окон

WIND1 DEFB 5
        DEFB 3
        DEFB 7
        DEFB 20
        DEFB %00001111
        DEFB %00000001
        DEFB 8
        DEFB 20

      ;текст окна 1

        DEFM "Рестарт adrwt(66)"
        DEFB #0D
        DEFM "позволяет печатать"
        DEFB #0D
        DEFM "текст, следующий"
        DEFB #0D
        DEFM "за вектором окна."
        DEFB #03

WIND2 DEFB 5
        DEFB 12
        DEFB 7
        DEFB 20
        DEFB %00001111
        DEFB %00000001
        DEFB 8
        DEFB 20

      ;текст окна 2

        DEFM "Строки разделяются"
        DEFB #0D
        DEFM "символом с кодом #0D"
        DEFB #0D
        DEFM "конец текста - "
        DEFB #0D
        DEFM "символ с кодом #03"
        DEFB #03
```

#67 (103) $lenwt

Так же предназначен для распечатки текста в окне, однако текст не закрепляется за вектором конкретного окна, а может находиться в произвольном месте программы.

Основное преимущество рестарта lenwt - возможность печати одного текста в нескольких окнах и наоборот - возможность печати разных текстов в одном окне.

Входные параметры: C - код рестарта
IX - адрес вектора окна
HL - адрес текста

При работе рестарта lenwt содержимое регистра IX и регистровой пары HL сохраняется.

Пример:

Листинг 6 Печать при помощи $lenwt(#67)

```
        ORG #5D64

;открытие окон

        CALL CLS
        LD IX,WIND1
        LD C,#61
        LD A,2
        RST #10
        LD IX,WIND2
        LD C,#61
        LD A,0
        RST #10
        CALL WAIT

;печать строк текста из буфера в окне 1

        LD IX,WIND1     ;адрес вектора окна
        LD HL,TEXT      ;адрес текста
        LD C,#67        ;код рестарта
        RST #10
        CALL WAIT

;печать строк текста из буфера в окне 2

        LD IX,WIND2
        LD HL,TEXT
        LD C,#67
        RST #10
        CALL WAIT
        JP EXIT

;векторы окон

WIND1   DEFB 5
        DEFB 3
        DEFB 7
        DEFB 20
        DEFB %00001111
        DEFB %00000001
        DEFB 8
        DEFB 20

WIND2   DEFB 5
        DEFB 12
        DEFB 7
        DEFB 20
        DEFB %00001111
        DEFB %00000001
        DEFB 8
```

Рестарт lenwt(#67) позволяет печатать текст, находящийся в любом месте программы

Рестарт lenwt(#67) позволяет печатать текст, находящийся в любом месте программы

Результат работы программы Листинг 6

```
        DEFB 20

    ;текстовый буфер

TEXT    DEFM "Рестарт lenwt(67)"
        DEFB #0D
        DEFM "позволяет печатать"
        DEFB #0D
        DEFM "текст, находящийся"
        DEFB #0D
        DEFM "в любом месте"
        DEFB #0D
        DEFM "программы"
        DEFB #03
```

#68 (104) $prstr

Рестарт prstr(#68) - самый мощный и универсальный из всех рестартов, предназначенных для печати строк текста в окне. Основным его достоинством является возможность автоматического выравнивания строки по левому и правому краю, а также центровки строки. Рестарт имеет 4 подфункции, позволяющие задать печатаемую строку различными способами. Код подфункции помещается в регистре E:

E=0 - распечатка строки по логическому имени, определенному в специальном массиве-анализаторе

E=1 - распечатка строки заданной длины, содержащейся по указанному адресу

E=2 - служебная подфункция, которая ни чего не печатает, но позволяет вы числить параметры для предыдущей подфункции, зная адрес начала текстового буфера и номер строки

E=3 - распечатка строки по номеру ее в тексте - фактически представляет собой комбинацию подфункций 2 и 1

Диспетчер подфункций имеет "защиту от дурака" - если Вы задали в регистре E номер, больший, чем 3, он будет автоматически приравнен к нулю.

Теперь рассмотрим более подробно работу с подфункциями рестарта prstr(#68).

Для начала - общие положения. При вызове рестарта в регистре A должно находиться число, определяющее два основных параметра печати - режим выравнивания (два старших бита) и отступ по Y от верхнего края окна (шесть младших битов). Режим выравнивания кодируется следующим образом:

00 - выравнивание по левому краю

01 - выравнивание по левому краю

10 - выравнивание по правому краю

11 - центровка строки

При работе рестарта содержимое регистра A не сохраняется.

Формат текстового буфера для prstr стандартный - символы располагаются последовательно, начиная со стартового адреса, строки отделяются друг от друга символом #0D, маркер конца текста - #03.

Для вывода символов на экран prstr использует рестарт ttyout(#0A) из уровня DOS.SYS. При формировании текстового буфера, задании отступа и числа печатаемых символов необходимо самостоятельно следить за тем, чтобы текст не вышел за пределы окна. При ширине строки, большей, чем ширина поля печати окна выравнивание автоматически осуществляется по левому краю, независимо от состояния двух старших битов регистра A.

Подфункция 0, как уже сообщалось выше, позволяет распечатать строки из текстового буфера по их логическим именам. Что это такое?

Допустим, Вам необходимо распечатать в окне текст, содержащий часто повторяющиеся строки. Вместо того, чтобы многократно вызывать prstr, каждый раз задавая все параметры и строго следя за тем, какую из строк печатать следующей, или, что еще хуже, забивать текстовый буфер одинаковыми строками, Вы можете просто составить массив, в котором будет установлена последовательность печати строк путем ссылок на их порядковые номера в буфере. Оперативно переключая разные массивы, Вы сможете легко перекомпоновывать Ваши тексты, менять местами строки, вставлять другие и т. п. При помощи подфункции 0 легко также организовать печать текстовых сообщений по коду из таблицы (по типу сообщений об ошибках в Spectrum BASIC)

Входные параметры:
- C - код рестарта (#68)
- E - код подфункции (0)
- IX - адрес вектора окна
- HL - адрес массива-анализатора
- A - режим центровки и отступ по Y (см. выше)
- B - логическое имя строки - любой код, кроме #FF

На выходе при нормальном завершении работы регистр HL содержит число, на единицу меньшее, чем адрес первого символа строки, следующей за напечатанной, сохраняется неизменным содержимое регистра IX, содержимое регистров A, B, C, E - теряется. Рестарт prstr также модифицирует системную переменную xypos (см. вектор символьного устройства вывода).

Возможные ошибки:

1. Если логическое имя строки не найдено в массиве-анализаторе, то рестарт возвращается в вызывающую программу с установленным флагом C и кодом ошибки 140 (нет имени в массиве) в регистре A.

2. Если текст не содержит строки с порядковым номером, указанным в массиве, рестарт возвращается с установленным флагом C и кодом ошибки 141 (нет строки в тексте) в регистре A.

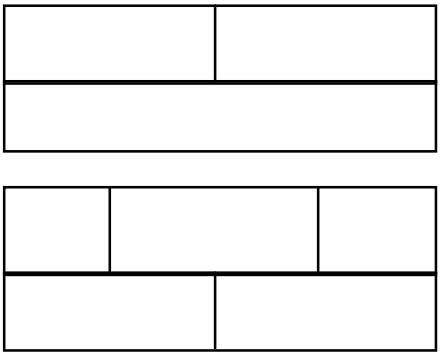
Массив-анализатор имеет следующую структуру: первые два байта - адрес текстового буфера, далее - двухбайтовые записи, в которых первый байт - логическое имя строки, а второй - ее порядковый номер в буфере. Завершается массив обязательно кодом #FF.

Рассмотрим пример. Изобразим на экране рамку из звездочек в двух разных вариантах для двух окон (см. рисунок):

Для рисования двух этих рамок при помощи подфункции 0 рестарта prstr необходимы только 4 вида строк:

1. "**********"
2. "* *"
3. "* * *"
4. "* * * *"

Составим программу, выполняющую эту задачу:



Результат работы программы

Листинг 7 Печать при помощи подфункции 0 рестарта $prstr

```
        ORG #5D60

;открытие окон

        CALL CLS        ;очистка экрана
        LD IX,WIND1     ;открыть окно 1
        LD C,#61
        LD A,2
        RST #10
        LD IX,WIND2     ;открыть окно 2
        LD C,#61
        LD A,0
        RST #10

;печать первой рамки

        LD IX,WIND1     ;адрес вектора окна
        LD HL,TABL1     ;адрес массива
        CALL PRINT      ;вызов процедуры печати текста

;печать второй рамки

        LD IX,WIND2
        LD HL,TABL2
        CALL PRINT

        CALL WAIT       ;ожидание клавиши
        JP EXIT         ;выход в IS-DOS

;процедура печати

PRINT   LD B,#09        ;число строк, которые необходимо распечатать, одновременно - логическое имя первой строки
                        ;режим выравнивания - по центру, отступ - 1 строка
        LD A,%11000001
M1      PUSH HL         ;сохранить на стеке адрес массива
        PUSH AF         ;сохранить на стеке режим выравнивания и текущую величину отступа по Y
        LD C,#68        ;код рестарта
        LD E,#00        ;код подфункции
        RST #10         ;вызов рестарта
        POP AF          ;снять со стека текущую величину отступа
        INC A           ;увеличить ее на 1 для печати следующей строки
        POP HL          ;восстановить в HL адрес массива
        DJNZ M1         ;повторить, пока значение B не уменьшится до 0
```

```
        ;значение B служит одновременно счетчиком в цикле печати и логическим именем очередной печатаемой
строки
            RET         ;возврат в основную программу

        ;векторы окон

WIND1   DEFB 0
        DEFB 0
        DEFB 11
        DEFB 30
        DEFB %00001111
        DEFB #FF
        DEFB 3
        DEFB 38

WIND2   DEFB 0
        DEFB 12
        DEFB 11
        DEFB 30
        DEFB %00001111
        DEFB #FF
        DEFB 3
        DEFB 38

        ;текстовый буфер

TEXT    DEFM "************************"
        DEFB #0D
        DEFM "* *"
        DEFB #0D
        DEFM "* * *"
        DEFB #0D
        DEFM "* * * *"
        DEFB #0D
        DEFB #03

        ;массивы-анализаторы

TABL1   DEFW TEXT
        DEFB #09
        DEFB 1
        DEFB #08
        DEFB 3
        DEFB #07
        DEFB 3
        DEFB #06
        DEFB 1
        DEFB #05
        DEFB 2
        DEFB #04
        DEFB 2
        DEFB #03
        DEFB 2
        DEFB #02
        DEFB 2
        DEFB #01
        DEFB 1
        DEFB #FF

TABL2     DEFW TEXT
          DEFB #09
          DEFB 1
          DEFB #08
          DEFB 4
          DEFB #07
```

```
DEFB 4
DEFB #06
DEFB 4
DEFB #05
DEFB 1
DEFB #04
DEFB 3
DEFB #03
DEFB 3
DEFB #02
DEFB 3
DEFB #01
DEFB 1
DEFB #FF
```

Подфункция 1 позволяет распечатать строку заданной длины, хранящуюся в памяти по указанному адресу. Эта подфункция вычисляет координаты печати строки, учитывая количество символов, отступ, режим выравнивания и ширину поля печати. При этом она пользуется той же процедурой, что и подфункция 0, поэтому за соответствием размеров окна размерам текста приходится следить самостоятельно.

После вычисления координат подфункция 1 вызывает тот же рестарт $ttyout(#0A) в цикле до тех пор, пока не будет напечатано требуемое количество символов.

Входные параметры: C - код рестарта (#68)
E - код подфункции (#01)
IX - адрес вектора окна
HL - адрес начала строки
B - длина строки
A - режим выравнивания и отступ по Y

Длина печатаемой строки определяется исключительно значением регистра B. Символ перевода строки #0D в этом случае заменяется пробелом, остальные символы с кодами меньше #20 применять не рекомендуется во избежание появления "мусора" на экране.

Пример использования подфункции 1:

Листинг 8 Пример печати при помощи $prstr(#68) подфункция 1

```
    ORG #5D64

;открытие окна

    CALL CLS
    LD IX,WIND
    LD C,#61
    LD A,2
    RST #10
    CALL WAIT

;печать строки текста из буфера в окне через подфункцию 1

    LD IX,WIND      ;вектор окна
    LD C,#68        ;код рестарта
    LD E,1          ;код подфункции
    LD HL,TEXT1     ;адрес строки
    LD A,%11000010

;два старших бита - режим выравнивания (в данном случае - центровка) шесть младших - отступ сверху (2 строки)

    LD B,30         ;кол-во символов в строке
    RST #10         ;вызов рестарта
    CALL WAIT

;печать следующей строки

    LD IX,WIND      ;вектор окна
    LD C,#68        ;код рестарта
    LD E,1          ;код подфункции
    LD HL,TEXT2     ;адрес начала строки
```

```
        LD A,%11000100

    ;печать без выравнивания и отступ сверху - 4 строки

        LD B,36         ;кол-во символов в строке
        RST #10         ;вызов рестарта
        CALL WAIT
        JP EXIT

    ;вектор окна

WIND    DEFB 1
        DEFB 1
        DEFB 8
        DEFB 30
        DEFB %00001111
        DEFB %00000001
        DEFB 3
        DEFB 38

    ;текстовый буфер

TEXT1   DEFM "Рестарт prstr(68)"
        DEFM "подфункция 1"
TEXT2   DEFM "печать строки в окне"
        DEFM "с выравниванием"
```

Подфункция 2 позволяет вычислить входные параметры для подфункции 1 (адрес и длину строки) по ее порядковому номеру в текстовом буфере.

Входные параметры: C - код рестарта (#68)
E - код подфункции (#02)
HL - адрес начала текста
B - порядковый номер строки

Выходные параметры: HL - адрес строки
B - длина строки

Содержимое регистров C и E не сохраняется. Формат текста - стандартный. Если значение порядкового номера превышает число строк в тексте, рестарт возвращается с установленным флагом C и кодом ошибки 141 (нет строки в тексте) в регистре A.

Пример использования подфункции 2:

Листинг 9 Пример использования $prstr(#68) подфункция 2

```
        ORG #5D64

    ;открытие окна

        CALL CLS
        LD IX,WIND
        LD C,#61
        LD A,2
        RST #10
        CALL WAIT
```

;пример использования подфункции 2 для определения параметров для подфункции 1 по известному номеру строки в тексте

```
        LD C,#68        ;код рестарта
        LD E,2          ;код подфункции 2
        LD HL,TEXT      ;адрес начала текста
        LD B,1          ;номер строки
        RST #10
```

;на выходе в HL' - адрес строки, а в B - длина

```
        EXX
        LD C,#68
        LD E,1          ;код подфункции 1
        LD A,%11000001
        RST #10
        LD C,#68
        LD E,2          ;определение
        LD HL,TEXT      ;параметров следующей
        LD B,2          ;строки
        RST #10
        EXX
        LD C,#68
        LD E,1          ;печать
        LD A,%11000011
        RST #10
        LD C,#68
        LD E,2
        LD HL,TEXT
        LD B,3
        RST #10
        EXX
        LD C,#68
        LD E,1
        LD A,%11000101
        RST #10
        CALL WAIT
        JP EXIT

    ;вектор окна

WIND    DEFB 1
        DEFB 1
        DEFB 8
        DEFB 30
        DEFB %00001111
        DEFB %00000001
        DEFB 3
        DEFB 38

    ;текстовый буфер

TEXT    DEFM "Рестарт prstr(68) "
        DEFM "подфункция 2"
        DEFB #0D
        DEFM "определяет параметры для"
        DEFB #0D
        DEFM "подфункции 1"
        DEFB #0D
        DEFB #03
```

Подфункция 3 - комбинация подфункций 2 и 1. В качестве входных параметров используются адрес начала текста и порядковый номер строки в нем.

Сначала подфункция 3 при помощи подфункции 2 вычисляет значения адреса и длины строки, после чего управление передается подфункции 1.

Входные параметры: C - код рестарта (#68)
E - код подфункции (#03)
IX - адрес вектора окна
HL - адрес начала текста
B - номер строки
A - режим выравнивания и отступ по Y

Как и в подфункции 2, отсутствие в тексте строки с указанным порядковым номером приводит к возврату в вызывающую программу с установленным флагом C и кодом ошибки 141 (нет строки в тексте) в регистре A.

Пример использования подфункции 3:

Листинг 10 Пример использования $prstr(#68) подфункция 3

```
        ORG #5D64

    ;открытие окна

        CALL CLS
        LD IX,WIND
        LD C,#61
        LD A,2
        RST #10
        CALL WAIT

    ;печать строк текста из буфера в окне через подфункцию 3

        LD IX,WIND      ;вектор окна
        LD C,#68        ;код рестарта
        LD E,3          ;код подфункции
        LD HL,TEXT      ;адрес начала текста
        LD A,%11000010

    ;печать с центровкой и отступ сверху - 2 строки

        LD B,1          ;номер строки в тексте
        RST #10         ;вызов рестарта
        CALL WAIT
        LD IX,WIND      ;вектор окна
        LD C,#68        ;код рестарта
        LD E,3          ;код подфункции
        LD A,%11000101
        LD B,2          ;печать следующей
        RST #10         ;строки в окне
        CALL WAIT
        JP EXIT

    ;вектор окна

WIND    DEFB 1
        DEFB 1
        DEFB 8
        DEFB 30
        DEFB %00001111
        DEFB %00000001
        DEFB 3
        DEFB 38

    ;текстовый буфер

TEXT    DEFM "Рестарт prstr(68)"
        DEFM "подфункция 3"
        DEFB #0D
        DEFM "печать строки по номеру"
        DEFB #0D
        DEFB #03
```

#6A (106) $panel

Работа с панелью. Код подфункции передаётся в регистре A (см. **ПРИЛОЖЕНИЕ 15**).

#6B (107) $wtpos

Рестарт wtpos вычисляет абсолютные логические координаты печати по значениям Y-coord и Print pos. из вектора окна, на который указывает содержимое регистра IX, с учетом смещения по Y, задаваемого в регистре H и по X, задаваемого в регистре L, и помещает их в системную переменную XYPOS, находящуюся в векторе символьного устройства вывода. Эта системная переменная активно используется рестартом ttyout(#0A), на основе которого созданы многие из рестартов печати уровня WIND.SYS.

Следует упомянуть о том, что для задания координат печати можно пользоваться также и рестартом prapd(#0C) из уровня DOS.SYS, который помещает в системную переменную XYPOS значения регистров H (Y-coord в строках) и L (X-coord в знакоместах 6*8).

Пример с использованием prapd:

Листинг 13 Установка координат для $lnstr(#6D)

```
        ORG #5D64
        CALL CLS        ;очистка экрана
        LD H,5          ;Y-coord
        LD L,2          ;X-coord
        LD C,#0C        ;код рестарта #0C из уровня DOS.SYS
        RST #10
        LD HL,TEXT  ;адрес текста
        LD C,#6D        ;код рестарта lnstr
        LD B,37         ;кол-во символов
        RST #10         ;печать строки
        CALL WAIT
        JP EXIT

    ;текстовый буфер

TEXT    DEFM "Печать строки"
        DEFM "с координатами"
        DEFM "X=2, Y=5"
```

То же с использованием $wtpos:
Листинг 14 Установка координат wtpos для $lnstr(#6D)

```
        ORG #5D64

    ;исходные данные:
    ;       Y-coord окна =3
    ;       X-coord окна =0
    ;       Print pos=1

    ;требуемые координаты печати:
    ;       X=2
    ;       Y=5

    ;следовательно:
    ;       H=2 (смещение по Y)
    ;       L=1 (смещение по X)

        CALL CLS        ;очистка экрана
        LD IX,WIND  ;вектор окна
        LD A,2          ;открытие окна
        LD C,#61
        RST #10
        LD H,2          ;Y-смещение
        LD L,1          ;X-смещение
        LD C,#6B        ;код рестарта wtpos
        RST #10
        LD HL,TEXT  ;адрес текста
        LD C,#6D        ;код рестарта lnstr
        LD B,37         ;кол-во символов
        RST #10         ;печать строки
        CALL WAIT
        JP EXIT

    ;вектор окна

WIND    DEFB 0
        DEFB 3
```

```
        DEFB 7
        DEFB 30
        DEFB %00001111
        DEFB %00000001
        DEFB 1
        DEFB 40

    ;текстовый буфер

TEXT    DEFM "Печать строки"
        DEFM "с координатами"
        DEFM "X=2, Y=5"
```

#6C (108) $str

Рестарт str(#6C) - печать в текущей позиции экрана строки, ограниченной кодом #0D. Адрес строки задается в регистровой паре HL. Этот рестарт удобен для печати строк на экране за пределами окон, не требует задания координат, а использует текущую позицию печати, хранящуюся в системной переменной XYPOS.

Рестарт str - это всего лишь циклическое обращение к уже упоминавшемуся выше рестарту $ttyout(#0A) из уровня DOS.SYS, осуществляемое до тех пор, пока не встретится код перевода строки #0D. Для определения координат печати используется системная переменная XYPOS, располагающаяся в векторе символьного устройства вывода. Доступ к этой переменной можно получить, используя рестарты $wtpos(#6B) и $prapd(#0C), о чем несколько позже.

При работе рестарта str символ перевода строки рассматривается только как ограничитель текста, и реальный перевод строки при печати не производится. Следует отметить, что размер поля печати при вызове str ограничивается одной из третей экрана - верхней, средней или нижней, а именно той, в пределах которой находилась текущая позиция печати до вызова. При достижении правой границы экрана печать переносится на следующую строку, причем последний символ может быть "разорван" при переносе, а при достижении нижней границы соответствующей трети экрана печать переносится на первую из восьми строк этой трети.

Входные параметры: C - код рестарта (#6C)
HL - адрес строки
XYPOS - системная переменная - текущая позиция печати.

Пример:

Листинг 11 Пример использования $str(#6C)

```
        ORG #5D64
        CALL CLS        ;очистка экрана
        LD HL,TEXT1     ;адрес текста
        LD C,#6C        ;код рестарта
        RST #10         ;печать строки
        CALL WAIT
        LD HL,TEXT2     ;печать второй
        LD C,#6C        ;строки
        RST #10
        CALL WAIT
        LD HL,TEXT3     ;печать третей
        LD C,#6C        ;строки
        RST #10
        CALL WAIT
        JP EXIT

    ;текстовый буфер

TEXT1   DEFM "Рестарт str (#6C)"
        DEFB #0D
TEXT2   DEFM " позволяет печатать"
        DEFB #0D
TEXT3   DEFM "в абсолютных"
        DEFM " координатах"
        DEFB #0D
```

#6D (109) $lnstr

Рестарт lnstr(#6D) - так же, как и предыдущий, обращается в цикле к $ttyout(#0A), но распечатывает только заданное количество символов. Координаты печати также хранятся в системной переменной XYPOS.

Входные параметры: C - код рестарта (#6D)
HL - адрес текста строки
B - количество символов
XYPOS - системная переменная - текущая позиция печати.

При печати символы перевода строки #0D заменяются пробелами, другие же символы с кодами меньшими #20 применять не рекомендуется. Как и в случае со str, поле печати ограничивается одной из третей экрана.

Для задания координат через системную переменную XYPOS пользуйтесь рестартами $prapd(#0C) из уровня DOS.SYS и $wtpos(#6B).

Пример:

Листинг 12 Пример работы $lnstr(#6D)

```
    ORG #5D64
    CALL CLS        ;очистка экрана
    LD HL,TEXT      ;адрес текста
    LD C,#6D        ;код рестарта
    LD B,81
    RST #10         ;печать строки
    CALL WAIT
    JP EXIT

;текстовый буфер

TEXT  DEFM "Рестарт str (#6C)"
      DEFM "позволяет печатать"
      DEFM " в абсолютных"
      DEFM "координатах"
```

#6E (110) $smbgt

Строковый мобильный редактор. Сам печатает содержимое буфера.

Вход: HL - адрес буфера,
A - ширина окна (X),
B (биты 0..4) - высота окна (Y), Старшие биты регистра B:
бит 7: курсор позиционируется на первый пробел с конца(0)/начала(1),
бит 6: на конец(0)/начало(1) буфера,
бит 5: режим "overtype"/"ME"(очистка буфера по нажатии).
D - M^CSR, E - K^CSR. (см. **ПРИЛОЖЕНИЕ 2**)

Выход: с обработкой ошибок: **Флаг C**: ошибка ввода/вывода (нет драйвера например).
Иначе (NC): **флаг Z**: Выход по клавише <Enter>, A = длина строки без <ВК>,
флаг NZ: Выход по одной из 5 специальных клавиш: CS/9, CS/SS, SS/A, SS/Space, SS/Enter (коды от #0E до #12), A = код клавиши.

#70 (112) $scrol

Еще один чрезвычайно полезный при работе с текстами рестарт - scrol(#70). Он позволяет смещать содержимое рабочей области окна на строку вверх или вниз.

При работе рестарта предусмотрено программирование заполнения крайнего левого и крайнего правого знакомест в освободившейся строке (нижней при движении вверх и наоборот).

Входные параметры: IX - вектор окна
HL - значение крайних байтов заполнения освободившейся строки (образ кодируется в двоичном представлении, например: одинарная линия по краю кодируется как 10000000 00000001 и т.п.)
A - режим работы: A=0 - настройка рестарта на вектор окна - вызывается один раз перед началом работы в данном окне. Настройка сохраняется на все время работы до следующего вызова scrol с нулевым значением в регистре A.
A=1 - скроллинг на строку вверх
A=2 - скроллинг на строку вниз

Для нормальной работы scrol необходимы еще два параметра, добавляемые к стандартному вектору окна и располагающиеся непосредственно перед ним:

IX-1 скорость скроллинга - возможные значения - 1, 2, 4, 8 определяет время, в течение которого строка перемещается на 8 пикселей по вертикали

IX-2 задержка скроллинга - возможные значения от 0 до 255 - регулирует время паузы между последовательными вызовами рестарта

Подбором этих двух параметров достигается наибольшая плавность движения при требуемой скорости скроллинга.

Пример:

Листинг 15 Применение $scrol(#70)

```
        ORG #5D64

    ;открытие окна

        CALL CLS
        LD IX,WIND
        LD C,#61
        LD A,2
        RST #10

    ;настройка процедуры scroll на вектор окна:

        LD IX,WIND      ;вектор окна
        LD C,#70        ;код рестарта
        LD HL,#0000     ;заполнение
        XOR A           ;в регистре A - 0
        RST #10

    ;печать строки prstr подфункция 3

        LD IX,WIND
        LD HL,TEXT
        LD C,#68
        LD E,3
        LD A,%00001010
        LD B,1
        RST #10

    ;скроллинг окна вверх в цикле на 5 строк

        LD B,5

M1      PUSH BC
        LD C,#70        ;код рестарта
        LD A,1          ;скроллинг вверх
        LD IX,WIND      ;вектор окна
        LD HL,#0000     ;заполнение
        RST #10
        POP BC
        DJNZ M1
        CALL WAIT
        JP EXIT

    ;вектор окна + 2 байта для рестарта $scrol(#70)

        DEFB 25         ;задержка
        DEFB 01         ;скорость
WIND    DEFB 0
        DEFB 5
        DEFB 12
        DEFB 25
        DEFB %00000111
        DEFB %00000001
        DEFB 2
        DEFB 25
```

```
        ;текст для печати

TEXT    DEFM "Так работает scrol(#70)"
        DEFB #0D
        DEFB #03
```

Применение рестарта scrol вместе с рестартами печати (например prstr) позволяет организовать на экране режим "бегущего текста".

Рассмотрим еще один пример:

Листинг 16 Программа "бегущий текст"

```
        ORG #5D64

        ;процедура открытия окна

        CALL CLS
        LD IX,WIND
        LD C,#61
        LD A,2
        RST #10

        ;настройка процедуры scrol на вектор окна:

        LD IX,WIND
        LD C,#70
        LD HL,0
        XOR A
        RST #10

        ;входные параметры для процедуры печати

        LD A,%00001010

        ;в рег. A - биты 7 и 6 определяют режим выравнивания
        ;биты 5...0 определяют отступ от верхнего края окна

        LD B,1      ;номер строки в тексте (строки отделяются друг от друга символом с кодом #0D)
M1      PUSH AF     ;сохранить регистр A
        PUSH BC     ;сохранить BC

        ;печать строки по номеру ее в тексте

        LD IX,WIND  ;адрес вектора окна
        LD HL,TEXT  ;адрес начала текста
        LD C,#68    ;код prstr
        LD E,3      ;код подфункции
        RST #10
        CALL SCROLL ;вызов scrol
        POP BC      ;восстановить BC
        INC B       ;увеличить номер строки в тексте
        LD A,B      ;проверка на число
        CP 9        ;напечатанных строк
        JP Z,END    ;выход, если все строки напечатаны
        POP AF      ;восстановить A
        JR M1       ;переход к печати следующей строки

        ;вектор окна + 2 байта для рестарта $scrol(#70)

        DEFB 25     ;задержка
        DEFB 01     ;скорость
WIND    DEFB 1
        DEFB 1
        DEFB 12
        DEFB 28
        DEFB %00000111
```

```
        DEFB %00000001
        DEFB 4
        DEFB 35

    ;текст для печати

TEXT    DEFM "Эта программа"
        DEFB 13
        DEFM "представляет собой пример"
        DEFB 13
        DEFM "организации в системе"
        DEFB 13
        DEFM "IS-DOS работы с экраном"
        DEFB 13
        DEFM "в режиме 'бегущий текст'"
        DEFB 13
        DEFM "Вывод текста организуется"
        DEFB 13
        DEFM "при помощи рестартов"
        DEFB 13
        DEFM "scrol(#70) и prstr(#68)"
        DEFB 13

    ;Дополнительные процедуры:

SCROLL LD C,#70     ;процедура скрол-
        LD A,1      ;линга окна на
        LD IX,WIND  ;строку вверх
        LD HL,#0000
        RST #10
        RET

END     POP AF      ;завершение работы
        CALL WAIT
        JR EXIT
```

#71 (113) $tylin

Распечатка строки по вектору для edstr от XS (позиции курсора) до конца видимой области.
Вход: IX - адрес вектора ("ошибок быть не может")(см. **ПРИЛОЖЕНИЯ 10..13**)

#72 (114) $g_scr

Возвращение в регистре HL' адреса вектора экрана и beep'a (см. **ПРИЛОЖЕНИЕ 14**)

#73 (115) $cls

Если при вызове рестарта cls в регистре A находится 0, то происходит полная очистка экрана - все пиксели сбрасываются, область цветовых атрибутов заполняется текущим значением системной переменной PAPER, находящейся в векторе экрана, адрес которого можно получить, используя рестарт $q_scr(#72). Также при этом цвет бордюра приводится в соответствие с текущим значением системной переменной BORD из того же вектора. Формат задания значений цветов стандартный, атрибуты бордюра кодируются тремя младшими байтами, остальные байты игнорируются.

Если же перед вызовом cls в регистре A находится число, отличное от нуля, то сброс пикселей не происходит, а инициализируется только область атрибутов экрана, причем значение регистра A трактуется как байт атрибутов в стандартном представлении. Изменения цвета бордюра в этом случае не происходит.

Как уже говорилось выше, доступ к системным переменным PAPER и BORD можно получить через вектор экрана, адрес которого возвращает в регистре HL' рестарт $q_scr(#72). Вектор экрана (не путать с вектором окна) представляет собой область системных переменных, уровня WIND.SYS. Необходимые нам переменные располагаются в нем следующим образом:

Если в регистре IX - адрес вектора экрана, то
(IX+0) - значение PAPER
(IX+1) - значение BORD

Остальные системные переменные этого вектора будут рассмотрены позднее.

Процедура для занесения необходимых значений цветов в соответствующие системные переменные может выглядеть примерно так:

```
LD C,#72      ;код рестарта q_scr
RST #10
EXX           ;HL - адрес вектора
PUSH HL       ;переносим его
POP IX        ;в регистр IX
LD A,%00101010
LD (IX+0)     ;загрузка атрибутов экрана
LD A,%00000100
LD (IX+1)     ;загрузка атрибутов бордюра
```

После этого Вы смело можете обнулять регистр A и вызывать cls:

```
XOR A
LD C,#73
RST #10
```

значения цветов будут установлены именно так, как Вы и хотели.

Рассмотрим пример:

Листинг 17 Пример работы $cls

```
    ORG #5D64

;установка цветовых атрибутов без сброса пикселей (A - байт атрибутов)

    LD A,%00111000
    LD C,#73          ;код рестарта $cls
    RST #10
    CALL WAIT

;установка значений системных переменных PAPER и BORD

    LD C,#72      ;получение адреса
    RST #10       ;вектора экрана
    EXX           ;в HL - адрес
    PUSH HL       ;перенос адреса
    POP IX        ;в регистр IX
    LD A,%00101010
    LD (IX+0),A   ;установка цветов экрана
    LD A,%00000011
    LD (IX+1),A   ;установка цветов бордюра
```

```
;полная очистка экрана, цвета берутся из системных переменных PAPER и BORD

    XOR A           ;в A - 0
    LD C,#73
    RST #10
    CALL WAIT
    JR EXIT
```

#76 (118) $y, #77 (119) $n

Еще два рестарта отвечают за включение и выключение в текущей позиции экрана мигающего курсора.

Положение курсора определяется уже знакомой нам системной переменной XYPOS, находящейся в векторе символьного устройства вывода.

Рестарт $y(#76) включает курсор в текущей позиции, а рестарт $n(#77) - выключает. Координаты курсора абсолютные.

Процедура изображения мигающего курсора добавляется в цепочку процедур, вызываемых по прерываниям от таймера при помощи рестарта $im2(#1E) из уровня DOS.SYS.

При включении следующего курсора не забывайте убирать предыдущий, хотя никаких серьезных последствий, кроме "засорения" экрана останками курсоров это не вызовет, а рестарт $n(#77) спокойно уберет их все за один прием.

Пример:

Листинг 19 Пример работы с курсором

```
    ORG #5D64

;открытие окна

    CALL CLS
    LD IX,WIND
    LD C,#61
    LD A,2
    RST #10

;печать строк текста из буфера в окне через подфункцию 3 рестарта $prstr

    LD IX,WIND
    LD C,#68
    LD E,3
    LD HL,TEXT
    LD A,%11000010
    LD B,1
    RST #10
    LD C,#68
    LD E,3
    LD B,2
    LD HL,TEXT
    LD A,%11000100
    RST #10
    CALL WAIT

;включение и перемещение курсора по экрану

    LD B,12
    LD H,4          ;Y-смещение для wtpos
    LD L,5          ;X-смещение для $wtpos
M1  PUSH BC
    PUSH HL
    LD C,#6B        ;вызов wtpos, уста-
    RST #10         ;новка начальных координат курсора
    LD C,#76        ;включение курсора
    RST #10
    CALL WAIT       ;ожидание клавиши
    LD C,#77        ;выключение курсора
    RST #10
```

```
        POP HL          ;увеличение смеще-
        INC HL          ;ния координат
        POP BC
        DJNZ M1         ;перейти к началу цикла с новыми координатами
        CALL WAIT
        JP EXIT

    ;вектор окна

WIND    DEFB 1
        DEFB 1
        DEFB 8
        DEFB 30
        DEFB %00001111
        DEFB %00000001
        DEFB 3
        DEFB 38

    ;текстовый буфер

TEXT    DEFM "Рестарты y(#76)"
        DEFM "и n(#77)"
        DEFB #0D
        DEFM "включение-выключение"
        DEFM "курсора"
        DEFB #0D
        DEFB #03
```

#7C (124) $d_a, #7D (125) $a_d

Еще два рестарта могут быть необходимы Вам при работе с числами. Они позволяют переводить целые числа из стандартного 2-х или 4-х байтового представления в ASCII-строку и наоборот. При помощи этих двух рестартов Вы сможете привести любое число в форму, приемлемую для печати на экране, а также сможете сделать понятным компьютеру число, введенное в символьной форме.

Рестарт d_a(#7C) - преобразует целое 2-х или 4-х байтовое число в ASCII-строку

Входные параметры:
- C - код рестарта (#7C)
- DE - число, которое надо преобразовать
- HL - адрес выходного буфера
- A - длина выходного буфера
- B - основание системы счисления (2, 8, 10, 16)
- Состояние флага C определяет формат числа:
  - флаг C=0 - 2-х байтовое число
  - флаг C=1 - 4-х байтовое число

На выходе образуется требуемое число в символьной форме, причем строка внутри буфера выравнивается по правой границе, а оставшаяся левая часть дополняется пробелами. При переполнении буфера рестарт возвращается в вызывающую программу с установленным флагом C (независимо от его первоначального состояния) и кодом ошибки 1 (переполнение буфера) в регистре A.

При переводе чисел в 16-ричную, 8-ричную и двоичную системы символы этих систем (#, o, % и т. п.) не применяются и в буфер не вносятся, поэтому за принадлежностью числа к той или иной системе следите самостоятельно.

Рассмотрим пример:

Листинг 20 Перевод чисел в символьную форму и распечатка их в окне

```
        ORG #5D64

    ;открытие окна

        CALL CLS
        LD IX,WIND
        LD C,#61
        LD A,2
        RST #10

    ;перевод числа из 2-х байтовой формы в символьную (в DE - число)
```

```
        LD C,#7C          ;код рестарта
        LD DE,#9C40     ;исходное число
        LD HL,TEXT      ;адрес буфера
        LD A,5            ;длина буфера
        SCF               ;сбросим
        CCF               ;флаг C
        LD B,10           ;десятичная система счисления
        RST #10

    ;печать числа в символьной форме через подфункцию 1 рестарта $prstr

        LD IX,WIND
        LD C,#68
        LD E,1
        LD HL,TEXT
        LD A,%11000010
        LD B,5
        RST #10
        CALL WAIT

    ;перевод числа из 4-х байтовой формы в символьную (в DE - адрес числа)

        LD C,#7C          ;код рестарта
        LD HL,TEXT      ;адрес буфера
        LD DE,TABL      ;адрес числа
        LD A,8            ;длина буфера
        LD B,16           ;16-ричная система счисления
        SCF               ;установим флаг C
        RST #10

    ;печать числа в окне

        LD IX,WIND
        LD C,#68
        LD E,1
        LD HL,TEXT
        LD A,%11000100
        LD B,8
        RST #10
        CALL WAIT
        JP EXIT

    ;вектор окна

WIND    DEFB 1
        DEFB 1
        DEFB 8
        DEFB 30
        DEFB %00001111
        DEFB %00000001
        DEFB 3
        DEFB 38

    ;текстовый буфер

TEXT    DEFM "          "

    ;исходное 4-х байтовое число

TABL    DEFW #0000      ;младшие 2 байта
        DEFW #9C40      ;старшие 2 байта
```

Рестарт $a_d(#7D) предназначен для обратного перевода числа из символьной формы в стандартную четырехбайтовую. Он преобразует ASCII-строку указанной длины, или завершенную кодом #0D, игнорируя ведущие пробелы. Результат располагается в альтернативных регистрах - младшие байты в регистровой паре

HL', старшие байты - в регистровой паре DE', при этом в регистровой паре BC' остается адрес продолжения входного буфера.

Входные параметры: C - код рестарта (#7D)
HL - адрес входного буфера
A - длина входного буфера; если A=0, то преобразуется строка до символа с кодом #0D
B - основание системы счисления по умолчанию. Основание системы счисления не должно быть больше 16, причем 0 трактуется как 10 (десятичная система)

Система счисления может также определяться первым символом ASCII-строки:

h H - 16-ричная система
d - десятичная система
o O - восьмеричная система
%b - двоичная система

Выходные параметры: DE'- старшие байты числа
HL'- младшие байты числа
BC'- адрес следующего символа ASCII буфера

Этот рестарт в комплексе с предыдущим позволяет производить арифметические действия над числами, хранящимися в символьной форме (например внутри текста).

Рассмотрим пример. Пусть необходимо напечатать строку "10000+12=", выполнить указанное в ней арифметическое действие и напечатать результат (см. Листинг 21).

Листинг 21 Пример использования $a_d(#7D) арифметические действия над числом в символьной форме

```
    ORG #5D64

;открытие окна

    CALL CLS
    LD IX,WIND
    LD A,2
    LD C,#61
    RST #10

;печать строки с исходным числом

    LD HL,TEXT
    LD C,#68
    LD E,1
    LD B,9          ;длина всей строки
    LD A,%11000001
    RST #10
    CALL WAIT

;перевод числа в 4-х байтовую форму

    LD HL,TEXT      ;адрес входного буфера
    LD A,5          ;кол-во символов в исходном числе
    LD B,10         ;система счисления
    LD C,#7D        ;код рестарта
    RST #10
    EXX

;на выходе в HL - младшие байты, а в DE - старшие

    LD BC,12        ;сложение чисел
    ADD HL,BC
    PUSH DE         ;помещение резуль-
    PUSH HL         ;тата в буфер
    LD HL,TABL
    POP DE
    LD (HL),E
    INC HL
    LD (HL),D
    INC HL
    POP DE
```

```
        LD (HL),E
        INC HL
        LD (HL),D

    ;перевод числа в символьную форму результат помещается в текст на место исходного числа

        LD C,#7C
        LD HL,TEXT
        LD A,5          ;кол-во символов числа - результата
        LD B,10
        LD DE,TABL
        SCF
        RST #10

    ;печать результата

        LD C,#68
        LD E,1
        LD IX,WIND
        LD HL,TEXT
        LD B,5
        LD A,%11000011
        RST #10
        CALL WAIT
        JP EXIT

    ;вектор окна

WIND    DEFB 1
        DEFB 1
        DEFB 5
        DEFB 30
        DEFB %00000111
        DEFB %00000001
        DEFB 3
        DEFB 35

    ;текстовый буфер

TEXT    DEFM "10000+12="

    ;буфер числа

TABL    DEFB #00
        DEFB #00
        DEFB #00
        DEFB #00
```

Еще одним практическим примером использования рестартов $a_d и $d_a может служить Листинг 22 - программа перевода чисел из одной системы счисления в другую

Листинг 22 Перевод чисел в 16-ричную систему (пример использования $d_a)

```
        ORG #5D64

    ;открытие окна

        CALL CLS
        LD IX,WIND
        LD C,#61
        LD A,2
        RST #10

    ;печать пояснительного текста

        LD C,#66
```

```
        RST #10

    ;ввод числа с клавиатуры

START   LD C,#88        ;рестарт $ed_dig
        LD A,17         ;из уровня SHELL
        LD DE,#0402     ;будет рассмотрен
        LD HL,#0000     ;в следующих
        LD IX,WIND      ;выпусках серии
        RST #10
        JP C,START
        JP NZ,EXIT

    ;помещение числа в буфер

        EXX
        EX DE,HL
        LD HL,TABL
        LD (HL),E
        INC HL
        LD (HL),D

    ;перевод числа из 4-х байтовой формы в символьную (в DE - адрес числа)

        LD C,#7C        ;код рестарта
        LD HL,TEXT      ;адрес буфера
        LD DE,(TABL)    ;адрес числа
        LD A,16         ;длина буфера
        LD B,16         ;система счисления
        SCF
        CCF             сбросим флаг C
        RST #10

    ;печать числа в окне

        LD IX,WIND
        LD C,#68
        LD E,1
        LD HL,TEXT
        LD A,%10000100
        LD B,16
        RST #10
        CALL WAIT
        CP #10
        JP Z,EXIT
        JP START

    ;вектор окна

  WIND  DEFB 1
        DEFB 1
        DEFB 8
        DEFB 30
        DEFB %00001111
        DEFB %00000001
        DEFB 3
        DEFB 36

    ;пояснительный текст

        DEFM "Перевод чисел в 16-ричную"
        DEFM "систему"
        DEFB #0D
        DEFM " Входное значение"
        DEFM "   Результат   "
        DEFB #0D
```

```
        DEFB #0D
        DEFM "                  "
        DEFM "              0"
        DEFB #0D
        DEFB #0D
        DEFM "   (C) PENCRAFT 1994 "
        DEFM "SS+A - Выход"
        DEFB #0D
        DEFB #03

    ;текстовый буфер

TEXT    DEFM "            "

    ;буфер числа

TABL    DEFW #0000
```

#7E (126) $analys

Рестарт analys(#7E) позволяет организовать в Вашей программе таблицу процедур. Он находит в таблице адрес необходимой процедуры по ее коду и передает ей управление.

Входные параметры: HL - адрес таблицы процедур
A - код процедуры

Таблица состоит из записей по три байта, из которых первый - код процедуры, а два оставшихся - ее адрес в стандартном формате (первый байт - младший).

Последней записью должна быть процедура обработки ситуации "код не найден", для нее зарезервирован код #FF, после которого и следует поместить ее адрес.

На выходе из рестарта при передаче управления процедуре в регистре HL сохраняется адрес таблицы, в A - код команды, а регистр BC' содержит адрес процедуры, которой передается управление. Остальные регистры (кроме альтернативного набора) не изменяются и могут быть использованы для передачи параметров. Переход к процедуре осуществляется путем помещения ее адреса рестартом analys на стек и подачей команды RET, поэтому передача параметров через стек в данном случае нежелательна во избежание путаницы с адресами.

Все процедуры должны заканчиваться инструкцией RET, которая возвращает управление в точку, следующую за вызовом рестарта analys.

При помощи рестарта analys можно организовать, например, программу для обработки клавиш управления курсором передавая управление процедурам перемещения его в разных направлениях в зависимости от кода нажатой клавиши (см. Листинг 23).

Листинг 23 Вызов процедур по таблице рестарт $analys(#7E)

```
        ORG #5D64
        CALL CLS

    ;установка координат курсора

        LD H,10         ;Y-coord курсора
        LD L,20         ;X-coord курсора
        LD C,#0C        ;вызов prapd, уста-
        RST #10         ;новка начальных координат курсора

    ;основная программа

MAIN    LD C,#76        ;включение курсора
        RST #10
        LD C,#12        ;вызов вектора
        RST #10         ;символьного устройства
        EXX             ;вывода, HL - адрес
        INC HL        ;координат курсора
        EX DE,HL      ;помещаем его в DE
        CALL WAIT     ;ожидание клавиши, в A - код клавиши
        PUSH AF       ;сохраним его
        LD C,#77      ;выключение курсора
        RST #10
        POP AF        ;восстановим регистр A
        LD HL,TABL    ;адрес таблицы процедур
        LD C,#7E      ;код рестарта analys
        RST #10       ;вызов рестарта

    ;управление передается по адресу из таблицы в соответствии с кодом клавиши по окончании отработки процедуры - возврат по RET на следующую после вызова рестарта команду

        CP #F4          ;проверка на
        RET Z           ;окончание работы
        JP MAIN         ;повторить цикл

    ;таблица процедур

TABL    DEFB #0B        ;CS+7
        DEFW UP
        DEFB #0A        ;CS+6
        DEFW DOWN
        DEFB #08        ;CS+5
        DEFW LEFT
        DEFB #09        ;CS+8
        DEFW RIGHT
        DEFB #10        ;SS+A
        DEFW END
        DEFB #FF        ;остальные клавиши
        DEFW NO_KEY

    ;процедуры

UP      EX DE,HL        ;увеличение
        INC HL          ;координаты Y
        DEC (HL)
        RET

DOWN    EX DE,HL        ;уменьшение
        INC HL          ;координаты Y
        INC (HL)
        RET

LEFT    EX DE,HL        ;уменьшение
```

```
        DEC (HL)        ;координаты X
        RET

RIGHT   EX DE,HL        ;увеличение
        INC (HL)        ;координаты X
        RET

NO_KEY  RET             ;"ключ не найден"

END     XOR A           ;установка парамет-
        LD A,#F4        ;ров для выхода в
        RET             ;IS-DOS и флага Z для выхода из цикла MAIN
```

;после отработки процедур инструкция RET осуществляет переход к следующей после вызова рестарта analys команде.

#7F (127) $edstr

Редактирование строки в тексте. Отрабатывает все печатные коды (символы) + 6 управляющих: клавиши 1,2,3,5,8,0 с Caps Shift'ом.

Вход: IX - вектор, (на входе необходимо инициировать LNST, XS и COM)

Выход: CARRY - системная ошибка (см. **ПРИЛОЖЕНИЯ 10..13**).

ОБОЛОЧКА (SHELL.SYS)

Это самый верхний уровень стандартной конфигурации системы iS-DOS. Выше него могут устанавливаться дополнительно еще три. Обычно это два уровня баз данных или уровень отладчика ВиОС.sys, 7-ой уровень временно занимается текстовым редактором. Замечу при этом, что уровни iSDOS не должны обязательно находиться в памяти один рядом с другим. Редактор, например, содержит 7-ой уровень прямо в себе, т.е. в файле edit+.com, и всего лишь помещает адрес своей таблицы рестартов в таблицу уровней, находящуюся в 0-ом уровне (DOS).

4-ый уровень, или уровень SHELL, содержит рестарты, позволяющие передавать управление нортонообразной оболочке, вызывать некоторые из ее функций, а также включает такие полезные подпрограммы, как редактирование десятичного числа и меню. Программы, не использующие данные функции, могут временно снимать верхние 2 уровня (3-ий и 4-ый) для увеличения свободной памяти в iSDOS Classic. Так поступает, скажем FORTH. Так работал копировщик copy_all.com. Обе программы запускались из bat-файла, в самом начале которых вызывалась программа set.com, снимавшая все лишние резиденты и эти 2 уровня. На выходе эти уровни можно было поставить обратно. Но, поскольку 4-й уровень (shell.sys) после установки необходимо инициализировать программой shell.com, то проще обычно перезагрузиться. В системе iSDOS Chic необходимость снимать верхние уровни отпала, да и выигрыш там был бы при этом очень небольшим - всего 1063 байта shel.sys + 764 байта twira.sys. Их адреса Вы можете посмотреть сами в программе show.com. Да и панелью пользоваться удобно во многих программах. Работать с отмеченными файлами, например, или открыть какой-нибудь файл, не выходя из программы, с помощью панели куда удобнее слепого ввода его пути и имени. Я уже не говорю про рестарт $menu. Покажите мне хоть одну программу сложности чуть выше минимальной, не использующую эту удобную вещь! Итак, рестарты один за другим:

#80 - $shel0, #81 - $shel1, #82 - $shel2

Эти три рестарта перехватывают стек и отдают управление в 4-й уровень iSDOS, т.е. возвращают его 2-панельному файл-менеджеру. Пользуются им программы крайне редко, т.к. такой выход в систему, естественно, прерывает выполнение "вышестоящей" программы, которой была вызвана текущая программа, например, bat-файла, mon.com или mon.res, menu.com и т.д. Т.е. Вы вернётесь, но не туда, откуда Вас вызывали, а прямо в систему. Причём, флаг C в этом случае роли не играет, т.е. по ошибке Вы вернулись или ещё как, система не узнает, и пользователя проинформировать не сможет. Нормальный выход в систему - это всегда выход по RET. Печатать же панели из своей программы для временного "выхода" в файл-менеджер надо рестартами $shpan(#90) и $shsub(#8e) (см. для примера zB_shsub.asm).

#80 (128) $shel0

Главный выход в оболочку: создание виртуального диска. (Размер: g_cnfg(#10): HL'-6)

) Все 3 shel-рестарта перехватывают стек и принудительно задают pnCSR=%01010, nLOG=1 (см. **ПРИЛОЖЕНИЯ 16,17), причем shel1 является точкой входа в shel0, а shel2 - в shel1.*

#81 (129) $shel1

Выход с очисткой экрана, инициализацией ERDEV, QUEST (см. **ПРИЛОЖЕНИЕ 3**), USPNL, PAPER, BORD (см. **ПРИЛОЖЕНИЕ 14**), переключением на основное (0-ое) устройство печати /swtyp(#1B)/ в прямом режиме /prcpl(#0B)/. В блочном драйвере устанавливается 7-ой бит в 9-ом байте (см. **ПРИЛОЖЕНИЯ 3,6**).

) Все 3 shel-рестарта перехватывают стек и принудительно задают pnCSR=%01010, nLOG=1 (см. **ПРИЛОЖЕНИЯ 16,17), причем shel1 является точкой входа в shel0, а shel2 - в shel1.*

#82 (130) $shel2

Выход с перерисовкой и подкраской верхней строки подсказок и печатью обеих панелей.

) Все 3 shel-рестарта перехватывают стек и принудительно задают pnCSR=%01010, nLOG=1 (см. **ПРИЛОЖЕНИЯ 16,17), причем shel1 является точкой входа в shel0, а shel2 - в shel1.*

#83 (131) $pnview

Сей рестарт предназначен для подготовки грамотного выхода в систему по RET. Например, Вы сотворили нечто в каталоге своей программой (создали новый файл, скажем) и хотите, выходя перепечатать из чувств минимализма лишь текущую панель. Имеете полное право, если рисовали все окна только на ней или вообще их не рисовали. Но что делать, если две панели одинаковые? Тогда надо перепечатать их обе. Вот этой проверкой на их одинаковость и занимается данная точка входа в систему. Заодно, она ожидает отпуска клавиш. Если

панели одинаковые, в регистре A получите код cshel2 (#F2), а ежели разные - код cvwpnl (#F0). (Список всех подобных кодов см. в **ПРИЛОЖЕНИИ 16**) Флаги, по идее, уже должны быть все установлены. Останется лишь дописать RET и вывалиться в систему.

Пример из программы delete.com:

```
          LD C,$menu      ;Единственное меню
          RST 16          ;программы
          RET C           ;Error?
QUIT
STACK     LD SP,0         ;возвращаем на место
EXCSR     LD A,0          ;20, если удалялся файл
          OR A            ;0, если каталог или отказ
          JR NZ,PNVW
      ;Если удалялся каталог, на всякий пожарный перепечатаем обе панели (f2):

AQUI      LD A,#F8        ;c_sh Если отказ - f8
          RET             ; Если каталог - f2
PNVW
      ;Удалялся файл.
          LD C,$pnvw      ;<-- ВОТ ОНА!
          RST 16
          RET             ; выход из delete.com
```

Пример из программы rename.com:

```
      ;После переименовывания файла

          CALL $SVSTL     ;Сохраняем стиль в
          RET C           ;файле rename.com
          LD C,$flush     ;Сохраняем блоки
          RST 16          ;кэша на диске
          RET C
          LD A,#F2        ;Перепечатка 2 панелей
          LD (AQUI+1),A
          EXIT
      ;Сюда прыгаем по отказу или после меню

          LD C,$kwait     ;Ждём отпуска клавиатуры
          RST 16
EXCSR     LD A,0          ;0, если каталог
          OR A
          JR NZ,PNVW

AQUI      LD A,0          ;F2, если были в меню или меняли имя каталога (см. выше) 0, т.е. "ничего не перепечатывать", если ничего не делали
          RET

PNVW
      ;файл
          LD C,$pnvw      ;<-- ВОТ ОН!
          RST 16
          CP #F2          ;если панели разные
          RET Z           ;то обе и перепечатывать
          XOR A           ;ИНАЧЕ: ничего не делать
          RET
```

#84 (132) $shout

Выход в оболочку с выполнением команды shell. Если флаг C установлен, то в регистре A - код ошибки, если же Carry флаг сброшен, а флаг Z поднят, то регистр A воспринимается как код команды. Например:

AF=5245: Флаг C поднят, A=#52, получите сообщение: "Error 82". Или: AF=#F244: флаг C сброшен, флаг Z поднят, выполняется команда "Перепечатать 2 панели".

Данный рестарт, как и рестарты #80-82, перехватывает стек, поэтому пользоваться им надо лишь в исключительных случаях, когда по RET, выходить просто некуда. Не шагать же, в самом деле, в открытый

космос! Или когда возвращаться уже просто незачем. Все мосты сожжены. Однако всё, что было здесь сказано про флаги и регистр A, будет работать и при выходе по RET. Правда, лишь в том случае, когда Вы при выходе попадаете прямо в shell, а скажем, не в bat-файл. exebat.com, получив от Вас код в регистре A, передаст его оболочке только при поднятом флаге C, т.е. вывалится далее с кодом ошибки. Т.о. весь этот интерфейс предназначен для обычного вызова com-файлов по Enter или через extent.txt либо extkey.txt. Тем не менее, эти коды широко применяются во многих программах и их полезно знать. Полностью они приведены в **ПРИЛОЖЕНИИ 16**.

Пример из программы uni_boot.sys, загрузчика iSDOS Chic с HDD для машин KAY:

```
...

    CALL READ           ;читаем is_dos.sys
    RET C
    LD A,(BUFF+32+22)   ;Регистр I
    LD I,A              ;из описателя файла

;Система загружена. Теперь можно пользоваться её рестартами:

;Создаём cache:
    LD C,$g_cnf         ;Главный вектор
    RST 16              ;конфигурации
    EXX                 ;системы
    LD BC,-6
    ADD HL,BC
    LD A,(HL)           ;Размер cache
    LD C,$creat         ;Создать Кэш
    RST 16
    JR C,ERROR          ;C означает ошибку

;Берём дату с диска и устанавливаем её в систему:
    LD C,$g_com
    RST 16
    EXX
    DEC HL
    DEC HL
    LD DE,(BUFF+30)     ;Дата из 0-го
    LD (HL),D           ;блока диска
    DEC HL
    LD (HL),E

;Open disk "S":

    LD A,"S"
    LD C,$dvtrn         ;Преобразуем в A=0..7
    RST 16
    LD B,A
    LD C,$swblk         ;Переключаемся на S
    RST 16
    JR C,ERROR

    LD C,$open          ;...и открываем
    RST 16
    JR C,ERROR

;BASIC & TR-DOS variables:

    LD HL,TRDBUF        ;Загружаем 2 блока
    LD DE,#5C00         ;переменных TR-DOS'a
    LD BC,#200
    LDIR

;Вот ещё один редкостный RST 4-го уровня.
;Запускаем autoexec.bat и, если всё ОК, то управления нам он уже не отдаст...
    LD C,$shexe
    IM 2
```

```
            JR NC,1$

        ;А ВОТ И ОН! Сюда попадаем по ошибке
            LD C,$shout
1$          EI
            RST 16
ERROR
        ;Или сюда:
            LD C,$shout
            RST 16
        ;-----
TRDBUF DEFS #200
```

#85 (133) $p^csr

Устанавливает маску состояния файла в панели для пометки и проверки файла.

Вход: A= маска.

По умолчанию маска=1, т.е. уровень SHELL метит файлы, поднимая 1-ой бит.

#86 (134) $^csr

По номеру файла в каталоге (в регистре E) возвращает: A=(HL')= байт состояния отметки файла (отметка есть, когда бит 1 установлен в 1);
(HL'-1)= порядковый номер отметки.

Номер отметки начинается с 1. Если (HL'-1) равен 0, файл не отмечен.

#87 (135) $g_mpan

Запрос о состоянии текущей панели.

Выход: A - число отмеченных файлов на текущей панели,
L' и H' - минимальный и максимальный номера отмеченных файлов, а именно: L'=0, H'= где-то между общим числом файлов в каталоге, включая удаленные, но без каталогов, и числом неудалённых файлов, но с каталогами.
DE' - адрес вектора рабочей панели -48,
BC' - адрес вектора оболочки (см. **ПРИЛОЖЕНИЯ 17,18**).

#88 (136) $ed_dig

Редактирование десятичного числа в окне (для menu, например).

Вход: A - число цифр,
DE - координаты для wtpos,
HL - начальное значение числа,
IX - вектор окна.

Выход: Флаг C: ошибка (переполнение числа, ввода/вывода и пр.);
NC, Z: HL'= новое значение числа;
NZ: выход по одной из 5 специальных клавиш (см. $smbgt(#6E).

Следует обратить внимание, что при режиме ввода INS (вставка символов) этот рестарт работать не будет (не будет вводиться ни один символ). К примеру, если режим ввода в строковом редакторе mon.res вы переключите режим ввода на INS, то он таким и останется даже после выхода из mon.res. Режим ввода можно посмотреть при установленном резиденте indi.res, который при вызове любого строкового редактора будет показывать режим его работы.

#89 (137) $mmenu

Перерисовка верхней строки подсказок оболочки с подкраской или без в зависимости от 0-го бита nLOG (см. **ПРИЛОЖЕНИЕ 17**).

#8A (138) $g_curs

Возвращает параметры панельного курсора.
Выход: A=E'= номер файла,
D'= позиция Y в окне (=H для wtpos),
B' - номер текущей панели (0 - левая, 1 - правая),
HL' - адрес вектора окна панели (**ПРИЛОЖЕНИЯ 17,18**).

#8B (139) $mwait

Пишет "Please Wait" на месте курсора

#8C (140) $g_dev

Переключение устройств. Ввод символа с клавиатуры с миганием курсора в окне.
Вход: HL=смещение для wtpos,
Carry Set: A=номер устройства,
NC: иначе: текущее.
Флаг Z: Выход через RST $dvtrn(#4A),
NZ: через smbgt(#6E).
Выход: CF – ошибка;
NZ – выход по SS+A;
NC,Z – всё ок.

#8E (142) $shsubr

Выполнение команд из прикладных задач.
Вход: A - код команды (**ПРИЛОЖЕНИЕ 16**).

#8F (143) $cpfil

Сравнение 11-байтового шаблона(DE), (например результата RST $trans(#46)), с 11-байтовым описателем(HL) файла (имя и расширение). Если равны, то флаг Z, иначе NZ.

#90 (144) $shpanl

Выполнение команд работы с панелями:
Вход: B - регистр состояния pnCSR (**ПРИЛОЖЕНИЕ 17**).
A=0 -аналог shel0, A=1 -аналог shel1, A=2 -аналог shel2, A=3 - движение курсора по панели,
A=5 -pnDR - печать обеих панелей, A=6 -TYPNL- печать одной панели.

#91 (145) $menu

Этот рестарт осуществляет работу с меню. Перед вызовом рестарта надо вывести окно при помощи рестарта #61, текст в окне печатает сам рестарт $menu (#91).
Вход: IX - адрес вектора окна меню.
Вектор окна меню представляет собой обычный вектор окна (см. описание рестарта #61). Но к нему добавлены данные до и после самого вектора. Сам вектор меню может быть двух видов – старого и нового (см. далее).

Формат вектора меню:

Смещение	Длина	Описание
-12	1	Цвет курсора при нажатии на Enter
-11	1	Управляющий регистр меню. Биты (0/1): 0 – тип курсора (awt/awtc). Курсор **awt** заходит краями на рамку, а **awtc** – края курсора внутри рамки меню 1 – старый/новый формат вектора меню 7 – вызывать программу пользователя по смещению -6(2) до/после нажатия клавиши
-10	1	Цвет курсора меню
-9	1	Позиция курсора в меню (нумерация начинается с 1)
-8	2	Адрес программы инициализации, вызываемой при входе в меню и при каждом выходе из

		строковой функции с NZ, A=1. Если значение равно 0, то программа не вызывается.
-6	2	Адрес программы пользователя, выполняющейся при нажатии любой клавиши. Входных данных у этой программы нет, т.е. код клавиши в неё не передаётся. Если значение равно 0, то программа не вызывается.
-4	2	Адрес списка дополнительных функций (т.н. «горячих клавиш»). Этот список должен состоять из трёхбайтных значений: 0(1) – код клавиши 1(2) – адрес дополнительной процедуры В этом случае на входе в процедуру регистр B содержит код клавиши, курсор не перемещается. Или можно задать так: 0(1) - код клавиши 1(1) - номер строки 2(1) = 0 В данном случае клавиша обрабатывается как нажатие Enter на строке с указанным номером. Курсор (IX-9) принимает соответствующее значение. Признаком конца списка служит маркер #FF. Если адрес списка равен 0, то считается, что "горячие клавиши" не обрабатываются в меню.
-2	2	Адрес списка функций отработки клавиши Enter. Список должен состоять из двухбайтовых адресов процедур, количество которых должно соответствовать количеству строк в меню. Если один из адресов функций или адрес всего списка равны 0, то при нажатии на Enter ничего не произойдёт.
0	8	Стандартный вектор окна

Дальше начинаются различия между старым и новым векторами меню.
По старому стандарту:

8	n	Буфер (CSR-массив), длина которого равна количеству строк **n** в меню. Если значение равно 0, курсор по этой строке не ходит. Если значение равно 1, курсор может перемещаться по этой строке.
8+n		Далее идёт текст меню. Формат - как у обычного окна (см. рестарт #66)

По новому стандарту:

8	2	Адрес CSR-массива-1
10	2	Адрес текста

Как видно, по новому стандарту вектора меню текст и CSR-массив могут располагаться отдельно от вектора меню.

Функции по адресам -2, -4 могут управлять дальнейшими событиями флагом Z/NZ и содержимым регистра A: Z, A=0: ничего не делать,
A>0 – выполнить ту строку в меню, номер которой будет в регистре A.
NZ: A=#FF - выйти из меню,
A=1 - вызвать процедуру инициализации.
Если на выходе функция устанавливает флаг CF, то происходит выход из меню с сохранением регистра A. Это удобно при экстренном выходе с возвратом кода ошибки.

#92 (146) $funct

Выполнение функций-утилит по текстовому файлу.
Вход: HL=путь файла,
A=код функции.
Используется оболочкой для отработки нажатых клавиш (файл S:SHELL\extkey.txt). Среда сохраняется. Для обозначения служебных символов используется десятичное число, начинающееся с одиночной кавычки. Так сама кавычка будет выглядеть как '39. путь и ключи функции отделяются пробелом (см. extkey.hlp).

#93 (147) $shexe

Выход в оболочку с запуском RST $exebat(#44) с A=0 (см. **ПРИЛОЖЕНИЕ 17**).