HDD_boot

Дизассемблирование и комментарии к ассемблерному тексту - © Prusak, 2002, http://zxbyte.ru
Работа выполнена на основе оригинального кода загрузчика из ПЗУ компьютера KAY-1024.

Программа HDD_boot является загрузчиком системы, который в оригинальном ПЗУ для компьютера KAY находится в области ПЗУ TR-DOS, откуда может вызываться для запуска системы с винчестера. Загрузчик копируется из ПЗУ в память по адресу #5E00 и запускается на выполнение по этому же адресу. В процессе работы загрузчик сканирует на винчестере сектор 0, головку 0 треки от 0 до 255 в поисках признака "KAY" по смещению #7D от начала блока. После этих трёх букв по смещению #80 находится описатель файла uni_boot.sys, который загружается в память и запускается на выполнение, откуда уже непосредственно запускается система iS-DOS.

Дизассемблер загрузчика:

```
        ORG #5E00

;Перед запуском загрузчика необходимо запретить прерывания!!!

;2 точки входа в загрузчик:
;#5E00 - сканирование винчестера в поисках загрузчика uni_boot.sys
;#5E03 - процедура чтения с винчестера

        JP SCAN     ;#5E00
        JP READ     ;#5E03

;тут находится кусок заголовка драйвера винчестера, где расположены параметры самого винчестера
;изначально тут ничего нет. После выполнения сканирования винчестера сюда будет перенесена информация
;о винчестере

HEAD    DEFB 0      ;количество головок
SEC     DEFB 0      ;количество секторов
        DEFW 0      ;не используется
HS2     DEFW 0      ;HEAD·SEC·2
        DEFW 0      ;не используется
TRK     DEFW 0      ;начальный трек текущего устройства
SDH     DEFB #A0    ;маска для регистра SDH винчестера (#B0 - для чтения с устройства Slave)

;процедура RESET осуществляет программный сброс винчестера и его рекалибровку

        IN A,(#F0)          ;назначение этого куска программы неясно, т.к. его не вызывает ни одна
        CP #50              ;процедура. Скорее всего, этот кусок кода был началом процедуры RESET.
        RET Z

RESET   LD A,(SDH)          ;маска SDH
        OUT (#D0),A         ;заносим в SDH-регистр
        LD A,%00001100      ;устанавливаем бит SRST - бит программного сброса накопителя
        OUT (#C8),A         ;заносим в Fixed disk register
        LD B,0              ;делаем паузу
        DJNZ $
        LD A,%00001000      ;сбрасываем бит SRST
        OUT (#C8),A
RES1    IN A,(#F0)          ;проверка готовности накопителя
        RLCA                ;проверяем бит BSY
        JR C,RES1
        LD A,#10            ;команда Recalibrate
        OUT (#F0),A
RES2    IN A,(#F0)          ;ждём готовности после рекалибровки
        BIT 7,A
        JR NZ,RES2
        CP #50              ;проверка на нормальное завершение
        RET Z
        JR RESET            ;если что-то было не так, то повторяем процедуру сброса
```

```
;процедура READ осуществляет чтение блоков с винчестера
;Вход:   HL - куда читать блоки
         DE - с какого блока читать
         B - сколько блоков читать
         в TRK - номер трека, от которого будет вестись отсчёт
Выход:  C - ошибка
         NC - ОК

READ     CALL CALC      ;вычисляем номер трека и головки
         JR C,READ3     ;если читаем один блок с начала сектора
READ1    CALL C_SEC     ;высчитываем, сколько секторов на треке надо прочитать и откуда читать
         PUSH BC        ;количество секторов за пределами трека, которые ещё надо прочитать
         CALL MLTRD     ;читаем на текущем треке нужные сектора
         POP BC
         RET C          ;по ошибке выходим
         LD A,B         ;если больше не надо читать сектора
         OR A
         JR Z,READ2     ;то переход
         CALL SET       ;переход на следующую головку и трек
         JR READ1       ;читаем снова

READ2    CALL CHECK     ;проверяем, не надо ли дочитывать ещё полсектора
         RET Z          ;не надо

;дочитываем полсектора

READ3    LD A,E         ;номер текущего сектора
         INC A          ;в HDD нумерация секторов начинается с 1
         OUT (#70),A    ;заносим в регистр сектора
         LD A,1         ;читаем 1 сектор
         OUT (#50),A    ;Sector counter
         LD A,#21       ;команда Read sector(s)
         OUT (#F0),A    ;Command register
         CALL READY     ;ожидаем готовности накопителя
         RET C          ;если была ошибка
         LD B,0         ;256 байт
READ4    LD C,#10       ;читаем
         INI
         INC C
         INI
         JR NZ,READ4
READ5    IN A,(#10)     ;остальные 256 байт, которые нам не надо, читаем "вхолостую"
         IN A,(#F0)
         BIT 3,A        ;пока DRQ не сбросится в 0
         JR NZ,READ5
         RRCA           ;бит ERR загоняем в флаг C
         RET            ;выходим
```

```
;процедура CALC выполняет подсчёт номера трека, номера головки.
Вход:   DE - номер блока, с которого надо читать
        B - сколько блоков надо читать
Выход   флаг C - читаем 1 блок (256 байт) с начала сектора
                 E - номер сектора винчестера, с которого надо начать чтение
                 A - то же самое
        флаг NC - A≠#FF  1) читаем чётное количество блоков с начала сектора
                         2) читаем нечётное количество блоков со второй половины сектора
                         E - номер блока (256 байт) с начала дорожки, с которого надо читать
                         A - номер сектора винчестера, с которого надо начать чтение
                         в этом случае B - реальное количество целых секторов, которое надо почитать
                   A=#FF 1) читаем нечётное количество блоков с середины сектора
                         2) читаем чётное количество блоков с начала сектора
                         E - номер блока (256 байт) с начала дорожки, с которого надо читать
                         в этом случае B - количество секторов, которые надо прочитать (без последней половинки)
           HL - сохраняется

CALC       PUSH DE          ;нам надо номер блока
           EXX              ;остальное сохраняем
           POP DE
           LD HL,(HS2)      ;Block/(HEAD·SEC·2)=номер трека
           EX DE,HL
           XOR A
           LD B,A
           LD C,A
CALC1      SBC HL,DE        ;делим
           INC BC
           JR NC,CALC1
           DEC BC           ;BC - номер трека
           ADD HL,DE        ;HL - остаток от деления
           LD A,(SEC)
           RLCA
           LD E,A
           XOR A
           LD D,A           ;DE - количество секторов на дорожку
CALC2      SBC HL,DE        ;остаток делим на DE
           INC A
           JR NC,CALC2
           DEC A            ;номер головки
           ADD HL,DE
           EX DE,HL         ;DE - остаток от деления
           LD D,A           ;номер головки
           LD HL,(TRK)
           ADD HL,BC        ;суммируем с начальным треком текущего устройства
           LD A,E           ;номер блока (256 байт) от начала дорожки
           EX AF,AF'
           CALL SET1        ;заносим номер дорожки и головки в регистры винчестера
           EX AF,AF'
           SRL B            ;проверка B на 1 (а заодно и делим на 2 - получаем сколько надо прочитать
                            ;целых секторов на винчестере
           LD E,A
           JR Z,CALC7       ;если надо прочитать 1 блок, то переход
           JR C,CALC6       ;если надо прочитать нечётное количество блоков
           RRCA             ;делим номер блока на 2 (проверка на чётность)
           JR C,CALC4       ;если номер блока был нечётным, т.е. читаем со второй половины сектора
CALC3      LD A,#FF         ;тот случай, когда читаем чётное количество блоков с начала сектора
CALC4      LD (CHECK+1),A   ;установка признака чтения
CALC5      OR A             ;установка NC
           RET
```

```
;обработка случая, когда читаем нечётное количество блоков

CALC6   RRCA        ;делим номер блока от начала дорожки на 2 (проверка на чётность)
        JR NC,CALC4 ;если читаем нечётное количество блоков от начала сектора
        INC B       ;тут получаем целое количество секторов, которое надо прочитать
        JR CALC3    ;установка признака #FF

;если читаем только один блок

CALC7   INC B       ;после деления у нас B=0, поэтому увеличиваем B до 1 сектора
        RRCA        ;проверка на чётность читаемого блока (заодно делим номер блока на 2)
        JR C,CALC3  ;читаем с середины сектора

;тут читаем 1 блок с начала сектора

        LD E,A      ;номер сектора, с которого читаем
        SCF         ;установка признака того, что читаем 1 блок с начала сектора
        RET

;процедура C_SEC определяет, откуда читать очередной блок - с начала сектора или со второй его половины. Также процедура определяет, сколько секторов нужно прочитать до конца трека.
;Вход:  E - номер блока длиной 256 байт (с начала трека), откуда надо начать чтение
        B - сколько секторов (512 байт) надо прочитать
;Выход  C - начинать чтение со второй половины сектора
        NC - начинать чтение с начала сектора
        A - с какого сектора начинать чтение
        E - сколько секторов до конца дорожки надо прочитать
        B - сколько секторов останется прочитать после этого трека
;процедура, исходя из номера блока, с которого надо начинать чтение и количества секторов для чтения определяет, сколько секторов до конца дорожки можно прочитать и сколько секторов останется ещё прочитать после этой дорожки. Дело в том, что файл читается блоками по несколько секторов (в пределах дорожки).

C_SEC   SRL E       ;делим номер блока на 2 - получаем номер сектора, с которого надо читать
        LD A,E      ;если блок чётный (начало сектора), то получаем NC, если блок нечётный
                    ;(вторая половина сектора), то получаем C
        PUSH AF     ;сохраняем флаговый регистр
        LD A,(SEC)  ;количество секторов на дорожку
        SUB E       ;сколько секторов еще осталось на треке
        LD E,A
        LD A,B      ;сколько секторов надо прочитать
        SUB E       ;сколько секторов надо прочитать за пределами трека
        JR NC,C_SEC1 ;если читать секторов надо больше, чем осталось на треке

;тот случай, когда все сектора, которые надо прочитать, находятся на одном треке

        XOR A       ;за пределами трека ничего не надо читать (0 сектров)
        LD E,B      ;сколько надо прочитать секторов
C_SEC1  LD B,A      ;сколько секторов надо прочитать за пределами трека
        POP AF      ;восстанавливаем флаг C
        RET

;процедура SET осуществляет переход на следующий трек и номер головки (в том случае, если все сектора на треке уже прочитаны)
;Вход:  D' - текущий номер головки
        HL' - номер трека

SET     LD E,0      ;через передаётся номер блока на винчестере, а так как на треке все блоки уже
                    ;прочитаны, то переходим на новый трек, а нумерация блоков на треке начинается
                    ;с нуля
        EXX         ;в альтернативном наборе хранятся номер трека и номер головки
        INC D       ;следующая головка
        LD A,(HEAD) ;количество головок винчестера
        CP D        ;все головки на треке?
        JR NZ,SET1  ;если не все, то устанавливаем новый номер головки, ничего больше не меняя
```

```
        LD D,0          ;головка 0
        INC HL          ;увеличиваем номер трека на 1
SET1    IN A,(#F0)      ;ожидаем готовности винчестера
        CP #50
        JR NZ,SET1
        LD A,L          ;заносим номер трека в регистры винчестера
        OUT (#90),A     ;регистр Cylinder low
        LD A,H
        OUT (#B0),A     ;регистр Cyinder high
        LD A,(SDH)      ;маска SDH-регистра
        OR D            ;накладываем на номер головки
        OUT (#D0),A     ;заносим в SDH-регистр
        EXX             ;все параметры устанавливаем в альтернативный набор
        RET
```

;Процедура **CHECK** проверяет, надо ли в конце файла читать только полсектора, а вторую половину пропустить.
;Выход: Z - не надо ничего дочитывать
NZ - надо дочитать ещё полсектора, при этом процедура сама переходит на тот сектор, половину которого надо дочитать.

```
CHECK   LD A,0          ;сюда записывается признак того, надо ли дочитывать полсектора
        CP #FF          ;если было #FF, от не надо
        RET Z           ;если ничего не надо дочитывать, то выход с флагом Z
        IN A,(#70)      ;текущий сектор
        LD E,A
        LD A,(SEC)      ;количество секторов на трек
        CP E            ;текущий сектор является последним?
        JR Z,SET        ;если да, то переходим на следующий сектор с помощью программы SET
        RET             ;выход с флагом NZ
```

;процедура **MLTRD** загружает с винчестера E секторов.
;Вход: A - с какого сектора грузить (нумерация начинается с 0)
E - сколько секторов грузить (если у последнего сектора надо читать только половинку, то длина даётся без последнего сектора)
HL - куда грузить сектора
NC - начинать с начала сектора
C - начинать со второй половины сектора
;Выход: C - ошибка чтения с винчестера
;процедура грузит сектора только на текущей дорожке

```
MLTRD   PUSH AF         ;сохраняем флаг C
        INC A           ;увеличиваем номер сектора на 1, потому что у HDD сектора нумеруются с 1
        OUT (#70),A     ;заносим номер сектора в Sector number
        POP AF          ;восстанавливаем флаг C
        LD A,E          ;сколько секторов надо читать
        OUT (#50),A     ;заносим в Sector count
        LD A,#21        ;команда Read sector(s)
        OUT (#F0),A     ;заносим в Command register
        JR NC,MLT3      ;если начинаем чтение с начала сектора, то переход на чтение 512 байт
```

;Тут надо читать только вторую половину сектора, поэтому первые 256 байт сектора читаем "вхолостую"

```
        CALL READY      ;ожидаем готовности накопителя
        RET C           ;по ошибке отваливаем с флагом C
        LD B,#80        ;128 раз будем читать порт #10 - это равнозначно прочтению 256 байт
                        ;по чтению из порта #10 передаются два байта - один непосредственно в порт #10,
                        ;а второй - при чтении из порта #11, но его можно и не читать
MLT1    IN A,(#10)      ;читаем 128 раз порт #10
        DJNZ MLT1
MLT2    LD C,#10        ;а тут читаем 256 байт второй половинки сектора
        INI             ;IN (#10)
        INC C
        INI             ;IN (#11)
        JR NZ,MLT2      ;если 512 байт не посчитаны, то читаем ещё
        IN A,(#50)      ;все сектора почитали?
```

```
        OR A            ;если да, то выход
        RET Z

;тут читаем весь сектор целиком

MLT3    CALL READY      ;ожидаем готовность накопителя
        RET C           ;по ошибке отваливаем
        LD C,#10
MLT4    INI             ;читаем
        INC C
        INI
        DEC C
        IN A,(#F0)      ;читаем до тех пор, пока винчестер не снимет сигнал DRQ, что означает, что он
        BIT 3,A         ;передал все данные
        JR NZ,MLT4
        IN A,(#50)      ;ещё надо читать сектора?
        OR A
        JR NZ,MLT3      ;если надо, то читаем снова
        RET
```

;процедура **READY** ожидает готовности винчестера. В случае ошибки возвращается с флагом C и кодом ошибки - 7 в регистре A

```
READY   IN A,(#F0)      ;читаем Status register
        BIT 7,A         ;проверка на BSY=0
        JR NZ,READY     ;иначе ждём
        BIT 3,A         ;поверка на DRQ=1
        JR Z,READY      ;иначе ждём
        RRCA            ;проверка на ERR=0
        RET NC          ;если так, то выход с NC
        CALL RESET      ;если были ошибки, то пытаемся сбросить винчестер
        LD A,7          ;код ошибки - 7
        SCF             ;установка флага C
        RET
```

;процедура **SCAN** осуществляет сканирование треков винчестера с номерами [0-255] в поисках загрузчика uni_boot.sys
;в случае обнаружения такового он загружается в память и запускается на выполнение, иначе сканирование продолжается. В случае завершения сканирования (все треки просканированы) процедура "зависает"

```
SCAN    LD SP,#5DC0
        CALL CALC5      ;назначение этого вызова непонятно
SCAN1   LD A,1          ;в процессе работы меняется цвет бордюра, начинаем с синего (просто как
```
правило, загрузчик находится на системном устройстве, а оно в свою очередь - на треке 0 винчестера. В данном случае трек 0 будет найден первым, а система iS-DOS имеет синий цвет оболочки. Так что сразу же установится синий цвет при загрузке системы)
```
        OUT (#FE),A
        CALL ANALYS     ;загрузка трека в память и его анализ
```

;процедура ANALYS в случае обнаружения на треке загрузчика uni_boot.sys запускает его на выполнение, поэтому тот факт, что она отдала управление обратно, говорит о том, что или загрузчик отсутствует или была какая-либо ошибка, поэтому продолжаем сканирование со следующего трека винчестера

```
        LD HL,SCAN1+1   ;меняем цвет бордюра
        INC (HL)
        LD HL,TRK       ;номер трека для сканирования увеличиваем на 1
        INC (HL)
        JR NZ,SCAN1     ;если не было переполнения больше 255, то сканируем снова
        HALT            ;255 треков просканировано и ничего не найдено, поэтому "виснем"
```

;процедура **ANALYS** загружает трек с номером в TRK в память и производит его анализ на наличие на треке
;загрузчика uni_boot.sys. Если таковой имеется, то он запускается на выполнение

```
ANALYS  LD HL,BUFER     ;грузим в буфер 0-й блок на треке
        LD B,1
```

```
        LD DE,0
        CALL READ
        RET C                   ;была ошибка при загрузке, поэтому отваливаем обратно

        LD HL,(BUFER+#7D) ;проверка на наличие слова "KAY" по смещению #7D
        LD DE,#414B             ;"KA"
        LD A,(BUFER+#7F)
        XOR #59                 ;"Y"
        RET NZ                  ;последняя буква не "Y", поэтому выход
        SBC HL,DE
        RET NZ                  ;первые две буквы не совпали, поэтому выход
```

;раз слово 'KAY" обнаружено, то можно считать, что этот трек подходит нам.
;теперь из буфера надо взять информацию о конфигурации винчестера, которая была записана туда программой ;uni_con.com и перенести ее в загрузчик. Это надо для дальнейшей работы процедуры READ для загрузки файлов. Все значения проверяются на 0. Это сделано потому, что может быть это не тот блок, который нам надо. Вдруг на устройстве есть ещё один такой блок с текстом "KAY" по смещению #7D, но без всех остальных данных. В этом случае есть вероятность, что по одному из нужных адресов будет 0. Вот это и проверяем.

```
        LD HL,(BUFER+97)  ;данные о количестве головок винчестера хранятся в буфере
        LD A,L                  ;прочитали информацию о HEAD и SEC
        OR H                    ;проверка их значения на 0
        RET Z
        LD (HEAD),HL            ;заносим в сам загрузчик
        LD HL,(BUFER+115) ;HEAD·SEC·2
        LD A,L                  ;проверка на 0
        OR H
        RET Z
        LD (HS2),HL             ;сохраняем в загрузчике
```

;Теперь берём всю нужную информацию о файле uni_boot.sys и загружаем его в память, после чего запускаем на ;выполнение. Вся информация хранится в описателе файла uni_boot.sys по смещению #80 от начала буфера.

```
        LD DE,(BUFER+#80+17)  ;номер блока, с которого будем грузить uni_boot.sys
        LD A,E                  ;проверка на 0
        OR D
        RET Z
        LD HL,(BUFER+#80+12)  ;адрес загрузки
        LD A,L
        OR H
        RET Z
        LD A,(BUFER+#80+15)   ;старший байт длины файла
        OR A
        RET Z
        INC A                   ;увеличиваем на 1 - получаем длину в блоках
        LD B,A
        PUSH HL                 ;сохраняем стартовый адрес файла, чтобы потом запустить его на
                                ;выполнение
        CALL READ               ;читаем файл uni_boot.sys
        RET NC                  ;если не было ошибок, то делаем переход на начало файла
        POP HL                  ;возвращаем на место HL, потому что была ошибка
        RET                     ;выход

BUFER   EQU #6000
```