Recoded by Evgeny Muchkin 06.10.1998
SysOp of PALLY_STATION tel: 176-74-19

Translation of this document is
 16th of September, 2001

Write me: iliks@chat.ru
with any comments, translation work offers etc

(á) STINGER &
(á) CYRAX- (*)

General Sound Programming Manual

Version v1.04. Edition 004.

1. Overview of technical characteristics of GS

CPU : Z80, 12MHz, without wait cycles
ROM : 32k, 27256
RAM : Static Ram 128k total, 112k is available for modules and samples in base version
INT : 37.5 KHz Channels: 4 independent 8-bit channels, every channel has 6-bit volume controller.

2. Short overview of GS

GS is a soundcard, designed for playback of musical modules and individual samples (sound effects).

GS modules are standard 4-channels MOD modules from Amiga and PC.
GS samples - both Amiga signed samples and PC unsigned samples.

MOD files player in GS is practically a full analogue of ProTracker on Amiga, and was designed with intensive use of ProTracker v2.1A sources.

It supports the whole list of ProTracker`s commands except two ones:

E01 Filter on This command is Amiga-specific, enables low-pass filter.

EFX Invert Loop. I haven't seen a player, who supports this command. It's possible, however, that it is supported in some old players.

GS is, in fact, microprocessor complex with its own CPU, ROM, RAM and IO ports, and it is absolutely independent from Spectrum`s CPU, what makes you able to, for example, load your favourite module, reset Spectrum, load assembler and to write masterpieces with your favourite music in the background. Software in GS ROM plays module invisibly for you. Programming for GS could be squeezed to just several commands, as: Play module, Set global volume, run sample #09 in channel #02 etc.

If you would like to load module with samples, then you should load module itself first, and only after that - samples.

While loading of module it's highly recommended to keep 2k of RAM free, ie to load modules with maximal length of 110k. This condition isn't necessary, but its usage is very recommended, with the aim to supply further compatibility.

Also it's recommended to keep 80 bytes for each sample free, for example

if you want to load 63kb module and 18 samples, then the whole amount of samples, which could be loaded is:

Total_Sample_Length=112*1024-63*1024-2*1024-18*80=46688 ¡ ©â

If you want to calculate the amount of 2k samples, which could be stored in GS memory, then it should be calculated the following way:

112*1024/(2048+80)=53 samples.

GS has 4 physical channels, which play sound.

Left channels: 0 and 1; Right channels: 2 and 3.

3. Interface with Spectrum

For communication with the rest of the world, GS has 4 registers:

1. Command register - it's - right - register of commands, it is port with address 187 (#BB), which is write-enabled. In this port you should write your commands to GS.

2. Status register - read-enabled port with address equal to 187 (#BB)

Bits of the register

7 - Data bit, flag of data
6 - undefined
5 - undefined
4 - undefined
3 - undefined
2 - undefined
1 - undefined
0 - Command bit, flag of commands

This register allows you to determine the GS status, in particular you could know, whether you are able to read or write next data byte, or to give next command etc.

3. Data register - port 179 (#B3), available for writing. In this port Spectrum writes data, for example, arguments of commands.

4. Output register - port 179 (#B3), available for reading. Spectrum read data from GS from this register.

Command bit in status register is set by hardware after current command has been stored in commands` register. Only GS could reset it to 0, which is a signal about the current stage of processing of a command.

Data bit in status register could be set or reset either by Spectrum's will, or by GS` will: it is set to 1 by hardware when Spectrum writes in data register, and it is reset to 0 after GS has read data from data register. When GS writes in output register, data bit is set to 1 by hardware. And after reading by Spectrum from this port, the bit is reset by hardware to 0.

In spite of data register and output register are situated by the only port address, they are two independent registers. Value, which has been stored in one of this registers stays unchanged until new record.

The value of data bit is undefined very often, and it is impossible to make any guesses about its value.

4. System of GS commands

At first, little comment. In the current version (1.03) of GS ROM it's possible to load one module and/or up to 32 samples.

Every sample gets his unique identifier, which is needed in commands which require sample number. First sample gets handle 1, next - 2, etc.

The same is for modules, and this one loaded module will have handle=1 after loading.

The speciality of this version is that you should load module first, and only then samples.

Specialities of commands explanations:

They are defined as follows:

1. Hex-code of command
2. Name of command
3. Actions which take place after command
4. Command's format.
5. Comments

Command's formate is defined this way:

```
GSCOM EQU 187
GSDAT EQU 179
```

```
SC #NN    : Send command code into commands register

          LD A,#NN
          OUT (GSCOM),A

WC      : Wait for reset of Command bit

     WCLP IN A,(GSCOM)
          RRCA
          JR C,WCLP

SD Data : Send data into data register

          LD A,Data
          OUT (GSDAT),A

WD      : Wait for reset of Data bit, ie wait while GS receives its data

     WDLP IN A,(GSCOM)
          RLCA
          JR C,WDLP

GD Data : Get data from data register

          IN A,(GSDAT)

WN      : Wait for 1 in Data bit, ie wait until GS send you any data

     WNLP IN A,(GSCOM)
          RLCA
          JR NC,WNLP
```

GS commands:

#00 Reset flags
Resets Data bit and Command bit flags

```
SC #00
WC
```

(Data bit=0, Command bit=0)

#01 Set silence (*)
Outputs #80 in DACs of all four channels. Sets silence.

```
SC #01
WC
```

#02 Set low volume (*)
Sets the volume of all DACs in null.

```
SC #02
WC
```

#03 Set high volume (*)
Sets the maximum volume of all DACs.

```
SC #03
WC
```

#04 Set 'E' 3bits (*)
Sets 3 lower bits in 'E' register of GS with the specified value (2 lower bits are number of channel #00-#03).

```
SD Chan (#00-#07)
SC #04
WC
```

#05 Out volume port (*)
Sets the volume of the channel, which number is stored in 'E', to specified value. (Works only when 'E' contains numbers from #00 to #03)

```
SD Volume (#00-#3F)
SC #05
WC
```

#06 Send to DAC (*)
Outputs specified byte in DAC of channel, specified by 'E'.

```
SD Byte
SC #06
WC
```

#07 Send to DAC and to volume port (*)
Outputs byte in DAC ('E') with specified volume.

```
SD Byte
SC #07
WC
SD Volume
WD
```

#08 - The same as #00 command

#09 Sets one's byte volume. (*)
    Sets the volume of channel, which number is specified in 2 higher bits.

    SD Byte (ccvvvvvv)
    SC #09
    WC

    cc - channel number
    vvvvvv - volume

#0A DAC output (*)
    One more intermediate output to DAC.

    SD Byte
    SC #0A
    WC
    SD Chan (#00-#03)
    WD

#0B DAC and Volume output (*)
    One more output to DAC with specified volume

    SD Fbyte
    SC #0B
    WC
    SD Sbyte (ccvvvvvv)
    WD

    The maning of Sbyte is the same as in command #09

    Commands #01 - #0B are, in general, for building of various Covox'es and
    players, without going in depths.

    vvvvvv - its volume.

#0C DAC output (*)
    One more output to DAC.

#0C Call SounDrive Covox mode (*)
    Calls 4-channel Covox mode. Stores, in serial way, the data register to
    all channels. Exits from this mode right after output of 4th byte.

    SD CH1
    SC #0C
    WC
    SD CH2
    WD
    SD CH3
    WD
    SD CH4
    WD

#0D Call Ultravox mode (*)
    Calls mode of universal Covox, sequentially stores data register to
    channels, the number of which is controlled (1-4). Synchronization, unli-
    ke previous command, isn't made. Also exits after sending of 4th byte.

    SD CHANS
    SC #0D
    WC

    SD CH1
    SD CH2
    SD CH3

```
    SD CH4

    CHANS (4 lower bits) shows which channels will be enabled - set the corre-
    sponding bit for each channel. If the channel is off, then next sent byte
    goes to next enabled channel.

#0E Go to LPT Covox mode
    Sets 1-channel Covox mode. Stores data register into DACs of left and
    right channels. Store #00 into commands register to exit from this mode.

    SC #0E
    WC

    SD  \
    SD   \
    ...   Output in DACs
        /
    SD  /

    SC #00
    WC

#0F Go in Profi Covox mode (*)
    Sets the two-channel Covox mode. Stores data register into DAC of one
    channel, and commands register - into DAC of second channel. To exit
    from this mode, send #4E into data register, then (sequentially)
    #0F and #AA into commands register.

    SD #59
    SC #0F
    WC

    SD  \
    SC   \
    SD    \
    SC      output in DACs
    ...   /
    SD   /
    SC  /

    SD #4E
    WD
    SC #0F
    WC
    SC #AA
    WC

#10 Out to any port (*)
    Outputs byte into internal port of GS (#00-#09)

    SD Port
    SC #10
    WC
    SD Data
    WD

#11 In from any port (*)
    Reads byte from internal port of GS (#00-#09)

    SD Port
    SC #10
    WC
    GD Data
    WN
```

```
#12 OUT to 0 port (*)
    Outputs byte into configuration port of GS (#00)

    SD Data
    SC #12
    WC

#13 Jump to Address (*)
    Goto's to specified address.

    SD ADR.L
    SC #13
    WC
    SD ADR.H
    WD

#14 Load memory block (*)
    Load memory block from specified address with specified length.

    SD LEN.L
    SC #14
    SD LEN.H
    WD
    SD ADR.L
    WD
    SD ADR.H

    SD \
    WD  \
    SD   \
    WD     Block of data with the length LEN
    ...  /
    SD  /
    WD /

#15 Get memory block (*)
    Gets memory block from specified address with specified length.

    SD LEN.L
    SC #15
    SD LEN.H
    WD
    SD ADR.L
    WD
    SD ADR.H

    GD \
    WN  \
    GD   \
    WN     Data block with length LEN.
    ...  /
    GD  /
    WN /

#16 Poke to address (*)
    Saves one byte to specified address

    SD ADR.L
    SC #16
    WC
    SD ADR.H
    WD
    SD Byte
```

```
    WD

#17 Peek from address (*)
    Reads one byte from specified address

    SD ADR.L
    SC #17
    WC
    SD ADR.H
    WD
    GD Byte
    WN

#18 Load DE Pair (*)
    Loads specified word into DE register (inside the GS, don't mess up with
    Main CPU)

    SD B.E
    SC #18
    WC
    SD B.D
    WD

#19 Poke to (DE) address (*)
    Saves byte to address, specified by DE.

    SD Byte
    SC #19
    WC

#1A Peek from (DE) address (*)
    Reads value from address, specified by DE.

    SC #1A
    WC
    GD Byte
    WN

#1B Increment of DE Pair (*)
    Increments DE by 1.

    SC #1B
    WC

#1C Poke to (#20XX) address (*)
    Stores specified byte to address, higher byte of which is equal to #20

    SD ADR.L
    SC #1C
    WC
    SD Byte
    WD

#1D Peek from (#20XX) address (*)
    Reads byte from address, higher byte of which is equal to #20.

    SD ADR.L
    SC #1D
    WC
    GD Byte
    WN

#1E - #1F Reserved
#F1 - #F2 Reserved
```

#F3 Warm restart
    Resets GS, but skips testing of memory, which speeds up initialization.

    SC #F3
    WC

#F4 Cold restart
    Complete restart of GS with all tests. By other words - JP #0000

    SC #F4
    WC

#F5 Busy on
    Sets busy flag in #FF

    SC #F5
    WC

#F6 Busy off
    Sets busy flag in #00

    SC #F6
    WC

   From the beginning Busy=#00. Execution of all GS commands is performed in the main cycle of commands interpreter. This cycle could be shows as:

 1 if Command bit=0 then go to 1
 2 Execute Command
 3 if Command bit=1 then go to 2
 4 if Playing=0 then go to 1
 5 if Busy=#FF then go to 1
 6 Process Sound
 7 go to 1

  With Busy command you could initialize, for example, playback of samples in all channels, then change the playback parameters, and then run it simultaneously. Without Busy command the following situation is possible: first sample is initialised (sample will start to play, but only then second sample will be initialized etc)

#F7 Get HX Register (*)
    Get value of HX (GS) register.
    HX participates in determination of Busy flag

    SC #F7
    WC
    GD HX
    WN

#F8 - #F9 Reserved/

#FA Out zero_to_zero
    Stores null into #00 (configuration) port of GS. Stops music until next read from any port.

    SC #FA

    WC

#FB - #FF Reserved

#20 Get total RAM

```
    Get amount of RAM in GS (112k in base version)

    SC #20
    WC
    GD RAM.L (low part)
    WN
    GD RAM.M (middle part)
    WN
    GD RAM.H (high part)

Total RAM=65536*RAM.H+256*RAM.M+RAM.L

#21 Get free RAM
    Get amount of free GS RAM

    SC #20
    WC
    GD RAM.L (low part)
    WN
    GD RAM.M (middle)
    WN
    GD RAM.H (high part)

 Free_RAM=65536*RAM.H+256*RAM.M+RAM.L

#23 Get number of GS RAM Pages

    SC #23
    WC
    GD Number_RAM_Pages

    There are 3 pages in base version.

#24 - #29 Reserved

#2A Set Module Master Volume

    SD Module_Master_Volume [#00..#40]
    SC #2A
    WC
    [GD Old_Master_Volume] - old volume

    Little example:

    (suppose module is being played)

           LD B,#40

     LOOP: LD A,B
           OUT (GSDAT),A
           LD A,#2A
           OUT (GSCOM),A
           EI
           HALT
           DJNZ LOOP

           LD A,#32
           OUT (GSCOM),A

  This slightly lowers the volume of module, and then stops it.

#2B Set FX Master Volume

    SD FX_Master_Volume [#00..#40]
```

```
    SC #2B
    WC
    [GD Old_FX_Volume] - old volume
```

Analogue of the previous command, but affects samples playback

You could control balance of volumes of samples and module with this two commands.

#2E Set Current FX

Loads specified number into CURFX variable. If some commands requires number of sample (sample handle), then you could give it #00 and interpreter will change this with the CURFX value. (look commands #38-#4F)

```
    SD Cur_FX
    SC #2E
    WC
```

#30 Load Module in memory

```
    SC #30
    WC
   [GD Module_Handle] - number of module
   (Command bit=0, Data bit=0)
    SC #D1 (Open Stream)
    WC

    SD \
    WD  \
    ...   bytes of module
    SD  /
    WD /

    SC #D2 (Close Stream)
    WC

    Example:
             LD HL,Mod_adress
             LD DE,0-Mod_length
             LD C,GSCOM

             LD A,#30
             CALL SENDCOM
             LD A,#D1
             CALL SENDCOM

             LD A,(HL)
     LOOP:   IN B,(C)
             JP P,READY
             IN B,(C)
             JP M,LOOP
     READY:  OUT (GSDAT),A
             INC HL
             LD A,(HL)
             INC E
             JP NZ,LOOP
             INC D
             JP NZ,LOOP
     WAIT:   IN B,(C)  ;wait for end of receiving
             JP M,WAIT ;of the last byte
             LD A,#D2
             CALL SENDCOM
             IN A,(GSDAT) ; number of module
```

```
            OUT (GSDAT),A
            LD A,#31

    SENDCOM: OUT (GSCOM),A
    WAITCOM: IN A,(GSCOM)
            RRCA
            JR C,WAITCOM
            RET

#31 Play module

    SD Module_Handle - number of module
    SC #31
    WC

#32 Stop module

    SC #32
    WC

#33 Continue module
    Continue playback after pause.

    SC #33
    WC

#35 Set Module Volume

    SD Module_Master_Volume [#00..#40]
    SC #35
    WC
   [GD Old_Master_Volume] - old volume

#36 Data on (*)
    Stores #FF into data register

    SC #36
    WC
   [GD Data (#FF) ]

#37 Reinitialisation (*)
    Restores default values into internal variables.

    SC #37
    WC

#38 Load FX
    Loads sample in memory. Uses unsigned samples (PC type)

    SC #38
    WC
   [GD FX_Handle] -  sample number
   (Command bit=0, Data bit=0)
    SC #D1 (Open Stream)
    WC

    SD \
    WD  \
    ...   sample bytes
    SD  /
    WD /

    SC #D2 (Close Stream)
    WC
```

After loading of every sample, GS creates header for this sample. This parameters are set in the following values:
Note=60, Volume=#40, FineTune=0, SeekFirst=#0F, SeekLast=#0F, Priority=#80, No Loop, CurFX=FX_Handle.

After this you could enter your own values, by commands #40-42, #45-#47. This is needed because command #39 uses header values for initalization of sample playback.

In their original form samples are usually packed bad, but one can improve compression ratio by transforming them into delta-form, ie not to store absolute sample values, but relative. Example of transformation:

```
            LD HL,Start_of_sample
            LD DE,0-Length_of_sample
            LD C,#00

      LOOP: LD A,(HL)
            SUB C
            LD C,(HL)
            LD (HL),A
            INC E
            JP NZ,LOOP
            INC D
            JP NZ,LOOP
```

And here's how to load such sample:

```
             LD IX,Parameters
             LD HL,Sample_adress
             LD DE,0-Sample_length
             LD C,GSCOM

             LD A,#38
             CALL SENDCOM
             LD A,#D1
             CALL SENDCOM

             LD A,(HL)
     LOOP:   IN B,(C)
             JP P,READY
             IN B,(C)
             JP M,LOOP
     READY:  OUT (GSDAT),A
             INC HL
             ADD A,(HL)
             INC E
             JP NZ,LOOP
             INC D
             JP NZ,LOOP
     WAIT:   IN B,(C)  ;wait for end of receiving
             JP M,WAIT ;of the last byte
             LD A,#D2
             CALL SENDCOM

      ; Now let's define sample parameters

             LD A,(IX+#00)
             OUT (GSDAT),A  ; note
             LD A,#40
             CALL SENDCOM
             LD A,(IX+#01)
             OUT (GSDAT),A  ; volume
```

```
          LD A,#41

SENDCOM: OUT (GSCOM), A
WAITCOM: IN A,(GSCOM)
          RRCA
          JR C,WAITCOM
          RET
```

#39 Play FX

```
SD FX_Handle - sample number
SC #39
WC
```

The way of how this command is performed: channels, which are specified in SeekFirst parameter of our sample, are being examined, and, if one of them is free, then sample is played in this channel, otherwise channels from SeekLast are being examined, and if one of them is fress, then sample is played in this channel. If there are no free channels, then all channels from SeekLast are being examined, and the channel with the most low priority is selected. Then it is compared with the priority of our sample, and if the priority of our sample is higher, then old sample will be stopped, and replaced with our sample.

In general, in order to play sample with needed parameters, you could set them after loading of sample and then use command #39. If parameters could be changed, then we can do the following: set the sample as current with the command #2E, and then change its parameters by commands #4x, and then run it with the command #39.

The alternate method of sample playback is provided with the commands #80-#9F, with these commands you specify the channel, where you want your sample to be played and also you could specify desired note or volume with which the sample should be played.

#3A Stop FX in channels

Stops effects playback in channels, specified in Channel Mask. In this mask 1 in bit #n shows that you want to stop effect in channel number n.

```
SD Channel_Mask
SC #3A
WC
```

Explained above is ideal variant, but unfortunately at the current moment the command doesn't work exactly this way: 1 in 7th bit stops sample in channel #0 etc. In the following version this will be corrected. Now I could only recommend you to stop all samples with #FF mask.

#3D Set FX Volume

```
SD FX Volume [#00..#40]
SC #3D
WC
[GD Old_FX_Volume] - old volume
```

#3E Load FX (Extended version)

Loads sample into memory. Uses signed samples (Amiga type)

```
SD #01 (Signed sample)
SC #3E
WC
[GD FX_Handle] - sample number
(Command bit=0, Data bit=0)
 SC #D1 (Open Stream)
```

```
    WC

    SD \
    WD  \
    ...   sample bytes
    SD  /
    WD /

    SC #D2 (Close Stream)
    WC

#40 Set FX Sample Playing Note
    Sets the default note for the current effect

    SD Note [0..95]
    SC #40
    WC

Note=
 0 C-0
 1 C#0
12 C-1
24 C-2
36 C-3 (C-1 in Amiga)
48 C-4 (C-2 in Amiga)
60 C-5 (C-3 in Amiga)
72 C-6
84 C-7

  In the current version of Sound Generators Wave 2, 3 could play octaves
3, 4 and 4, therefore Note parameter should lie in [36..71] range.

#41 Set FX Sample Volume
    Sets the default volume for the current effect

    SD FX_Volume [#00..#40]
    SC #41
    WC

#42 Set FX Sample Finetune for the current effect

    SD FX_Finetune [#00..#40]
    SC #42
    WC

#43 - #44 Reserved

#45 Set FX Sample Priority (look command #39)

    SD FX_Priority [#01..#fe]
    SC #45
    WC

#46 Set FX Sample Seek First parameter (look command #39)

    SD FX_SeekFirst
    SC #46
    WC

#47 Set FX Sample Seek Last parameter (look command #39)

    SD FX_SeekLast
    SC #47
    WC
```

#48 Set FX Sample Loop Begin for current effect (*)

```
    SD LEN.L
    SC #48
    WC
    SD LEN.M
    WD
    SD LEN.H
    WD
```

    When LEN.H is equal to #FF, then looping isn't made.

#49 Set FX Sample Loop End for the current effect (*)

```
    SD LEN.L
    SC #49
    WC
    SD LEN.M
    WD
    SD LEN.H
    WD
```

#4A - #4F Reserved

#51 - #5F Reserved

#60 Get Song Position in the current module

```
    SC #60
    WC
    GD Song_Position  [#00..#FF]
```

  You could interpret this as the number of played patterns. After start this parameter is 0, and is increased after end of each pattern. This variable could be used for synchronization of Spectrum actions with the module playback. For this you could, for example, in the beginning of the interrupt procedure call SC #60, then run your graphic procedures (for reasonable delay for command processing), and then read value from port #179 (GD Song_Position), and then compare it with required value, and jump to other part of demo, if they are equal, ie:

```
  if (Song_Position==My_Position)
  then goto Next_Part_Of_Demo
```

#61 Get Pattern Position in the current module

```
    SC #61
    WC
    GD Pattern_Position  [#00..#3F]
```

  Returns the current row in a pattern, usage is the same as in the previous command, but mind - this value changes quite fast, and therefore

```
 if (Pattern_Position>=My_Position) then goto Next_Part_Of_Demo
```

#62 Get Mixed Position

```
    SC #62
    WC
    GD Mixed_Position
```

    Mixed_Position: (by bits)

```
    7-Song_Position.1
    6-Song_Position.0
    5-Pattern_Position.5
    4-Pattern_Position.4
    3-Pattern_Position.3
    2-Pattern_Position.2
    1-Pattern_Position.1
    0-Pattern_Position.0
```

Ie, if to receive Mixed_Position and then use AND #3F with it, then you will get Pattern_Postion, and if to receive Mixed_Position, and then apply RLCA, RLCA, AND #02, then it will be two lower bytes of Song_Position. Look comments for commands #60 and #61.

#63 Get current Channel Notes

```
    SC #63
    WC
    GD Note_of_channel_0
    WN
    GD Note_of_channel_1
    WN
    GD Note_of_channel_2
    WN
    GD Note_of_channel_3
```

If in some channel note has changed since the last call of command #63, then bit #7 of returned value

```
            Note_of_channel_N
```

will be zero; if the value is the same as earlier, then this bit will be 1. Lower 7 bits are notes from 0 to 95; if this value is 127, then it means, that the channel is free now. This command is needed for various analyzers.

#64 Get Channel Volumes

```
    SC #64
    WC
    GD Volume_of_channel_0
    WN
    GD Volume_of_channel_1
    WN
    GD Volume_of_channel_2
    WN
    GD Volume_of_channel_3
```

Look comments for command #63

#65 Jump to position (*)
Goes to specified position

```
    SD Position
    SC #65
    WC
```

#66 Set speed/tempo (*)
Sets speed/tempo in the range of #01-#1F. When this value is #20-#FF, it is considered as tempo. The values of tempo are equal to original values when speed is equal to #06.

```
    SD Speed/Tempo
    SC #66
    WC
```

#67 Get current speed value (*)

```
    SC #67
    WC
    GD Speed
    WD
```

#68 Get current tempo value (*)

```
    SC #68
    WC
    GD Tempo
    WD
```

#6A - #7F Reserved

#80 Direct Play FX Sample (#80..#83)

```
    Sample playback in specified channel.

    SD Sample_Number
    SC #80..#83 (lower bits define number of channel)
    WC
```

#88 Direct Play FX Sample (#88..#8B)

```
    Sample playback with specified note.

    SD Sample_Number
    SC #88..#8B (lower bits define number of channel)
    WC
    SD Note [0..95]
    WD
```

#90 Direct Play FX Sample (#90..#93)

```
    Sample playback with specified volume.

    SD Sample_Number
    SC #90..#93 (lower bits define number of channel)
    WC
    SD Volume [#00..#40]
    WD
```

#98 Direct Play FX Sample (#98..#9B)

```
    Sample playback in specified channel with given note and volume.

    SD Sample_Number
    SC #98..#9B (lower bits define volume)
    WC
    SD Note [0..95]
    WD
    SD Volume [#00..#40]
    WD
```

#B0 - #F0 Reserved

Comment:
1. Commands, marked with (*) are undocumented command, and are guaranteed to be available only in version up to 1.04. Author (2) of this document won't be responsible for any errors.

2. All registers (their names), specified in this document, relate _only_ to internal GS registers and have no connection with registers of the Main CPU of Spectrum.

5. A bit of lyrics

GS Authors: (2 ones ;)

--- Slava Dangerous ---
(X-Trade)

He owns the idea of GS, hardware implementation of it, some suggestions towards software part of GS, and also Amiga 1200, which was used by me for various experiments. He is the only manufacturer and trader of General Sound.

--- Stinger ---

It's me, author of this document. I am the author of internal GS software. AT the current moment ROM of GS is about 20K and I'm quite tired, but anyway, I have the following plans for future versions of GS:

- Wave 4 Sound Generators, all octaves
- Acceleration of sound generation on 30-40%
- STM playback from PC
- Good command system
- Special effects with samples
- Compressed patterns (15% of original size)
- And more

Any software should work well on any future versions of ROM, if it was written with my recommendations.

I plan to add more commands and I wait your concrete suggestions

So, if you have some problems and want some features to appear in GS, then phone me and say your suggestions (translator: telephone wasn't attached)

Sanx 4 moral support:
Dima (X-Trade)
SParker (XLD)

-= THE END =-