SPECTRA^2

TMS9900 Arcade Game Library

for the

Texas Instruments TI-99/4A

REFERENCE MANUAL

version 1.0 - March 2011

REVISION

Date	Author	Revision
08-MAR-2011	Filip van Vooren	Initial release

Introduction

License spectra^2

License Honeycomb Rapture

How it all started

The idea for the initial implementation of SPECTRA was born while I was working on Pitfall!, my first homebrew game for the Texas Instruments TI-99/4A.

During that time I was studying the Colecovision disassembly of the game very closely and I learned that the game called multiple subroutines stored in the consoles' built-in ROM. Doing some research in the internet revealed that this Colecovision ROM contains a BIOS; a collection of game routines called OS7.

Thanks to the wonderful work of Daniel Bienvenu who documented most of these subroutines, I was able to understand what they were actually for. It inspired me to start working on a similar library for the TI-99/4A Home Computer.
I wanted an open-source library that allows me to concentrate on the development of the game itself, without having to start writing all subroutines from scratch over and over again.

SPECTRA^2 takes that approach one step further and acts as a miniature operating system for running homebrew games and software from the cartridge space on the unexpanded TI-99/4A.
The library is designed for minimal memory usage, the main target being the TI-99/4A with its 256 bytes of scratchpad memory.

Compatibility

SPECTRA^2 is a library targeted for cross-development on a PC compatible environment. Even on an older PC, assembly times are so fast that I don't see much benefit in reusing already assembled object files. I do see some huge benefits in programming TMS9900 assembly on your desktop or netbook:

Besides the fact that you can always carry your development environment with you (e.g. on a USB stick), the biggest advantage for me are the TI-99/4A emulators and their powerful built-in debuggers. Using such an environment will seriously speed-up your development cycle, while allowing more flexibility.

The source code of SPECTRA^2 is compatible with Burrsofts' Asm994A Assembler V3.008
This great cross-assembler for Windows is not part of SPECTRA^2, but can be obtained directly at BurrSoft[1]
The assembler is part of the Win994A emulator package and is considered freeware by the author. For further details and verification please check the license conditions at the mentioned BurrSoft page.

Serviceable parts inside

The library has been tested to some extent, but comes without any warranties whatsoever. There may still be plenty of bugs inside and if you find any, let me know and I'll try to fix them.

The runtime library

Installation

The installation process is very easy, download the spectra^2 zip-file from **http://www.retroclouds.de/spectra2/spectra2.zip** and extract/copy all files to your working directly.
If you want a minimal installation, then it's sufficient to copy the **runlib.a99** file.
This assembly source file is the core of the library and contains all required equates and subroutines for running your first program.

Hello world! example

Take a look at the below "Hello World!" program. This is pretty much how the assembly source should be laid out.

We can identify 4 major parts:

- A) The cartridge header
- B) Include required assembly source files
- C) Equates for controlling library startup behaviour
- D) The main program

```
********@*****@*********************@**************************
        AORG  >6000                 ; cartridge space >6000 - >7FFF
*--------------------------------------------------------------
* A - Cartridge header
*--------------------------------------------------------------
GRMHDR  BYTE  >AA,1,1,0,0,0
        DATA  PROG
        BYTE  0,0,0,0,0,0,0,0
PROG    DATA  0
        DATA  RUNLIB
HW      BYTE  12                    ; # of chars in 'HELLO WORLD!'
        TEXT  'HELLO WORLD!'
*--------------------------------------------------------------
* B - Include required files
*--------------------------------------------------------------
        COPY  "D:\Projekte\spectra2\tms9900\runlib.a99"
*--------------------------------------------------------------
* C - SPECTRA2 startup options
*--------------------------------------------------------------
SPVMOD  EQU   GRAPH1                 ; Video mode.   See VIDTAB for details.
SPFONT  EQU   FNOPT7                 ; Font to load. See LDFNT for details.
SPFCLR  EQU   >F0                    ; Foreground/Background color for font.
SPFBCK  EQU   >08                    ; Screen background color.
***************************************************************
* D - Main
********@*****@*********************@**************************
MAIN    BL    @PUTAT
        DATA  >0B0A,HW              ; "Hello World!" on row >0B, column >0A
        B     @TMGR                 ; Handle FCTN-QUIT key, etc.
        END
```

This example is included as file **example1.a99** in the spectra^2 samples directory.

A) The cartridge header

The TI cartridge space is in the range from >6000 to >7FFF. It's important to know that the cartridge header must start at >6000 in order to be recognized as a valid header by the TI Operating System.

For most projects it's sufficient to change the program title for the TI selection screen. This string has to be prefixed with a length byte and may not contain any lower-case characters.

```
********@*****@*******************@*************************
        AORG  >6000                 ; cartridge space >6000 - >7FFF
*--------------------------------------------------------------
* Cartridge header
*--------------------------------------------------------------
GRMHDR  BYTE  >AA,1,1,0,0,0
        DATA  PROG
        BYTE  0,0,0,0,0,0,0,0
PROG    DATA  0
        DATA  RUNLIB
HW      BYTE  12                    ; # of chars in HELLO WORLD!
        TEXT  'HELLO WORLD!'
```

The TI cartridge selection screen should look as seen in the screenshot below:

B) Include required files

Use the COPY directive to include the spectra^2 runtime library **runlib.a99** in your source code. Change the file path so that it matches the directory containing your version of the runlib.a99 file.

```
*--------------------------------------------------------------
* Include required files
*--------------------------------------------------------------
        COPY  "D:\Projekte\spectra2\tms9900\runlib.a99"
```

C) Equates for controlling library startup behaviour

The below equate values are used for initializing the TI-99/4A environment. The specified values are inserted in the source code of the spectra[2] initialisation routine and video mode table during the assembly process.

```
*--------------------------------------------------------------
* SPECTRA2 startup options
*--------------------------------------------------------------
SPVMOD  EQU   GRAPH1               ; Video mode.   See VIDTAB for details.
SPFONT  EQU   FNOPT7               ; Font to load. See LDFNT for details.
SPFCLR  EQU   >F0                  ; Foreground/Background color for font.
SPFBCK  EQU   >08                  ; Screen background color.
```

- **SPVMOD EQU GRAPH1**

  This directive is used for initializing the VDP in graphic mode 1 (32 columns mode). Actually GRAPH1 is the address of the included video mode table. The table is used by the **VIDTAB** subroutine for setting all 7 VDP registers.

  See the VIDTAB subroutine on page 70 for further details.

- **SPFONT EQU FNOPT7**

  Load the TI-Basic upper and lower case font from GROM and make the font bold. This is handled by the **LDFNT** subroutine.

  See the LDFNT subroutine on page 89 for further details.

- **SPCLR EQU >F0**

  Set foreground color to white.

- **SPFBCK EQU >08**

  Set background color to red.

For further details also refer to the section "Library startup options" on page 21.

D) The main program

After the library has completely initialized, it will automatically do a "B @MAIN" for returning control to the main program.

```
*****************************************************************
* Main
*********@*****@*********************@**************************
MAIN    BL    @PUTAT
        DATA  >0B0A,HW              ; "Hello World!" on row >0B, column >0A
        B     @TMGR                 ; Handle FCTN-QUIT key, etc.
        END
```

- **BL @PUTAT**

  By calling PUTAT with the specified "DATA >0B0A,HW" statement, the cursor is set to row >0B, column >0A. It then displays the length-byte prefixed string 'HELLO WORLD!', which was also used in the cartridge header.

  See the PUTAT subroutine on page 92 for details on how to display a string.

- **B @TMGR**

  Control is now handed over to TMGR, the thread scheduler. This subroutine is the main-loop for all programs using the spectra[2] library. It does many tasks, such as scanning the keyboard, handling FCTN-QUIT, running speech & sound player, etc.

  See the thread scheduler section on page 42 for further details.

- **END**

  The assembler END directive.

Library initialisation

The initialisation subroutine **RUNLIB** is the entry point into the spectra^2 library. This subroutine is normally called via a "**B @RUNLIB**" upon program start.

If the program is in the cartridge space, then RUNLIB gets called when the corresponding option is chosen from the TI cartridge selection screen. For this to work, it's required that the address of RUNLIB is used in the cartridge header.
See the "Hello World!" program on page 14 for an example.

The tasks done by RUNLIB are:

1) Disable interrupts and set workspace to >8300.
2) Clear CPU scratch-pad memory from >8306->83FF.
3) Set random seed and determine if VDP handles PAL or NTSC.
4) Copy machine code into scratch-pad memory.
5) Determine TI-99/4A operating system version.
6) Initialize used registers, set defaults and mute the sound generators.
7) Setup VDP registers, clear 16K of VRAM, load color table and startup font.
9) Jump into the main program via "**JMP MAIN**".

Now let's take a look at all these steps in detail:

1) Disable interrupts and set workspace to >8300.

To avoid any conflicts with the ISR routine in the consoles' OS, interrupts are disabled. The register workspace is then set to the top of scratchpad memory (>8300).

2) Clear CPU scratchpad memory from >8306 - >83FF.

In the previous step the workspace was set to >8300. We now clear all scratch-pad memory starting at >8306 (location of register R4).

3) Set random seed and determine if VDP handles PAL or NTSC.

The init routine copies the random seed set by the monitor OS into its proper memory location. Additionally the init subroutine now determines if the VDP is a PAL or NTSC version. It does that by continuously checking the VDP interrupt flag while running a loop counter.

The result of the test (PAL or NTSC) is stored in bit 12 of the CONFIG register (R12).

Note that this step uses registers R1-R3 for temporary storage.

See the VDP Programmers Guide[1] for further details on the VDP interrupt flag.

4) Copy machine code into scratch-pad memory.

In this step 6 bytes of machine code are copied into the scratch-pad memory location >8320. The machine code is mainly used for speeding up the filling and copying of large memory blocks between CPU and VDP memory. Having this code in scratch-pad memory reduces wait-states.

See Thierry Nouspikel's Technical pages[2] for further details on scratch-pad memory and the multiplexer.

5) Determine TI-99/4A operating system version.

The GROM memory in the TI-99/4(A) console is scanned to determine the operating system version.

The result is stored in bit 10 of the CONFIG register.

If the OS can't be determined, then spectra^2 assumes it's running on an unsupported platform.

It's important to know, that spectra^2 doesn't support the original TI-99/4 (without a) Home Computer.

This step will exit to the TI title screen if an unsupported system such as the TI-99/4 is detected.

6) **Initialize used registers, set defaults and mute the sound generators.**

- The registers R1-R3 used in the previous steps are now cleared.
- The stack register (R9) is loaded with address >8400 (that's outside scratch-pad memory. You need to do a "DECT STACK" first).
- The register R15 is loaded with the address of the VDP data write port.
- All sound generators are muted.

7) **Setup VDP registers and clear 16K of VRAM.**

- All VDP registers are set according to the values in the specified video mode table. This is handled by calling the VIDTAB subroutine using the specified equates.
- The 16K of VRAM gets cleared.
- The color table is loaded into VRAM using the specified equates.
- The startup font is loaded into VRAM using the specified equates.

See the Library startup options on page 21 for further details on the equate values to use.

8) **Hand-over control to MAIN**

The initialisation has completed and control is given to the MAIN subroutine by issuing a "B @MAIN".

Note that register R0 is not cleared during the library initialisation. This can be useful for passing-through a value from your custom pre-init routine to MAIN.

See file "**/samples/example6.a99**" for an example.

Library startup options

There are a few equates that must be set in the main source file. They control the spectra^2 startup options such as:
VDP video mode, font style, etc.

Equate	Description
SPVMOD	Address of video mode table to use on startup. Use GRAPH1 for 32 columns mode (with sprites). Use TXTMOD for 40 columns mode (no sprites). It's also possible to use your own video mode table.
SPFONT	Built-in system font to load on startup. Note that there are no fonts included in RUNLIB. The fonts are loaded into VDP memory from the GROMs in the TI-99/4A console. Possible values to use are: NOFONT ; Do not load font on startup FNOPT1 ; Load TI title screen font FNOPT2 ; Load upper case font FNOPT3 ; Load upper/lower case font FNOPT4 ; Load lower case font FNOPT5 ; Load TI title screen font & make fat FNOPT6 ; Load upper case font & make fat FNOPT7 ; Load upper/lower case font & make fat FNOPT8 ; Load lower case font & make fat
SPFCLR	Foreground and background color for textmode This value goes into VDP#register 7 when using a textmode video table. The SPFCLR equate is not used in any of the graphics video mode tables.
SPFBCK	Background color for graphic modes. This value goes into VDP#register 7 when using a graphics video mode table. The SPFBCK equate is not used in the TXTMOD video mode table.

For further details see documentation on VIDTAB (page 70) and LDFNT (page 89).

Reset to TI title screen

You can safely exit the program and return to the TI title screen, by setting register R1 to >FFFF and doing a "**B @RUNLI1**". The advantage over a "BLWP @>0000", is that scratchpad memory gets properly cleared first.

```
        SETO  R1
        B     @RUNLI1              ; Exit to title screen
```

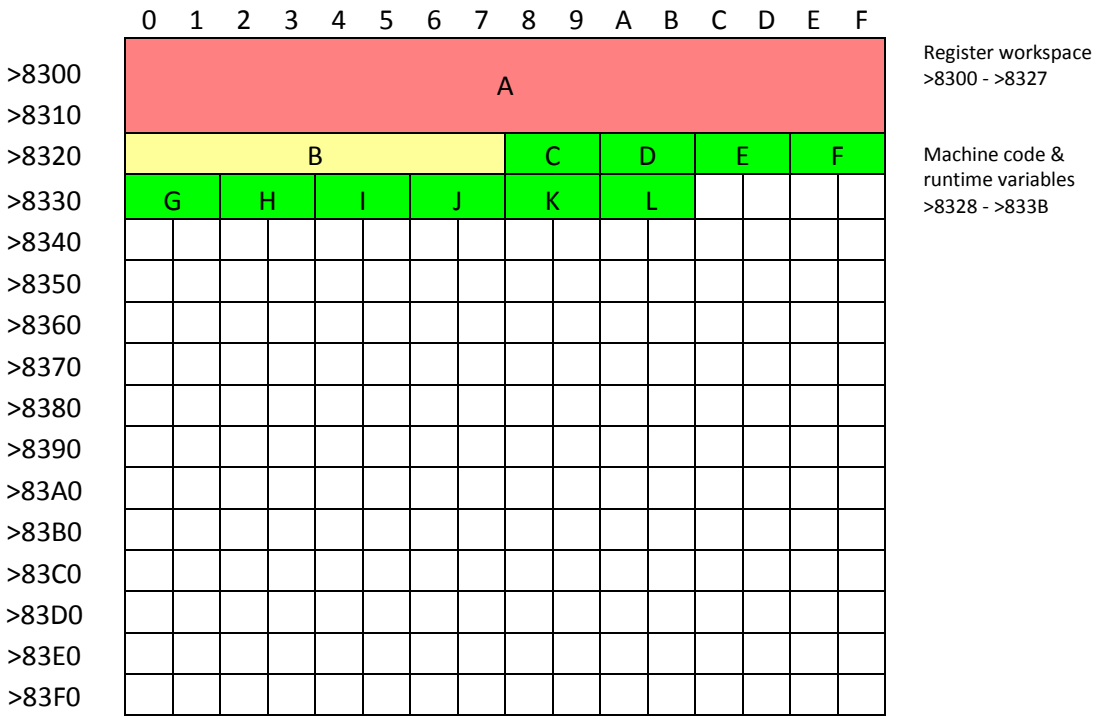
Scratch-pad memory setup

The TI-99/4A has 256 bytes of memory on the motherboard, which is referred to as "scratch-pad" memory. It has the address range >8300 - >83FF and is on the 16-bit bus. It can be accessed without wait states and is really fast compared to the memory on the 8-bit bus (e.g. 32K memory expansion). You should use scratch-pad memory where possible.

Of the 256 bytes available, spectra^2 uses 60 bytes for storing the register workspace and all variables it needs for housekeeping tasks, etc.

Depending on the features used, some memory can be recovered and used for other purposes. This is controlled by multiple flags in the CONFIG register.

Let’s take a detailed look at each of the used memory locations.



A	Register workspace	>8300 - >831F	32
B	Machine code for loops/speech/...	>8320 - >8327	8
C	PNT BASE address	>8328 - >8329	2
D	Cursor YX position	>832A - >832B	2
E	Timers: Address of timer table	>832C - >832D	2
F	Timers: Address of user hook	>832E - >832F	2
G	Timers: Internal use	>8330 - >8331	2
H	Virtual keyboard flags	>8332 - >8333	2
I	Sound player: Address of tune	>8334 - >8335	2
J	Sound player: Internal use	>8336 - >8337	2
K	Speech player: Address of LPC data	>8338 - >8339	2
L	Seed for random subroutine	>833A- >833B	2
	Size in bytes		**60**

A) Register workspace (>8300 - >8319)

There are only 3 hardware registers in a 9900 CPU: PC (program counter), WP (workspace pointer), ST (status register). All other registers are stored in CPU memory.

That is why we need 32 bytes of scratchpad-memory for holding the 16 (16-bit) registers R0-R15.

See section "Register usage" on page 31 for further details.

B) Machine code (>8320 - >8327)

The below 8 bytes of machine code are copied into scratchpad memory >8320 upon library startup. The machine code is mainly used for speeding up loops. It's used by several spectra^2 low-level routines (e.g. CPYM2V)

You can use the "tight-loop" routine for your own purpose by overwriting 2 bytes of machine code at >8320. Note that the routine should be called with BL @>8320. It expects TMP2 (R6) to contain the number of times the loop should be executed.

```
*--------------------------------------------------------------
* ; Machine code for tight loop.
* ; The MOV operation at MCLOOP must be injected by the
* ; calling routine.
*--------------------------------------------------------------
*       DATA  >????               ; \ MCLOOP  MOV   ...
MCCODE  DATA  >0606               ; |         DEC   R6 (TMP2)
        DATA  >16FD               ; |         JNE   MCLOOP
        DATA  >045B               ; /         B     *R11
```

When running the speech player (SPPLAY), the following 4 bytes of machine code get copied to >8320, overwriting part of the "tight loop" code. The tight loop code is automatically restored upon player exit.

```
*--------------------------------------------------------------
* ; Machine code for reading from the speech synthesizer
* ; The SRC instruction takes 12 uS for execution in scratchpad RAM.
* ; Is required for the 12 uS delay. It destroys R5.
*--------------------------------------------------------------
SPCODE  DATA  >D114                ; \         MOVB  *R4,R4 (TMP0)
        DATA  >0BC5                ; /         SRC   R5,12  (TMP1)
```

C) PNT base address (>8328 - >8329)

This memory location holds the address of the Pattern Name Table (PNT) in VRAM. The PNT table in VRAM contains all tiles to display on screen. The address is automatically set by spectra^2 if a video mode table is loaded with the VIDTAB subroutine and is used by many of the VDP subroutines included in the library (e.g. YX2PNT).

```
Equates
WBASE   EQU   >8328                ; 02 - PNT base address
```

You can also manually set the (using the WBASE equate) for creating multiple "virtual" screens. Basically you'd set it to a VDP memory location outside the window addressed by the VDP#2 write-only register. You can then use all available spectra^2 subroutines for drawing the screen.
For instant display, you then only have to switch the VDP#2 write-only register to the new address.

Please refer to the VDP Programmer's Guide page for further details on the Pattern Names Table.

D) Cursor YX position (>832A - >832B)

This is the memory address used for holding the cursor position. There is no real cursor in spectra[2], but many VDP routines in the library use this location for calculating the VRAM target address of the corresponding PNT entry.

```
Equates
WYX     EQU   >832A              ; 02 - Cursor YX position
BY      EQU   WYX                ;      Cursor Y position
BX      EQU   WYX+1              ;      Cursor X position
```

Note that the cursor position always starts with Y=0, X=0. So if you want to display something on row 6, column 10 you would load >0509 into memory location @WYX.

Here's an example on how to use the cursor for displaying the string "Hello World!" on row 6, column 10.

```
TEST1   LI    R0,>0509           ; Row 6, column 10
        MOV   R0,@WYX            ; Load cursor
        BL    @PUTSTR            ; Display string
        DATA  HW                 ; String to display
        JMP   $                  ; Soft-halt
HW      BYTE  12
        TEXT  'HELLO WORLD!'
```

E) Timers: Address of timer table (>832C - >832D)

This memory address points to a table in CPU memory that contains required base data when running timers. You normally fill the timer table by using the MKSLOT routine.

```
Equates
WTITAB  EQU  >832C               ; 02 - Address of timer table
```

See section "Thread Scheduler" on page 43 for further details.

F) Timers: Address of user hook (>832E - >832F)

This memory address contains the address of the user hook, a user-supplied subroutine that is executed **at least** every 1/60th (NTSC) or 1/50th (PAL) of a second.

The idea is that you use the user hook for stuff that isn't bound to the VDP interrupt.

```
Equates
WTIUSR  EQU  >832E              ; 02 - Address of user hook
```

See section "Thread Scheduler" on page 43 for further details.

G) Timers: Internal use (>8330 - >8331)

Used by the Thread Scheduler subroutine (TMGR) for storing internal variables.

```
Equates
WTITMP  EQU  >8330              ; 02 - Internal use
```

See the "Thread Scheduler" section on page 43 for further details.

H) Virtual keyboard flags (>8332 - >8333)

This memory location holds 16 1-bit flags, representing the keys pressed on the spectra2 virtual keyboard. That is when calling the VIRTKB subroutine.

```
Equates
WVRTKB  EQU  >8332              ; 02 - Virtual keyboard flags
```

See the "Virtual keyboard" section on page 52 for further details.

I) Sound player: Address of tune (>8334 - >8335)

Points to a table in CPU memory or VRAM containing the the sound list data for playback with the built-in sound player routine (SPPLAY).

```
Equates
WSDLST  EQU  >8334              ; 02 – Tune address
```

See the "Sound & speech subroutines" section on page 103 for further details.

J) Sound player: Temporary use (>8336 - >8337)

Contains some internal variables used by the sound player routine (SDPLAY).

```
Equates
WSTMP  EQU   >8336              ; 02 – Tune address
```

See the "Sound & speech subroutines" section on page 103 for further details.

K) Speech player: Address of LPC data (>8338 - >8339)

The spectra^2 library offers the possibility to playback speech samples, when a speech synthesizer is connected to the TI-99/4A console. Speech samples are encoded in LPC format (Linear Predictive Coding) and must be stored in CPU memory for playback with the SPPLAY subroutine.
This memory location holds the address of the LPC data stream.

```
Equates
WSPEAK  EQU  >8338              ; 02 – Address of LPC data
```

See the "Sound & speech subroutines" section on page 103 for further details.

L) Seed for random subroutine (>833A - >833B)

For generating pseude-random numbers we need a seed value. The WSEED memory location is automatically setup by the spectra^2 initialisation routine. It copies the seed value set by the monitor OS.

Equates

```
WSEED  EQU  >833A              ; 02 - Seed for random subroutine
```

See the RND subroutine on page 123 for further details.

Register usage

The 16 available registers play a very important role when using the spectra^2 library. Some of the registers have a special purpose, e.g. for passing parameters or speeding-up memory access.

Let's take a detailed look at each of the registers.

- **General purpose registers (R0 … R3)**

  The registers R0 – R3 aren't used by any of the subroutines in the spectra^2 library.

  With the only exception being that registers R1-R3 are used during the library initialisation. Nonetheless, once your program (MAIN) takes over, you'll have R0-R3 to your exclusive disposal.

- **Temporary registers (R4 … R8)**

  The registers R4 … R8 are registers used for temporary storage of parameters, counters, etc.

  These registers should never be addressed with their R4 … R8 label.

  Instead they should be referred to using the TMP0 … TMP4 label.

Equates

```
TMP0    EQU   R4                ; Temp register 0
TMP1    EQU   R5                ; Temp register 1
TMP2    EQU   R6                ; Temp register 2
TMP3    EQU   R7                ; Temp register 3
TMP4    EQU   R8                ; Temp register 4
```

Keep in mind, that when calling any of the spectra^2 subroutines, it is likely that some or all of the temporary registers will be destroyed.

- **The stack pointer or temporary register TMP5 (R9)**

  Now for sure you already know that there is no hardware stack pointer in a TMS9900 CPU. As a workaround the stack pointer can be simulated by using the general purpose register R9.

  Depending on your requirement you should use one of the below equates:

```
Equates
STACK   EQU   R9    ; Stack pointer
TMP5    EQU   R9    ; Temp register 5
```

  Note that when the runtime library gets initialized, R9 is loaded with the value >8400.

  Please refer to page 40 for further details on stack usage.

  If you decide not to use a stack, then you can use R9 as temporary register **TMP5**.

- **Highest slot in use & internal counter for timers (R10)**

  R10 is exclusively used by the thread scheduler.

  - The high byte of R10 keeps track of the highest slot used in the thread scheduler timer table.
  - The low byte of R10 is the thread scheduler tick counter and is updated every 1/50th (VDP) or 1/60th (NTSC) of a second.

  Please refer to page 43 for further details on the thread scheduler.

- **Subroutine return address (R11)**

  Contains the subroutine return address when issuing a branch-and-link **"BL xxxx"**.

- **The CONFIG register (R12)**

  R12 is the spectra^2 configuration register. It's used for storing 16 individual status flags and should be referenced using the CONFIG label.

```
Equates
CONFIG  EQU   R12    ; SPECTRA configuration register
```

  Please refer to page 37 for further details on the bit flags available in the CONFIG register.

- **Copy of VDP status byte & counter for sound player (R13)**

  R13 is exclusively used by spectra^2:

  - The high byte of R13 contains a copy of the VDP status register byte. The byte is continuously copied by the TMGR thread scheduler.
  - The low byte of R13 is used as an internal counter when the sound player is running.

```
Equates
BVDPST   EQU   WS1+26   ; Copy of VDP status register (HI byte R13)
```

- **Copy of VDP register #0 and VDP register #1 (R14)**

  R14 is exclusively used by spectra^2.

  - The high byte of R14 contains a copy of VDP write-only register #0.
  - The low byte of R14 contains a copy of VDP write-only register #1.

Equates

```
VDPR01   EQU   R14     ; Copy of VDP#0 and VDP#1 bytes
VDPR0    EQU   WS1+28  ; High byte of R14. Is VDP#0 byte
VDPR1    EQU   WS1+29  ; Low byte of R14.  Is VDP#1 byte
```

This register is used for easily doing bit-operations when setting/getting current video mode, sprite magnification, etc.
See the sections "VDP low level subroutines" on page 65 and "VDP tiles & patterns subroutines" on page 88 for further details on the available VDP support routines.

- **VDP write address or temporary register (R15)**

  R15 contains the address of the VDP read or write port. By storing the port address in the register, it's possible to write more compact and faster code. The VDP low-level routines in spectra^2 use this register a lot.

Equates

```
VDPRW    EQU   R15     ; Contains VDP read/write address
TMP6     EQU   R15     ; Temp register 6
VDPR     EQU   >8800   ; VDP read data window address
VDPW     EQU   >8C00   ; VDP write data window address
VDPS     EQU   >8802   ; VDP status register
VDPA     EQU   >8C02   ; VDP address register
```

Note that when a spectra^2 VDP low-level routine is called, it will load R15 with VDPW or VDPR depending if writing or reading.

It's also possible to use R15 as temporary register TMP6. However you'll have to ensure, that R15 is reloaded with the correct VDP write or read address before calling any of the VDP subroutines.

Equates & constants

A large set of equates is included in the library source code. Please use the equates instead of the corresponding values where possible.
That way, the migration to a new spectra^2 release will be less cumbersome.

In particular equates exist for:

- Registers
  - Temporary registers (R3-R9)
  - Hi- or Lo- byte of all registers (R0-R15)
  - Stack pointer (R9)
  - Special purpose registers (R10-R15)

- Bit-level operations
  - All flags in the config register (R12)
  - Bit 0-15 of a word

- Spectra^2 routines & parameters
  - Virtual keyboard
  - Sound player options
  - Speech player options

- Spectra^2 memory
  - Cursor YX position
  - Task scheduler variables
  - Virtual keyboard, Sound/Speech player
  - …

- Hardware
  - VDP & sound addresses
  - GROM, Speech, etc.

The spectra^2 library also includes some constants

- For setting bits 0-15 of a word
- For loading a byte with decimal value 0-9

For further details please check the runlib.a99 file (spectra^2 source code).

The config register

Introducing the CONFIG register

Many of the features in spectra^2 are controlled by 16 individual bit flags of the configuration (CONFIG) register. This is currently mapped to R12 but that might change in the future. Therefore please use the CONFIG label instead.

```
Equates
CONFIG  EQU   R12    ; SPECTRA configuration register
```

The reason why we use a register instead of a memory location, is that a register allows for easy bit compare and manipulation.

```
                  1 1 1 1 1 1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | +-- Sound player: tune source (1=ROM/RAM, 0=VDP)
| | | | | | | | | | | | | |
| | | | | | | | | | | | | +---- Sound player: repeat tune
| | | | | | | | | | | | |
| | | | | | | | | | | | +------ Sound player: enabled
| | | | | | | | | | | |
| | | | | | | | | | | +-------- Keyboard: mode (1=Real, 0=virtual)
| | | | | | | | | | |
| | | | | | | | | | +---------- Keyboard: ANY key pressed
| | | | | | | | | |
| | | | | | | | | +------------ TI-99/4A v2.2 OS (*)
| | | | | | | | |
| | | | | | | | +-------------- Timer: Kernel thread enabled
| | | | | | | |
| | | | | | | +---------------- Timer: Block kernel thread
| | | | | | |
| | | | | | +------------------ Timer: User hook enabled
| | | | | |
| | | | | +-------------------- Timer: Block user hook
| | | | |
| | | | +---------------------- Speech player: external voice
| | | |
| | | +------------------------ Speech player: busy
| | |
| | +-------------------------- Speech player: enabled (*)
| |
| +---------------------------- VDP9918 version (1=PAL/50, 0=NTSC/60) (*)
| |
| +------------------------------ Subroutine state flag 2
|
+-------------------------------- Subroutine state flag 1

(*) = Read-only flag. Set by RUNLIB subroutine
```

The subroutine state flags

Bit 0 and 1 in the CONFIG register are used to control the behaviour of some of the subroutines in the spectra^2 library. They can be seen as toggles that turn certain features on/off.
The MKNUM subroutine for example uses bit 0 in the CONFIG register to determine if the converted number should be displayed on screen.

You can use bit 0 and 1 of the CONFIG register for your own purposes. Just keep in mind that they may be overwritten when calling some of the spectra^2 subroutines.

For further details please check the runlib.a99 file (spectra^2 source code).

The stack

Introducing the stack

Now for sure you already know that there is no hardware stack pointer in a TMS9900 CPU. As a workaround the stack pointer can be simulated by using one of general purpose registers.
In its current release spectra[2] uses register R9 as stack pointer, that might change in a future release. Please use the STACK equate instead of R9.

```
Equates
STACK   EQU   R9    ; Stack pointer
TMP5    EQU   R9    ; Temp register 5
```

The stack grows toward low memory. **It means you have to decrease the stack pointer before pushing a value on the stack.**

```
MYSUB   DECT  STACK
        MOV   R11,*STACK          ; Push R11
        DECT  STACK
        MOV   R0,*STACK           ; Push R0
        DECT  STACK
        MOV   R1,*STACK           ; Push R1
        ...
        B     @POPR1              ; Pop R1,R0,R11 and return to caller
```

When the runtime library gets initialized, the STACK pointer (R9) is loaded with >8400, that is just above scratch-pad memory. By issuing a "DECT STACK" before pushing, we get to address >83FE which is the highest address in scratch-pad memory. Did I mention that >8400 is the address of the sound port? You'll get strange results when trying to push a value to that address...

The POPR(0-3) and POPRT subroutine

Instead of writing inline code upon subroutine exit, you can branch to the POPR(0-3) subroutine to pop the registers from the stack and return to the calling program.

If for example, you pushed registers R11, R0, R1 & R2 on the stack, you would do a "B @POPR2" to pop the registers and exit your subroutine.

```
****************************************************************
* POPR. - Pop registers & return to caller
****************************************************************
*  B  @POPRG.
*--------------------------------------------------------------
*  REMARKS
*  R11 must be at stack bottom
********@******@*********************@**************************
POPR3   MOV   *STACK+,R3
POPR2   MOV   *STACK+,R2
POPR1   MOV   *STACK+,R1
POPR0   MOV   *STACK+,R0
POPRT   MOV   *STACK+,R11
        B     *R11
```

spectra^2 stack usage

It's important to know, that none of the routines in the spectra^2 library internally make use of the stack. This is a major difference compared to the initial spectra release, which fully relied on the presence of a stack.

The reason for this change, is that spectra^2 is targeting the unexpanded TI-99/4A with its 256 bytes of scratch-pad memory. We don't want to waste any memory and instruction cycles on pushing/popping values from the stack.

That for sure doesn't mean that a stack is bad. As a matter of fact, based on the complexity of your game project, it's probably a good idea to use a stack. That is especially true if you have a bunch of nested calls.

You can use R9 as temporary register **TMP5**, if you decide not to use a stack.

Threads

The thread scheduler

When writing arcade games, one is faced with the difficulty of having to control different things at the same time. You have to read the keyboard, move sprites, draw the screen, run some game logic, … all at the same time.
For your game to run fluently, you have to ensure that all of this is handled in a short time frame.
Now even though TMS9900 assembly language is lightning fast, it can be very cumbersome writing such routines.

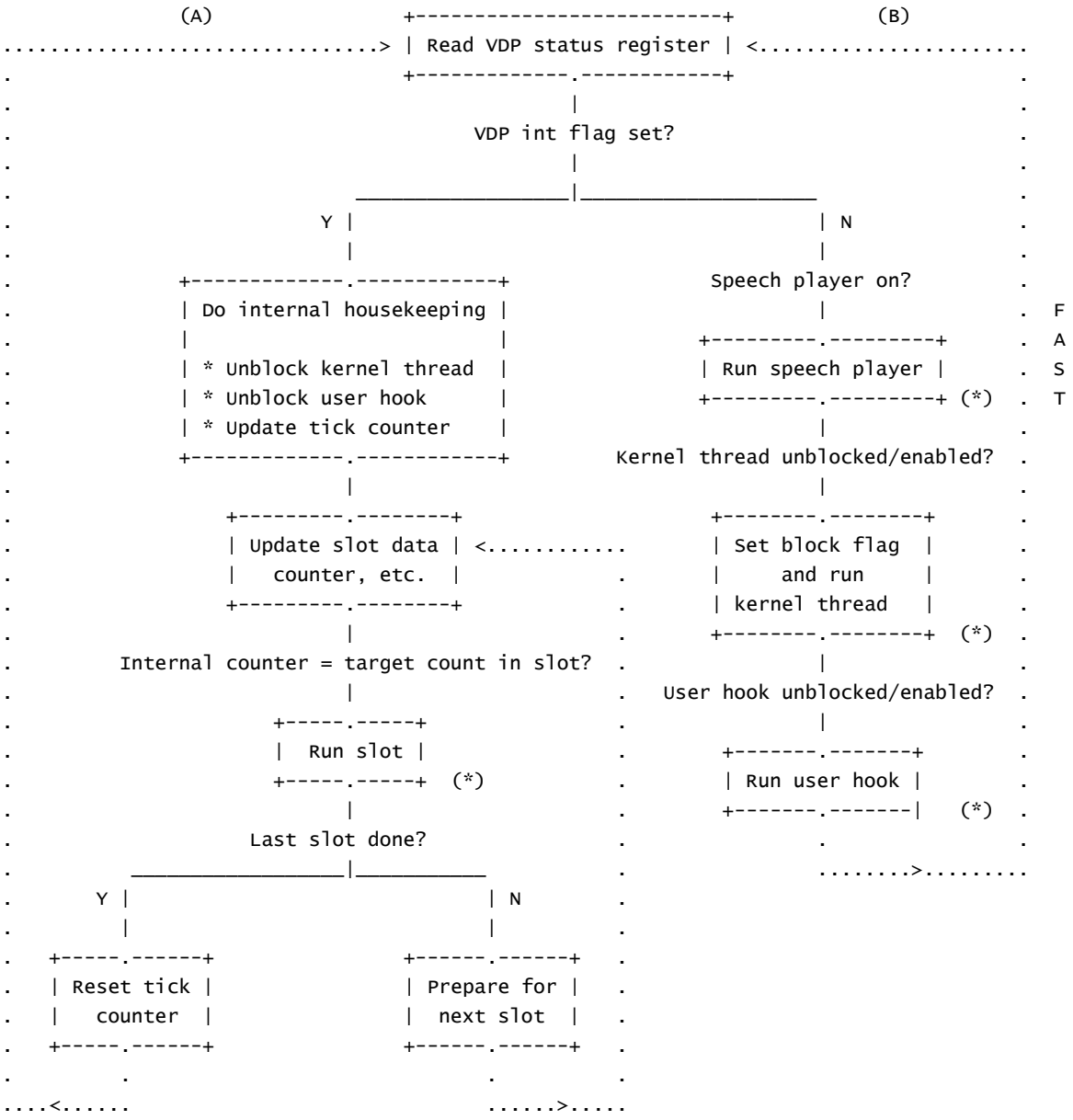
To help with that, a thread scheduler (TMGR) is included in spectra^2.
Basically the scheduler acts as your programs' main loop, periodically calling the subroutines you specify.
In order for this to work, the scheduler expects that the called subroutine will end in a timely matter. However, it can't enforce it. **A poorly designed subroutine may "hang" your game while consuming all of the CPU time for itself.**

The thread scheduler itself synchronizes with the VDP interrupt flag.
It means that -in best case- a thread can be executed every 1/60th (NTSC) or 1/50th (PAL) of a second.

For some tasks (e.g. sprite coincidence detection) this may be too slow.
That is why the thread scheduler offers the possibility to call a "kernel thread" and a "user hook" each time it reads the VDP status register.

The kernel thread is responsible for controlling the built-in music player and virtual keyboard.
The speech player is controlled by code embedded in the thread scheduler itself. This is to obtain the best possible performance.
Note that some of the bit-flags in the CONFIG register control the behaviour of the speech player, kernel thread, etc.

The below schema shows the thread scheduler workflow:



(A) = Executed once per frame (1/60th for NTSC, 1/50th for PAL)

(B) = User hook repeats until blocked from within user hook code.
Kernel thread (sound player, keyboard scan) runs once per frame.

(*) = Skipped depending on result of previous check

The timer table

The scheduler requires a work table in CPU memory for keeping track of the threads, when to fire, etc.
It's the programmers' responsibility to make sure there is enough free CPU memory for holding this table.
Make sure the table is properly initialized with >00 bytes, otherwise the thread scheduler may interpret memory as an allocated slot, execute this garbled slot and lock the computer.

You have to store the address of the table at memory location @WTITAB, as seen in the next example:

```
        ...
        MOV   @MYTAB,@WTITAB        ; Setup address of timer table
        BL    @MKSLOT
        DATA  >0002,MVBOX,EOL       ; Create new timer slot
        B     @TMGR                 ; Start thread scheduler
MYTAB   DATA  >8350                 ; Timer table address
```

For each running thread a timer slot must be allocated. A timer slot consists of 4 bytes and the initial setup is normally done by using the MKSLOT subroutine.

BYTE 0-1	BYTE 2	BYTE 3
Thread address	Interval	Internal tick counter

Thread This is the address of the subroutine that will be called by the thread scheduler when the slot is fired.
An empty slot must contain >0000 in BYTE 0-1.

Interval Determines at what interval the slot should be fired. This interval must be specified in ticks per second.
On a NTSC console we have 60 ticks per second.
On a PAL console we have 50 ticks per second.

Internal counter Is an internal counter used by the thread scheduler to keep track about when the slot should be fired.

Highest slot in use

The thread scheduler must know how many slots it needs to handle. This is controlled by the most signifcant byte of register R10.
Note that register R10 is set to 0 when the library is initialized. It means that by default only slot 0 gets executed.

In the below example, the highest slot in use is set to 2.

```
START   LI R10,>0200          ; Set highest slot to 2
```

The kernel thread

Both the built-in music player and the scanning of the virtual keyboard is handled transparantly by a background thread called the "kernel thread". You do not need to allocate a timer slot for it.

The kernel thread automatically runs once per frame. This is controlled by the "thread block flag". That's bit 8 in the CONFIG register. This block flag is set by the kernel thread upon exit and is reset by the thread scheduler once the next frame is reached.

The kernel thread feature can be completely turned off by resetting the "thread enabled" flag, that's bit 9 in the CONFIG register. However in that case there will be no automatic sound player and keyboard scanning.

Use the below code for turning off the kernel thread:

```
START   SZC @WBIT9,CONFIG   ; Turn off kernel thread
        B   @TMGR
```

The user hook

Calling a subroutine once per frame may be insufficient for certain tasks. That's especially the case if you want to reliably scan some of the VDP status register flags (e.g. coincidence detection, 5th sprite in a row, etc).

This is where the user hook comes to the rescue: Once loaded it will execute (using BRANCH!) each time the VDP status register is read.

The user hook is turned off by default, this is controlled by the "user-hook enabled" flag (bit 7 in the CONFIG register.).
The easiest way to setup a user hook, is by using the MKHOOK subroutine. **Note that spectra^2 only supports 1 user hook.**

You can delay the next execution until the next frame is reached, by setting the "block user-hook" flag in your hook code (bit 6 in the CONFIG register).
To exit the user hook code and return to the thread scheduler, you have to issue a "B @HOOKOK".

In the below example, a user hook is defined for checking the coincidence status flag. The thread scheduler automatically copies the value of the VDP status register into the high byte of R13.

```
        BL    @MKHOOK               ; Prepare user hook
        DATA  COINC
GAME2   B     @TMGR
***************************************************************
* User hook - Check for coincidence
***************************************************************
COINC   COC   @WBIT2,R13            ; Coincidence bit set ?
        JNE   COINCZ                ; No, exit
        …
COINCZ  B     @HOOKOK               ; Back to thread scheduler
```

Support routines

Following subroutines are available for dealing with threads:

- **TMGR**

  The TMGR subroutine is the entry point into the thread scheduler. It should be started with a "**B @TMGR**" after initialisation in the main program has completed.

  Make sure you checked the below before initiating TMGR, it will save you a lot of time searching for program crashes:

  - Memory address WTITAB (2 bytes) set with address of your timer table.
  - Timer table initialized with >00 bytes.
  - Memory address BTIHI (1 byte!) set with highest timer slot in use.

- **MKSLOT**

  The MKSLOT subroutine is used for allocating new timer slots. It allows you to allocate non-sequential slots, e.g. allocate slots 0,3,4,7 (without touching slots 1,2,5,6).
  If you have many slots to allocate at once, then you could copy a preset slot table from ROM to RAM without using the MKSLOT subroutine.
  Please refer to page 116 for further details.

- **CLSLOT**

  Use the CLSLOT subroutine to remove a single running slot.
  Please refer to page 118 for further details.

- **MKHOOK**

  The MKHOOK subroutine is used for allocating a user hook.
  Please refer to page 120 for further details.

- **KERNEL**

  The KERNEL subroutine runs as a thread and is responsible for running the sound player and reading the virtual keyboard. You should normally not call this subroutine from your program, it's automatically called by spectra[2].
  Please refer to page 119 for further details.

Support equates

Following equates are available for dealing with threads:

```
*--------------------------------------------------------------
* Equates for scratchpad memory locations
*--------------------------------------------------------------
WTITAB  EQU   >832C                ; 02 - Timers: Address of timer table
WTIUSR  EQU   >832E                ; 02 - Timers: Address of user hook
WTITMP  EQU   >8330                ; 02 - Timers: Internal use
*--------------------------------------------------------------
* Equates for CONFIG register
*--------------------------------------------------------------
ENUSR   EQU   >0100                ; bit 7=1   (Enable user hook)
ENKNL   EQU   >0040                ; bit 9=1   (Enable kernel thread)
```

Register usage

R10 is exclusively used by the thread scheduler:

- The high byte of R10 keeps track of the highest slot used in the thread scheduler timer table.
- The low byte of R10 is the thread scheduler tick counter and is updated every 1/50th (VDP) or 1/60th (NTSC) of a second.

Exiting a thread

There are many ways how one can exit a thread. Let's look at some of the possibilities:

B *R11

Use "B *R11" (2 bytes of machine code) to exit a thread and return to the thread scheduler if you didn't use any BL (Branch-and-link) instruction in your thread code.

```
THREAD1 BLABLA                    ; Some statements
        …
        B *R11                    ; Exit thread (2 bytes)
```

B @SLOTOK

Use "B @SLOTOK" (6 bytes of machine code) to exit a thread and return to the thread scheduler if you use a BL to call a subroutine from your thread.

```
THREAD1 BLABLA                    ; Some statements
        BL  @MYSUB1               ; Call some routine. R11 is overwritten
        B   @SLOTOK               ; Exit thread (6 bytes)
```

Save return address in other register

As an alternative you can save a copy of R11 and work with that. Remember that R0-R3 are not used by spectra^2 so they are good candidates.

```
THREAD1 MOV R11,R0              ; Save copy of R11 (2 bytes)
        BL  @MYSUB1             ; Call some routine. R11 is overwritten
        B   *R0                 ; Exit thread (2 bytes)
```

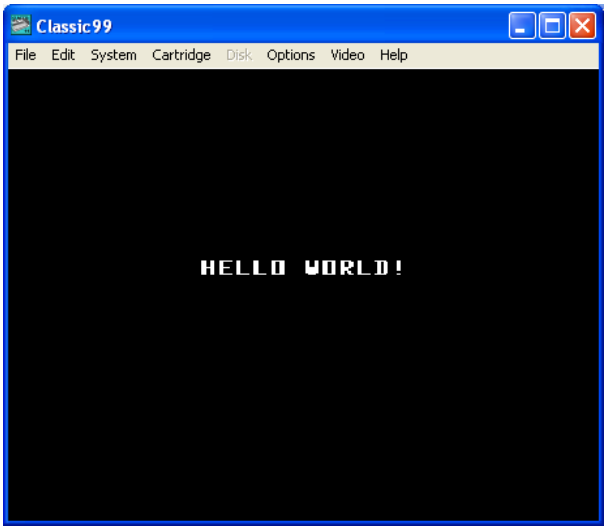
Save return address on stack

If you decide to set-up a return stack, you can do so by using the STACK register (R9). The STACK register is initialised to >8400 upon library initialisation. Use the POPRT subroutine to pop the return address from the stack and return.

```
THREAD1 DECT STACK              ; Set stack pointer (R9)
        MOV  R11,*STACK         ; Save return address on stack
        BL   @MYSUB1            ; Call some routine. R11 is overwritten
        B    @POPRT             ; Pop R11 from stack and return to caller
```

Example

In the next example, we start a thread for showing the blinking message 'HELLO WORLD!'. The thread interval is set to 15 ticks, which means that the text will effectively blink once every ½ second.



At the same time the speech player will be playing back a recorded speech sample and the kernel thread will scan the keyboard and handle FNCTN-QUIT.

This example is included as file **example2.a99** in the spectra^2 samples directory.

```
        AORG  >6000
*--------------------------------------------------------------
* Cartridge header
*--------------------------------------------------------------
GRMHDR  BYTE  >AA,1,1,0,0,0
        DATA  PROG
        BYTE  0,0,0,0,0,0,0,0
PROG    DATA  0
        DATA  RUNLIB
HW      BYTE  12
        TEXT  'HELLO WORLD!'
*--------------------------------------------------------------
* Include required files & startup options
*--------------------------------------------------------------
        COPY  "D:\Projekte\spectra2\tms9900\runlib.a99"
SPVMOD  EQU   GRAPH1          ; Video mode.   See VIDTAB for details.
SPFONT  EQU   FNOPT7          ; Font to load. See LDFNT for details.
SPFCLR  EQU   >F0             ; Foreground/Background color for font.
SPFBCK  EQU   >01             ; Screen background color.
***************************************************************
* Main
*********@*****@******************@*****************************
MAIN    BL    @FILV
        DATA  >0380,>F0,16        ; Set color table
        LI    R0,>8370
        MOV   R0,@WTITAB          ; Our timer table
        BL    @MKSLOT
        DATA  >000F,BLINK,EOL     ; Run thread every 15 ticks
        BL    @SPPREP
        DATA  ROCK,SPOPT1         ; Speech player on / Speak external
        MOVB  @BD1,@>8369         ; Set toggle
        B     @TMGR               ; Run scheduler
***************************************************************
* Thread
*********@*****@******************@*****************************
BLINK   NEG   @>8368              ; Switch toggle
        JLT   BLIN2
BLIN1   BL    @PUTAT
        DATA  >0A0A,HW            ; Show "Hello World!" message
        JMP   BLIN3
BLIN2   BL    @HCHAR
        BYTE  >0A,>0A,32,12       ; white space x
        DATA  EOL
BLIN3   B     @SLOTOK             ; Exit to Thread Scheduler
ROCK    BYTE TALKON               ; Speech data
        BYTE >00,>E0,>80,>E2,>3B,>13,>50,>DC,>64,>00,>AA,>E9,>3C,>69
        ...
        BYTE TALKOF
        END
```

Virtual keyboard

VIRTKB subroutine

The spectra^2 runtime supports a virtual TI-99/4A game keyboard controlled by the "kernel" thread. Each time the thread runs, it calls the subroutine VIRTKB for polling the keyboard/joystick status. It then maps the pressed keys as bit flags in the memory location @WVRTKB.

Benefit of the virtual keyboard is that you do not need to check both keyboard and joysticks. If you for example press 'S' on the keyboard, it reacts the same as if you pull joystick 1 to the left. They will both set the bit for the virtual key 'K1LF' in @WVRTKB to 1.

Note that the virtual keyboard does not support all keys, but it does handle enough keys for supporting an arcade game. The subroutine also checks for FNCTN-QUIT and exits to the TI-99/4A title screen when pressed.

It also handles multiple keys. If you for example pull joystick 1 diagonally up/left, then it will set both virtual keys 'K1UP' and 'K1LF'. Be aware, that the VIRTKB subroutine always scans the full keyboard. It is not possible to only scan left/right half of the keyboard.

The 'ANY' key

As soon as the VIRTKB routine detects that a key is pressed (or joystick pulled), it will set bit 11 (ANYKEY) in the CONFIG register.

Use the below code to check if any key was pressed:

```
CHECK  COC   @ANYKEY,CONFIG        ; ANY key pressed ?
       JEQ   MYLABL                ; YES
       B     *R11                  ; NO, exit
MYLABL …                           ; Process key
```

Support routines

Following subroutines in the spectra[2] library are available when dealing with the virtual keyboard:

- **VIRTKB**

  The VIRTKB subroutine handles the scanning of the keyboard and maps it the corresponding bit flags in @WVRTKB. Normally you should not call VIRTKB directly, because this is all handle in the background by the "KERNEL" thread.

  Check the "Thread scheduler" section at page 43 for further details on the kernel thread.

Support equates

Below are the equates for checking virtual keys:

```
*****************************************************************
* Equates for virtual keyboard (@WVRTKB)
*****************************************************************
* ; bit  0: ALPHA LOCK down              0=no  1=yes
* ; bit  1: ENTER                        0=no  1=yes
* ; bit  2: REDO                         0=no  1=yes
* ; bit  3: BACK                         0=no  1=yes
* ; bit  4: Pause                        0=no  1=yes
* ; bit  5: *free*                       0=no  1=yes
* ; bit  6: P1 Left                      0=no  1=yes
* ; bit  7: P1 Right                     0=no  1=yes
* ; bit  8: P1 Up                        0=no  1=yes
* ; bit  9: P1 Down                      0=no  1=yes
* ; bit 10: P1 Space / fire / Q          0=no  1=yes
* ; bit 11: P2 Left                      0=no  1=yes
* ; bit 12: P2 Right                     0=no  1=yes
* ; bit 13: P2 Up                        0=no  1=yes
* ; bit 14: P2 Down                      0=no  1=yes
* ; bit 15: P2 Space / fire / Q          0=no  1=yes
*****************************************************************
KALPHA  EQU   >8000                 ; Virtual key alpha lock
KENTER  EQU   >4000                 ; Virtual key enter
KREDO   EQU   >2000                 ; Virtual key REDO
KBACK   EQU   >1000                 ; Virtual key BACK
KPAUSE  EQU   >0800                 ; Virtual key pause
KFREE   EQU   >0400                 ; ***NOT USED YET***
*---------------------------------------------------------------
* Keyboard Player 1
*---------------------------------------------------------------
K1UPLF  EQU   >0280                 ; Virtual key up   + left
K1UPRG  EQU   >0180                 ; Virtual key up   + right
K1DNLF  EQU   >0240                 ; Virtual key down + left
K1DNRG  EQU   >0140                 ; Virtual key down + right
K1LF    EQU   >0200                 ; Virtual key left
K1RG    EQU   >0100                 ; Virtual key right
K1UP    EQU   >0080                 ; Virtual key up
K1DN    EQU   >0040                 ; Virtual key down
K1FIRE  EQU   >0020                 ; Virtual key fire
*---------------------------------------------------------------
* Keyboard Player 2
*---------------------------------------------------------------
K2UPLF  EQU   >0014                 ; Virtual key up   + left
K2UPRG  EQU   >000C                 ; Virtual key up   + right
K2DNLF  EQU   >0012                 ; Virtual key down + left
K2DNRG  EQU   >000A                 ; Virtual key down + right
K2LF    EQU   >0010                 ; Virtual key left
K2RG    EQU   >0008                 ; Virtual key right
K2UP    EQU   >0004                 ; Virtual key up
K2DN    EQU   >0002                 ; Virtual key down
K2FIRE  EQU   >0001                 ; Virtual key fire
```

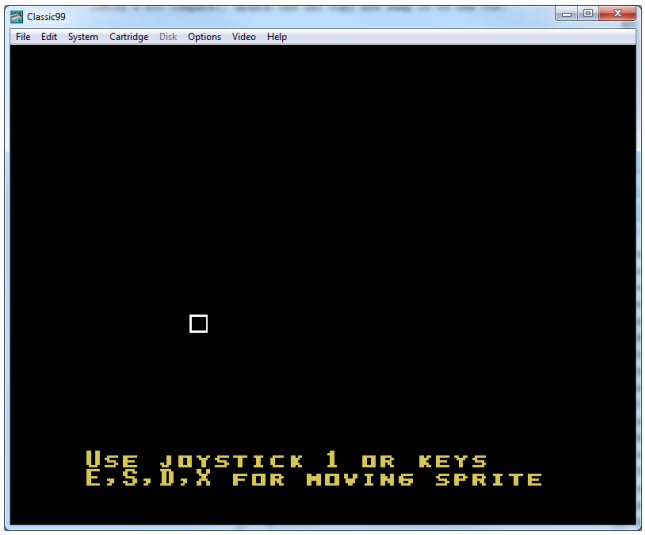
Example

In the next example, we will be moving a sprite using the keyboard or joystick 1.

The main program prepares a copy of the Sprite Attribute Table (SAT) in RAM, displays an information message on screen and allocates a new thread ("MVBOX") with a 1-tick repeat interval.

After the thread scheduler (TMGR) has taken over, it automatically starts the kernel thread and also execute the MVBOX thread every 1/50th or 1/60th of a second.

The keyboard scanning is automatically done by the kernel thread, the MVBOX subroutine only needs to check what virtual keys got pressed (using a bit compare), update the SAT copy and dump it to the VDP.



This example is included as file **example3.a99** in the spectra^2 samples directory.

```
        AORG  >6000
*--------------------------------------------------------------
* Cartridge header
*--------------------------------------------------------------
GRMHDR  BYTE  >AA,1,1,0,0,0
        DATA  PROG
        BYTE  0,0,0,0,0,0,0,0
PROG    DATA  0
        DATA  RUNLIB
HW      BYTE  15
        TEXT  'MOVE THE SPRITE'
        EVEN
*--------------------------------------------------------------
* Include required files
*--------------------------------------------------------------
        COPY  "D:\Projekte\spectra2\tms9900\runlib.a99"
*--------------------------------------------------------------
* SPECTRA2 startup options
*--------------------------------------------------------------
SPVMOD  EQU   GRAPH1               ; Video mode.   See VIDTAB for details.
SPFONT  EQU   FNOPT7               ; Font to load. See LDFNT for details.
SPFCLR  EQU   >A0                  ; Foreground/Background color for font.
SPFBCK  EQU   >01                  ; Screen background color.
*--------------------------------------------------------------
* Our constans and variables in scratchpad memory
*--------------------------------------------------------------
RAMSAT  EQU   >8340                ; Copy of mini-SAT in RAM memory (6 bytes)
RAMTAB  EQU   >8346                ; Timer table (4 bytes)
***************************************************************
* Main
********@*****@*********************@**************************
MAIN    BL    @CPYM2M
        DATA  SPRITE,RAMSAT,6      ; Copy 6 bytes from ROM into scratchpad RAM
        BL    @CPYM2V
        DATA  >1000,PAT1,8         ; Dump sprite pattern
        BL    @PUTBOX
        DATA  >1503,>1A02,INFO,EOL ; Show text in box
        MOV   @MYTAB,@WTITAB       ; Setup address of timer table
        BL    @MKSLOT
        DATA  >0002,MVBOX,EOL      ; Create new timer slot
        B     @TMGR                ; Handle FCTN-QUIT key, timers, etc.
***************************************************************
* THREAD Move sprite: This routine is called from TMGR
********@*****@*********************@**************************
MVBOX   MOV   R11,R0               ; Save R11 in R0
        COC   @WBIT11,CONFIG       ; ANY key pressed ?
        JNE   MVBOX5               ; No, so exit
        MOV   @WVRTKB,R1           ; Get keyboard flags
MVBOX1  COC   @KEY1,R1             ; Left ?
        JNE   MVBOX2
        SB    @BD2,@RAMSAT+1       ; X=X-2
MVBOX2  COC   @KEY2,R1             ; Right ?
        JNE   MVBOX3
        AB    @BD2,@RAMSAT+1       ; X=X+2
MVBOX3  COC   @KEY3,R1             ; Up ?
        JNE   MVBOX4
        SB    @BD2,@RAMSAT         ; Y=Y-2
MVBOX4  COC   @KEY4,R1             ; Down ?
        JNE   MVBOX5
        AB    @BD2,@RAMSAT         ; Y=Y+2
MVBOX5  BL    @CPYM2V              ; Dump copy of SAT to VDP SAT
        DATA  >0300,RAMSAT,6       ; ... R11 is overwritten
        B     *R0                  ; ... so return using copy in R0
KEY1    DATA  K1LF                 ; Left
KEY2    DATA  K1RG                 ; Right
KEY3    DATA  K1UP                 ; Up
KEY4    DATA  K1DN                 ; Down
***************************************************************
* Our constants
********@*****@*********************@**************************
MYTAB   DATA  RAMTAB               ; Location of timer table in scratchpad memory
SPRITE  DATA  >2020,>000F          ; Row >20, col >20, pattern >00, color white
        DATA  >0D00                ; No more sprites
PAT1    DATA  >FF81,>8181,>8181,>81FF
INFO    BYTE  52
        TEXT  'Use joystick 1 or keys    E,S,D,X for moving sprite'
        END
```

Memory / Copy subroutines

MEMORY/COPY

CPYM2M / XPYM2M

Copy ROM/RAM to RAM

CPYM2M – Parameter in DATA statement	
Call format	MYTEST BL @CPYM2M DATA P0,P1,P2
Input	P0 = ROM/RAM source address P1 = RAM destination address P2 = Number of bytes to copy
Example	/samples/example3.a99

XPYM2M – Parameter in register	
Call format	MYTEST BL @XPYM2M
Input	TMP0 = ROM/RAM source address TMP1 = RAM destination address TMP2 = Number of bytes to fill
Example	/samples/????.a99

Description:

Copy a CPU memory range. Source can be either in RAM or ROM. Destination must be RAM. Note that this subroutine uses some machinecode in scratch-pad memory for obtaining the best possible performance.

Example:

Copy 8K of cartridge ROM (>6000 - >7FFF) to high memory (>A000).

```
MAIN                    BL      @CPYM2M
                        DATA    >6000,>A000,8192


                        LI      TMP0,>6000
                        LI      TMP1,>A000
                        LI      TMP2,8192
                        BL      @XPYM2M
```

CPYM2V / XPYM2V

Copy ROM/RAM to VDP VRAM

CPYM2V – Parameter in DATA statement	
Call format	MYTEST BL @CPYM2V DATA P0,P1,P2
Input	P0 = VDP VRAM destination address P1 = ROM/RAM source address P2 = Number of bytes to copy
Example	/samples/example3.a99

XPYM2V – Parameter in register	
Call format	MYTEST BL @XPYM2V
Input	TMP0 = VDP VRAM destination address TMP1 = ROM/RAM source address TMP2 = Number of bytes to fill
Example	/samples/example6.a99

Description:

Copy a CPU memory range to VDP VRAM. Source can be either in RAM or ROM. Destination must be VRAM. Note that this subroutine uses some machinecode in scratch-pad memory for obtaining the best possible performance.
Has basically the same functionality as the Editor/Assembler VMBW utility.

Example:

Copy a color table from ROM to VDP RAM (>0380).

```
MAIN                    BL      @CPYM2V
                        DATA    >0380,COLTAB,16
                        JMP     $
COLTAB                  BYTE    >03,>03,>03,>03,>05,>05,>07,>0F
                        BYTE    >0F,>0F,>0F,>03,>03,>04,>04,>04
```

CPYV2M / XPYV2M

Copy VDP VRAM to RAM

CPYV2M – Parameter in DATA statement	
Call format	MYTEST BL @CPYV2M DATA P0,P1,P2
Input	P0 = VDP VRAM source address P1 = RAM destination address P2 = Number of bytes to copy
Example	/samples/????.a99

XPYV2M – Parameter in register	
Call format	MYTEST BL @XPYV2M
Input	TMP0 = VDP VRAM source address TMP1 = RAM destination address TMP2 = Number of bytes to copy
Example	/samples/????.a99

Description:

Copy a memory range from VDP VRAM to RAM. Note that this subroutine uses some machine code in scratch-pad memory for obtaining the best possible performance.
Has basically the same functionality as the Editor/Assembler VMBR utility.

Example:

Copy 16 bytes from VDP VRAM >0380 to scratchpad RAM >8370.

```
MAIN                    BL      @CPYV2M
                        DATA    >0380,>8370,16
                        JMP     $
```

CPYG2M / XPYG2M

Copy GROM to RAM

CPYG2M – Parameter in DATA statement	
Call format	MYTEST BL @CPYG2M DATA P0,P1,P2
Input	P0 = GROM source address P1 = RAM destination address P2 = Number of bytes to copy
Example	runlib.a99

XPYG2M – Parameter in register	
Call format	MYTEST BL @XPYG2M
Input	TMP0 = GROM source address TMP1 = RAM destination address TMP2 = Number of bytes to copy
Example	/samples/????.a99

Description:

Copy a memory range from GROM to RAM. Note that this subroutine uses some machine code in scratch-pad memory for obtaining the best possible performance.

CPYG2V / XPYG2V

Copy GROM to VDP VRAM

CPYG2V – Parameter in DATA statement	
Call format	MYTEST BL @CPYG2V DATA P0,P1,P2
Input	P0 = GROM source address P1 = VRAM destination address P2 = Number of bytes to copy
Example	/samples/????.a99

XPYG2M – Parameter in register	
Call format	MYTEST BL @XPYG2V
Input	TMP0 = GROM source address TMP1 = VRAM destination address TMP2 = Number of bytes to copy
Example	/samples/????.a99

Description:

Copy a memory range from GROM to VDP VRAM. Note that this subroutine uses some machine code in scratch-pad memory for obtaining the best possible performance.

FILM / XFILM

Fill RAM with byte

FILM – Parameter in DATA statement	
Call format	MYTEST BL @FILM DATA P0,P1,P2
Input	P0 = RAM start address P1 = Byte to fill P2 = Number of bytes to fill
Example	/samples/????.a99

XFILM – Parameter in register	
Call format	MYTEST BL @XFILM
Input	TMP0 = RAM start address TMP1 = Byte to fill TMP2 = Number of bytes to fill
Example	/samples/????.a99

Description:

This routine is used for filling the specified CPU RAM range with a byte value. Note that this subroutine uses some machine code in scratch-pad memory for obtaining the best possible performance.

Example:

Fill high-memory range >A000 - >B000 with byte >FF.

```
MAIN            BL      @FILM
                DATA    >6000,>FF,4096
```

FILV / XFILV

Fill VDP VRAM with byte

FILV – Parameter in DATA statement	
Call format	MYTEST BL @FILV DATA P0,P1,P2
Input	P0 = VDP VRAM start address P1 = Byte to fill P2 = Number of bytes to fill
Example	/samples/example2.a99

XFILV – Parameter in register	
Call format	MYTEST BL @XFILV
Input	TMP0 = VDP VRAM start address TMP1 = Byte to fill TMP2 = Number of bytes to fill
Example	/samples/????.a99

Description:

This routine is used for filling the specified VDP VRAM memory range with a byte value. Note that this subroutine uses some machinecode in scratch-pad memory for obtaining the best possible performance.

Example:

Clear the 16K of VDP VRAM memory (>0000 - >3FFF).

```
MAIN                    BL      @FILM
                        DATA    >0000,>00,16384
```

VDP low-level subroutines

VDWA

Setup VDP write address

VDWA – Parameter in register	
Call format	MYTEST BL @VDWA
Input	TMP0 = VDP VRAM destination address
Example	Runlib.a99

Description:

Setup the VDP destination address for writing. Specify the VDP destination address in register TMP0. Useful if you need to insert some inline VSBW/VMBW code in your subroutine.

VDRA

Setup VDP read address

VDWA – Parameter in register	
Call format	MYTEST BL @VDRA
Input	TMP0 = VDP VRAM destination address
Example	runlib.a99

Description:

Setup the VDP destination address for reading. Specify the VDP destination address in register TMP0. Useful if you need to insert some inline VSBR/VMBR code in your subroutine.

VPUTB / XVPUTB

Write a single byte to VDP VRAM

VPUTB – Parameter in DATA statement	
Call format	MYTEST BL @VPUTB DATA P0,P1
Input	P0 = VDP VRAM destination address P1 = Byte to write
Example	/samples/example4.a99

XVPUTB – Parameter in register	
Call format	MYTEST BL @XVPUTB
Input	TMP0 = VDP VRAM destination address TMP1 = Byte to write
Example	/samples/example5.a99

Dependencies	VDWA

Description:

Write a single byte to VDP VRAM. Has the same functionality as the Editor/Assembler VSBW utility.

VGETB / XVGETB

Read a single byte from VDP VRAM

VGETB – Parameter in DATA statement	
Call format	MYTEST BL @VGETB DATA P0
Input	P0 = VDP VRAM source address
Output	TMP0 = Byte read (in LO-byte)
Example	/samples/????.a99

XVGETB – Parameter in register	
Call format	MYTEST BL @XVGETB
Input	TMP0 = VDP VRAM source address
Output	TMP0 = Byte read (in LO-byte)
Example	/samples/????.a99

Dependencies	VDRA

Description:

Read a single byte from VDP VRAM. Has the same functionality as the Editor/Assembler VSBR utility. The byte read is returned in the low-byte of register TMP0

VIDTAB / XIDTAB

Dump video mode table to VDP registers

VIDTAB – Parameter in DATA statement	
Call format	MYTEST BL @VIDTAB DATA P0
Input	P0 = ROM/RAM address of video mode table
Example	runlib.a99

XIDTAB – Parameter in register	
Call format	MYTEST BL @XIDTAB
Input	TMP0 = ROM/RAM address of video mode table
Example	/samples/????.a99

Description:

Instead of individually loading each of the VDP write-only registers, you can use this subroutine to load all 7 VDP write-only registers at once. For doing so, you need a table holding the required byte value for each of the registers. There are some default video mode tables bundled with the runtime library (e.g. GRAPH1, TXTMOD).

Note that the subroutine also calculates the base address of the pattern name table by checking the value of VDP register #2. It then stores the calculated address in scratchpad memory location WBASE.

Please refer to the TMS9918 VDP programmer's guide for details on the 7 VDP registers.

See section "scratchpad memory setup" on page 23 (item c) for further details on the PNT base address.

Here's a sample video mode table (included in the runtime library):

```
TXTMOD  BYTE  >00,>F2,>00,>0E,>01,>06,>80,SPFCLR
*--------------------------------------------------------------
* Textmode (40 columns)
*--------------------------------------------------------------
* ; VDP#0 Control bits
* ;      bit 6=0: M3 | Graphics 1 mode
* ;      bit 7=0: Disable external VDP input
* ; VDP#1 Control bits
* ;      bit 0=1: 16K selection
* ;      bit 1=1: Enable display
* ;      bit 2=1: Enable VDP interrupt
* ;      bit 3=1: M1 \ TEXT MODE
* ;      bit 4=0: M2 /
* ;      bit 5=0: reserved
* ;      bit 6=1: 16x16 sprites
* ;      bit 7=0: Sprite magnification (1x)
* ; VDP#2 PNT (Pattern name table)       at >0000  (>00 * >400)
* ; VDP#3 PCT (Pattern color table)      at >0380  (>0E * >040)
* ; VDP#4 PDT (Pattern descriptor table) at >0800  (>01 * >800)
* ; VDP#5 SAT (sprite attribute list)    at >0300  (>06 * >080)
* ; VDP#6 SPT (Sprite pattern table)     at >0400  (>80 * >008)
* ; VDP#7 Set foreground/background color
*****************************************************************
```

Example:

Switch the TMS9918 VDP into 40 columns mode (text-mode)

```
MAIN                    BL          @VIDTAB
                        DATA        TXTMOD
                        JMP         $
```

PUTVR / PUTVRX

Load single VDP register with byte

PUTVR – Parameter in DATA statement	
Call format	MYTEST BL @PUTVR DATA P0
Input	P0 = MSB contains the VDP target register LSB contains byte to load
Example	/samples/game/hc_source2.a99

PUTVRX – Parameter in register	
Call format	MYTEST BL @PUTVRX
Input	TMP0 = MSB contains the VDP target register LSB contains byte to load
Example	/samples/????.a99

Description:

Load single VDP write-only register with specified byte. Same functionality as the Editor/Assembler VWTR utility.

PUTV01

Load VDP registers #0 and #1 from R14

PUTVRX – Parameter in register	
Call format	MYTEST BL @PUTV01
Input	R14 = MSB contains byte for VDP register #0 LSB contains byte for VDP register #1
Example	runlib.a99

Dependencies	PUTVRX

Description:

The spectra^2 library uses CPU register R14 for holding a copy of VDP write-only registers #0 and #1. Basically one first sets/resets the corresponding bit masks in R14 and then uses the PUTV01 subroutine for loading the byte values in VDP register #0 and #1.

The high byte of R14 contains a copy of VDP write-only register #0.
The low byte of R14 contains a copy of VDP write-only register #1.

Various features of the VDP are controlled by bit flags in VDP register #0 and #1, e.g. current video mode, sprite magnification, interupt enabling, etc.
Please refer to the TMS9918 VDP programmer's guide for further details.

SCROFF

Turn screen off

SCROFF – No parameter	
Call format	MYTEST BL @SCROFF
Example	/samples/game/hc_source2.a99

Dependencies	PUTV01, PUTVRX

Description:

This subroutine sets bit 1 in VDP write-only register #1 to 0.
As a result the VDP will turn off the screen display and will open a permanent window for CPU access.

You normally use this command before drawing a new screen. Once it is fully drawn, you can then use the SCRON subroutine for turning on the screen again.

Please refer to the TMS9918 VDP programmer's guide for further details.

Note that the corresponding bit in the VDP shadow register (R14) is also set to 0. See page 31 for details on R14.

SCRON

Turn screen on

SCRON – No parameter	
Call format	MYTEST BL @SCRON
Example	/samples/game/hc_source2.a99

Dependencies	PUTV01, PUTVRX

Description:

This subroutine sets bit 1 in VDP write-only register #1 to 1.
As a result the VDP will turn on the screen display again.

You normally call the SCRON subroutine after issuing a SCROFF and doing some screen manipulation.
Please refer to the TMS9918 VDP programmer's guide for further details.

Note that the corresponding bit in the VDP shadow register (R14) is also set to 1. See page 31 for details on R14.

INTOFF

Disable VDP interrupt

INTOFF – No parameter	
Call format	MYTEST BL @INTOFF
Example	/samples/????.a99

Dependencies	PUTV01, PUTVRX

Description:

This subroutine sets bit 2 in VDP write-only register #1 to 0.
As a result the VDP will NOT trigger the CPU interrupt line at the end of the active screen area.
Note that the spectra^2 thread scheduler (TMGR) continuously checks the VDP interrupt flag. **The scheduler will not work if you use INTOFF to disable VDP interrupts.**

Please refer to the TMS9918 VDP programmer's guide for further details.

Note that the corresponding bit in the VDP shadow register (R14) is also set to 0. See page 31 for details on R14.

INTON

Enable VDP interrupt

INTON – No parameter	
Call format	MYTEST BL @INTON
Example	/samples/????.a99

Dependencies	PUTV01, PUTVRX

Description:

This subroutine sets bit 2 in VDP write-only register #1 to 1.
As a result the VDP will trigger the CPU interrupt line at the end of the active screen display area, just before vertical retrace starts.
Note, that you can still mask the CPU interrupt by using the "LIMI 0" instruction.
This is the default setting when spectra^2 is initialized.

Please refer to the TMS9918 VDP programmer's guide for further details.

Note that the corresponding bit in the VDP shadow register (R14) is also set to 1. See page 31 for details on R14.

SMAG1X

Set sprite magnification 1X

SMAG1X – No parameter	
Call format	MYTEST BL @SMAG1X
Example	/samples/????.a99

Dependencies	PUTV01, PUTVRX

Description:

This subroutine sets bit 7 in VDP write-only register #1 to 0.
As a result the VDP will remove the sprite magnification,

Please refer to the TMS9918 VDP programmer's guide for further details.

Note that the corresponding bit in the VDP shadow register (R14) is also set to 0. See page 31 for details on R14.

SMAG2X

Set sprite magnification 2X

SMAG2X – No parameter	
Call format	MYTEST BL @SMAG2X
Example	/samples/????.a99

Dependencies	PUTV01, PUTVRX

Description:

This subroutine sets bit 7 in VDP write-only register #1 to 1.
As a result the VDP will install sprite magnification. This means that 8x8 sprites become 16x16 and 16x16 sprites become 32x32.

Please refer to the TMS9918 VDP programmer's guide for further details.

Note that the corresponding bit in the VDP shadow register (R14) is also set to 1. See page 31 for details on R14.

S8X8

Set sprite size to 8x8 pixels

S8X8 – No parameter	
Call format	MYTEST BL @S8X8
Example	/samples/????.a99

Dependencies	PUTV01, PUTVRX

Description:

This subroutine sets bit 6 in VDP write-only register #1 to 0.
As a result the VDP will set the sprite size to 8x8 pixels. It means that you need 8 bytes to define a sprite pattern.

Please refer to the TMS9918 VDP programmer's guide for further details.

Note that the corresponding bit in the VDP shadow register (R14) is also set to 0. See page 31 for details on R14.

S16X16

Set sprite size to 16x16 pixels

S16X16 – No parameter	
Call format	MYTEST BL @S16X16
Example	/samples/????.a99

Dependencies	PUTV01, PUTVRX

Description:

This subroutine sets bit 6 in VDP write-only register #1 to 1.
As a result the VDP will set the sprite size to 16x16 pixels. It means that you need 32 bytes to define a sprite pattern.

Please refer to the TMS9918 VDP programmer's guide for further details.

Note that the corresponding bit in the VDP shadow register (R14) is also set to 1. See page 31 for details on R14.

GTCLMN

Get number of columns per row

GTCLMN – No parameter	
Call format	MYTEST BL @S16X16
Output	TMP0 = Number of columns per row (32, 40, 64)
Example	runlib.a99

Description:

This subroutine checks the bit masks of the bytes in CPU register R14 (copy of VDP#0 & VDP#1) to determine how many columns there are in a row. This routine is used by some of the other VDP subroutines in spectra^2.

YX2PNT

Get VDP Pattern-Name-Table address for cursor YX position

YX2PNT - Parameter in memory location	
Call format	MYTEST BL @YX2PNT
Input	@WYX
Output	TMP0 = VDP destination address
Example	/samples/example4.a99

Description:

This subroutine calculates the VDP address of the entry in the Pattern Name Table that matches with the cursor YX position (@WYX). The formula used is:

VDP address = @WBASE + (Y * columns per row) + X

Note that the memory location @WBASE holds the VRAM base address of the VDP Pattern Name Table.
The subroutine checks the bit masks of the 2 bytes that make up CPU register R14 (copy of VDP#0 & VDP#1) to determine how many columns there are in a row. This routine is used by some of the other VDP subroutines in spectra^2.
You can use multiple "virtual screens" by first loading @WBASE with the address of another PNT table.

Please refer to the TMS9918 VDP programmer's guide for further details on the Pattern Name Table.

YX2PX / YX2PXX

Get pixel position for cursor YX position

YX2PX – Parameter in memory location	
Call format	MYTEST BL @YX2PX
Input	@WYX = YX value-pair (CONFIG:0 = 1) = Skip sprite adjustment
Output	TMP0HB = Y pixel position TMP0LB = X pixel position
Example	/samples/example4.a99

YX2PXX – Parameter in register	
Call format	MYTEST BL @YX2PXX
Input	TMP0 = YX value-pair (CONFIG:0 = 1) = Skip sprite adjustment
Output	TMP0HB = Y pixel position TMP0LB = X pixel position
Example	/samples/example4.a99

Description:

This subroutine converts the tile based cursor YX position into the corresponding Y,X pixel coordinates using the below formula:

Pixel Y = (Tile Y) * 8

Pixel X = (Tile X) * 8

On subroutine exit, the most significant byte of register TMP0 will contain the Y pixel position and the least significant byte of register TMP0 will contain the X pixel position.

The functionality is useful for setting the sprite position based on the position of a tile.

Note that for sprites the top of screen is at >FF, not at >00. The subroutine automatically does the necessary adjustment. This feature can be turned off by setting bit 0 in the CONFIG register.

Also note that the subroutine does not support multicolor and text mode.

Please refer to the TMS9918 VDP programmer's guide for further details on the Sprite Attribute Table.

PX2YX

Get tile YX position for pixel YX position

PX2YX - Parameter in register	
Call format	MYTEST BL @PX2YX
Input	TMP0 = YX value-pair (CONFIG:0 = 1) = Skip sprite adjustment
Output	TMP0HB = Y tile position TMP0LB = X tile position TMP1HB = Y pixel offset TMP1LB = X pixel offset
Example	/samples/????.a99

Description:

This subroutine converts a -sprite- YX pixel position into the corresponding Y,X tile coordinates using the below formula:

Tile Y = (Pixel Y) / 8

Tile X = (Pixel X) / 8

Offset Y = (Pixel Y) modulus 8

Offset X = (Pixel X) modulus 8

On subroutine exit, the most significant byte of register TMP0 will contain the Y tile position and the least significant byte of register TMP0 will contain the X tile position.

The most significant byte of register TMP1 contains the Y offset.
The least significant byte of register TMP1 contains the X offset.
Both the Y and X offset are expressed in pixels.

The functionality is useful for setting a character tile based on the position of a sprite.

Note that for sprites the top of screen is at >FF, not at >00. The subroutine automatically does the necessary adjustment. This feature can be turned off by setting bit 0 in the CONFIG register.

Also note that the subroutine does not support multicolor and text mode.

Please refer to the TMS9918 VDP programmer's guide for further details on the Sprite Attribute Table.

VDP tiles & patterns subroutines

LDFNT

Load TI-99/4A character font from GROM into VRAM

LDFNT – Parameter in DATA statement	
Call format	MYTEST BL @LDFNT DATA P0,P1
Input	P0 = VDP VRAM destination address P1 = Font options
Example	runlib.a99

Description:

The LDFNT subroutine is used to copy the built-in character font from the TI-99/4A operating system GROMs into VDP VRAM memory.
We can save valuable ROM space by using the fonts available in the TI-99/4A itself. Note that it's also possible to apply a "bold" effect to the fonts. That way you get a new font that looks nice for games.

Parameter P0 must contain the VDP destination address.

Below are the possible values for parameter P1.

```
FNOPT1  EQU   >0000      ; LDFNT => Load TI title screen font
FNOPT2  EQU   >0006      ; LDFNT => Load upper case font
FNOPT3  EQU   >000C      ; LDFNT => Load upper/lower case font
FNOPT4  EQU   >0012      ; LDFNT => Load lower case font
FNOPT5  EQU   >8000      ; LDFNT => Load TI title screen font  & make fat
FNOPT6  EQU   >8006      ; LDFNT => Load upper case font       & make fat
FNOPT7  EQU   >800C      ; LDFNT => Load upper/lower case font & make fat
FNOPT8  EQU   >8012      ; LDFNT => Load lower case font       & make fat
```

The LDFNT routine is automatically called when the spectra^2 library is initalized.
Please also see details on the SPFONT equate in the "Library startup options" section on page 21.

PUTSTR

Put length-byte prefixed string at cursor position

PUTSTR – Parameter in DATA statement	
Call format	MYTEST BL @PUTSTR DATA P0
Input	P0 = Pointer to length-byte prefixed string @WYX = Cursor YX position
Example	/samples/????.a99

XUSTSTR – Parameter in register	
Call format	MYTEST BL @XUTSTR
Input	TMP0 = Pointer to length-byte prefixed string @WYX = Cursor YX position
Example	/samples/????.a99

Dependencies	YX2PNT, XPYM2V

Description:

The PUTSTR subroutine is used to display a length-byte prefixed string at the current cursor position (@WYX). Both rows and columns start with 0.
In other words: the 1st row, 1st column is at YX position 0,0.

Parameter P0 must contain the address of the string to display.
The first byte of that string must contain the string length.

The subroutine supports string with a maximum length of 255 characters. There are no boundary checks. It is for example possible to display a string on row 85. That makes it possible to do some cool effects when working with multiple "virtual screens".

Example:

Display string "Hello World" on row 5, column 15

```
MAIN    LI    R0,>050F    ; Y=5, X=15
        MOV   R0,@WYX     ; Load cursor
        BL    @PUTSTR     ; Display string
        DATA  HELLOW
        B     @TMGR       ; Handle FCTN-QUIT key, etc.
HELLOW  BYTE  12
        TEXT  'HELLO WORLD!'
        END
```

PUTAT

Put length-byte prefixed string at position Y,X

PUTAT – Parameter in DATA statement	
Call format	MYTEST BL @PUTAT DATA P0
Input	P0 = YX position P1 = Pointer to length-byte prefixed string
Example	/samples/example1.a99

Dependencies	PUTSTR, YX2PNT, XPYM2V

Description:

The PUTAT subroutine is used to display a length-byte prefixed string at the cursor position specified in parameter P0.

The most-significant byte of parameter P0 must contain the row value, the least-significant byte of P0 contains the column value. Both rows and columns start with 0.
In other words: the 1st row, 1st column is at YX position 0,0.
Note that this subroutine overwrites the cursor YX position (@WYX).

Parameter P1 must contain the address of the string to display.
The first byte of that string must contain the string length.
The subroutine supports strings with a maximum length of 255 characters. There are no boundary checks. It is for example possible to display a string on row 85. That makes it possible to do some cool effects when working with multiple "virtual screens".

Example:

Display string "Hello World" on row 5, column 15

```
MAIN    BL    @PUTSTR    ; Display string
        DATA  >050F,HELLOW
        B     @TMGR      ; Handle FCTN-QUIT key, etc.
HELLOW  BYTE  12
        TEXT  'HELLO WORLD!'
        END
```

HCHAR

Repeat characters horizontally at position Y,X

HCHAR – Parameter in DATA statement	
Call format	MYTEST BL @HCHAR DATA P0,P1 … DATA EOL
Input	P0 = YX position P1 = MSB: Character to write LSB: Number of times to repeat
Example	/samples/example2.a99

Dependencies	YX2PNT, XFILV

Description:

The HCHAR subroutine is comparable to the TI-Basic CALL HCHAR statement. It repeats characters horizontally.

The most-significant byte of parameter P0 must contain the row value, the least-significant byte of P0 must contain the column value. Both rows and columns start with 0.
In other words: the 1st row, 1st column is at YX position 0,0.
Note that this subroutine overwrites the cursor YX position (@WYX).

The most-significant byte of Parameter P1 must contain the character to write. The least-significant byte of Parameter P1 must contain the number of times the character should be repeated.

The HCHAR subroutine expects a list of parameters. With one HCHAR call you can draw multiple horizontal lines. **You need to specify the End-Of-List marker in the last DATA statement by using the EOL equate.**

Also note that there are no boundary checks. It is for example possible to do a HCHAR on row 85. This feature is especially useful when working with multiple "virtual screens".

VCHAR

Repeat characters vertically at position Y,X

VCHAR – Parameter in DATA statement	
Call format	MYTEST BL @VCHAR DATA P0,P1 … DATA EOL
Input	P0 = YX position P1 = MSB: Character to write LSB: Number of times to repeat
Example	/samples/game/hc_source2.a99

Dependencies	GTCLMN, YX2PNT

Description:

The VCHAR subroutine is comparable to the TI-Basic CALL VCHAR statement. It repeats characters vertically.

The most-significant byte of parameter P0 must contain the row value. The least-significant byte of P0 must contain the column value. Both rows and columns start with 0.
In other words: the 1st row, 1st column is at YX position 0,0.
Note that this subroutine overwrites the cursor YX position (@WYX).

The most-significant byte of Parameter P1 must contain the character to write. The least-significant byte of Parameter P1 must contain the number of times the character should be repeated.

The VCHAR subroutine expects a list of parameters. With one VCHAR call you can draw multiple vertical lines. **You need to specify the End-Of-List marker in the last DATA statement by using the EOL equate.**

Also note that there are no boundary checks. It is for example possible to do a VCHAR on row 85. This feature is especially useful when working with multiple "virtual screens".

FILBOX

Fill box with characters at position Y,X

FILBOX – Parameter in DATA statement	
Call format	MYTEST BL @FILBOX DATA P0,P1 … DATA EOL
Input	P0HB = Upper left corner Y P0LB = Upper left corner X P1HB = Width P1LB = Height P2HB = >00 P2LB = Character to fill
Example	/samples/????.a99

Dependencies	YX2PNT, XFILV

Description:

The FILBOX subroutine fills the specified rectangular area with characters.

The most-significant byte of parameter P0 must contain the row value. The least-significant byte of P0 must contain the column value. Both rows and columns start with 0.
In other words: the 1st row, 1st column is at YX position 0,0.
Note that this subroutine overwrites the cursor YX position (@WYX).

The most-significant byte of parameter P1 specifies the width of the area. The least-significant byte of parameter P1 specifies the height of the area.

The most-significant byte of parameter P2 should be set to the byte value >00 and is not used. The least-significant byte of parameter P2 specifies the character for filling the area.
The FILBOX subroutine handles multiple data statements. **You need to specify the End-Of-List marker in the last DATA statement by using the EOL equate.**

Also note that there are no boundary checks. It is for example possible to do a FILBOX call for row 85. This feature is especially useful when working with multiple "virtual screens".

PUTBOX

Put length-prefixed string in box at position Y,X

PUTBOX – Parameter in DATA statement	
Call format	MYTEST BL @PUTBOX DATA P0,P1 … DATA EOL
Input	P0HB = Upper left corner Y P0LB = Upper left corner X P1HB = Width P1LB = Height P2 = Pointer to length-prefixed string
Example	/samples/game/hc_source2.a99

Dependencies	YX2PNT, XFILV

Description:

The PUTBOX subroutine fills the specified rectangular area with the length-prefixed string.

The most-significant byte of parameter P0 must contain the row value. The least-significant byte of P0 must contain the column value. Both rows and columns start with 0.
In other words: the 1st row, 1st column is at YX position 0,0.
Note that this subroutine overwrites the cursor YX position (@WYX).

The most-significant byte of parameter P1 specifies the width of the area. The least-significant byte of parameter P1 specifies the height of the area.

Parameter P2 must contain the address of the string to display in the area. The first byte of that string must contain the string length. The subroutine supports string with a maximum length of 255 characters.

Note that if the string is too short for filling the whole rectangular area, it will be automatically repeated until it fits.

The PUTBOX subroutine handles multiple data statements. **You need to specify the End-Of-List marker in the last DATA statement by using the EOL equate.**
Also note that there are no boundary checks. It is for example possible to do a PUTBOX call for row 85. This feature is especially useful when working with multiple "virtual screens".

MKNUM

Convert unsigned number to right-justified string

MKNUM – Parameter in DATA statement	
Call format	MYTEST BL @MKNUM DATA P0,P1,P2
Input	P0 = Pointer to 16 bit unsigned number P1 = Pointer to 5 byte string buffer P2HB = Offset for ASCII digit P2LB = Character for replacing leading 0's Optional (CONFIG:0 = 1) = Display number at cursor YX @WYX = Cursor YX position
Output	5 byte string buffer will contain converted number
Example	/samples/????.a99

Dependencies	XUTSTR

Description:

The MKNUM subroutine converts a 16 bit unsigned number (0-65535) into a right-justified string.

Parameter P0 must contain the address of the memory location holding the 16 bit unsigned number.

Parameter P1 must contain the address of a working buffer in RAM (5 bytes). This buffer will also contain the generated string.

The most-significant byte of parameter P2 must contain the ASCII offset for digit 0. The offset depends on what ASCII characters you use for holding the digits 0-9. If you for example load patterns for 0-9 overriding characters A-J, then you would load the byte value >41 (decimal 65).
This functionality is useful, if you have multiple characters sets for displaying a score (e.g. with different colours) or if you relocated the digits to a more suitable location in the pattern table.

The least-significant byte of parameter P2 must contain the ASCII value of the padding character. This character will be used for replacing the leading 0's. That could for example be a white-space character or the ASCII value of the character holding digit 0.
Suppose you have the value "123". Using the MKNUM subroutine you could convert it to the string "00123" or " 123".

Following equates are available for parameter P2:

```
NUM1    EQU   >3030                ; MKNUM  => ASCII 0-9, leading 0's
NUM2    EQU   >3020                ; MKNUM  => ASCII 0-9, leading spaces
```

You can optionally display the generated string at the current cursor YX position by setting bit 0 in the CONFIG register.

PUTNUM

Put unsigned number on screen

PUTNUM – Parameter in DATA statement	
Call format	MYTEST BL @PUTNUM DATA P0,P1,P2,P3
Input	P0 = YX position P1 = Pointer to 16 bit unsigned number P2 = Pointer to 5 byte string buffer P3HB = Offset for ASCII digit P3LB = Character for replacing leading 0's
Output	5 byte string buffer will contain converted number
Example	/samples/example5.a99

Dependencies	MKNUM, XUTSTR

Description:

The PUTNUM subroutine converts a 16 bit unsigned number (0-65535) into a right-justified string and displays it on screen.

The most-significant byte of parameter P0 must contain the row value.
The least-significant byte of P0 must contain the column value.
Both rows and columns start with 0.
In other words: the 1st row, 1st column is at YX position 0,0.
Note that this subroutine overwrites the cursor YX position (@WYX).

Parameter P1 must contain the address of the memory location holding the 16 bit unsigned number.
Parameter P2 must contain the address of a working buffer in RAM (5 bytes). This buffer will also contain the generated string.

The most-significant byte of parameter P3 must contain the ASCII offset for digit 0. The offset depends on what ASCII characters you use for holding the digits 0-9. If you for example load patterns for 0-9 overriding characters A-J, then you would load the byte value >41 (decimal 65).

This functionality is useful, if you have multiple characters sets for displaying a score (e.g. with different colors) or if you relocated the digits to a more suitable location in the pattern description table.

The least-significant byte of parameter P3 must contain the ASCII value of the padding character. This character will be used for replacing the leading 0's. That could for example be a white-space character or the ASCII value of the character holding digit 0.
Suppose you have the value "123". Using the PUTNUM subroutine you could display the string "00123" or " 123".

Following equates are available for parameter P3:

```
NUM1    EQU   >3030                ; MKNUM  => ASCII 0-9, leading 0's
NUM2    EQU   >3020                ; MKNUM  => ASCII 0-9, leading spaces
```

Sound & speech subroutines

MUTE

Mute all sound generators and clear sound pointer

MUTE – No parameter	
Call format	MYTEST BL @MUTE
Example	/samples/game/hc_source2.a99

Description:

The MUTE subroutine is used for muting all sound generators. It additionally clears memory location @WSDLST (address of tune currently playing) and turns off the sound player by resetting bit 13 in the CONFIG register.

For further details please refer to the SDPREP and SDPLAY subroutines.

MUTE2

Mute all sound generators

MUTE2 – No parameter	
Call format	MYTEST BL @MUTE2
Example	/samples/game/hc_source2.a99

Description:

The MUTE2 subroutine is used for muting all sound generators. It additionally turns off the sound player by resetting bit 13 in the CONFIG register.
However, subroutine MUTE2 does not clear memory location (@WSDLST).
So due to this, you basically use MUTE2 for pausing the sound player.

For further details please refer to the SDPREP and SDPLAY subroutines.

SDPREP

Prepare for playing sound

SDPREP – Parameter in DATA statement	
Call format	MYTEST BL @SDPREP DATA P0,P1
Input	P0 = Address of tune P1 = Option flags for sound player
Example	/samples/game/hc_source1.a99

Description:

The SDPREP subroutine initializes the CONFIG register bits 13-15 and sets some memory addresses (@WSDLST, @WSDTMP) used by the built-in sound player. It also loads the least-significant byte of R13 with 1. The sound player (SDPLAY) itself is automatically called by the kernel background thread on each VDP interrupt.

Parameter P0 contains the address of the tune to play. Note that the tune data must already be present in either ROM/RAM or VRAM.
The sound table format is compatible to the format supported by the ISR sound routine found in the console ROM.

Parameter P1 contains the option flags for the tune. It specifies if the tune should be played from ROM/RAM or VRAM. Additionally it specifies if the tune should automatically start over when finished.

The below equates are available for parameter P1

```
SDOPT1  EQU   7     ; SDPLAY => 111 (Player on, repeat, tune in CPU memory)
SDOPT2  EQU   5     ; SDPLAY => 101 (Player on, no repeat, tune in CPU memory)
SDOPT3  EQU   6     ; SDPLAY => 110 (Player on, repeat, tune in VRAM)
SDOPT4  EQU   4     ; SDPLAY => 100 (Player on, no repeat, tune in VRAM)
```

Please refer to the Editor/Assembler manual page 312 for details on the ISR sound table format.

SDPLAY

Run the sound player

SDPLAY – No parameter	
Call format	MYTEST BL @SDPLAY
Example	-

Description:

The SDPLAY subroutine is the built-in sound player. It is normally automatically called by the background kernel thread on each VDP interrupt. It means this code is executed 60 times a second on NTSC and 50 times a second on a PAL machine.
The sound format is compatible to the sound format of the ISR sound routine found in the console ROM.
It's still possible to call the SDPLAY subroutine from your program in case you are not using the background kernel thread. That'd allow for some custom effects like slowing down or speeding up a tune.

The SDPREP subroutine must be used for setting up memory before a tune can be played.

The sound player uses bit 13,14,15 in the CONFIG register. You can turn off the sound player by setting bit 13 to 0. You have to use the MUTE subroutine if a tune is already in progress.

Please refer to the Editor/Assembler manual page 312 for details on the ISR sound table format.

SPSTAT

Read status register byte from speech synthesizer

SPSTAT – No parameter	
Call format	MYTEST LI TMP2,MYRET B @SPSTAT
Output	MSB TMP0 = speech synth status code
Example	runlib.a99

Description:

The SPSTAT subroutine is used for checking the speech synth FIFO buffer fill grade. You normally do not need to run this subroutine in your program, as it's automatically handled by the built-in speech player (SPPLAY).
Nonetheless, should you need to call the SPSTAT subroutine, you'll have to use "B @SPSTAT" after loading register TMP2 with the return address to branch to upon subroutine exit.

Upon exit register TMP0 will contain the speech synthesizer status code.

Note that the 32K memory expansion is not available when the speech synthesizer status register is accessed. Therefore the SPSTAT subroutine loads and executes some machine code in scratchpad memory (>8320 - >8327).

Please refer to the Editor Assembler manual, section 22 page 352 for further details on using speech on the TI-99/4A.
Also see the TMS5220 Speech Synthesizer Data Manual, section 5.2 (FIFO Buffer) and section 5.4 (Status Register)

SPCONN

Check if speech synthesizer is connected

SPCONN – No parameter	
Call format	MYTEST BL @SPCONN
Output	MSB TMP0 = speech synth status code
Example	/samples/????.a99

Dependencies	SPSTAT

Description:

The SPCONN subroutine is used for checking if the speech synthesizer is connected. Upon exit the most-significant byte of register TMP0 will contain the speech synthesizer status code.
The latter will equal >AA if a speech synthesizer is connected.

You normally do not need to call this subroutine in your program. The RUNLIB subroutine does that for you upon library initialisation and stores the results in bit 3 of the CONFIG register.

For further details please refer to section 22.1.6 page 354 in the Editor/Assembler manual.
Please refer to the library initialisation section for further details on RUNLIB and the CONFIG register usage.

SPPREP

Prepare for playing speech

SPPREP – Parameter in DATA statement	
Call format	MYTEST BL @SPPREP DATA P0,P1
Input	P0 = Address of LPC speech data P1 = Speech player options
Example	/samples/example2.a99

Description:

The SPPREP subroutine prepares memory and the CONFIG register for playing speech. It loads the value of parameter P0 into memory location (@WSPEAK) and sets the CONFIG bits 3-5 according to the value specified in parameter P1.

The speech player (SPPLAY) itself is automatically called by the thread scheduler routine (TMGR).

Parameter P0 specifies the memory address (ROM/RAM) where the LPC encoded speech data can be found if P1 equals SPOPT1.

Parameter P1 specifies the speech player operating mode.

The below equates are available for parameter P1

```
SPOPT1  EQU   >1400                ; 0001010000000000 (Player on, external voice)
SPOPT2  EQU   >1000                ; 0001000000000000 (Player on, resident voice)
```

Note that in the current spectra^2 version the speech player (subroutine SPPLAY) only supports external voice mode (P1=SPOPT1).

SPPLAY

Run the speech player

SPPLAY – No parameter	
Call format	MYTEST BL @SPPLAY
Example	-

Description:

The SPPLAY subroutine is the built-in speech player. You normally do not need to call the SPPLAY subroutine from your program. This is automatically handled in the background by the thread scheduler (TMGR).
Communicating with the speech synthesizer device is very critical as far as timing is concerned. That is why the speech player code is called from inside the thread scheduler code itself.

The SPPREP subroutine must be used for setting up memory before speech can be activated.

The speech player (SPPLAY) included in spectra^2 supports multiple operating modes:

- **Playback recorded speech from an external source**
  In that case the LPC encoded voice data is either available in cartridge ROM or loaded into RAM.

- **Speak words from resident vocabulary**
  The LPC data is present in a ROM mounted inside the speech synthesizer device itself.

Note that the speech player uses bit 3,4,5 in the CONFIG register.
The speech player can be turned off by setting bit 3 in the CONFIG register to 0.

Note that in the current spectra^2 version the speech player (subroutine SPPLAY) only supports external voice mode (P1=SPOPT1).

Keyboard & joystick subroutines

KEYBOARD & JOYSTICKS

VIRTKB

The virtual keyboard implementation

KBSCAN – No parameter	
Call format	MYTEST BL @KBSCAN
Output	@WVRTKB
Example	runlib.a99

Description:

Spectra^2 knows the concept of a "virtual keyboard". It basically maps most game keys and joysticks 1 and 2 on a bit mask. The concept used to accomplish this is explained in the "Virtual Keyboard" section. Check there for examples, etc.

Normally there is no need to call the KBSCAN from your program. It's automatically handled by the background kernel thread (KERNEL) which is part of the runtime library.

The VIRTKB subroutine uses bit 11-12 in the CONFIG register. Upon completion, the keys presses are stored as bit flags in the memory word @WVRTKB.

Thread scheduler subroutines

Thread Scheduler

TMGR

The thread scheduler

TMGR – No parameter	
Call format	MYTEST B @TMGR
Example	/samples/example2.a99

Description:

The TMGR subroutine is the spectra^2 thread scheduler. It's pretty much the main subroutine responsible for running background jobs such as the kernel thread and any additional threads started by the user.

The "Thread Scheduler" section explains in detail how the scheduler works and how to use it. Check there for examples, etc.

You can start the scheduler with "**B @TMGR**" after initialisation in the main program has completed.

Make sure you checked the below before initiating TMGR, it will save you a lot of time searching for program crashes:

- Memory address WTITAB (2 bytes) set with address of your timer table.
- Timer table initialized with >00 bytes.
- Memory address BTIHI (1 byte!) set with highest timer slot in use.

MKSLOT

Allocate timer slots

TMGR – Parameter in DATA statement	
Call format	MYTEST BL @MKSLOT DATA P0,P1 … DATA EOL
Input	P0HB = Slot number P0LB = Repeat interval P1 = Subroutine to call
Example	/samples/example5.a99

Description:

The MKSLOT subroutine is used to allocate timer slots for running threads. The subroutine allows you to allocate non-sequential slots, e.g. allocate slots 0,3,4,7 (without touching slots 1,2,5,6).
See the "Threads" sections for details on timer table layout.

The most significant byte of parameter P0 is the slot number to use. The amount of available slots is determined by the size of the timer table in RAM memory.

The least significant byte of parameter P0 determines the interval at which the task scheduler should run the subroutine specified in parameter P1. The value for the interval is defined in ticks per second.

Parameter P1 contains the address of the subroutine to call via BL when the slot is fired.

The MKSLOT subroutine handles multiple data statements. **You need to specify the End-Of-List marker in the last DATA statement by using the EOL equate.**

Make sure that you set the memory word @WTITAB (2 bytes) with the address of your timer table before calling MKSLOT the first time.
Don’t forget to update the memory location @WBTIHI (1 byte!) with the highest slot in use.

Note that if you have many slots to allocate at once, you could copy a preset slot table from ROM into RAM without using the MKSLOT subroutine

CLSLOT

Clear allocated timer slot

CLSLOT – Parameter in DATA statement	
Call format	MYTEST BL @CLSLOT DATA P0
Input	P0 = Slot number
Example	/samples/game/hc_source2.a99

XLSLOT – Parameter in register	
Call format	MYTEST BL @XLSLOT
Input	TMP0 = Slot number
Example	/samples/????.a99

Description:

Use the CLSLOT subroutine to remove a single running slot. It means that the subroutine marked in the specified slot will no longer be executed.

Note that using CLSLOT does not re-arrange the remaining slots in the timer table. Due to this you can get holes in the timer table over time. It's pretty much the responsibility of the programmer to keep track of what slots can be reused for new threads.

Parameter P0 must contain the slot number of the slot to clear.

KERNEL

The kernel thread

KERNEL – No parameter	
Call format	MYTEST B @KERNEL
Input	P0 = Slot number
Example	-

Description:

The KERNEL subroutine is used for doing certain basic background tasks such as running the sound player (SDPLAY) and scanning the virtual keyboard (VIRTKB). You can't call the KERNEL subroutine directly from your program. It's completely controlled by the Thread Scheduler code (TMGR).

The kernel thread can be deactivated by resetting bit 9 in the CONFIG register.

Please refer to the "Threads" section for further details on the kernel thread.

MKHOOK

Allocate the user hook

MKHOOK – Parameter in DATA statement	
Call format	MYTEST BL @MKHOOK DATA P0
Input	P0 = Address of user hook
Example	/samples/game/hc_source2.a99

Description:

The MKHOOK subroutine is responsible for allocating the user hook.

The idea is that you use the user hook for stuff that isn't bound to the VDP interrupt and that needs to be executed very often (more than 50 or 60 times a second), e.g. checking the coincidence flag in the VDP status register.

Parameter P0 contains the address of the user hook, a user-supplied subroutine that is executed each time the VDP status register is read.

The MKHOOK routine will move the address in P0 to memory location @WHOOK. It then sets bit 7 and resets bit 8 in the CONFIG register.

Note that the user hook code must always exit with a "B @HOOKOK" for returning to the thread scheduler.

Please refer to the "Threads" section for the full details on the user hook concept.

Miscellaneous subroutines

POPR(0-3) or POPRT

Pop registers & return to caller

POPR(0-3) or POPRT – No parameter	
Call format	MYTEST B @POPR3 MYTEST B @POPR2 MYTEST B @POPR1 MYTEST B @POPR0 MYTEST B @POPRT
Example	/samples/????.a99

Description:

These routines pop the specified registers from the stack and then returns to the caller. It expects that the return address (R11) is at the bottom.

Use POPRT if you only want to pop R11 and return.

Note that –by default- STACK is an equate for R9.

See the "Stack" section on page 40 for further details.

Example:

Suppose you have a subroutine MYTEST that changes R0 and R1. You want to make sure that upon subroutine exit R0 and R1 keep their original values.

```
MAIN            LI      R0,15
                LI      R1,22
                BL      @MYTEST         ; Upon return; R0=15, R1=22
                JMP     $               ; Soft halt
MYTEST          DECT    STACK
                MOV     R11,*STACK      ; Push R11 (return address)
                DECT    STACK
                MOV     R0,*STACK       ; Push R0 on stack (value 15)
                DECT    STACK
                MOV     R1,*STACK       ; Push R1 on stack (value 22)
                LI      R0,99           ; Overwrite R0
                CLR     R1              ; Overwrite R1
                B       @POPR1          ; Pop R1,R0,R11 from stack and return
```

RND / RNDX

Generate random number

RND – Parameter in DATA statement	
Call format	MYTEST BL @RND DATA P0
Input	P0 = Highest random number allowed
Output	TMP0 = Random number
Example	/samples/example5.a99

RNDX – Parameter in register	
Call format	MYTEST BL @RNDX
Input	TMP0 = Highest random number allowed
Output	TMP0 = Random number
Example	/samples/example5.a99

Description:

The RND subroutine generates a new random number in the range between 0 and P0. The subroutine uses and updates the seed value stored in memory location @WSEED.

Parameter P0 must contain the highest number allowed.
The generated random number is returned in register TMP0.

The seed value in memory location @WSEED is populated for the first time when the library gets initialized. The value is copied from scratch-pad memory location @>83C0 which is set by the monitor OS.

The original seed value is based on a counter waiting for a key-press in the TI selection screen.

RUNLIB

Initialize spectra^2 runtime library

RND – No parameter	
Call format	MYTEST B @RUNLIB
Example	/samples/example1.a99

Dependencies	CPYM2M, CPYG2M, FILV, MUTE, VIDTAB, LDFNT

Description:

The RUNLIB subroutine initializes the spectra^2 runtime library. It must be the first thing that gets executed when a program is started. It does many tasks as clearing RAM and VDP VRAM memory, setting the VDP in a defined state, checking the console it's running on, etc.

It will jump to the main program (label MAIN), once it has completed the initialisation process.

For the full details please refer to the "Runtime library initialisation" section.

Appendix:
examples & source code

```
********@*****@**********************@**************************
        AORG  >6000
*--------------------------------------------------------------
* Cartridge header
*--------------------------------------------------------------
GRMHDR  BYTE  >AA,1,1,0,0,0
        DATA  PROG
        BYTE  0,0,0,0,0,0,0,0
PROG    DATA  0
        DATA  RUNLIB
HW      BYTE  12
        TEXT  'HELLO WORLD!'
*--------------------------------------------------------------
* Include required files
*--------------------------------------------------------------
        COPY  "D:\Projekte\spectra2\tms9900\runlib.a99"
*--------------------------------------------------------------
* SPECTRA2 startup options
*--------------------------------------------------------------
SPVMOD  EQU   GRAPH1                ; Video mode.   See VIDTAB for details.
SPFONT  EQU   FNOPT7                ; Font to load. See LDFNT  for details.
SPFCLR  EQU   >F0                   ; Foreground/Background color for font.
SPFBCK  EQU   >08                   ; Screen background color.
***************************************************************
* Main
********@*****@**********************@**************************
MAIN    BL    @PUTAT
        DATA  >0B0A,HW              ; Show "Hello World!" message on row >0B, column >0A
        B     @TMGR                 ; Handle FCTN-QUIT key, etc.
        END
```

```
        AORG  >6000
*--------------------------------------------------------------
* Cartridge header
*--------------------------------------------------------------
GRMHDR  BYTE  >AA,1,1,0,0,0
        DATA  PROG
        BYTE  0,0,0,0,0,0,0,0
PROG    DATA  0
        DATA  RUNLIB
HW      BYTE  12
        TEXT  'HELLO WORLD!'
*--------------------------------------------------------------
* Include required files & startup options
*--------------------------------------------------------------
        COPY  "D:\Projekte\spectra2\tms9900\runlib.a99"
SPVMOD  EQU   GRAPH1          ; Video mode.   See VIDTAB for details.
SPFONT  EQU   FNOPT7          ; Font to load. See LDFNT  for details.
SPFCLR  EQU   >F0             ; Foreground/Background color for font.
SPFBCK  EQU   >01             ; Screen background color.
***************************************************************
* Main
********@*****@******************@*****************************
MAIN    BL    @FILV
        DATA  >0380,>F0,16        ; Set color table
        LI    R0,>8370
        MOV   R0,@WTITAB          ; Our timer table
        BL    @MKSLOT
        DATA  >000F,BLINK,EOL     ; Run thread every 15 ticks
        BL    @SPPREP
        DATA  ROCK,SPOPT1         ; Speech player on / Speak external
        MOVB  @BD1,@>8369         ; Set toggle
        B     @TMGR               ; Run scheduler
***************************************************************
* Thread
********@*****@******************@*****************************
BLINK   NEG   @>8368              ; Switch toggle
        JLT   BLIN2
BLIN1   BL    @PUTAT
        DATA  >0A0A,HW            ; Show "Hello World!" message
        JMP   BLIN3
BLIN2   BL    @HCHAR
        BYTE  >0A,>0A,32,12       ; white space x
        DATA  EOL
BLIN3   B     @SLOTOK             ; Exit to Thread Scheduler
ROCK    BYTE TALKON
        BYTE >00,>E0,>80,>E2,>3B,>13,>50,>DC,>64,>00,>AA,>E9,>3C,>69
        BYTE >53,>1B,>EE,>9E,>F8,>E4,>55,>4D,>BA,>75,>9A,>53,>54,>DD
        BYTE >65,>5E,>AD,>0F,>90,>6C,>45,>01,>AA,>CE,>08,>40,>15,>1D
        BYTE >01,>68,>AA,>3C,>00,>55,>97,>17,>20,>F9,>D6,>63,>67,>3B
        BYTE >61,>39,>49,>4F,>5A,>75,>65,>45,>66,>3D,>40,>B6,>ED,>0B
        BYTE >C8,>BA,>35,>01,>D9,>8C,>25,>A0,>9A,>F6,>00,>64,>37,>91
        BYTE >80,>18,>46,>8E,>9B,>5D,>87,>E5,>64,>39,>4E,>D1,>D5,>EE
        BYTE >DD,>EE,>00,>D9,>B5,>0F,>A0,>E8,>8A,>04,>14,>53,>11,>80
        BYTE >6A,>3A,>1C,>50,>7C,>FB,>F1,>8B,>99,>34,>EF,>24,>C7,>2D
        BYTE >B6,>53,>B3,>DB,>1C,>A7,>AA,>CA,>CA,>6C,>77,>80,>EC,>27
        BYTE >1A,>90,>6D,>67,>02,>B2,>9F,>74,>40,>89,>1B,>06,>C8,>6E
        BYTE >FC,>84,>55,>77,>99,>76,>9D,>E3,>57,>DD,>ED,>52,>75,>4E
```

```
BYTE >55,>EC,>94,>79,>D5,>3E,>76,>53,>53,>A1,>13,>E7,>D8,>4D
BYTE >F5,>B8,>55,>EA,>00,>54,>3D,>61,>80,>6C,>27,>15,>50,>54
BYTE >B7,>00,>8A,>EE,>2A,>66,>13,>5D,>A9,>55,>FB,>D8,>55,>F7
BYTE >98,>76,>DD,>63,>57,>55,>1D,>52,>55,>8E,>D9,>D4,>A6,>EB
BYTE >34,>69,>40,>51,>13,>0E,>A8,>62,>CA,>00,>55,>6D,>08,>A0
BYTE >C9,>CD,>14,>2C,>55,>BD,>EA,>49,>47,>52,>E5,>94,>47,>26
BYTE >39,>6E,>D3,>D3,>E6,>59,>F7,>98,>55,>6F,>69,>54,>DD,>63
BYTE >57,>37,>29,>D1,>71,>8F,>DD,>F5,>B4,>C9,>26,>59,>79,>57
BYTE >DB,>E1,>D1,>7A,>F4,>5D,>4F,>9B,>D6,>92,>92,>2F,>E3,>5D
BYTE >16,>6D,>92,>35,>64,>75,>49,>36,>71,>E6,>54,>39,>6D,>DE
BYTE >B4,>99,>C5,>4E,>4A,>56,>9C,>63,>56,>33,>25,>D1,>4D,>8E
BYTE >5B,>F4,>B4,>E4,>C4,>39,>5E,>D1,>D3,>E6,>5D,>E7,>78,>5D
BYTE >4E,>05,>77,>5D,>07,>54,>31,>A1,>80,>AA,>36,>05,>50,>F9
BYTE >25,>03,>AA,>E8,>20,>40,>91,>E5,>02,>88,>A9,>E4,>78,>55
BYTE >4F,>99,>55,>DD,>E3,>35,>D5,>ED,>5A,>75,>8F,>57,>4D,>B7
BYTE >FA,>D4,>3D,>5E,>55,>57,>1A,>1D,>F5,>B8,>55,>6D,>B9,>55
BYTE >D5,>E3,>56,>33,>6D,>3A,>75,>8F,>5B,>ED,>94,>69,>D5,>3D
BYTE >6E,>71,>53,>6A,>5D,>F7,>B8,>D5,>4C,>9B,>76,>D5,>E3,>55
BYTE >3D,>ED,>56,>55,>8F,>57,>75,>77,>68,>57,>3D,>5E,>31,>53
BYTE >61,>53,>F5,>78,>55,>75,>A7,>F6,>94,>E3,>55,>5D,>1B,>3A
BYTE >55,>8F,>DB,>55,>76,>69,>35,>3D,>6E,>55,>35,>A1,>35,>E7
BYTE >B8,>4D,>E5,>86,>D6,>94,>E3,>36,>95,>93,>9A,>53,>8E,>57
BYTE >75,>4D,>58,>4D,>39,>5E,>D5,>DD,>EE,>3D,>FB,>78,>45,>4F
BYTE >BB,>F7,>EC,>E3,>17,>B3,>65,>D1,>73,>8E,>5F,>CD,>B4,>E9
BYTE >D4,>3D,>41,>B5,>DB,>66,>55,>F7,>04,>D5,>6D,>A9,>55,>DD
BYTE >13,>56,>BB,>AD,>DE,>49,>4F,>58,>ED,>B4,>5A,>27,>3D,>51
BYTE >D5,>5B,>E6,>5D,>F7,>C4,>D5,>6C,>69,>74,>D3,>93,>54,>73
BYTE >A9,>5E,>4D,>4F,>52,>D5,>96,>7A,>27,>3D,>49,>D5,>97,>9A
BYTE >95,>F4,>24,>D5,>6C,>9A,>67,>93,>93,>56,>B3,>69,>5E,>4D
BYTE >4E,>5A,>DC,>B9,>45,>D5,>39,>59,>71,>E7,>1A,>D9,>E4,>E4
BYTE >C5,>5C,>B8,>67,>9D,>93,>57,>79,>6E,>51,>95,>03,>90,>DC
BYTE >8A,>00,>22,>2F,>21,>80,>93,>05,>CE,>1C,>72,>3C,>C5,>6C
BYTE >07,>73,>E9,>4D,>0F,>95,>55,>CC,>A5,>B7,>DD,>DD,>8E,>01
BYTE >2A,>75,>32,>C0,>E4,>29,>06,>18,>7A,>D4,>00,>53,>B7,>3B
BYTE >60,>E8,>76,>07,>4C,>5D,>E9,>80,>A9,>32,>0C,>30,>55,>98
BYTE >01,>96,>CE,>54,>C0,>70,>6D,>02,>E8,>C2,>99,>00,>95,>3B
BYTE >11,>20,>51,>23,>50,>40,>57,>E1,>01,>98,>2A,>A2,>01,>53
BYTE >67,>2C,>60,>69,>CF,>05,>6C,>1D,>35,>80,>25,>CB,>03,>B0
BYTE >58,>B8,>03,>A6,>CC,>50,>C0,>10,>65,>02,>18,>C4,>EC,>04
BYTE >CD,>A7,>9B,>6B,>97,>53,>34,>9F,>1A,>E6,>6D,>4E,>D6,>63
BYTE >8B,>66,>26,>39,>7E,>8F,>45,>96,>B9,>BA,>01,>59,>9B,>06
BYTE >20,>3B,>66,>03,>14,>C3,>A4,>80,>6A,>8D,>19,>50,>9D,>53
BYTE >02,>6E,>F0,>68,>C0,>B2,>E5,>C7,>5C,>71,>C2,>CC,>64,>1F
BYTE >6B,>E8,>54,>33,>53,>7C,>CC,>12,>43,>C3,>D9,>DD,>31,>4B
BYTE >34,>F7,>E0,>2E,>27,>28,>29,>D3,>C5,>D7,>9E,>B0,>E8,>09
BYTE >77,>DF,>7A,>CC,>A6,>26,>CC,>A3,>ED,>31,>9B,>E8,>90,>CA
BYTE >35,>C7,>1C,>3A,>4B,>D5,>1A,>2F,>7B,>CA,>08,>57,>69,>D2
BYTE >EC,>65,>B2,>CA,>C5,>4D,>B2,>86,>AA,>08,>F1,>B5,>C1,>EC
BYTE >66,>3A,>4C,>9B,>86,>B0,>E9,>2E,>57,>6D,>5B,>80,>2D,>32
BYTE >07,>30,>54,>DB,>29,>46,>E9,>30,>65,>37,>27,>EF,>2D,>C3
BYTE >8D,>35,>9F,>AC,>F9,>08,>0D,>51,>7B,>B2,>66,>3D,>AD,>58
BYTE >CB,>49,>9B,>F6,>D6,>24,>2F,>27,>AB,>D2,>DB,>92,>93,>9C
BYTE >AC,>28,>EB,>2C,>AC,>75,>F2,>62,>64,>C7,>C1,>F5,>CA,>93
BYTE >A9,>8D,>04,>2F,>AB,>88,>A6,>37,>0A,>62,>0F,>33,>29,>9F
BYTE >18,>72,>B8,>CC,>68,>B2,>35,>25,>CF,>31,>53,>B0,>94,>90
BYTE >3E,>C7,>AA,>51,>9D,>4B,>DA,>1F,>AB,>7B,>75,>49,>6D,>7B
BYTE >AC,>E9,>25,>AC,>B4,>ED,>B1,>86,>D6,>96,>B4,>B5,>C7,>EA
```

```
BYTE >CA,>5A,>22,>56,>1F,>BB,>0B,>1F,>55,>EF,>7A,>9C,>26,>A3
BYTE >5D,>AC,>DB,>71,>9B,>F0,>0E,>D6,>AC,>C7,>2B,>3A,>5B,>44
BYTE >BA,>9C,>A0,>9A,>4C,>31,>D5,>BA,>C2,>62,>23,>38,>C4,>E9
BYTE >0A,>8B,>89,>A0,>52,>A7,>C7,>2F,>3E,>82,>8A,>DC,>1E,>B7
BYTE >7A,>73,>6F,>EC,>72,>AC,>64,>33,>A2,>B4,>EF,>31,>BB,>9A
BYTE >CA,>D0,>34,>C7,>EC,>6A,>C6,>C2,>B6,>1E,>AF,>CB,>6D,>8D
BYTE >58,>7A,>8A,>A6,>A6,>34,>72,>E9,>C9,>AA,>99,>96,>A8,>75
BYTE >C7,>AE,>A6,>46,>B2,>B7,>9E,>A4,>84,>0E,>A9,>68,>7D,>BC
BYTE >1A,>36,>38,>22,>ED,>B0,>9A,>EC,>F6,>CA,>36,>CD,>EE,>BA
BYTE >3A,>2C,>DA,>06,>A0,>71,>F5,>04,>4C,>AD,>56,>80,>E9,>3A
BYTE >8F,>B9,>53,>47,>8A,>4A,>3E,>E6,>4A,>1D,>0A,>DA,>E4,>98
BYTE >A3,>76,>20,>79,>E2,>63,>8E,>D6,>46,>1A,>69,>4F,>D0,>9B
BYTE >AB,>78,>6C,>3D,>7E,>B5,>66,>1E,>BE,>F5,>78,>D5,>94,>07
BYTE >5B,>DF,>93,>16,>51,>99,>21,>6B,>8F,>5B,>79,>95,>A6,>AC
BYTE >3B,>49,>15,>93,>1A,>BA,>EE,>D8,>55,>6F,>9A,>C9,>BA,>63
BYTE >17,>3B,>E9,>A2,>DB,>4E,>5C,>DC,>A6,>A9,>AE,>3B,>49,>55
BYTE >93,>66,>DA,>EE,>24,>59,>4C,>AA,>FB,>BA,>93,>67,>DA,>93
BYTE >C9,>4B,>56,>91,>D9,>B4,>85,>4E,>19,>76,>61,>55,>66,>52
BYTE >67,>D8,>59,>54,>85,>71,>94,>65,>67,>51,>55,>8E,>75,>8E
BYTE >99,>45,>55,>18,>2D,>3D,>45,>91,>D3,>1E,>34,>FB,>14,>55
BYTE >74,>4B,>DA,>EC,>93,>4D,>1E,>23,>29,>4B,>4E,>DA,>55,>64
BYTE >26,>AD,>3E,>69,>37,>99,>E9,>B4,>EA,>64,>5D,>4E,>4B,>CA
BYTE >E2,>93,>37,>BD,>C3,>E1,>AB,>4F,>DE,>D4,>B4,>BA,>2D,>3E
BYTE >79,>93,>DD,>EA,>B6,>EA,>64,>43,>D4,>48,>E8,>A2,>93,>77
BYTE >51,>ED,>A1,>8B,>4E,>3E,>58,>8F,>BA,>CF,>39,>C5,>E4,>31
BYTE >11,>7C,>FB,>94,>55,>4D,>9B,>FA,>AB,>53,>65,>3B,>AD,>92
BYTE >8F,>4E,>9D,>C2,>16,>6B,>3D,>3E,>66,>76,>DB,>2C,>F9,>E4
BYTE >98,>D1,>4F,>89,>F6,>9B,>63,>26,>33,>2D,>52,>CF,>8E,>9D
BYTE >F4,>8C,>6A,>BC,>3B,>6E,>52,>33,>CA,>B9,>EE,>78,>49,>CD
BYTE >28,>FB,>FA,>E3,>67,>DE,>6B,>64,>EB,>4E,>98,>F9,>AE,>A2
BYTE >2D,>3D,>71,>E6,>BB,>8A,>B6,>F4,>C4,>99,>EF,>18,>DA,>DA
BYTE >93,>64,>B1,>23,>18,>4B,>4E,>D2,>50,>4C,>09,>B7,>1D,>69
BYTE >21,>DD,>01,>BE,>65,>64,>15,>4D,>06,>C4,>E6,>96,>55,>1A
BYTE >95,>E0,>AB,>47,>DE,>85,>94,>73,>38,>6E,>45,>63,>E6,>26
BYTE >6E,>37,>14,>89,>A6,>2B,>47,>DB,>96,>6D,>15,>2A,>EA,>B6
BYTE >57,>BA,>54,>9A,>8A,>DB,>1B,>69,>95,>A5,>2E,>B5,>64,>A4
BYTE >55,>86,>05,>D7,>92,>96,>35,>51,>1A,>1C,>8B,>5A,>3E,>79
BYTE >58,>91,>37,>2A,>E5,>10,>A5,>89,>9E,>28,>E5,>43,>8E,>3A
BYTE >79,>A2,>90,>4D,>19,>D2,>EC,>B6,>5C,>32,>54,>9A,>A3,>3B
BYTE >22,>80,>A1,>EE,>00,>00,>00,>00,>00,>00,>00,>00,>00,>00
BYTE >00,>00,>00,>00,>00,>00,>00,>F0
BYTE TALKOF
END
```

```
********@*****@**********************@**************************
        AORG  >6000
*--------------------------------------------------------------
* Cartridge header
*--------------------------------------------------------------
GRMHDR  BYTE  >AA,1,1,0,0,0
        DATA  PROG
        BYTE  0,0,0,0,0,0,0,0
PROG    DATA  0
        DATA  RUNLIB
HW      BYTE  15
        TEXT  'MOVE THE SPRITE'
        EVEN
*--------------------------------------------------------------
* Include required files
*--------------------------------------------------------------
        COPY  "D:\Projekte\spectra2\tms9900\runlib.a99"
*--------------------------------------------------------------
* SPECTRA2 startup options
*--------------------------------------------------------------
SPVMOD  EQU   GRAPH1                ; Video mode.   See VIDTAB for details.
SPFONT  EQU   FNOPT7                ; Font to load. See LDFNT  for details.
SPFCLR  EQU   >A0                   ; Foreground/Background color for font.
SPFBCK  EQU   >01                   ; Screen background color.
*--------------------------------------------------------------
* Our constans and variables in scratchpad memory
*--------------------------------------------------------------
RAMSAT  EQU   >8340                 ; Copy of mini-SAT in RAM memory (6 bytes)
RAMTAB  EQU   >8346                 ; Timer table (4 bytes)
***************************************************************
* Main
********@*****@**********************@**************************
MAIN    BL    @CPYM2M
        DATA  SPRITE,RAMSAT,6       ; Copy 6 bytes from ROM into scratchpad RAM
        BL    @CPYM2V
        DATA  >1000,PAT1,8          ; Dump sprite pattern
        BL    @PUTBOX
        DATA  >1503,>1A02,INFO,EOL  ; Show text in box on row >15, col >03 with width >1A, height >02.
        MOV   @MYTAB,@WTITAB        ; Setup address of timer table
        BL    @MKSLOT
        DATA  >0002,MVBOX,EOL       ; Create new timer slot
        B     @TMGR                 ; Handle FCTN-QUIT key, timers, etc.
***************************************************************
* THREAD Move sprite: This routine is called from TMGR
********@*****@**********************@**************************
MVBOX   COC   @WBIT11,CONFIG        ; ANY key pressed ?
        JNE   MVBOX5                ; No, so exit
        MOV   @WVRTKB,R1            ; Get keyboard flags
MVBOX1  COC   @KEY1,R1              ; Left ?
        JNE   MVBOX2
        SB    @BD2,@RAMSAT+1        ; X=X-2
MVBOX2  COC   @KEY2,R1              ; Right ?
        JNE   MVBOX3
        AB    @BD2,@RAMSAT+1        ; X=X+2
MVBOX3  COC   @KEY3,R1              ; Up ?
        JNE   MVBOX4
        SB    @BD2,@RAMSAT          ; Y=Y-2
```

```
MVBOX4  COC   @KEY4,R1              ; Down ?
        JNE   MVBOX5
        AB    @BD2,@RAMSAT          ; Y=Y+2
MVBOX5  BL    @CPYM2V               ; Dump copy of SAT to VDP SAT
        DATA  >0300,RAMSAT,6
        B     @SLOTOK               ; Return to Thread Scheduler
KEY1    DATA  K1LF                  ; Left
KEY2    DATA  K1RG                  ; Right
KEY3    DATA  K1UP                  ; Up
KEY4    DATA  K1DN                  ; Down
*****************************************************************
* Our constants
********@*****@**********************@**************************
MYTAB   DATA  RAMTAB                ; Location of timer table in scratchpad memory
SPRITE  DATA  >2020,>000F           ; Row >20, col >20, pattern >00, color white
        DATA  >0D00                 ; No more sprites
PAT1    DATA  >FF81,>8181,>8181,>81FF
INFO    BYTE  52
        TEXT  'Use joystick 1 or keys    E,S,D,X for moving sprite'
        END
```

```
********@*****@**********************@**************************
        AORG  >6000
*--------------------------------------------------------------
* Cartridge header
*--------------------------------------------------------------
GRMHDR  BYTE  >AA,1,1,0,0,0
        DATA  PROG
        BYTE  0,0,0,0,0,0,0,0
PROG    DATA  0
        DATA  RUNLIB
MSG0    BYTE  11
        TEXT  'LEAVE TRAIL'
*--------------------------------------------------------------
* Include required files
*--------------------------------------------------------------
        COPY  "D:\Projekte\spectra2\tms9900\runlib.a99"
*--------------------------------------------------------------
* SPECTRA2 startup options
*--------------------------------------------------------------
SPVMOD  EQU   GRAPH1                ; Video mode.   See VIDTAB for details.
SPFONT  EQU   FNOPT7                ; Font to load. See LDFNT  for details.
SPFCLR  EQU   >60                   ; Foreground/Background color for font.
SPFBCK  EQU   >01                   ; Screen background color.
*--------------------------------------------------------------
* Variables
*--------------------------------------------------------------
BUFFER  EQU   >8340                 ; Buffer for PUTNUM       (5 bytes)
RAMTAB  EQU   >8346
RAMSAT  EQU   >8370
***************************************************************
* Main
********@*****@**********************@**************************
MAIN    BL    @PUTBOX
        DATA  >0003,>1A02,INFO,EOL  ; Show text in box on row >00, col >03 with width >1A, height >02.
        LI    R0,>0709
        MOV   R0,@WYX
        BL    @CPYM2M
        DATA  SPRITE,RAMSAT,6       ; Copy 6 bytes from ROM into scratchpad RAM
        BL    @CPYM2V
        DATA  >1000,PAT1,8          ; Dump sprite pattern
        MOV   R0,TMP0
        BL    @YX2PXX
        MOV   TMP0,@RAMSAT          ; Update RAMSAT

        MOV   @MYTAB,@WTITAB        ; Setup timer table
        BL    @MKSLOT               ; Allocate slot 0
        DATA  >0004,MVBOX,EOL
        B     @TMGR
***************************************************************
* Move sprite: This routine is called as timer slot from TMGR
********@*****@**********************@**************************
MVBOX   MOV   R11,R2                ; Save R11
        COC   @WBIT11,CONFIG        ; ANY key pressed ?
        JNE   MVBOX5                ; No, so exit
        MOV   @WVRTKB,R1            ; Get keyboard flags
MVBOX1  COC   @KEY1,R1              ; Left ?
        JNE   MVBOX2
```

```
        SB    @BD1,@BX
MVBOX2  COC   @KEY2,R1              ; Right ?
        JNE   MVBOX3
        AB    @BD1,@BX
MVBOX3  COC   @KEY3,R1              ; Up ?
        JNE   MVBOX4
        SB    @BD1,@BY
MVBOX4  COC   @KEY4,R1              ; Down ?
        JNE   MVBOX5
        AB    @BD1,@BY
MVBOX5  MOV   @WYX,R0
        BL    @YX2PX                ; YX tile position to sprite pixel position
        MOV   TMP0,@RAMSAT          ; Update YX in SAT copy
        BL    @CPYM2V               ; Dump SAT copy to VDP SAT
        DATA  >0300,RAMSAT,6        ; ... R11 is overwritten
        BL    @YX2PNT               ; Get VDP destination address
        LI    TMP1,65
        BL    @XVPUTB               ; Write character

        MOV   R0,TMP0
        SRL   TMP0,8                ; Right align Y
        BL    @PUTNUM
        DATA  >0500,TMP0HB,BUFFER,NUM2

        MOV   R0,TMP0
        ANDI  TMP0,>00FF            ; Only keep X
        BL    @PUTNUM
        DATA  >050A,TMP0HB,BUFFER,NUM2
        MOV   R0,@WYX
        B     *R2                   ; ... so return using copy in R2
KEY1    DATA  K1LF                  ; Left
KEY2    DATA  K1RG                  ; Right
KEY3    DATA  K1UP                  ; Up
KEY4    DATA  K1DN                  ; Down
***************************************************************
* Our constants
********@*****@**********************@**************************
MYTAB   DATA  RAMTAB                ; Location of timer table in scratchpad memory
SPRITE  DATA  >0000,>000F           ; Row >00, Col >00, pattern >00, color white
        DATA  >0D00                 ; No more sprites
PAT1    DATA  >FF81,>8181,>8181,>81FF
INFO    BYTE  52
        TEXT  'Use joystick 1 or keys    E,S,D,X for moving sprite'
        END
```

```
********@*****@*********************@**************************
       AORG  >6000
*--------------------------------------------------------------
* Cartridge header
*--------------------------------------------------------------
GRMHDR BYTE  >AA,1,1,0,0,0
       DATA  PROG
       BYTE  0,0,0,0,0,0,0,0
PROG   DATA  0
       DATA  RUNLIB
MSG0   BYTE  14
       TEXT  'RANDOM NUMBERS'
*--------------------------------------------------------------
* Include required files
*--------------------------------------------------------------
       COPY  "D:\Projekte\spectra2\tms9900\runlib.a99"
*--------------------------------------------------------------
* SPECTRA2 startup options
*--------------------------------------------------------------
SPVMOD EQU   GRAPH1                ; Video mode.   See VIDTAB for details.
SPFONT EQU   FNOPT2                ; Font to load. See LDFNT  for details.
SPFCLR EQU   >30                   ; Foreground/Background color for font.
SPFBCK EQU   >01                   ; Screen background color.
*--------------------------------------------------------------
* Variables
*--------------------------------------------------------------
BUFFER EQU   >8340                 ; Buffer for PUTNUM       (5 bytes)
TIMERS EQU   >8370                 ; Address of timer table (12 bytes)
***************************************************************
* Main
********@*****@*********************@**************************
MAIN   BL    @PUTAT
       DATA  >000A,MSG0            ; Show "RANDOM NUMBERS" on row 0, column 10
       BL    @PUTAT
       DATA  >0302,MSG1            ; Show "RANGE 0-65536...:" on row 3, column 2
       BL    @PUTAT
       DATA  >0502,MSG2            ; Show "RANGE 0-100.....:" on row 5, column 2
***************************************************************
* Prepare threads
********@*****@*********************@**************************
       MOV   @ZTITAB,@WTITAB       ; Setup timer table
       MOVB  @BD2,@BTIHI           ; Set highes slot in use
       BL    @MKSLOT               ; Allocate 3 timers
       DATA  >0001,SLOT0           ; Slot 0, run every tick
       DATA  >0120,SLOT1           ; Slot 1, run every 32 ticks
       DATA  >0201,SLOT2,EOL       ; Slot 2, run every tick
       B     @TMGR                 ; Handle FCTN-QUIT key, etc.
ZTITAB DATA  TIMERS                ; Address of timer table
***************************************************************
* Thread 0 - Display random number on row 3, column 21
********@*****@*********************@**************************
SLOT0  BL    @RND
       DATA  >FFFF                 ; Random number in range 0-65536, returned in TMP0
       BL    @PUTNUM
       DATA  >0315,TMP0HB,BUFFER,>3030
       B     @SLOTOK               ; Exit thread
***************************************************************
```

```
* Thread 1 - Display random number on row 5, column 21
********@*****@*********************@**************************
SLOT1    BL     @RND
         DATA   100                    ; Random number in range 0-100, returned in TMP0
         BL     @PUTNUM
         DATA   >0515,TMP0HB,BUFFER,>3020
         B      @SLOTOK                ; Exit thread
*****************************************************************
* Thread 2 - Display random characters on lower part of screen
********@*****@*********************@**************************
SLOT2    BL     @RND
         DATA   544
         AI     TMP0,7*32
         BL     @XVPUTB                ; Put character on screen
         B      @SLOTOK                ; Exit thread
MSG1     BYTE   17
         TEXT   'RANGE 0-65536...:'
MSG2     BYTE   17
         TEXT   'RANGE 0-100.....:'
         END
```

```
********@*****@**********************@**************************
        AORG  >6000
*--------------------------------------------------------------
* Cartridge header
*--------------------------------------------------------------
GRMHDR  BYTE  >AA,1,1,0,0,0
        DATA  PROG4                   ; Address of last menu item
        BYTE  0,0,0,0,0,0,0,0
PROG4   DATA  PROG3                   ; Address of next menu item
        DATA  INIT4
MSG4    BYTE  14
        TEXT  'FOURTH PROGRAM'
PROG3   DATA  PROG2                   ; Address of next menu item
        DATA  INIT3
MSG3    BYTE  13
        TEXT  'THIRD PROGRAM'
PROG2   DATA  PROG1                   ; Address of next menu item
        DATA  INIT2
MSG2    BYTE  14
        TEXT  'SECOND PROGRAM'
PROG1   DATA  0                       ; No more menu items following.
        DATA  INIT1
MSG1    BYTE  13
        TEXT  'FIRST PROGRAM'
        EVEN
*--------------------------------------------------------------
* Include required files
*--------------------------------------------------------------
        COPY  "D:\Projekte\spectra2\tms9900\runlib.a99"
*--------------------------------------------------------------
* SPECTRA2 startup options
*--------------------------------------------------------------
SPVMOD  EQU   GRAPH1                  ; Video mode.   See VIDTAB for details.
SPFONT  EQU   FNOPT7                  ; Font to load. See LDFONT for details.
SPFCLR  EQU   >F0                     ; Foreground/Background color for font.
SPFBCK  EQU   >01                     ; Screen background color.
***************************************************************
* Execute this before RUNLIB
********@*****@**********************@**************************
INIT1   LI    R0,MSG1                 ; String 'First program'
        JMP   RUNINI
INIT2   LI    R0,MSG2                 ; String 'Second program'
        JMP   RUNINI
INIT3   LI    R0,MSG3                 ; String 'Third program'
        JMP   RUNINI
INIT4   LI    R0,MSG4                 ; String 'Fourth program'
RUNINI  MOV   R0,@>8300               ; R0 in the SPECTRA2 workspace, not the GPL workspace (!)
        B     @RUNLIB                 ; Initialize SPECTRA2 library
***************************************************************
* Main
********@*****@**********************@**************************
MAIN    LI    TMP0,>0169              ; VDP target address in PNT table (Pattern Name Table), row 11/col 9.
        MOVB  *R0+,TMP2               ; Get string length into TMP2.
        SRL   TMP2,8                  ; Move high byte into low byte.
        MOV   R0,TMP1                 ; Address of string to display.
        BL    @XPYM2V                 ; Dump string to VDP memory.
        B     @TMGR                   ; Handle FCTN-QUIT key, etc.
```

```
END
```

```
****************************************************************
* Honeycomb Rapture - (c) 2010 by Owen Brand
*
* Cartridge conversion by Retroclouds
*
* This file: hc.a99
****************************************************************
* Main program
********@*****@*********************@**************************
        AORG  >6000
*--------------------------------------------------------------
* Cartridge header
*--------------------------------------------------------------
GRMHDR  BYTE  >AA,1,1,0,0,0
        DATA  PROG
        BYTE  0,0,0,0,0,0,0,0
PROG    DATA  0
        DATA  RUNLIB
        BYTE  17
        TEXT  'HONEYCOMB RAPTURE'
*--------------------------------------------------------------
* Include required files
*--------------------------------------------------------------
        COPY  "D:\Projekte\spectra2\tms9900\runlib.a99"
*--------------------------------------------------------------
* SPECTRA2 startup options
*--------------------------------------------------------------
SPVMOD  EQU   GRAPH1                ; Video mode.   See VIDTAB for details.
SPFONT  EQU   FNOPT7                ; Font to load. See LDFONT for details.
SPFCLR  EQU   >A0                   ; Foreground/Background color for font.
SPFBCK  EQU   >01                   ; Screen background color.
*--------------------------------------------------------------
* Game memory setup
*--------------------------------------------------------------
SCORE   EQU   >8340                 ; Score
LIVES   EQU   >8342                 ; Lives
LEVEL   EQU   >8344                 ; Level
GASTAT  EQU   >8346                 ; Game status flags
BSPEED  EQU   >8348                 ; Killer bee speed variable
BUFFER  EQU   >834A                 ; Working buffer (6 bytes)
TIMERS  EQU   >8350                 ; Timer table
RAMSAT  EQU   >8370                 ; Our SAT in scratchpad memory
COUNTER EQU   BUFFER                ; Temporary counter
****************************************************************
* Initial game setup
********@*****@*********************@**************************
MAIN    BL    @CPYM2V
        DATA  >0808,TLTPAT,8*8      ; Load patterns for title letters
        BL    @CPYM2V
        DATA  >0908,LETTRS,73*8     ; Load font into VDP
        BL    @FILV
        DATA  >0380,>1A,16          ; Fill color table
        BL    @PUTVR
        DATA  >070A                 ; VDP#7 - Set background color
        BL    @CPYM2V
        DATA  >1000,SPRPAT,11*32    ; Load sprite patterns into VDP
*--------------------------------------------------------------
```

```
*    Initialise variables
*---------------------------------------------------------------
        CLR   @SCORE                ; Score 0
        LI    R0,1
        MOV   R0,@LEVEL             ; Level 1
        LI    R0,3
        MOV   R0,@LIVES             ; 3 Lives
        CLR   @GASTAT               ; Clear all game flags
        LI    R0,>0400
        MOV   R0,@BSPEED            ; Killer bee start speed
*---------------------------------------------------------------
*    Game
*---------------------------------------------------------------
        COPY  "D:\Projekte\spectra2\tms9900\hc_source1.a99"
        COPY  "D:\Projekte\spectra2\tms9900\hc_source2.a99"
        COPY  "D:\Projekte\spectra2\tms9900\hc_source3.a99"
        COPY  "D:\Projekte\spectra2\tms9900\hc_source4.a99"
        END
```

```
*****************************************************************
* Honeycomb Rapture - (c) 2010 by Owen Brand
*
* Cartridge conversion by Retroclouds
*
* This file: hc_source1.a99
*****************************************************************
* Title Screen
********@*****@**********************@**************************
        BL    @PUTAT
        DATA  >0200,TITLE1          ; Y=2  X=0 - Honeycomb
        BL    @PUTAT
        DATA  >0A00,TITLE2          ; Y=10 X=0 - Rapture
        BL    @PUTAT
        DATA  >1308,TXT1            ; Y=19 X=8  'PRESS ANY KEY TO'
        BL    @PUTAT
        DATA  >140C,TXT2            ; Y=20 X=12 'CONTINUE'
        BL    @CPYM2M
        DATA  SPRSAT,RAMSAT,4       ; Copy SAT from ROM to RAM
*---------------------------------------------------------------
*    Prepare tune & timers
*---------------------------------------------------------------
        BL    @SDPREP
        DATA  SOUND1,SDOPT1         ; Prepare tune for playing
        MOV   @TIMTAB,@WTITAB       ; Set pointer to timer table
        BL    @MKSLOT               ; Create slots
        DATA  >0001,DMPSAT          ; Slot 0 = Dump to VDP
        DATA  >0101,MVPLAN          ; Slot 1 = Move plane
        DATA  >0201,START,EOL       ; Slot 2 = Wait for game start
        MOVB  @BD2,R10              ; Set highest slot in use
        B     @TMGR                 ; Start timer manager

*****************************************************************
* Thread START - Wait for game start
*****************************************************************
START   COC   @WBIT11,R12           ; ANY key pressed ?
        JEQ   START1                ; Start game
        B     *R11                  ; Exit
START1  B     @GAME
```

```
****************************************************************
* Honeycomb Rapture - (c) 2010 by Owen Brand
*
* Cartridge conversion by Retroclouds
*
* This file: hc_source2.a99
****************************************************************
* Game
********@*****@*********************@**************************

****************************************************************
* Setup game screen
********@*****@*********************@**************************
GAME    BL    @SCROFF                ; Screen off
        BL    @FILV
        DATA  >0000,00,768           ; Fill screen
        BL    @CPYM2M
        DATA  SPRSAT,RAMSAT,6*4      ; Copy SAT from ROM to RAM
        LI    TMP0,>0700             ; Y=>07 X=>00
        MOV   TMP0,@RAMSAT           ; Reposition sprite plane
        BL    @FILV
        DATA  >0000,32,7*32
        BL    @PUTAT
        DATA  >0002,TXT3             ; Display text 'Score'
        BL    @PUTAT
        DATA  >000E,TXT4             ; Display text 'Level'
        BL    @PUTAT
        DATA  >0018,TXT5             ; Display text 'Lives'
*--------------------------------------------------------------
*    Draw honeycomb & flower bed
*--------------------------------------------------------------
        BL    @PUTBOX
        DATA  >071B,>0202,HCOMB
        DATA  >1103,>0303,FLOWER
        DATA  >1107,>0303,FLOWER
        DATA  >110B,>0303,FLOWER
        DATA  >110F,>0303,FLOWER
        DATA  >1113,>0303,FLOWER
        DATA  >1117,>0303,FLOWER
        DATA  >111B,>0303,FLOWER
        DATA  >1401,>0303,FLOWER
        DATA  >1405,>0303,FLOWER
        DATA  >1409,>0303,FLOWER
        DATA  >140D,>0303,FLOWER
        DATA  >1411,>0303,FLOWER
        DATA  >1415,>0303,FLOWER
        DATA  >1419,>0303,FLOWER,EOL
        BL    @VCHAR
        DATA  >1404,>5B03
        DATA  >1408,>5B03
        DATA  >140C,>5B03
        DATA  >1410,>5B03
        DATA  >1414,>5B03
        DATA  >1418,>5B03
        DATA  >141C,>5B03,EOL
*--------------------------------------------------------------
*  Display game stats (Score, Level, Lives)
```

```
*--------------------------------------------------------------
GAME1   BL    @PUTNUM               ; Display score, fill with '0'
        DATA  >0102,SCORE,BUFFER,>3030
        BL    @PUTNUM               ; Display level, fill with ' '
        DATA  >010C,LEVEL,BUFFER,>3020
        BL    @PUTNUM               ; Display lives, fill with ' '
        DATA  >0116,LIVES,BUFFER,>3020
        BL    @MKSLOT
        DATA  >0201,MVBEE
        DATA  >0302,MVKBEE
        DATA  >0402,MVKBEE
        DATA  >0501,MVKBEE
        DATA  >0630,CLRMSG,EOL      ; Allocate timer slots
        MOVB  @BD6,R10              ; Set highest slot in use
        BL    @MKHOOK
        DATA  COINC
        BL    @CPYM2V
        DATA  >0380,COLORS,16       ; Load color table into VDP
        BL    @SCRON                ; Screen on
GAME2   B     @TMGR



***************************************************************
* User hook - Check for coincidence
***************************************************************
COINC   COC   @WBIT2,R13            ; Coincidence bit set ?
        JNE   COINCZ                ; No, exit
        MOV   @GASTAT,R0            ; Sequence already busy
        COC   @WBIT0,R0
        JEQ   COINCZ
*--------------------------------------------------------------
* Bee gets killed by killer bee
*--------------------------------------------------------------
        BL    @MUTEX                ; Pause sound player
        SOC   @WBIT0,@GASTAT        ; Bee killed = 1
        BL    @CPYM2V
        DATA  >0380,COLOR2,16       ; Load 2nd color table into VDP
        BL    @PUTVR
        DATA  >0706                 ; VDP#7 - Set background color
        BL    @MKSLOT
        DATA  >0207,SPNBEE,EOL      ; Start "spin bee" sequence
        CLR   @COUNTER
COINCZ  B     @HOOKOK               ; Exit



***************************************************************
* Thread DMPSAT - Dump SAT to VDP
***************************************************************
DMPSAT  MOV   @GASTAT,R1            ; Get game status flags
        COC   @WBIT0,R1             ; Bee kill sequence busy ?
        JEQ   DMPSA1
*--------------------------------------------------------------
*  Dump SAT containing killer bees
*--------------------------------------------------------------
        BL    @CPYM2V
        DATA  >0300,RAMSAT,6*4      ; Dump SAT to VDP
```

```
        JMP    DMPSAZ
*--------------------------------------------------------------
*  Dump SAT for bee died sequence
*--------------------------------------------------------------
DMPSA1  BL     @CPYM2V
        DATA   >0300,RAMSAT,2*4       ; Dump SAT to VDP
        BL     @FILV
        DATA   >0308,>D0,1             ; End of sprite processing
DMPSAZ  B      @SLOTOK                 ; Exit




********@*****@*********************@**************************
DOSCOR  A      @ADD100,@SCORE
        BL     @PUTNUM                 ; Display score, fill with '0'
        DATA   >0102,SCORE,BUFFER,>3030
        MOV    @SPRSAT+4,@RAMSAT+4     ; Restore bee start position
        SB     @BD1,@BSPEED            ; Increase killer bee speed
        BL     @SDPREP
        DATA   SOUND3,SDOPT2           ; Start "score" tune
        B      @SLOTOK                 ; Return to Thread Scheduler


****************************************************************
* Thread MVBEE - Move the bee
****************************************************************
MVBEE   COC    @WBIT11,CONFIG          ; ANY key pressed ?
        JEQ    MVBEE1                  ; Yes, check keys
        MOV    @MVLIST+40,@RAMSAT+6    ; Update pattern/color
        B      *R11                    ; Exit
MVBEE1  MOV    @WVRTKB,TMP0            ; Get keyboard flags
        LI     TMP1,MVLIST             ; List
        LI     TMP2,8                  ; List counter
*--------------------------------------------------------------
*    Check direction
*--------------------------------------------------------------
MVBEE2  COC    *TMP1,TMP0
        JNE    MVBEE3
        SZC    *TMP1+,TMP0             ; Remove this key combination
        AB     *TMP1+,@RAMSAT+4        ; Update Y
        AB     *TMP1+,@RAMSAT+5        ; Update X
        MOV    *TMP1+,@RAMSAT+6        ; Update pattern/color
        JMP    MVBEE4
MVBEE3  AI     TMP1,6
MVBEE4  DEC    TMP2
        JNE    MVBEE2
*--------------------------------------------------------------
*    Prepare for checking Y-boundaries
*--------------------------------------------------------------
        ANDI   CONFIG,>7FFF            ; CONFIG register bit 0=0
        MOV    @RAMSAT+4,TMP0          ; Sprite YX in TMP0
        MOV    @MVLIM,TMP1             ; Get Y-boundaries
*--------------------------------------------------------------
*       Compare boundaries
*--------------------------------------------------------------
MVBEE5  CB     TMP0,TMP1               ; Compare min
```

```
        JHE   MVBEE6
        MOVB  TMP1,TMP0             ; Set sprite min
MVBEE6  SWPB  TMP1                  ; Swap min/max
        CB    TMP0,TMP1             ; Compare max
        JLE   MVBEE7
        MOVB  TMP1,TMP0             ; Set sprite max
*--------------------------------------------------------------
*    Prepare for checking X-boundaries
*--------------------------------------------------------------
MVBEE7  COC   @WBIT0,CONFIG         ; X-already checked ?
        JEQ   MVBEE8
        ORI   CONFIG,>8000          ; CONFIG register bit 0=1
        SWPB  TMP0                  ; Bee YX -> XY
        MOV   @MVLIM+2,TMP1         ; Get X-boundaries
        JMP   MVBEE5
*--------------------------------------------------------------
*    Update RAMSAT
*--------------------------------------------------------------
MVBEE8  SWPB  TMP0                  ; Bee XY -> YX
        MOV   TMP0,@RAMSAT+4        ; Save updated YX in RAMSAT
        ANDI  CONFIG,>7FFF          ; CONFIG register bit 0=0
*--------------------------------------------------------------
*    Check if bee reached honeycomb
*--------------------------------------------------------------
MVBEE9  CB    TMP0,@MVCOMB          ; Y position greater than >48 ?
        JGT   MVBEEZ
        SLA   TMP0,8
        CB    TMP0,@MVCOMB+1        ; X position less than >D0 ?
        JLT   MVBEEZ
        CB    TMP0,@MVCOMB+2        ; X position greater than >E0 ?
        JGT   MVBEEZ
*--------------------------------------------------------------
*    Yes, now update score
*--------------------------------------------------------------
        A     @ADD100,@SCORE
        BL    @PUTNUM               ; Display score, fill with '0'
        DATA  >0102,SCORE,BUFFER,>3030
        MOV   @SPRSAT+4,@RAMSAT+4   ; Restore bee start position
        SB    @BD1,@BSPEED          ; Increase killer bee speed
        BL    @SDPREP
        DATA  SOUND3,SDOPT2         ; Start "score" tune
        BL    @MKSLOT
        DATA  >0630,MVBEEY,EOL      ; Restart "game" tune in 1 second
        JMP   MVBEEZ
*--------------------------------------------------------------
*    Restart "game" tune
*--------------------------------------------------------------
MVBEEY  BL    @SDPREP
        DATA  SOUND1,SDOPT1         ; Start tune
        BL    @CLSLOT               ; Clear slot
        DATA  6
*--------------------------------------------------------------
*    Exit
*--------------------------------------------------------------
MVBEEZ  B     @SLOTOK               ; Exit
MVLIST  DATA  K1UPLF,>FFFF,>2001    ; Up-left
        DATA  K1UPRG,>FF01,>0801    ; Up-right
```

```
        DATA  K1DNLF,>01FF,>1801    ; Down-left
        DATA  K1DNRG,>0101,>1001    ; Down-right
        DATA  K1LF,>00FF,>1C01      ; Left
        DATA  K1RG,>0001,>0C01      ; Right
        DATA  K1UP,>FF00,>0401      ; Up
        DATA  K1DN,>0100,>1401      ; Down
MVLIM   DATA  >38A7,>05F0           ; Bee screen boundaries (y-min,y-max,x-min,x-max)
MVCOMB  BYTE  >40,>D4,>DC,>00       ; Honeycomb boundaries  (y-max,X-min,x-max,dummy)
ADD100  DATA 100                    ; 100 points

***************************************************************
* Thread MVKBEE - Move the killer bees
***************************************************************
MVKBEE  MOV  R10,TMP0               ; Get slot number
        ANDI TMP0,>00FF             ; Get rid of high byte
*--------------------------------------------------------------
*  index = ((slot - 1) << 2) + 1
*--------------------------------------------------------------
        DEC  TMP0,1                 ; index = slot - 1
        SLA  TMP0,2                 ; index << 2
        INC  TMP0                   ; index++
        MOV  R10,TMP1               ; Get slot number
        S    @BSPEED,TMP1           ; Higher slots go faster
        SRL  TMP1,1
        SB   TMP1,@RAMSAT(TMP0)     ; RAMSAT address of killer bee X position
        B    *R11

***************************************************************
* Thread SPNBEE - Do spin bee sequence
********@*****@*********************@**************************
SPNBEE  MOV   @BEEPOS,@RAMSAT+4     ; Reposition bee
        MOV   @COUNTER,R1
        CI    R1,2                  ; Already spinned 2 times ?
        JNE   SPNBE1                ; No continue
*--------------------------------------------------------------
*  Show message "Oops be careful"
*--------------------------------------------------------------
        BL    @PUTAT
        DATA  >0508,OOPS            ; Ooops be careful....
        BL    @MKSLOT
        DATA  >0240,GOGAME,EOL      ; Set delay
        B     @SLOTOK               ; Exit
*--------------------------------------------------------------
*  Spin the bee
*--------------------------------------------------------------
SPNBE1  CLR   R1
        MOVB  @RAMSAT+6,R1
        CI    R1,>2000              ; Last pattern ?
        JNE   SPNBE2
        LI    R1,>0400
        INC   @COUNTER              ; Counter = Counter + 1
        JMP   SPNBE3
SPNBE2  AI    R1,>0400              ; pattern = pattern + 4
SPNBE3  MOVB  R1,@RAMSAT+6
        B     *R11                  ; Exit
BEEPOS  DATA  >5A78                 ; Y=90, X=120
```

```
****************************************************************
* Thread MVPLAN - Move plane accross the screen
****************************************************************
MVPLAN  CB    @RAMSAT+1,@TMPDAT     ; Screen boundary reached ?
        JEQ   MVPLA1                ; Yes, X=0
        INC   @RAMSAT               ; No,  X=X+1
        JMP   MVPLA2                ; Exit
MVPLA1  MOVB  @BD0,@RAMSAT+1        ; X=0
MVPLA2  B     *R11                  ; Exit
TMPDAT  DATA  >FE00


****************************************************************
* Thread CLRMSG - Clear message
********@*****@**********************@**************************
CLRMSG  BL    @HCHAR
        DATA  >0508,>200F           ; Clear 1st message line
        DATA  >060A,>200F,EOL       ; Clear 2nd message line
        BL    @CLSLOT
        DATA  6                     ; Clear slot 6
        B     @SLOTOK               ; Exit
```

```
****************************************************************
* Honeycomb Rapture - (c) 2010 by Owen Brand
*
* Cartridge conversion by Retroclouds
*
* This file: hc_source3.a99
****************************************************************
* Game over
********@*****@*********************@**************************



****************************************************************
* Thread GOGAME - Resume or game over
********@*****@*********************@**************************
GOGAME  DEC   @LIVES                  ; Lives = Lives - 1
        JEQ   GOGAM1                  ; GAME OVER ?
        MOV   @SPRSAT+4,@RAMSAT+4     ; Restore bee start position
        BL    @PUTAT
        DATA  >060A,GO                ; Here we go ....
        SZC   @WBIT0,@GASTAT          ; Reset "bee killed" flag
        SOC   @WBIT13,CONFIG          ; Resume sound player
        BL    @PUTVR
        DATA  >070A                   ; VDP#7 - Set background color
        B     @GAME1                  ; Update stats & resume game
*--------------------------------------------------------------
*  Game over
*--------------------------------------------------------------
GOGAM1  BL    @SCROFF                 ; Screen off
        BL    @VPUTB
        DATA  >0300,>D0               ; End of sprite processing
        BL    @FILV
        DATA  >0000,00,768            ; Fill screen
        BL    @FILV
        DATA  >0380,>16,16            ; Set colors black/red
        BL    @SCRON                  ; Screen on
        BL    @MKSLOT
        DATA  >0020,SHOW1,EOL         ; Display msg with 0.5 seconds interval
        CLR   R10                     ; Set highest slot to 0
        B     @SLOTOK                 ; Back to Thread Scheduler


****************************************************************
* Thread SHOW. - Display "game over" messages
********@*****@*********************@**************************
SHOW1   BL    @PUTAT
        DATA  >0505,MSG1              ; Line 1
        BL    @SDPREP
        DATA  SOUND2,SDOPT2           ; Start "game over" tune
        MOV   @TMP50,@TIMERS          ; Update slot 0 thread address
        JMP   SHOWZ
SHOW2   BL    @PUTAT
        DATA  >0703,MSG2              ; Line 2
        JMP   SHOWY
SHOW3   BL    @PUTAT
        DATA  >0905,MSG3              ; Line 3
        JMP   SHOWY
SHOW4   BL    @PUTAT
```

```
        DATA  >0B09,MSG4            ; Line 4
        JMP   SHOWY
SHOW5   BL    @PUTAT
        DATA  >0F07,MSG5            ; Line 5
        BL    @PUTNUM
        DATA  >0F15,SCORE,BUFFER,>3030
        BL    @MKHOOK
        DATA  WAIKEY                ; User hook - restart game when key pressed
        BL    @MKSLOT
        DATA  >000A,BLKMSG,EOL      ; Blink text "PRESS ANY KEY TO PLAY AGAIN"
        SETO  R0                    ; Use R0 as toggle for BLKMSG
        JMP   SHOWZ
*--------------------------------------------------------------
*  Update subroutine address in slot 0
*--------------------------------------------------------------
SHOWY   A     @ADD10,@TIMERS        ; Update slot 0 thread address
SHOWZ   B     @SLOTOK               ; Back to Thread Scheduler
TMP50   DATA  SHOW2
ADD10   DATA  10                    ; Offset between SHOW1 -> SHOW2 -> SHOW3 -> ... is fixed !!!



*****************************************************************
* User hook TSTKEY - Restart game upon keypress
********@*****@**********************@**************************
WAIKEY  COC   @ANYKEY,R12
        JNE   WAIKE1
        B     @RUNLIB               ; Restart game
WAIKE1  B     @HOOKOK

*****************************************************************
* THREAD - Blink message "PRESS ANY KEY TO PLAY AGAIN"
********@*****@**********************@**************************
BLKMSG  INV   R0
        JEQ   BLKMS1
        BL    @PUTAT
        DATA  >1303,MSG6            ; Show messsage
        JMP   BLKMS2
BLKMS1  BL    @HCHAR
        DATA  >1303,>001C,EOL       ; Erase line 6
BLKMS2  B     @SLOTOK               ; Exit
```

```
****************************************************************
*                          SPECTRA2
*                     Arcade Game Library
*                           for
*               the Texas Instruments TI-99/4A
*                   2010 by Filip Van Vooren
*
****************************************************************
* Special thanks to Mark Wills, Mathew Hagerty & sometimes99er
*
* 1) Speech code based on version of Mark Wills.
* 2) Fat font style based on work of sometimes99er.
* 2) Number conversion based on work of Mathew Hagerty.
****************************************************************
* This file: runlib.a99
****************************************************************
* v1.0.0  2011/02   Initial version
****************************************************************




*//////////////////////////////////////////////////////////////
*                       RUNLIB MEMORY SETUP
*//////////////////////////////////////////////////////////////

* Equates and Memory setup ...
****************************************************************
* >8300 - >833A     Scratchpad memory layout
********@*****@*********************@**************************
WS1     EQU   >8300                 ; 32 - Primary workspace
MCLOOP  EQU   >8320                 ; 08 - Machine code for loop & speech
WBASE   EQU   >8328                 ; 02 - PNT base address
WYX     EQU   >832A                 ; 02 - Cursor YX position
WTITAB  EQU   >832C                 ; 02 - Timers: Address of timer table
WTIUSR  EQU   >832E                 ; 02 - Timers: Address of user hook
WTITMP  EQU   >8330                 ; 02 - Timers: Internal use
WVRTKB  EQU   >8332                 ; 02 - Virtual keyboard flags
WSDLST  EQU   >8334                 ; 02 - Sound player: Tune address
WSDTMP  EQU   >8336                 ; 02 - Sound player: Temporary use
WSPEAK  EQU   >8338                 ; 02 - Speech player: Address of LPC data
WSEED   EQU   >833A                 ; 02 - Seed for random subroutine
****************************************************************
BY      EQU   WYX                   ;      Cursor Y position
BX      EQU   WYX+1                 ;      Cursor X position
MCSPRD  EQU   MCLOOP+2              ;      Speech read routine
****************************************************************
* Register usage
* R0-R3   General purpose registers
* R4-R8   Temporary registers
* R9      Stack pointer
* R10     Highest slot in use + Timer counter
* R11     Subroutine return address
* R12     Configuration register
* R13     Copy of VDP status byte and counter for sound player
* R14     Copy of VDP register #0 and VDP register #1 bytes
* R15     VDP read/write address
****************************************************************
```

```
* Workspace and register equates
********@*****@**********************@**************************
R0      EQU   0
R1      EQU   1
R2      EQU   2
R3      EQU   3
R4      EQU   4
R5      EQU   5
R6      EQU   6
R7      EQU   7
R8      EQU   8
R9      EQU   9
R10     EQU   10
R11     EQU   11
R12     EQU   12
R13     EQU   13
R14     EQU   14
R15     EQU   15
R0HB    EQU   WS1                   ; HI byte R0
R0LB    EQU   WS1+1                 ; LO byte R0
R1HB    EQU   WS1+2                 ; HI byte R1
R1LB    EQU   WS1+3                 ; LO byte R1
R2HB    EQU   WS1+4                 ; HI byte R2
R2LB    EQU   WS1+5                 ; LO byte R2
R3HB    EQU   WS1+6                 ; HI byte R3
R3LB    EQU   WS1+7                 ; LO byte R3
R4HB    EQU   WS1+8                 ; HI byte R4
R4LB    EQU   WS1+9                 ; LO byte R4
R5HB    EQU   WS1+10                ; HI byte R5
R5LB    EQU   WS1+11                ; LO byte R5
R6HB    EQU   WS1+12                ; HI byte R6
R6LB    EQU   WS1+13                ; LO byte R6
R7HB    EQU   WS1+14                ; HI byte R7
R7LB    EQU   WS1+15                ; LO byte R7
R8HB    EQU   WS1+16                ; HI byte R8
R8LB    EQU   WS1+17                ; LO byte R8
R9HB    EQU   WS1+18                ; HI byte R9
R9LB    EQU   WS1+19                ; LO byte R9
R10HB   EQU   WS1+20                ; HI byte R10
R10LB   EQU   WS1+21                ; LO byte R10
R11HB   EQU   WS1+22                ; HI byte R11
R11LB   EQU   WS1+23                ; LO byte R11
R12HB   EQU   WS1+24                ; HI byte R12
R12LB   EQU   WS1+25                ; LO byte R12
R13HB   EQU   WS1+26                ; HI byte R13
R13LB   EQU   WS1+27                ; LO byte R13
R14HB   EQU   WS1+28                ; HI byte R14
R14LB   EQU   WS1+29                ; LO byte R14
R15HB   EQU   WS1+30                ; HI byte R15
R15LB   EQU   WS1+31                ; LO byte R15
TMP0    EQU   R4                    ; Temp register 0
TMP1    EQU   R5                    ; Temp register 1
TMP2    EQU   R6                    ; Temp register 2
TMP3    EQU   R7                    ; Temp register 3
TMP4    EQU   R8                    ; Temp register 4
TMP5    EQU   R9                    ; Temp register 5
TMP6    EQU   R15                   ; Temp register 6
```

```
TMP0HB  EQU   WS1+8                   ; HI byte R4
TMP0LB  EQU   WS1+9                   ; LO byte R4
TMP1HB  EQU   WS1+10                  ; HI byte R5
TMP1LB  EQU   WS1+11                  ; LO byte R5
TMP2HB  EQU   WS1+12                  ; HI byte R6
TMP2LB  EQU   WS1+13                  ; LO byte R6
TMP3HB  EQU   WS1+14                  ; HI byte R7
TMP3LB  EQU   WS1+15                  ; LO byte R7
TMP4HB  EQU   WS1+16                  ; HI byte R8
TMP4LB  EQU   WS1+17                  ; LO byte R8
TMP5HB  EQU   WS1+16                  ; HI byte R8
TMP5LB  EQU   WS1+17                  ; LO byte R8
TMP6HB  EQU   WS1+30                  ; HI byte R15
TMP6LB  EQU   WS1+31                  ; LO byte R15
*****************************************************************
* Equates for VDP, GROM, SOUND, SPEECH ports
********@*****@*********************@**************************
SOUND   EQU   >8400                   ; Sound generator address
VDPR    EQU   >8800                   ; VDP read data window address
VDPW    EQU   >8C00                   ; VDP write data window address
VDPS    EQU   >8802                   ; VDP status register
VDPA    EQU   >8C02                   ; VDP address register
GRMWA   EQU   >9C02                   ; GROM set write address
GRMRA   EQU   >9802                   ; GROM set read address
GRMRD   EQU   >9800                   ; GROM read byte
GRMWD   EQU   >9C00                   ; GROM write byte
SPCHRD  EQU   >9000                   ; Address of speech synth Read Data Register
SPCHWT  EQU   >9400                   ; Address of speech synth Write Data Register
*****************************************************************
* Equates for registers
********@*****@*********************@**************************
STACK   EQU   R9                      ; Stack pointer
BTIHI   EQU   WS1+20                  ; Highest slot in use (HI byte R10)
CONFIG  EQU   R12                     ; SPECTRA configuration register
BVDPST  EQU   WS1+26                  ; Copy of VDP status register (HI byte R13)
VDPR01  EQU   R14                     ; Copy of VDP#0 and VDP#1 bytes
VDPR0   EQU   WS1+28                  ; High byte of R14. Is VDP#0 byte
VDPR1   EQU   WS1+29                  ; Low byte  of R14. Is VDP#1 byte
VDPRW   EQU   R15                     ; Contains VDP read/write address
*****************************************************************
* The config register equates
*---------------------------------------------------------------
* Configuration flags
* ===================
*
* ; 15  Sound player: tune source       1=ROM/RAM      0=VDP MEMORY
* ; 14  Sound player: repeat tune       1=yes          0=no
* ; 13  Sound player: enabled           1=yes          0=no (or pause)
* ; 12  Keyboard: mode                  1=real         0=virtual
* ; 11  Keyboard: ANY key pressed       1=yes          0=no
* ; 10  TI-99/4A v2.2 OS                1=yes          0=no
* ; 09  Timer: Kernel thread enabled    1=yes          0=no
* ; 08  Timer: Block kernel thread      1=yes          0=no
* ; 07  Timer: User hook enabled        1=yes          0=no
* ; 06  Timer: Block user hook          1=yes          0=no
* ; 05  speech player: external voice   1=yes          0=no
* ; 04  Speech player: busy             1=yes          0=no
```

```
* ; 03  Speech player: enabled         1=yes         0=no
* ; 02  VDP9918 PAL version            1=yes(50)     0=no(60)
* ; 01  Subroutine state flag 1        1=on          0=off
* ; 00  Subroutine state flag 0        1=on          0=off
********@*****@**********************@**************************
PALON   EQU   >2000                   ; bit 2=1   (VDP9918 PAL version)
ENUSR   EQU   >0100                   ; bit 7=1   (Enable user hook)
ENKNL   EQU   >0040                   ; bit 9=1   (Enable kernel thread)
V22OS   EQU   >0020                   ; bit 10=1  (TI-99/4A V2.2 OS)
****************************************************************
* Subroutine parameter equates
****************************************************************
EOL     EQU   >FFFF                   ; End-Of-List
NOFONT  EQU   >FFFF                   ; Skip loading font in RUNLIB
NUM1    EQU   >3030                   ; MKNUM  => ASCII 0-9, leading 0's
NUM2    EQU   >3020                   ; MKNUM  => ASCII 0-9, leading spaces
SDOPT1  EQU   7                       ; SDPLAY => 111 (Player on, repeat, tune in CPU memory)
SDOPT2  EQU   5                       ; SDPLAY => 101 (Player on, no repeat, tune in CPU memory)
SDOPT3  EQU   6                       ; SDPLAY => 110 (Player on, repeat, tune in VRAM)
SDOPT4  EQU   4                       ; SDPLAY => 100 (Player on, no repeat, tune in VRAM)
FNOPT1  EQU   >0000                   ;  => Load TI title screen font
FNOPT2  EQU   >0006                   ; LDFNT => Load upper case font
FNOPT3  EQU   >000C                   ; LDFNT => Load upper/lower case font
FNOPT4  EQU   >0012                   ; LDFNT => Load lower case font
FNOPT5  EQU   >8000                   ; LDFNT => Load TI title screen font  & make fat
FNOPT6  EQU   >8006                   ; LDFNT => Load upper case font       & make fat
FNOPT7  EQU   >800C                   ; LDFNT => Load upper/lower case font & make fat
FNOPT8  EQU   >8012                   ; LDFNT => Load lower case font       & make fat
*--------------------------------------------------------------
*   Speech player
*--------------------------------------------------------------
SPOPT1  EQU   >1400                   ; 0001010000000000 (Player on, external voice)
SPOPT2  EQU   >1000                   ; 0001000000000000 (Player on, resident voice)
TALKON  EQU   >60                     ; 'start talking' command code for speech synth
TALKOF  EQU   >FF                     ; 'stop talking' command code for speech synth
SPKON   EQU   >6000                   ; 'start talking' command code for speech synth
SPKOFF  EQU   >FF00                   ; 'stop talking' command code for speech synth
****************************************************************
* Virtual keyboard equates
****************************************************************
* ; bit  0: ALPHA LOCK down           0=no  1=yes
* ; bit  1: ENTER                     0=no  1=yes
* ; bit  2: REDO                      0=no  1=yes
* ; bit  3: BACK                      0=no  1=yes
* ; bit  4: Pause                     0=no  1=yes
* ; bit  5: *free*                    0=no  1=yes
* ; bit  6: P1 Left                   0=no  1=yes
* ; bit  7: P1 Right                  0=no  1=yes
* ; bit  8: P1 Up                     0=no  1=yes
* ; bit  9: P1 Down                   0=no  1=yes
* ; bit 10: P1 Space / fire / Q       0=no  1=yes
* ; bit 11: P2 Left                   0=no  1=yes
* ; bit 12: P2 Right                  0=no  1=yes
* ; bit 13: P2 Up                     0=no  1=yes
* ; bit 14: P2 Down                   0=no  1=yes
* ; bit 15: P2 Space / fire / Q       0=no  1=yes
****************************************************************
```

```
KALPHA  EQU   >8000                  ; Virtual key alpha lock
KENTER  EQU   >4000                  ; Virtual key enter
KREDO   EQU   >2000                  ; Virtual key REDO
KBACK   EQU   >1000                  ; Virtual key BACK
KPAUSE  EQU   >0800                  ; Virtual key pause
KFREE   EQU   >0400                  ; ***NOT USED YET***
*--------------------------------------------------------------
* Keyboard Player 1
*--------------------------------------------------------------
K1UPLF  EQU   >0280                  ; Virtual key up   + left
K1UPRG  EQU   >0180                  ; Virtual key up   + right
K1DNLF  EQU   >0240                  ; Virtual key down + left
K1DNRG  EQU   >0140                  ; Virtual key down + right
K1LF    EQU   >0200                  ; Virtual key left
K1RG    EQU   >0100                  ; Virtual key right
K1UP    EQU   >0080                  ; Virtual key up
K1DN    EQU   >0040                  ; Virtual key down
K1FIRE  EQU   >0020                  ; Virtual key fire
*--------------------------------------------------------------
* Keyboard Player 2
*--------------------------------------------------------------
K2UPLF  EQU   >0014                  ; Virtual key up   + left
K2UPRG  EQU   >000C                  ; Virtual key up   + right
K2DNLF  EQU   >0012                  ; Virtual key down + left
K2DNRG  EQU   >000A                  ; Virtual key down + right
K2LF    EQU   >0010                  ; Virtual key left
K2RG    EQU   >0008                  ; Virtual key right
K2UP    EQU   >0004                  ; Virtual key up
K2DN    EQU   >0002                  ; Virtual key down
K2FIRE  EQU   >0001                  ; Virtual key fire
****************************************************************
* Misc equates (bank switching, etc.)
********@*****@**********************@**************************
BANK0   EQU   >6000
BANK1   EQU   >6002
****************************************************************
*                      Some constants
********@*****@**********************@**************************
        EVEN                         ; Just in case
WBIT0   DATA  >8000                  ; Binary 1000000000000000
WBIT1   DATA  >4000                  ; Binary 0100000000000000
WBIT2   DATA  >2000                  ; Binary 0010000000000000
WBIT3   DATA  >1000                  ; Binary 0001000000000000
WBIT4   DATA  >0800                  ; Binary 0000100000000000
WBIT5   DATA  >0400                  ; Binary 0000010000000000
WBIT6   DATA  >0200                  ; Binary 0000001000000000
WBIT7   DATA  >0100                  ; Binary 0000000100000000
WBIT8   DATA  >0080                  ; Binary 0000000010000000
WBIT9   DATA  >0040                  ; Binary 0000000001000000
WBIT10  DATA  >0020                  ; Binary 0000000000100000
WBIT11  DATA  >0010                  ; Binary 0000000000010000
WBIT12  DATA  >0008                  ; Binary 0000000000001000
WBIT13  DATA  >0004                  ; Binary 0000000000000100
WBIT14  DATA  >0002                  ; Binary 0000000000000010
WBIT15  DATA  >0001                  ; Binary 0000000000000001
WHFFFF  DATA  >FFFF                  ; Binary 1111111111111111
BD0     BYTE  0                      ; Digit 0
```

```
BD1     BYTE  1                       ; Digit 1
BD2     BYTE  2                       ; Digit 2
BD3     BYTE  3                       ; Digit 3
BD4     BYTE  4                       ; Digit 4
BD5     BYTE  5                       ; Digit 5
BD6     BYTE  6                       ; Digit 6
BD7     BYTE  7                       ; Digit 7
BD8     BYTE  8                       ; Digit 8
BD9     BYTE  9                       ; Digit 9
BD208   BYTE  208                     ; Digit 208 (>E0)
        EVEN
*--------------------------------------------------------------
* The equates for constants
*--------------------------------------------------------------
ANYKEY  EQU   WBIT11                  ; BIT 11 in the CONFIG register
BBIT0   EQU   WBIT0
BBIT1   EQU   WBIT1
BBIT2   EQU   WBIT2
BBIT3   EQU   WBIT3
BBIT4   EQU   WBIT4
BBIT5   EQU   WBIT5
BBIT6   EQU   WBIT6
BBIT7   EQU   WBIT7
BH10    EQU   WBIT11+1                ; >10
BH20    EQU   WBIT10+1                ; >20
BH40    EQU   WBIT9+1                 ; >40
BH80    EQU   WBIT8+1                 ; >80
WH100   EQU   WBIT7                   ; >0100
WH4000  EQU   WBIT1                   ; >4000

*****************************************************************
*                     Video mode tables
*****************************************************************
* Graphics mode 1 (32 columns)
*--------------------------------------------------------------
GRAPH1  BYTE  >00,>E2,>00,>0E,>01,>06,>02,SPFBCK
* ; VDP#0 Control bits
* ;      bit 6=0: M3 | Graphics 1 mode
* ;      bit 7=0: Disable external VDP input
* ; VDP#1 Control bits
* ;      bit 0=1: 16K selection
* ;      bit 1=1: Enable display
* ;      bit 2=1: Enable VDP interrupt
* ;      bit 3=0: M1 \ Graphics 1 mode
* ;      bit 4=0: M2 /
* ;      bit 5=0: reserved
* ;      bit 6=1: 16x16 sprites
* ;      bit 7=0: Sprite magnification (1x)
* ; VDP#2 PNT (Pattern name table)       at >0000  (>00 * >400)
* ; VDP#3 PCT (Pattern color table)      at >0380  (>0E * >040)
* ; VDP#4 PDT (Pattern descriptor table) at >0800  (>01 * >800)
* ; VDP#5 SAT (sprite attribute list)    at >0300  (>06 * >080)
* ; VDP#6 SPT (Sprite pattern table)     at >0400  (>80 * >008)
* ; VDP#7 Set screen background color
*****************************************************************
* Textmode (40 columns)
*--------------------------------------------------------------
```

```
TXTMOD  BYTE  >00,>F2,>00,>0E,>01,>06,>80,SPFCLR
* ; VDP#0 Control bits
* ;      bit 6=0: M3 | Graphics 1 mode
* ;      bit 7=0: Disable external VDP input
* ; VDP#1 Control bits
* ;      bit 0=1: 16K selection
* ;      bit 1=1: Enable display
* ;      bit 2=1: Enable VDP interrupt
* ;      bit 3=1: M1 \ TEXT MODE
* ;      bit 4=0: M2 /
* ;      bit 5=0: reserved
* ;      bit 6=1: 16x16 sprites
* ;      bit 7=0: Sprite magnification (1x)
* ; VDP#2 PNT (Pattern name table)       at >0000  (>00 * >400)
* ; VDP#3 PCT (Pattern color table)      at >0380  (>0E * >040)
* ; VDP#4 PDT (Pattern descriptor table) at >0800  (>01 * >800)
* ; VDP#5 SAT (sprite attribute list)    at >0300  (>06 * >080)
* ; VDP#6 SPT (Sprite pattern table)     at >0400  (>80 * >008)
* ; VDP#7 Set foreground/background color
*****************************************************************



*****************************************************************
*                 Data used by runtime library
********@*****@**********************@**************************
KDATA   DATA  KERNEL                  ; Address of kernel thread
*---------------------------------------------------------------
* ; Machine code for tight loop.
* ; The MOV operation at MCLOOP must be injected by the calling routine.
*---------------------------------------------------------------
*       DATA  >????                   ; \ MCLOOP  MOV   ...
MCCODE  DATA  >0606                   ; |         DEC   R6 (TMP2)
        DATA  >16FD                   ; |         JNE   MCLOOP
        DATA  >045B                   ; /         B     *R11
*---------------------------------------------------------------
* ; Machine code for reading from the speech synthesizer
* ; The SRC instruction takes 12 uS for execution in scratchpad RAM.
* ; Is required for the 12 uS delay. It destroys R5.
*---------------------------------------------------------------
SPCODE  DATA  >D114                   ; \         MOVB  *R4,R4 (TMP0)
        DATA  >0BC5                   ; /         SRC   R5,12  (TMP1)
        EVEN
*...



*///////////////////////////////////////////////////////////////
*                      FILL & COPY FUNCTIONS
*///////////////////////////////////////////////////////////////



* FILM   (DATA P0,P1,P2) / XFILM ..:
*****************************************************************
* FILM - Fill CPU memory with byte
*****************************************************************
*  BL   @FILM
*  DATA P0,P1,P2
*---------------------------------------------------------------
```

```
*  P0 = Memory start address
*  P1 = Byte to fill
*  P2 = Number of bytes to fill
*--------------------------------------------------------------
*  BL   @XFILM
*
*  TMP0 = Memory start address
*  TMP1 = Byte to fill
*  TMP2 = Number of bytes to fill
********@*****@**********************@**************************
FILM    MOV   *R11+,TMP0            ; Memory start
        MOV   *R11+,TMP1            ; Byte to fill
        MOV   *R11+,TMP2            ; Repeat count
*--------------------------------------------------------------
* Fill memory with 16 bit words
*--------------------------------------------------------------
XFILM   MOV   TMP2,TMP3
        ANDI  TMP3,1                ; TMP3=1 -> ODD else EVEN
        JEQ   FILM1
        DEC   TMP2                  ; Make TMP2 even
FILM1   MOVB  @TMP1LB,@TMP1HB       ; Duplicate value
FILM2   MOV   TMP1,*TMP0+
        DECT  TMP2
        JNE   FILM2
*--------------------------------------------------------------
* Fill last byte if ODD
*--------------------------------------------------------------
        MOV   TMP3,TMP3
        JEQ   FILMZ
        MOVB  TMP1,*TMP0
FILMZ   B     *R11
* ...

* FILV   (DATA P0,P1,P2) / XFILV ...
***************************************************************
* FILV - Fill VRAM with byte
***************************************************************
*  BL   @FILV
*  DATA P0,P1,P2
*--------------------------------------------------------------
*  P0 = VDP start address
*  P1 = Byte to fill
*  P2 = Number of bytes to fill
*--------------------------------------------------------------
*  BL   @XFILV
*
*  TMP0 = VDP start address
*  TMP1 = Byte to fill
*  TMP2 = Number of bytes to fill
********@*****@**********************@**************************
FILV    MOV   *R11+,TMP0            ; Memory start
        MOV   *R11+,TMP1            ; Byte to fill
        MOV   *R11+,TMP2            ; Repeat count
*--------------------------------------------------------------
*    Setup VDP write address
*--------------------------------------------------------------
XFILV   ORI   TMP0,>4000
```

```
        SWPB  TMP0
        MOVB  TMP0,@VDPA
        SWPB  TMP0
        MOVB  TMP0,@VDPA
*--------------------------------------------------------------
*    Fill bytes in VDP memory
*--------------------------------------------------------------
        LI    R15,VDPW              ; Set VDP write address
        SWPB  TMP1
        MOV   @FILZZ,@MCLOOP        ; Setup move command
        B     @MCLOOP               ; Write data to VDP
FILZZ   DATA  >D7C5                 ; MOVB TMP1,*R15
*...

* CPYM2M (DATA P0,P1,P2) / XPYM2M ...:
****************************************************************
* CPYM2M - Copy CPU memory to CPU memory
****************************************************************
*  BL   @CPYM2M
*  DATA P0,P1,P2
*--------------------------------------------------------------
*  P0 = Memory source address
*  P1 = Memory target address
*  P2 = Number of bytes to copy
*--------------------------------------------------------------
*  BL @XPYM2M
*
*  TMP0 = Memory source address
*  TMP1 = Memory target address
*  TMP2 = Number of bytes to copy
********@*****@**********************@**************************
CPYM2M  MOV   *R11+,TMP0            ; Memory source address
        MOV   *R11+,TMP1            ; Memory target address
        MOV   *R11+,TMP2            ; Number of bytes to copy
*--------------------------------------------------------------
* Do some checks first
*--------------------------------------------------------------
XPYM2M  ANDI  CONFIG,>7FFF          ; Clear CONFIG bit 0
        MOV   TMP0,TMP3
        ANDI  TMP3,1
        JNE   CPYODD                ; Odd source address handling
CPYM1   MOV   TMP1,TMP3
        ANDI  TMP3,1
        JNE   CPYODD                ; Odd target address handling
*--------------------------------------------------------------
* 8 bit copy
*--------------------------------------------------------------
CPYM2   COC   @WBIT0,CONFIG         ; CONFIG bit 0 set ?
        JNE   CPYM3
        MOV   @TMP011,@MCLOOP       ; Setup byte copy command
        B     @MCLOOP               ; Copy memory and exit
*--------------------------------------------------------------
* 16 bit copy
*--------------------------------------------------------------
CPYM3   MOV   TMP2,TMP3
        ANDI  TMP3,1                ; TMP3=1 -> ODD else EVEN
        JEQ   CPYM4
```

```
        DEC   TMP2                    ; Make TMP2 even
CPYM4   MOV   *TMP0+,*TMP1+
        DECT  TMP2
        JNE   CPYM4
*--------------------------------------------------------------
* Copy last byte if ODD
*--------------------------------------------------------------
        MOV   TMP3,TMP3
        JEQ   CPYMZ
        MOVB  *TMP0,*TMP1
CPYMZ   B     *R11
*--------------------------------------------------------------
* Handle odd source/target address
*--------------------------------------------------------------
CPYODD  ORI   CONFIG,>8000          ; Set CONFIG bot 0
        JMP   CPYM2
TMP011  DATA  >DD74                 ; MOVB *TMP0+,*TMP1+


* ...

* CPYM2V (DATA P0,P1,P2) / XPYM2V ...
****************************************************************
* CPYM2V - Copy CPU memory to VRAM
****************************************************************
*  BL   @CPYM2V
*  DATA P0,P1,P2
*--------------------------------------------------------------
*  P0 = VDP start address
*  P1 = RAM/ROM start address
*  P2 = Number of bytes to copy
*--------------------------------------------------------------
*  BL @XPYM2V
*
*  TMP0 = VDP start address
*  TMP1 = RAM/ROM start address
*  TMP2 = Number of bytes to copy
********@*****@**********************@**************************
CPYM2V  MOV   *R11+,TMP0            ; VDP Start address
        MOV   *R11+,TMP1            ; RAM/ROM start address
        MOV   *R11+,TMP2            ; Bytes to copy
*--------------------------------------------------------------
*    Setup VDP write address
*--------------------------------------------------------------
XPYM2V  ORI   TMP0,>4000
        SWPB  TMP0
        MOVB  TMP0,@VDPA
        SWPB  TMP0
        MOVB  TMP0,@VDPA
*--------------------------------------------------------------
*    Copy bytes from CPU memory to VRAM
*--------------------------------------------------------------
        LI    R15,VDPW              ; Set VDP write address
        MOV   @TMP008,@MCLOOP       ; Setup copy command
        B     @MCLOOP               ; Write data to VDP
TMP008  DATA  >D7F5                 ; MOVB *TMP1+,*R15
*...
```

```
* CPYV2M (DATA P0,P1,P2) / XPYV2M ..:
****************************************************************
* CPYV2M - Copy VRAM to CPU memory
****************************************************************
*  BL @CPYV2M
*  DATA P0,P1,P2
*--------------------------------------------------------------
*  P0 = VDP source address
*  P1 = RAM target address
*  P2 = Number of bytes to copy
*--------------------------------------------------------------
*  BL @XPYV2M
*
*  TMP0 = VDP source address
*  TMP1 = RAM target address
*  TMP2 = Number of bytes to copy
********@*****@**********************@**************************
CPYV2M  MOV   *R11+,TMP0            ; VDP source address
        MOV   *R11+,TMP1            ; Target address in RAM
        MOV   *R11+,TMP2            ; Bytes to copy
*--------------------------------------------------------------
*    Setup VDP read address
*--------------------------------------------------------------
XPYV2M  SWPB  TMP0
        MOVB  TMP0,@VDPA
        SWPB  TMP0
        MOVB  TMP0,@VDPA
*--------------------------------------------------------------
*    Copy bytes from VDP memory to RAM
*--------------------------------------------------------------
        LI    R15,VDPR              ; Set VDP read address
        MOV   @TMP007,@MCLOOP       ; Setup copy command
        B     @MCLOOP               ; Read data from VDP
TMP007  DATA  >DD5F                 ; MOVB *R15,*TMP+
*...

* CPYG2M (DATA P0,P1,P2) / XPYG2M ..:
****************************************************************
* CPYG2M - Copy GROM memory to CPU memory
****************************************************************
*  BL @CPYG2M
*  DATA P0,P1,P2
*--------------------------------------------------------------
*  P0 = GROM source address
*  P1 = CPU target address
*  P2 = Number of bytes to copy
*--------------------------------------------------------------
*  BL @CPYG2M
*
*  TMP0 = GROM source address
*  TMP1 = CPU target address
*  TMP2 = Number of bytes to copy
********@*****@**********************@**************************
CPYG2M  MOV   *R11+,TMP0            ; Memory source address
        MOV   *R11+,TMP1            ; Memory target address
        MOV   *R11+,TMP2            ; Number of bytes to copy
```

```
*--------------------------------------------------------------
* Setup GROM source address
*--------------------------------------------------------------
XPYG2M  MOVB  TMP0,@GRMWA
        SWPB  TMP0
        MOVB  TMP0,@GRMWA
*--------------------------------------------------------------
*    Copy bytes from GROM to CPU memory
*--------------------------------------------------------------
        LI    TMP0,GRMRD            ; Set TMP0 to GROM data port
        MOV   @TMP003,@MCLOOP       ; Setup copy command
        B     @MCLOOP               ; Copy bytes
TMP003  DATA  >DD54                 ; MOVB *TMP0,*TMP1+
*...

* CPYG2V (DATA P0,P1,P2) / XPYG2V ..:
****************************************************************
* CPYG2V - Copy GROM memory to VRAM memory
****************************************************************
*  BL @CPYG2V
*  DATA P0,P1,P2
*--------------------------------------------------------------
*  P0 = GROM source address
*  P1 = VDP target address
*  P2 = Number of bytes to copy
*--------------------------------------------------------------
*  BL @CPYG2V
*
*  TMP0 = GROM source address
*  TMP1 = VDP target address
*  TMP2 = Number of bytes to copy
********@*****@**********************@**************************
CPYG2V  MOV   *R11+,TMP0            ; Memory source address
        MOV   *R11+,TMP1            ; Memory target address
        MOV   *R11+,TMP2            ; Number of bytes to copy
*--------------------------------------------------------------
* Setup GROM source address
*--------------------------------------------------------------
XPYG2V  MOVB  TMP0,@GRMWA
        SWPB  TMP0
        MOVB  TMP0,@GRMWA
*--------------------------------------------------------------
* Setup VDP target address
*--------------------------------------------------------------
        ORI   TMP1,>4000
        SWPB  TMP1
        MOVB  TMP1,@VDPA
        SWPB  TMP1
        MOVB  TMP1,@VDPA            ; Set VDP target address
*--------------------------------------------------------------
*    Copy bytes from GROM to VDP memory
*--------------------------------------------------------------
        LI    TMP3,GRMRD            ; Set TMP3 to GROM data port
        LI    R15,VDPW              ; Set VDP write address
        MOV   @TMP002,@MCLOOP       ; Setup copy command
        B     @MCLOOP               ; Copy bytes
TMP002  DATA  >D7D7                 ; MOVB *TMP3,*R15
```

```
*...


*////////////////////////////////////////////////////////////////
*                       VDP LOW LEVEL FUNCTIONS
*////////////////////////////////////////////////////////////////

* VDWA   () / VDRA () ...
*****************************************************************
* VDWA / VDRA - Setup VDP write or read address
*****************************************************************
*  BL   @VDWA
*
*  TMP0 = VDP destination address for write
*---------------------------------------------------------------
*  BL   @VDRA
*
*  TMP0 = VDP source address for read
********@*****@**********************@**************************
VDWA    ORI   TMP0,>4000            ; Prepare VDP address for write
VDRA    SWPB  TMP0
        MOVB  TMP0,@VDPA
        SWPB  TMP0
        MOVB  TMP0,@VDPA            ; Set VDP address
        B     *R11
* ...

* VPUTB  (DATA P0, P1) / XVPUTB ...
*****************************************************************
* VPUTB - VDP put single byte
*****************************************************************
*  BL @VPUTB
*  DATA P0,P1
*---------------------------------------------------------------
*  P0 = VDP target address
*  P1 = Byte to write
********@*****@**********************@**************************
VPUTB   MOV   *R11+,TMP0            ; Get VDP target address
        MOV   *R11+,TMP1
XVPUTB  MOV   R11,TMP2              ; Save R11
        BL    @VDWA                 ; Set VDP write address
        SWPB  TMP1                  ; Get byte to write
        MOVB  TMP1,*R15             ; Write byte
        B     *TMP2                 ; Exit
*...

* VGETB  (DATA P0) / XVGETB ...
*****************************************************************
* VGETB - VDP get single byte
*****************************************************************
*  BL @VGETB
*  DATA P0
*---------------------------------------------------------------
*  P0 = VDP source address
********@*****@**********************@**************************
VGETB   MOV   *R11+,TMP0            ; Get VDP source address
XVGETB  MOV   R11,TMP2              ; Save R11
```

```
        BL    @VDRA                 ; Set VDP read address
        MOVB  @VDPR,TMP0            ; Read byte
        SRL   TMP0,8                ; Right align
        B     *TMP2                 ; Exit
*...

* VIDTAB (DATA P0) / XIDTAB ...:
*****************************************************************
* VIDTAB - Dump videomode table
*****************************************************************
*  BL   @VIDTAB
*  DATA P0
*---------------------------------------------------------------
*  P0 = Address of video mode table
*---------------------------------------------------------------
*  BL   @XIDTAB
*
*  TMP0 = Address of video mode table
*---------------------------------------------------------------
*  Remarks
*  TMP1 = MSB is the VDP target register
*         LSB is the value to write
********@*****@**********************@**************************
VIDTAB  MOV   *R11+,TMP0            ; Get video mode table
XIDTAB  MOV   *TMP0,R14             ; Store copy of VDP#0 and #1 in RAM
*---------------------------------------------------------------
* Calculate PNT base address
*---------------------------------------------------------------
        MOV   TMP0,TMP1
        INCT  TMP1
        MOVB  *TMP1,TMP1            ; Get value for VDP#2
        ANDI  TMP1,>FF00            ; Only keep MSB
        SLA   TMP1,2                ; TMP1 *= 400
        MOV   TMP1,@WBASE           ; Store calculated base
*---------------------------------------------------------------
* Dump VDP shadow registers
*---------------------------------------------------------------
        LI    TMP1,>8000            ; Start with VDP register 0
        LI    TMP2,8
VIDTA1  MOVB  *TMP0+,@TMP1LB        ; Write value to VDP register
        SWPB  TMP1
        MOVB  TMP1,@VDPA
        SWPB  TMP1
        MOVB  TMP1,@VDPA
        AI    TMP1,>0100
        DEC   TMP2
        JNE   VIDTA1                ; Next register
        B     *R11
*...

* PUTVR  (DATA P0) / PUTVRX ...:
*****************************************************************
* PUTVR  - Put single VDP register
*****************************************************************
*  BL   @PUTVR
*  DATA P0
*---------------------------------------------------------------
```

```
*  P0 = MSB is the VDP target register
*       LSB is the value to write
*--------------------------------------------------------------
*  BL   @PUTVRX
*
*  TMP0 = MSB is the VDP target register
*         LSB is the value to write
********@*****@*********************@**************************
PUTVR   MOV   *R11+,TMP0
PUTVRX  ORI   TMP0,>8000
        SWPB  TMP0
        MOVB  TMP0,@VDPA
        SWPB  TMP0
        MOVB  TMP0,@VDPA
        B     *R11
*...

* PUTV01 () ...
***************************************************************
* PUTV01  - Put VDP registers #0 and #1
***************************************************************
*  BL   @PUTV01
********@*****@*********************@**************************
PUTV01  MOV   R11,TMP4              ; Save R11
        MOV   R14,TMP0
        SRL   TMP0,8
        BL    @PUTVRX               ; Write VR#0
        LI    TMP0,>0100
        MOVB  @R14LB,@TMP0LB
        BL    @PUTVRX               ; Write VR#1
        B     *TMP4                 ; Exit
* ...

* SCROFF () ...
***************************************************************
* SCROFF - Disable screen display
***************************************************************
*  BL @SCROFF
********@*****@*********************@**************************
SCROFF  SZC   @WBIT9,R14            ; VDP#R1 bit 1=0 (Disable screen display)
        JMP   PUTV01
*...

* SCRON  () ...
***************************************************************
* SCRON - Disable screen display
***************************************************************
*  BL @SCRON
********@*****@*********************@**************************
SCRON   SOC   @WBIT9,R14            ; VDP#R1 bit 1=1 (Enable screen display)
        JMP   PUTV01
*...

* INTOFF () ...
***************************************************************
* INTOFF - Disable VDP interrupt
***************************************************************
```

```
*  BL @INTOFF
********@*****@*********************@**************************
INTOFF  SZC   @WBIT10,R14           ; VDP#R1 bit 2=0 (Disable VDP interrupt)
        JMP   PUTV01
*...

* INTON  () ...
****************************************************************
* INTON - Enable VDP interrupt
****************************************************************
*  BL @INTON
********@*****@*********************@**************************
INTON   SOC   @WBIT10,R14           ; VDP#R1 bit 2=1 (Enable VDP interrupt)
        JMP   PUTV01
*...

* SMAG1X () ...
****************************************************************
* SMAG1X - Set sprite magnification 1x
****************************************************************
*  BL @SMAG1X
********@*****@*********************@**************************
SMAG1X  SZC   @WBIT14,R14           ; VDP#R1 bit 7=0 (Sprite magnification 1x)
        JMP   PUTV01
*...

* SMAG2X () ...
****************************************************************
* SMAG2X - Set sprite magnification 2x
****************************************************************
*  BL @SMAG2X
********@*****@*********************@**************************
SMAG2X  SZC   @WBIT14,R14           ; VDP#R1 bit 7=1 (Sprite magnification 2x)
        JMP   PUTV01
*...

* S8X8   () ...
****************************************************************
* S8X8 - Set sprite size 8x8 bits
****************************************************************
*  BL @S8X8
********@*****@*********************@**************************
S8X8    SZC   @WBIT12,R14           ; VDP#R1 bit 6=0 (Sprite size 8x8)
        JMP   PUTV01
*...

* S16X16 () ...
****************************************************************
* S16X16 - Set sprite size 16x16 bits
****************************************************************
*  BL @S16X16
********@*****@*********************@**************************
S16X16  SOC   @WBIT12,R14           ; VDP#R1 bit 6=1 (Sprite size 16x16)
        JMP   PUTV01
*...

* GTCLMN () ...
```

```
****************************************************************
* GTCLMN - Get number of columns per row
****************************************************************
*  BL   @GTCOLM
*--------------------------------------------------------------
*  OUTPUT
*  TMP0 = Number of columns per row
*--------------------------------------------------------------
*  See VDP Programmers guide, Page 5-1
*
*  m1      m2      m3
*  bit10   bit11   bit6    cols    mask    Mode
*  0       1       0       64      >008    Multicolor mode
*  1       0       0       40      >010    Textmode
*  0       0       0       32      >000    Graphics 1/2 mode
********@*****@**********************@**************************
GTCLMN  MOV   R14,TMP0               ; Get VDP#0 & VDP#1
        SZC   @TMP009,TMP0           ; Remove all inrelevant flags
        JEQ   GTCLM1                 ; Graphic 1/2 mode
        CI    TMP0,>010
        JEQ   GTCLM2                 ; Text mode
        JMP   GTCLM3                 ; Multicolor mode !
GTCLM1  LI    TMP0,32                ; 32 columns per row
        JMP   GTCLMZ
GTCLM2  LI    TMP0,40                ; 40 columns per row
        JMP   GTCLMZ
GTCLM3  LI    TMP0,64                ; 64 columns per row
GTCLMZ  B     *R11                   ; Exit
TMP009  DATA  >FDE7                  ; 1111110111100111
* ...

* YX2PNT (@YX) ...
****************************************************************
* YX2PNT - Get VDP PNT address for current YX cursor position
****************************************************************
*  BL   @YX2PNT
*--------------------------------------------------------------
*  INPUT
*  @WYX   = Cursor YX position
*--------------------------------------------------------------
*  OUTPUT
*  TMP0 = VDP address for entry in Pattern Name Table
*--------------------------------------------------------------
*  Register usage
*  TMP0, TMP6 (R15)
********@*****@**********************@**************************
YX2PNT  MOV   @WYX,TMP6
        ANDI  TMP6,>FF00             ; Get rid of LSB
        MOV   R14,TMP0               ; Get VDP#0 & VDP#1
        SZC   @TMP009,TMP0           ; Remove all inrelevant flags
        JEQ   YX2PN2                 ; 32 cols per row
        CI    TMP0,>008              ; Multicolor mode ?
        JEQ   YX2PN3
*--------------------------------------------------------------
* Text mode
*--------------------------------------------------------------
        MOV   TMP6,TMP0
```

```
        SRL   TMP6,3                ; pos1 = Y * 32
        SRL   TMP0,5                ; pos2 = Y * 8
        A     TMP0,TMP6             ; pos  = pos1 + pos2
        JMP   YX2PN4
*--------------------------------------------------------------
* Graphics 1/2 mode
*--------------------------------------------------------------
YX2PN2  SRL   TMP6,3                ; pos = Y * 32
        JMP   YX2PN4
*--------------------------------------------------------------
* Multicolor mode
*--------------------------------------------------------------
YX2PN3  SRL   TMP6,4                ; pos = Y * 64
*--------------------------------------------------------------
* Do rest of calculation
*--------------------------------------------------------------
YX2PN4  AB    @BX,@TMP6LB           ; pos = pos + X
        A     @WBASE,TMP6           ; pos = pos + (PNT base address)
YX2PN5  MOV   TMP6,TMP0
        LI    R15,VDPW              ; Set VDP write address
        B     *R11
* ...

* YX2PX (@YX) / YX2PXX ...:
****************************************************************
* YX2PX - Get pixel position for cursor YX position
****************************************************************
*  BL   @YX2PX
*
*  (CONFIG:0 = 1) = Skip sprite adjustment
*--------------------------------------------------------------
*  INPUT
*  @WYX   = Cursor YX position
*--------------------------------------------------------------
*  OUTPUT
*  TMP0HB = Y pixel position
*  TMP0LB = X pixel position
*--------------------------------------------------------------
*  Remarks
*  This subroutine does not support multicolor or text mode
********@*****@**********************@**************************
YX2PX   MOV   @WYX,TMP0
YX2PXX  MOV   R11,TMP2              ; Save return address
        SWPB  TMP0                  ; Y<->X
        MOVB  TMP0,TMP1             ; Copy X to TMP1
        SLA   TMP1,3                ; X=X*8
        SLA   TMP0,3                ; Y=Y*8
        MOVB  TMP1,TMP0
        SWPB  TMP0                  ; X<->Y
YX2PI1  COC   @WBIT0,CONFIG         ; Skip sprite adjustment ?
        JEQ   YX2PI3                ; Yes, exit
*--------------------------------------------------------------
* Adjust for Y sprite location
* See VDP Programmers Guide, Section 9.2.1
*--------------------------------------------------------------
YX2PI2  SB    @BD1,TMP0             ; Adjust Y. Top of screen is at >FF
        CB    @BD208,TMP0           ; Y position = >D0 ?
```

```
        JEQ   YX2PI2                ; Yes, but that's not allowed, correct
YX2PI3  B     *TMP2                 ; Exit
BDIG32  DATA  >2000
* ...

* PX2YX ...:
*****************************************************************
* PX2YX - Get YX tile position for specified YX pixel position
*****************************************************************
*  BL   @PX2YX
*---------------------------------------------------------------
*  INPUT
*  TMP0   = Pixel YX position
*
*  (CONFIG:0 = 1) = Skip sprite adjustment
*---------------------------------------------------------------
*  OUTPUT
*  TMP0HB = Y tile position
*  TMP0LB = X tile position
*  TMP1HB = Y pixel offset
*  TMP1LB = X pixel offset
*---------------------------------------------------------------
*  Remarks
*  This subroutine does not support multicolor or text mode
********@*****@**********************@**************************
PX2YX   COC   @WBIT0,CONFIG         ; Skip sprite adjustment ?
        JEQ   PX2YX1
        AI    TMP0,>0100            ; Adjust Y. Top of screen is at >FF
PX2YX1  MOV   TMP0,TMP1             ; Copy YX
        MOV   TMP0,TMP2             ; Copy YX
*---------------------------------------------------------------
* Calculate Y tile position
*---------------------------------------------------------------
        SRL   TMP0,11               ; Y: Move to TMP0LB & (Y = Y / 8)
*---------------------------------------------------------------
* Calculate Y pixel offset
*---------------------------------------------------------------
        MOV   TMP0,TMP3             ; Y: Copy Y tile to TMP3LB
        SLA   TMP3,11               ; Y: Move to TMP3HB & (Y = Y * 8)
        NEG   TMP3
        AB    TMP1,TMP3             ; Y: offset = Y pixel old + (-Y) pixel new
*---------------------------------------------------------------
* Calculate X tile position
*---------------------------------------------------------------
        ANDI  TMP1,>00FF            ; Clear TMP1HB
        SLA   TMP1,5                ; X: Move to TMP1HB & (X = X / 8)
        MOVB  TMP1,TMP0             ; X: TMP0 <-- XY tile position
        SWPB  TMP0                  ; XY tile position <-> YX tile position
*---------------------------------------------------------------
* Calculate X pixel offset
*---------------------------------------------------------------
        ANDI  TMP1,>FF00            ; X: Get rid of remaining junk in TMP1LB
        SLA   TMP1,3                ; X: (X = X * 8)
        NEG   TMP1
        SWPB  TMP2                  ; YX <-> XY
        AB    TMP2,TMP1             ; offset X = X pixel old  + (-X) pixel new
        SWPB  TMP1                  ; X0 <-> 0X
```

```
        MOVB  TMP3,TMP1             ; 0X --> YX
        B     *R11                  ; Exit
*  ...



*//////////////////////////////////////////////////////////////
*                       VDP TILE FUNCTIONS
*//////////////////////////////////////////////////////////////

* LDFNT  (DATA P0,P1) ..:
*****************************************************************
*   - Load TI-99/4A font from GROM into VDP
*****************************************************************
*   BL   @LDFNT
*   DATA P0,P1
*---------------------------------------------------------------
*   P0 = VDP Target address
*   P1 = Font options
********@*****@**********************@**************************
LDFNT   MOV   R11,TMP4              ; Save R11
        INCT  R11                   ; Get 2nd parameter (font options)
        MOV   *R11,TMP0             ; Get parameter value
        ANDI  CONFIG,>7FFF          ; CONFIG register bit 0=0
        COC   @WBIT0,TMP0
        JNE   LDFNT1
        ORI   CONFIG,>8000          ; CONFIG register bit 0=1
        ANDI  TMP0,>7FFF            ; Parameter value bit 0=0
LDFNT1  MOV   @TMP006(TMP0),TMP0    ; Load GROM index address ...
        LI    TMP1,TMP3HB           ; ... into register TMP3
        LI    TMP2,2
        BL    @XPYG2M               ; Get font table address
*---------------------------------------------------------------
* Setup GROM source and VDP target address
*---------------------------------------------------------------
        MOVB  TMP3,@GRMWA
        SWPB  TMP3
        MOVB  TMP3,@GRMWA           ; Setup GROM address for reading
        MOV   *TMP4,TMP0            ; Get 1st parameter (VDP destination)
        BL    @VDWA                 ; Setup VDP destination address
        INCT  TMP4                  ; R11=R11+2
        MOV   *TMP4,TMP1            ; Get font options into TMP1
        ANDI  TMP1,>7FFF            ; Parameter value bit 0=0
        MOV   @TMP006+2(TMP1),TMP2  ; Get number of patterns to copy
        MOV   @TMP006+4(TMP1),TMP1  ; 7 or 8 byte pattern ?
*---------------------------------------------------------------
* Copy from GROM to VRAM
*---------------------------------------------------------------
LDFNT2  SRC   TMP1,1                ; Carry set ?
        JOC   LDFNT4                ; Yes, go insert a >00
        MOVB  @GRMRD,TMP0
*---------------------------------------------------------------
*   Make font fat
*---------------------------------------------------------------
        COC   @WBIT0,CONFIG         ; Fat flag set ?
        JNE   LDFNT3                ; No, so skip
        MOVB  TMP0,TMP6
```

```
        SRL   TMP6,1
        SOC   TMP6,TMP0
*--------------------------------------------------------------
*   Dump byte to VDP and do housekeeping
*--------------------------------------------------------------
LDFNT3  MOVB  TMP0,@VDPW            ; Dump byte to VRAM
        DEC   TMP2
        JNE   LDFNT2
        INCT  TMP4                  ; R11=R11+2
        LI    R15,VDPW              ; Set VDP write address
        ANDI  CONFIG,>7FFF          ; CONFIG register bit 0=0
        B     *TMP4                 ; Exit
LDFNT4  MOVB  @BD0,@VDPW            ; Insert byte >00 into VRAM
        JMP   LDFNT2
TMP006  DATA  >004C,64*8,>0000      ; Pointer to TI title screen font
        DATA  >004E,64*7,>0101      ; Pointer to upper case font
        DATA  >004E,96*7,>0101      ; Pointer to upper & lower case font
        DATA  >0050,32*7,>0101      ; Pointer to lower case font
*...

* PUTSTR (DATA P0) / XUTSTR ...
*****************************************************************
* Put length-byte prefixed string at current YX
*****************************************************************
*  BL   @PUTSTR
*  DATA P0
*
*  P0 = Pointer to string
*--------------------------------------------------------------
*  REMARKS
*  First byte of string must contain length
********@*****@**********************@**************************
PUTSTR  MOV   *R11+,TMP1
        MOVB  *TMP1+,TMP2           ; Get length byte
XUTSTR  MOV   R11,TMP3
        BL    @YX2PNT               ; Get VDP destination address
        MOV   TMP3,R11
        SRL   TMP2,8                ; Right justify length byte
        B     @XPYM2V               ; Display string
*...

* PUTAT  (DATA P0) ...
*****************************************************************
* Put length-byte prefixed string at YX
*****************************************************************
*  BL   @PUTAT
*  DATA P0,P1
*
*  P0 = YX position
*  P1 = Pointer to string
*--------------------------------------------------------------
*  REMARKS
*  First byte of string must contain length
********@*****@**********************@**************************
PUTAT   MOV   *R11+,@WYX            ; Set YX position
        B     @PUTSTR
*...
```

```
* HCHAR  (DATA P0,P1,...,EOL) ..:
*****************************************************************
* Repeat characters horizontally at YX
*****************************************************************
*  BL    @HCHAR
*  DATA  P0,P1
*  ...
*  DATA  EOL                       ; End-of-list
*---------------------------------------------------------------
*  P0HB = Y position
*  P0LB = X position
*  P1HB = Byte to write
*  P1LB = Number of times to repeat
********@*****@**********************@**************************
HCHAR   MOV   *R11+,@WYX            ; Set YX position
        MOVB  *R11+,TMP1
        SRL   TMP1,8                ; Byte to write
        MOVB  *R11+,TMP2
        SRL   TMP2,8                ; Repeat count
        MOV   R11,TMP3
        BL    @YX2PNT               ;Get VDP address into TMP0
*---------------------------------------------------------------
*    Draw line
*---------------------------------------------------------------
        LI    R11,HCHAR1
        B     @XFILV                ; Draw
*---------------------------------------------------------------
*    Do housekeeping
*---------------------------------------------------------------
HCHAR1  C     *TMP3,@WHFFFF         ; End-Of-List marker found ?
        JEQ   HCHAR2                ; Yes, exit
        MOV   TMP3,R11
        JMP   HCHAR                 ; Next one
HCHAR2  INCT  TMP3
        B     *TMP3                 ; Exit
*...

* VCHAR  (DATA P0,P1,...,EOL) ..:
*****************************************************************
* Repeat characters vertically at YX
*****************************************************************
*  BL    @VCHAR
*  DATA  P0,P1
*  ...
*  DATA  EOL                       ; End-of-list
*---------------------------------------------------------------
*  P0HB = Y position
*  P0LB = X position
*  P1HB = Byte to write
*  P1LB = Number of times to repeat
********@*****@**********************@**************************
VCHAR   MOV   *R11+,@WYX            ; Set YX position
        MOV   R11,TMP3              ; Save R11 in TMP3
VCHAR1  BL    @GTCLMN               ; Get cols per row into TMP0
        MOV   TMP0,TMP4             ; Copy to TMP4
        BL    @YX2PNT               ; Get VDP address into TMP0
```

```
        MOVB  *TMP3+,TMP1           ; Byte to write
        MOVB  *TMP3+,TMP2
        SRL   TMP2,8                ; Repeat count
*--------------------------------------------------------------
*    Setup VDP write address
*--------------------------------------------------------------
VCHAR2  ORI   TMP0,>4000
VCHAR3  SWPB  TMP0
        MOVB  TMP0,@VDPA
        SWPB  TMP0
        MOVB  TMP0,@VDPA
*--------------------------------------------------------------
*    Dump tile to VDP and do housekeeping
*--------------------------------------------------------------
        MOVB  TMP1,*R15             ; Dump tile to VDP
        A     TMP4,TMP0             ; Next row
        DEC   TMP2
        JNE   VCHAR3
        C     *TMP3,@WHFFFF         ; End-Of-List marker found ?
        JEQ   VCHAR4                ; Yes, exit
        MOV   *TMP3+,@WYX           ; Save YX position
        JMP   VCHAR1                ; Next one
VCHAR4  INCT  TMP3
        B     *TMP3                 ; Exit
*...

* FILBOX (DATA P0,P1,P2,...,EOL) ...
****************************************************************
* FILBOX - Fill box with character
****************************************************************
*  BL   @FILBOX
*  DATA P0,P1,P2
*  ...
*  DATA EOL
*--------------------------------------------------------------
*  P0HB = Upper left corner Y
*  P0LB = Upper left corner X
*  P1HB = Width
*  P1LB = Height
*  P2HB = >00
*  P2LB = Character to fill
********@*****@**********************@**************************
FILBOX  MOV   *R11+,@WYX             ; Upper left corner
        MOVB  *R11+,TMP2            ; Width in TMP2
        MOVB  *R11+,TMP3            ; Height in TMP3
        MOV   *R11+,TMP1            ; Byte to fill
        MOV   R11,TMP4              ; Save R11
        SRL   TMP2,8                ; Right-align width
        SRL   TMP3,8                ; Right-align height
*--------------------------------------------------------------
*  Do single row
*--------------------------------------------------------------
FILBO1  BL    @YX2PNT               ; Get VDP address into TMP0
        LI    R11,FILBO2            ; New return address
        B     @XFILV                ; Fill VRAM with byte
*--------------------------------------------------------------
*  Recover width & character
```

```
*--------------------------------------------------------------
FILBO2  MOV   TMP4,TMP0
        AI    TMP0,-4               ; R11 - 4
        MOV   *TMP0+,TMP2           ; Get Width/Height
        SRL   TMP2,8                ; Right align
        MOV   *TMP0,TMP1            ; Get character to fill
*--------------------------------------------------------------
*  Housekeeping
*--------------------------------------------------------------
        A     @WH100,@BY            ; Y=Y+1
        DEC   TMP3
        JGT   FILBO1                ; Process next row
        C     *TMP4,@WHFFFF         ; End-Of-List marker found ?
        JEQ   FILBO3                ; Yes, exit
        MOV   TMP4,R11
        JMP   FILBOX                ; Next one
FILBO3  INCT  TMP4
        B     *TMP4                 ; Exit
*...

* PUTBOX (DATA P0,P1,P2,...,EOL) ..:
****************************************************************
* PUTBOX - Put tiles in box
****************************************************************
*  BL   @PUTBOX
*  DATA P0,P1,P2
*  ...
*  DATA EOL
*--------------------------------------------------------------
*  P0HB = Upper left corner Y
*  P0LB = Upper left corner X
*  P1HB = Width
*  P1LB = Height
*  P2   = Pointer to length-byte prefixed string
********@*****@**********************@**************************
PUTBOX  MOV   *R11+,@WYX             ; Upper left corner
        MOVB  *R11+,TMP2            ; Width in TMP2
        MOVB  *R11+,TMP3            ; Height in TMP3
        MOV   *R11+,TMP1            ; Pointer to string
        MOV   R11,TMP4              ; Save R11
        AB    @BX,TMP2
        AB    @BY,TMP3
        ANDI  CONFIG,>7FFF          ; Reset bit 0 (state flag)
*--------------------------------------------------------------
*  Setup VDP write address
*--------------------------------------------------------------
PUTBO1  BL    @YX2PNT               ; Get VDP address into TMP0
        BL    @VDWA                 ; Set VDP write address
*--------------------------------------------------------------
*  Prepare string for processing
*--------------------------------------------------------------
        COC   @WBIT0,CONFIG         ; state flag 0 set ?
        JEQ   PUTBO2                ; Yes, so skip
        MOVB  *TMP1+,TMP0           ; Get length byte
        SRL   TMP0,8                ; Right justify
        JMP   PUTBO3
PUTBO2  MOV   TMP2,TMP0             ; Recover counter
```

```
        ANDI  TMP0,>00FF            ; Counter is in LB
        ANDI  TMP2,>FF00            ; Only keep with in TMP2
*--------------------------------------------------------------
*  Write line of tiles in box
*--------------------------------------------------------------
PUTBO3  MOVB  *TMP1+,*R15           ; Write to VDP
        DEC   TMP0
        JEQ   PUTBO6                ; End of string, reset to begin
*--------------------------------------------------------------
*  Adjust cursor
*--------------------------------------------------------------
PUTBO4  INC   @WYX                  ; X=X+1
        CB    @BX,TMP2              ; Right boundary reached ?
        JLT   PUTBO3                ; Not yet, continue
        A     @WH100,@BY            ; Y=Y+1
        CB    @BY,TMP3              ; Bottom boundary reached ?
        JEQ   PUTBO7                ; Yes, exit
*--------------------------------------------------------------
*  Recover starting column
*--------------------------------------------------------------
        MOVB  @TMP0LB,@TMP2LB       ; Save counter
        MOV   TMP4,TMP0
        AI    TMP0,-5
        MOVB  *TMP0,@BX             ; Recover X
        ORI   CONFIG,>8000          ; CONFIG register bit 0=1
        JMP   PUTBO1                ; Draw next line
*--------------------------------------------------------------
*  Recover string pointer
*--------------------------------------------------------------
PUTBO6  MOV   TMP4,TMP0
        AI    TMP0,-2               ; R11 - 2
        MOV   *TMP0,TMP1            ; Get string pointer
        MOVB  *TMP1+,TMP0           ; Get length byte
        SRL   TMP0,8                ; Right justify
        JMP   PUTBO4                ; Adjust cursor
PUTBO7  C     *TMP4,@WHFFFF         ; End-Of-List marker found ?
        JEQ   PUTBO8                ; Yes, exit
        MOV   TMP4,R11
        JMP   PUTBOX                ; Next one
PUTBO8  ANDI  CONFIG,>7FFF          ; CONFIG register bit 0=0
        B     *TMP4                 ; Exit
*...


* MKNUM  (DATA P0,P1,P2) ..:
****************************************************************
* MKNUM - Convert unsigned number to string
****************************************************************
*  BL   @MKNUM
*  DATA P0,P1,P2
*
*  P0   = Pointer to 16 bit unsigned number
*  P1   = Pointer to 5 byte string buffer
*  P2HB = Offset for ASCII digit
*  P2LB = Character for replacing leading 0's
*
*  (CONFIG:0 = 1) = Display number at cursor YX
********@*****@*********************@**************************
```

```
MKNUM   LI    TMP3,5                ; Digit counter
        MOV   *R11+,TMP1            ; \ Get 16 bit unsigned number
        MOV   *TMP1,TMP1            ; /
        MOV   *R11+,TMP4            ; Pointer to string buffer
        AI    TMP4,4                ; Get end of buffer
        LI    TMP2,10               ; Divide by 10 to isolate last digit
*--------------------------------------------------------------
*  Do string conversion
*--------------------------------------------------------------
MKNUM1  CLR   TMP0                  ; Clear the high word of the dividend
        DIV   TMP2,TMP0             ; (TMP0:TMP1) / 10 (TMP2)
        SWPB  TMP1                  ; Move to high-byte for writing to buffer
        AB    *R11,TMP1             ; Add offset for ASCII digit
        MOVB  TMP1,*TMP4            ; Write remainder to string buffer
        MOV   TMP0,TMP1             ; Move integer result into R4 for the next digit
        DEC   TMP4                  ; Adjust string pointer for next digit
        DEC   TMP3                  ; Decrease counter
        JNE   MKNUM1                ; Do next digit
*--------------------------------------------------------------
*  Replace leading 0's with fill character
*--------------------------------------------------------------
        LI    TMP3,4                ; Check first 4 digits
        INC   TMP4                  ; Too far, back to buffer start
        MOV   *R11,TMP0
        SLA   TMP0,8                ; Only keep fill character in HB
MKNUM2  CB    *TMP4,*R11            ; Digit = 0 ?
        JEQ   MKNUM4                ; Yes, replace with fill character
MKNUM3  INCT  R11
        COC   @WBIT0,CONFIG         ; Check if 'display' bit is set
        JEQ   MKNUM5                ; Yes, so show at current YX position
        B     *R11                  ; Exit
MKNUM4  MOVB  TMP0,*TMP4+           ; Replace leading 0 with fill character
        DEC   TMP3                  ; 4th digit processed ?
        JEQ   MKNUM3                ; Yes, exit
        JMP   MKNUM2                ; No, next one
*--------------------------------------------------------------
*  Display integer on screen at current YX position
*--------------------------------------------------------------
MKNUM5  ANDI  CONFIG,>7FFF          ; Reset bit 0
        MOV   R11,TMP0
        AI    TMP0,-4
        MOV   *TMP0,TMP1            ; Get buffer address
        LI    TMP2,>0500            ; String length = 5
        B     @XUTSTR               ; Display string

* ...

* PUTNUM (DATA P0,P1,P2,P3) ...
****************************************************************
* PUTNUM - Put unsigned number on screen
****************************************************************
*  BL   @PUTNUM
*  DATA P0,P1,P2,P3
*--------------------------------------------------------------
*  P0   = YX position
*  P1   = Pointer to 16 bit unsigned number
*  P2   = Pointer to 5 byte string buffer
```

```
*  P3HB = Offset for ASCII digit
*  P3LB = Character for replacing leading 0's
********@*****@*********************@**************************
PUTNUM  MOV   *R11+,@WYX            ; Set cursor
        ORI   CONFIG,>8000          ; CONFIG register bit 0=1
        JMP   MKNUM                 ; Convert number and display
*...


*////////////////////////////////////////////////////////////////
*                              SOUND
*////////////////////////////////////////////////////////////////

* MUTE   () / MUTE2  ...
*****************************************************************
* MUTE - Mute all sound generators
*****************************************************************
*  BL  @MUTE
*  Mute sound generators and clear sound pointer
*
*  BL  @MUTE2
*  Mute sound generators without clearing sound pointer
********@*****@*********************@**************************
MUTE    CLR   @WSDLST               ; Clear sound pointer
MUTE2   SZC   @WBIT13,CONFIG        ; Turn off/pause sound player
        LI    TMP0,MUTTAB
        LI    TMP1,SOUND            ; Sound generator port >8400
        MOVB  *TMP0+,*TMP1          ; Generator 0
        MOVB  *TMP0+,*TMP1          ; Generator 1
        MOVB  *TMP0+,*TMP1          ; Generator 2
        MOVB  *TMP0,*TMP1           ; Generator 3
        B     *R11
MUTTAB  BYTE  >9F,>BF,>DF,>FF       ; Table for muting all generators
* ...

* SDPREP (DATA P0,P1) ...
*****************************************************************
* SDPREP - Prepare for playing sound
*****************************************************************
*  BL   @SDPREP
*  DATA P0,P1
*
*  P0 = Address where tune is stored
*  P1 = Option flags for sound player
*---------------------------------------------------------------
*  REMARKS
*  Use the below equates for P1:
*
*  SDOPT1 => Tune is in CPU memory + loop
*  SDOPT2 => Tune is in CPU memory
*  SDOPT3 => Tune is in VRAM + loop
*  SDOPT4 => Tune is in VRAM
********@*****@*********************@**************************
SDPREP  MOV   *R11,@WSDLST          ; Set tune address
        MOV   *R11+,@WSDTMP         ; Set tune address in temp
        ANDI  R12,>FFF8             ; Clear bits 13-14-15
        SOC   *R11+,CONFIG          ; Set options
```

```
        MOVB  @BD1,@R13LB           ; Set initial duration
        B     *R11
*...

* SDPLAY () ...:
*****************************************************************
* SDPLAY - Sound player for tune in VRAM or CPU memory
*****************************************************************
*  BL  @SDPLAY
*---------------------------------------------------------------
*  REMARKS
*  Set config register bit13=0 to pause player.
*  Set config register bit14=1 to repeat (or play next tune).
********@*****@**********************@**************************
SDPLAY  COC   @WBIT13,CONFIG        ; Play tune ?
        JEQ   SDPLA1                ; Yes, play
        B     *R11
*---------------------------------------------------------------
* Initialisation
*---------------------------------------------------------------
SDPLA1  DEC   R13                   ; duration = duration - 1
        CB    @R13LB,@BD0           ; R13LB == 0 ?
        JEQ   SDPLA3                ; Play next note
SDPLA2  B     *R11                  ; Note still busy, exit
SDPLA3  COC   @WBIT15,CONFIG        ; Play tune from CPU memory ?
        JEQ   MMPLAY
*---------------------------------------------------------------
* Play tune from VDP memory
*---------------------------------------------------------------
VDPLAY  MOV   @WSDTMP,TMP0          ; Get tune address
        SWPB  TMP0
        MOVB  TMP0,@VDPA
        SWPB  TMP0
        MOVB  TMP0,@VDPA
        CLR   TMP0
        MOVB  @VDPR,TMP0            ; length = 0 (end of tune) ?
        JEQ   SDEXIT                ; Yes. exit
VDPLA1  SRL   TMP0,8                ; Right justify length byte
        A     TMP0,@WSDTMP          ; Adjust for next table entry
VDPLA2  MOVB  @VDPR,@>8400          ; Feed byte to sound generator
        DEC   TMP0
        JNE   VDPLA2
        MOVB  @VDPR,@R13LB          ; Set duration counter
VDPLA3  INCT  @WSDTMP               ; Adjust for next table entry, honour byte (1) + (n+1)
        B     *R11
*---------------------------------------------------------------
* Play tune from CPU memory
*---------------------------------------------------------------
MMPLAY  MOV   @WSDTMP,TMP0
        MOVB  *TMP0+,TMP1           ; length = 0 (end of tune) ?
        JEQ   SDEXIT                ; Yes, exit
MMPLA1  SRL   TMP1,8                ; Right justify length byte
        A     TMP1,@WSDTMP          ; Adjust for next table entry
MMPLA2  MOVB  *TMP0+,@>8400         ; Feed byte to sound generator
        DEC   TMP1
        JNE   MMPLA2
        MOVB  *TMP0,@R13LB          ; Set duration counter
```

```
        INCT  @WSDTMP               ; Adjust for next table entry, honour byte (1) + (n+1)
        B     *R11
*--------------------------------------------------------------
* Exit. Check if tune must be looped
*--------------------------------------------------------------
SDEXIT  COC   @WBIT14,CONFIG        ; Loop flag set ?
        JNE   SDEXI2                ; No, exit
        MOV   @WSDLST,@WSDTMP
        MOVB  @BD1,@R13LB           ; Set initial duration
SDEXI1  B     *R11                  ; Exit
SDEXI2  ANDI  CONFIG,>FFF8          ; Reset music player
        B     *R11                  ; Exit
* ...


*////////////////////////////////////////////////////////////////
*                               SPEECH
*////////////////////////////////////////////////////////////////

* SPSTAT () ...
*****************************************************************
* SPSTAT - Read status register byte from speech synthesizer
*****************************************************************
*  LI  TMP2,@>....
*  B   @SPSTAT
*--------------------------------------------------------------
* REMARKS
* Destroys R11 !
*
* Register usage
* TMP0HB = Status byte read from speech synth
* TMP1   = Temporary use  (scratchpad machine code)
* TMP2   = Return address for this subroutine
* R11    = Return address (scratchpad machine code)
********@*****@**********************@**************************
SPSTAT  LI    TMP0,SPCHRD           ; (R4) = >9000
        MOV   @SPCODE,@MCSPRD       ; \
        MOV   @SPCODE+2,@MCSPRD+2   ; / Load speech read code
        LI    R11,SPSTA1            ; Return to SPSTA1
        B     @MCSPRD               ; Run scratchpad code
SPSTA1  MOV   @MCCODE,@MCSPRD       ; \
        MOV   @MCCODE+2,@MCSPRD+2   ; / Restore tight loop code
        B     *TMP2                 ; Exit
*...

* SPCONN () ...
*****************************************************************
* SPCONN - Check if speech synthesizer connected
*****************************************************************
* BL  @SPCONN
*--------------------------------------------------------------
* OUTPUT
* TMP0HB = Byte read from speech synth
*--------------------------------------------------------------
* REMARKS
* See Editor/Assembler manual, section 22.1.6 page 354.
* Calls SPSTAT.
```

```
*
* Register usage
* TMP0HB = Byte read from speech synth
* TMP3   = Copy of R11
* R12    = CONFIG register
********@*****@*********************@**************************
SPCONN  MOV   R11,TMP3              ; Save R11
        MOVB  @BH10,@SPCHWT         ; Load >10
        LI    TMP2,SPCON1
        B     @SPSTAT               ; Read status byte
SPCON1  B     *TMP3                 ; Exit
*...

* SPPREP (DATA P0,P1) ..:
*****************************************************************
* SPPREP - Prepare for playing speech
*****************************************************************
*  BL   @SPPREP
*  DATA P0,P1
*
*  P0 = Address of LPC data for external voice
*       or index of word to speak if resident voice
*  P1 = Option flags for speech player
*---------------------------------------------------------------
*  REMARKS
*  Use the below equates for P1:
*
*  SPOPT1  => External voice
*  SPOPT2  => Resident voice
********@*****@*********************@**************************
SPPREP  MOV   *R11+,@WSPEAK         ; Set speech address
        ANDI  R12,>E3FF             ; Clear bits 3-4-5
        SOC   *R11+,CONFIG          ; Set options
        B     *R11
*...

* SPPLAY () ..:
*****************************************************************
* SPPLAY - Speech player
*****************************************************************
* BL  @SPPLAY
*---------------------------------------------------------------
* Register usage
* TMP3   = Copy of R11
* R12    = CONFIG register
********@*****@*********************@**************************
SPPLAY  CZC   @WBIT3,CONFIG         ; Player off ?
        JEQ   SPPLAZ                ; Yes, exit
SPPLA1  MOV   R11,TMP3              ; Save R11
        COC   @TMP010,CONFIG        ; Is on/busy/external ?
        JEQ   SPKEX3                ; Check FIFO buffer level
        COC   @WBIT5,CONFIG         ; Start speak external ?
        JEQ   SPKEXT                ; Yes, do it
*---------------------------------------------------------------
* Speak resident: ****
*---------------------------------------------------------------
*       NOT   YET
```

```
*--------------------------------------------------------------
* Speak external: Push LPC data to speech synthesizer
*--------------------------------------------------------------
SPKEXT  MOV   @WSPEAK,TMP0
        MOVB  *TMP0+,@SPCHWT        ; Send byte to speech synth
        JMP   $+2                   ; Delay
        LI    TMP2,16
SPKEX1  MOVB  *TMP0+,@SPCHWT        ; Send byte to speech synth
        DEC   TMP2
        JNE   SPKEX1
        ORI   CONFIG,>1800          ; bit 4=1 (busy) & bit 5=1 (external)
        MOV   TMP0,@WSPEAK          ; Update LPC pointer
        JMP   SPPLAZ                ; Exit
*--------------------------------------------------------------
* Speak external: Check synth FIFO buffer level
*--------------------------------------------------------------
SPKEX3  LI    TMP2,SPKEX4           ; Set return address for SPSTAT
        B     @SPSTAT               ; Get speech FIFO buffer status
SPKEX4  COC   @WH4000,TMP0          ; FIFO BL (buffer low) bit set ?
        JEQ   SPKEX5                ; Yes, refill
        JMP   SPPLAZ                ; No, exit
*--------------------------------------------------------------
* Speak external: Refill synth with LPC data if FIFO buffer low
*--------------------------------------------------------------
SPKEX5  MOV   @WSPEAK,TMP0
        LI    TMP2,8                ; Bytes to send to speech synth
SPKEX6  MOVB  *TMP0+,TMP1
        MOVB  TMP1,@SPCHWT          ; Send byte to speech synth
        CI    TMP1,SPKOFF           ; Speak off marker found ?
        JEQ   SPKEX8
        DEC   TMP2
        JNE   SPKEX6                ; Send next byte
        MOV   TMP0,@WSPEAK          ; Update LPC pointer
SPKEX7  JMP   SPPLAZ                ; Exit
*--------------------------------------------------------------
* Speak external: Done with speaking
*--------------------------------------------------------------
SPKEX8  SZC   @TMP010,CONFIG        ; bit 3,4,5=0
        CLR   @WSPEAK               ; Reset pointer
SPPLAZ  B     *TMP3                 ; Exit
TMP010  DATA  >1C00                 ; Binary 0001110000000000
*...



*//////////////////////////////////////////////////////////////
*                            KEYBOARD
*//////////////////////////////////////////////////////////////

* VIRTKB () ...:
****************************************************************
* VIRTKB - Read virtual keyboard and joysticks
****************************************************************
*  BL @VIRTKB
*--------------------------------------------------------------
*  COLUMN     0     1  2  3  4  5    6   7
*        +---------------------------------+------+
```

```
*  ROW 7  |   =     .  ,  M  N  /   JS1 JS2 | Fire |
*  ROW 6  | SPACE   L  K  J  H  :;  JS1 JS2 | Left |
*  ROW 5  | ENTER   O  I  U  Y  P   JS1 JS2 | Right|
*  ROW 4  |         9  8  7  6  0   JS1 JS2 | Down |
*  ROW 3  | FCTN    2  3  4  5  1   JS1 JS2 | Up   |
*  ROW 2  | SHIFT   S  D  F  G  A           +------|
*  ROW 1  | CTRL    W  E  R  T  Q                  |
*  ROW 0  |         X  C  V  B  Z                  |
*         +----------------------------------------+
*  See MG smart programmer 1986
*  September/Page 15 and November/Page 6
*  Also see virtual keyboard status for bits to check
*---------------------------------------------------------------
*  Register usage
*  TMP0     Keyboard matrix column to process
*  TMP1MSB  Keyboard matrix 8 bits of 1 column
*  TMP2     Virtual keyboard flags
*  TMP3     Address of entry in mapping table
*  TMP4     Copy of R12 (CONFIG REGISTER)
*  R12      CRU communication
********@*****@*********************@**************************
VIRTKB  SZC   @WBIT11,CONFIG        ; Reset ANY key
        MOV   CONFIG,TMP4           ; Save R12 (CONFIG REGISTER)
        CLR   TMP0                  ; Value in MSB! Start with column 0
        CLR   TMP2                  ; Erase virtual keyboard flags
        LI    TMP3,KBMAP0           ; Start with column 0
*---------------------------------------------------------------
* Check alpha lock key
*-------@-----@---------------------@--------------------------
        CLR   R12
        SBZ   >0015                 ; Set P5
        TB    7
        JEQ   VIRTK1
        LI    TMP2,KALPHA           ; Alpha lock key down
*---------------------------------------------------------------
* Scan keyboard matrix
*-------@-----@---------------------@--------------------------
VIRTK1  SBO   >0015                 ; Reset P5
        LI    R12,>0024             ; Scan full 8x8 keyboard matrix. R12 is used by LDCR
        LDCR  TMP0,3                ; Set keyboard column with a value from 0-7 (3=3 bits)
        LI    R12,>0006             ; Load CRU base for row. R12 required by STCR
        SETO  TMP1                  ; >FFFF
        STCR  TMP1,8                ; Bring 8 row bits into MSB of TMP1
        INV   TMP1
        JEQ   VIRTK2                ; >0000 ?
        SOC   @WBIT11,TMP4          ; Set ANY key in copy of CONFIG register
*---------------------------------------------------------------
* Process column
*-------@-----@---------------------@--------------------------
VIRTK2  COC   *TMP3+,TMP1           ; Check bit mask
        JNE   VIRTK3
        SOC   *TMP3,TMP2            ; Set virtual keyboard flags
VIRTK3  INCT  TMP3
        C     *TMP3,@KBEOC          ; End-of-column ?
        JNE   VIRTK2                ; No, next entry
        INCT  TMP3
*---------------------------------------------------------------
```

```
* Prepare for next column
*-------@-----@----------------------@--------------------------
VIRTK4  CI    TMP0,>0700             ; Column 7 processed ?
        JEQ   VIRTK6                 ; Yes, exit
        CI    TMP0,>0200             ; Column 2 processed ?
        JEQ   VIRTK5                 ; Yes, skip
        AI    TMP0,>0100
        JMP   VIRTK1
VIRTK5  LI    TMP0,>0500             ; Skip columns 3-4
        JMP   VIRTK1
*--------------------------------------------------------------
* Exit
*-------@-----@----------------------@--------------------------
VIRTK6  MOV   TMP4,CONFIG            ; Restore CONFIG register
        MOV   TMP2,@WVRTKB           ; Save virtual keyboard flags
        JNE   VIRTK7
        B     *R11                   ; Exit
VIRTK7  CI    TMP2,>FFFF             ; FCTN-QUIT pressed ?
        JNE   VIRTK8                 ; No
        SETO  R1                     ; Set exit flag
        B     @RUNLI1                ; Yes, reset computer
VIRTK8  CI    TMP2,KALPHA            ; Only alpha-lock pressed ?
        JNE   VIRTK9
        SZC   @WBIT11,CONFIG         ; Yes, so reset ANY key
VIRTK9  B     *R11                   ; Exit
*--------------------------------------------------------------
* Mapping table
*-------@-----@----------------------@--------------------------
*                                    ; Bit 01234567
KBMAP0  DATA  >1100,>FFFF            ; >11 00010001  FCTN QUIT
        DATA  >0200,K1FIRE           ; >02 00000010  spacebar
        DATA  >0400,KENTER           ; >04 00000100  enter
KBEOC   DATA  >FFFF
KBMAP1  DATA  >0800,KBACK            ; >08 00001000  FCTN BACK
        DATA  >2000,K1LF             ; >20 00100000  S (arrow left)
        DATA  >8000,K1DN             ; >80 10000000  X (arrow down)
        DATA  >FFFF
KBMAP2  DATA  >0800,KREDO            ; >08 00001000  FCTN REDO
        DATA  >2000,K1RG             ; >20 00100000  D (arrow right)
        DATA  >4000,K1UP             ; >80 01000000  E (arrow up)
        DATA  >FFFF
KBCOL5  DATA  >0800,KPAUSE           ; >08 00001000  P (pause)
        DATA  >8000,K1FIRE           ; >80 01000000  Q (fire)
        DATA  >FFFF
KBMAP6  DATA  >0100,K1FIRE           ; >01 00000001  joystick 1 FIRE
        DATA  >0200,K1LF             ; >02 00000010  joystick 1 left
        DATA  >0400,K1RG             ; >04 00000100  joystick 1 right
        DATA  >0800,K1DN             ; >08 00001000  joystick 1 down
        DATA  >1000,K1UP             ; >10 00010000  joystick 1 up
        DATA  >FFFF
KBMAP7  DATA  >0100,K2FIRE           ; >01 00000001  joystick 2 FIRE
        DATA  >0200,K2LF             ; >02 00000010  joystick 2 left
        DATA  >0400,K2RG             ; >04 00000100  joystick 2 right
        DATA  >0800,K2DN             ; >08 00001000  joystick 2 down
        DATA  >1000,K2UP             ; >10 00010000  joystick 2 up
        DATA  >FFFF
*...
```

```
*//////////////////////////////////////////////////////////////
*                               TIMERS
*//////////////////////////////////////////////////////////////

* TMGR   () ...:
*****************************************************************
* TMGR - X - Start Timer/Thread scheduler
*****************************************************************
*  B @TMGR
*---------------------------------------------------------------
*  REMARKS
*  Timer/Thread scheduler. Normally called from MAIN.
*  Don't forget to set BTIHI to highest slot in use.
*
*  Register usage in TMGR8 - TMGR11
*  TMP0  = Pointer to timer table
*  R10LB = Use as slot counter
*  TMP2  = 2nd word of slot data
*  TMP3  = Address of routine to call
********@*****@**********************@**************************
TMGR    LIMI  0                     ; No interrupt processing
*---------------------------------------------------------------
* Read VDP status register
*---------------------------------------------------------------
TMGR1   MOVB  @VDPS,TMP0            ; Get VDP status register
        MOVB  TMP0,R13              ; Save copy of VDP status register in R13
        COC   @WBIT0,TMP0           ; Interupt flag set ?
        JEQ   TMGR4                 ; Yes, process slots 0..n
*---------------------------------------------------------------
* Run speech player
*---------------------------------------------------------------
        COC   @WBIT3,CONFIG         ; Speech player on ?
        JNE   TMGR2
        BL    @SPPLA1               ; Run speech player
*---------------------------------------------------------------
* Run kernel thread
*---------------------------------------------------------------
TMGR2   COC   @WBIT8,CONFIG         ; Kernel thread blocked ?
        JEQ   TMGR3                 ; Yes, skip to user hook
        COC   @WBIT9,CONFIG         ; Kernel thread enabled ?
        JNE   TMGR3                 ; No, skip to user hook
        B     @KERNEL               ; Run kernel thread
*---------------------------------------------------------------
* Run user hook
*---------------------------------------------------------------
TMGR3   COC   @WBIT6,CONFIG         ; User hook blocked ?
        JEQ   TMGR1
        COC   @WBIT7,CONFIG         ; User hook enabled ?
        JNE   TMGR1
        MOV   @WTIUSR,TMP0
        B     *TMP0                 ; Run user hook
*---------------------------------------------------------------
* Do some internal housekeeping
*---------------------------------------------------------------
TMGR4   SZC   @TMDAT,CONFIG         ; Unblock kernel thread and user hook
```

```
        MOV   R10,TMP0
        ANDI  TMP0,>00FF            ; Clear HI byte
        COC   @WBIT2,CONFIG         ; PAL flag set ?
        JEQ   TMGR5
        CI    TMP0,60               ; 1 second reached ?
        JMP   TMGR6
TMGR5   CI    TMP0,50
TMGR6   JLT   TMGR7                 ; No, continue
        JMP   TMGR8
TMGR7   INC   R10                   ; Increase tick counter
*--------------------------------------------------------------
* Loop over slots
*--------------------------------------------------------------
TMGR8   MOV   @WTITAB,TMP0          ; Pointer to timer table
        ANDI  R10,>FF00             ; Use R10LB as slot counter. Reset.
TMGR9   MOV   *TMP0,TMP3            ; Is slot empty ?
        JEQ   TMGR11                ; Yes, get next slot
*--------------------------------------------------------------
*  Check if slot should be executed
*--------------------------------------------------------------
        INCT  TMP0                  ; Second word of slot data
        INC   *TMP0                 ; Update tick count in slot
        MOV   *TMP0,TMP2            ; Get second word of slot data
        CB    @TMP2HB,@TMP2LB       ; Slot target count = Slot internal counter ?
        JNE   TMGR10                ; No, get next slot
        ANDI  TMP2,>FF00            ; Clear internal counter
        MOV   TMP2,*TMP0            ; Update timer table
*--------------------------------------------------------------
*  Run slot, we only need TMP0 to survive
*--------------------------------------------------------------
        MOV   TMP0,@WTITMP          ; Save TMP0
        BL    *TMP3                 ; Call routine in slot
SLOTOK  MOV   @WTITMP,TMP0          ; Restore TMP0
*--------------------------------------------------------------
*  Prepare for next slot
*--------------------------------------------------------------
TMGR10  INC   R10                   ; Next slot
        CB    @R10LB,@BTIHI         ; Last slot done ?
        JGT   TMGR12                ; yes, Wait for next VDP interrupt
        INCT  TMP0                  ; Offset for next slot
        JMP   TMGR9                 ; Process next slot
TMGR11  INCT  TMP0                  ; Skip 2nd word of slot data
        JMP   TMGR10                ; Process next slot
TMGR12  ANDI  R10,>FF00             ; Use R10LB as tick counter. Reset.
        JMP   TMGR1
TMDAT   DATA  >0280                 ; Bit 8 (kernel thread) and bit 6 (user hook)
*...

* MKSLOT (DATA P0,P1,...) ..:
****************************************************************
* MKSLOT - Allocate timer slot(s)
****************************************************************
*  BL    @MKSLOT
*  BYTE  P0HB,P0LB
*  DATA  P1
*  ....
*  DATA  EOL                        ; End-of-list
```

```
*--------------------------------------------------------------
*  P0 = Slot number, target count
*  P1 = Subroutine to call via BL @xxxx if slot is fired
********@*****@*********************@**************************
MKSLOT  MOV   *R11+,TMP0
        MOV   *R11+,TMP1
*--------------------------------------------------------------
*  Calculate address of slot
*--------------------------------------------------------------
        MOV   TMP0,TMP2
        SRL   TMP2,6                ; Right align & TMP2 = TMP2 * 4
        A     @WTITAB,TMP2          ; Add table base
*--------------------------------------------------------------
*  Add slot to table
*--------------------------------------------------------------
        MOV   TMP1,*TMP2+           ; Store address of subroutine
        SLA   TMP0,8                ; Get rid of slot number
        MOV   TMP0,*TMP2            ; Store target count and reset tick count
*--------------------------------------------------------------
*  Check for end of list
*--------------------------------------------------------------
        C     *R11,@WHFFFF          ; End of list ?
        JEQ   MKSLO1                ; Yes, exit
        JMP   MKSLOT                ; Process next entry
*--------------------------------------------------------------
*  Exit
*--------------------------------------------------------------
MKSLO1  INCT  R11
        B     *R11                  ; Exit
*...

* CLSLOT (DATA P0) / XLSLOT ...:
**************************************************************
* CLSLOT - Clear single timer slot
**************************************************************
*  BL    @CLSLOT
*  DATA  P0
*--------------------------------------------------------------
*  P0 = Slot number
********@*****@*********************@**************************
CLSLOT  MOV   *R11+,TMP0
XLSLOT  SLA   TMP0,2                ; TMP0 = TMP0*4
        A     @WTITAB,TMP0          ; Add table base
        CLR   *TMP0+                ; Clear 1st word of slot
        CLR   *TMP0                 ; Clear 2nd word of slot
        B     *R11                  ; Exit
*...

* KERNEL () ...:
**************************************************************
* KERNEL - The kernel thread
*--------------------------------------------------------------
*  REMARKS
*  You shouldn't call the kernel thread manually.
*  Instead control it via the CONFIG register.
********@*****@*********************@**************************
KERNEL  SOC   @WBIT8,CONFIG         ; Block kernel thread
```

```
        COC   @WBIT13,CONFIG          ; Sound player on ?
        JNE   KERNE1
        BL    @SDPLA1                 ; Run sound player
KERNE1  COC   @WBIT12,CONFIG          ; Keyboard mode real ?
        JNE   KERNE2
*       ; STILL TO DO
        JMP   KERNEZ                  ; Exit
KERNE2  BL    @VIRTKB                 ; Scan virtual keyboard
KERNEZ  B     @TMGR3                  ; Exit

*...

* MKHOOK (DATA P0) ..:
*****************************************************************
* MKHOOK - Allocate user hook
*****************************************************************
*  BL    @MKHOOK
*  DATA  P0
*---------------------------------------------------------------
*  P0 = Address of user hook
*---------------------------------------------------------------
*  REMARKS
*  The user hook gets executed after the kernel thread.
*  The user hook must always exit with "B @HOOKOK"
********@*****@*********************@**************************
MKHOOK  MOV   *R11+,@WTIUSR           ; Set user hook address
        ORI   CONFIG,ENUSR            ; Enable user hook
MKHOO1  B     *R11                    ; Return
HOOKOK  EQU   TMGR1                   ; Exit point for user hook
*...


*////////////////////////////////////////////////////////////////
*                       MISC FUNCTIONS
*////////////////////////////////////////////////////////////////

* POPR.  (...) ..:
*****************************************************************
* POPR. - Pop registers & return to caller
*****************************************************************
*  B  @POPRG.
*---------------------------------------------------------------
*  REMARKS
*  R11 must be at stack bottom
********@*****@*********************@**************************
POPR3   MOV   *STACK+,R3
POPR2   MOV   *STACK+,R2
POPR1   MOV   *STACK+,R1
POPR0   MOV   *STACK+,R0
POPRT   MOV   *STACK+,R11
        B     *R11
*...

* RND    (DATA P0) / RNDX ..:
*****************************************************************
* RND - Generate random number
*****************************************************************
```

```
*  BL   @RND
*  DATA P0
*--------------------------------------------------------------
*  P0 = Highest random number allowed
*
*--------------------------------------------------------------
*  BL   @RNDX
*
*  TMP0 = Highest random number allowed
*--------------------------------------------------------------
*  OUTPUT
*  TMP0 = The generated random number
********@*****@**********************@**************************
RND     MOV   *R11+,TMP0            ; Highest number allowed
RNDX    CLR   TMP1
        MOV   @WSEED,TMP2           ; Get random seed
        JNE   RND1
        INC   TMP2                  ; May not be zero
RND1    SRL   TMP2,1
        JNC   RND2
        XOR   @RNDDAT,TMP2
RND2    MOV   TMP2,@WSEED           ; Store new random seed
        DIV   TMP0,TMP1
        MOV   TMP2,TMP0
        B     *R11                  ; Exit
RNDDAT  DATA  >0B400                ; The magic number
*...


*///////////////////////////////////////////////////////////////
*                    RUNLIB INITIALISATION
*///////////////////////////////////////////////////////////////

* RUNLIB () ..:
****************************************************************
*  RUNLIB - Runtime library initalisation
****************************************************************
*  B  @RUNLIB
*--------------------------------------------------------------
*  REMARKS
*  If R1 in WS1 equals >FFFF we return to the TI title screen
*  after clearing scratchpad memory.
*  Use 'B @RUNLI1' to exit your program.
********@*****@**********************@**************************
RUNLIB  CLR   @>8302                ; Reset exit flag (R1 in workspace WS1!)
*--------------------------------------------------------------
* Alternative entry point
*--------------------------------------------------------------
RUNLI1  LIMI  0                     ; Turn off interrupts
        LWPI  WS1                   ; Activate workspace 1
        MOV   @>83C0,R3             ; Get random seed from OS monitor
*--------------------------------------------------------------
* Clear scratch-pad memory from R4 upwards
*--------------------------------------------------------------
RUNLI2  LI    R2,>8308
RUNLI3  CLR   *R2+                  ; Clear scratchpad >8306->83FF
        CI    R2,>8400
```

```
        JNE   RUNLI3
*--------------------------------------------------------------
* Exit to TI-99/4A title screen ?
*--------------------------------------------------------------
        CI    R1,>FFFF              ; Exit flag set ?
        JNE   RUNLI4                ; No, continue
        BLWP  @0                    ; Yes, bye bye
*--------------------------------------------------------------
* Determine if VDP is PAL or NTSC
*--------------------------------------------------------------
RUNLI4  MOV   R3,@WSEED             ; Set random seed value
        CLR   R1                    ; Reset counter
        LI    R2,10                 ; We test 10 times
RUNLI5  MOV   @VDPS,R3
        COC   @WBIT0,R3             ; Interupt flag set ?
        JEQ   RUNLI6
        INC   R1                    ; Increase counter
        JMP   RUNLI5
RUNLI6  DEC   R2                    ; Next test
        JNE   RUNLI5
        CI    R1,>1250              ; Max for NTSC reached ?
        JLE   RUNLI7                ; No, so it must be NTSC
        ORI   CONFIG,PALON          ; Yes, it must be PAL, set flag
*--------------------------------------------------------------
* Prepare tight loop
*--------------------------------------------------------------
RUNLI7  BL    @CPYM2M
        DATA  MCCODE,MCLOOP+2,6     ; Copy machine code to scratchpad
*--------------------------------------------------------------
* Determine TI-99/4A operating system version
*--------------------------------------------------------------
        BL    @CPYG2M               ; Read GROM >0480 into TMP0
        DATA  >0480,R3HB,2
        C     R3,@TMP004            ; Check for TI-99/4
        JEQ   RUNLI8
        C     R3,@TMP005            ; Check for TI-99/4A V2.2
        JEQ   RUNLI9
        JMP   RUNLIA                ; It's a TI-99/4A v1
*--------------------------------------------------------------
*    It's a TI-99/4 .... PANIC !
*--------------------------------------------------------------
RUNLI8  SETO  R1                    ; Set reset flag
        JMP   RUNLI1                ; Bye bye
*--------------------------------------------------------------
*    It's a TI-99/4A v2.2
*--------------------------------------------------------------
RUNLI9  ORI   CONFIG,V22OS          ; Set V2.2 flag
*--------------------------------------------------------------
* Initialize registers, memory, ...
*--------------------------------------------------------------
RUNLIA  CLR   R1
        CLR   R2
        CLR   R3
        LI    STACK,>8400           ; Set stack
        LI    R15,VDPW              ; Set VDP write address
        BL    @MUTE                 ; Mute sound generators
*--------------------------------------------------------------
```

```
* Setup video memory
*---------------------------------------------------------------
        BL    @VIDTAB               ; Load video mode table into VDP
        DATA  SPVMOD                ; See VIDTAB for details
        BL    @FILV
        DATA  >0000,>00,16000       ; Clear VDP memory
        BL    @FILV
        DATA  >0380,SPFCLR,16       ; Load color table
*---------------------------------------------------------------
* Load font
*---------------------------------------------------------------
        LI    TMP0,SPFONT           ; Get font option
        INV   TMP0                  ; NOFONT (>FFFF) specified ?
        JEQ   RUNLIC                ; Yes, skip it
        BL    @LDFNT
        DATA  >0900,SPFONT          ; Load specified font
*---------------------------------------------------------------
* Branch to main program
*---------------------------------------------------------------
RUNLIC  ORI   CONFIG,ENKNL          ; Enable kernel thread
        B     @MAIN                 ; Give control to main program
TMP004  DATA  >48FC                 ; TI-99/4   1979
TMP005  DATA  >5632                 ; TI-99/4A  1983  V2.2
*       DATA  >0A01                 ; TI-99/4A  1981  V1
* ...
```