R65 System based on a 6502

1. History

Early days

After building an analogue model railroad, I was interested in digitizing it and controlling it with a microprocessor. I bought therefore 1977 a Kim-1 board to learn and apply microprocessor programming. At that time, no computer bases railroad control systems were available. While the KIM-1 board was quite powerful, it lacked many features required for what it thought would be a decent model railroad control system. I therefore started to build 1977 a whole microprocessor based computer with keyboard and TV display around the KIM-1.

I wrote the complete system software and most of the application software myself. I used some kind of a bootstrap process for this. Every time a new capability was added, it was used to design further capabilities into the software. For example, the very first assembler was written on paper in machine language, and transferred to the R65 computer using a borrowed teletype with paper tape reader and puncher.

Two audio casette tape drives were used to store programs and data, using the built in drivers of the original KIM-1. A first editor was written in assembly language and transferred to the system using the teletype. A TV display board and a keyboard were soon added, and the drivers for it stored in EPROM. Because the KIM-1 could not hold a complete assembler source file, the first editor was sequential. It kept reading text from one cassette tape, edited it in the display memory, which could hold 40 lines, and then storing it on the second tape drive. One could therefore not go backwards more than 40 lines during the editing process.

The R65 as a word processor

At that time, I finished the research for my thesis in molecular biology, and wrote the thesis using the R65 as a very early word processor. I had written the necessary word processing software for this project, still in its sequential form and using the audio tapes. This was quite likely the first thesis written on a personal word processor at the ETH in Zurich. I built the hardware and software myself to print it on an old golf type typewriter. At that time, no other low cost printers were available to fit my budget.

Making the R65 more powerful with Pascal

Using audio tape as a storage device was slow, cumbersome and not very reliable. I therefore built 1968 the hardware and software for two 160 KB floppy disk drives. This was a very major improvement for the R65. Now the R65 was powerful enough so that I could start developing a Pascal based software development system for the R65. I had that in mind for a while, because I got in contact with the work of Niklaus Wirth as a PHD student at the ETH. But rather than having the original ideas of Niklaus Wirth in mind, which I thought were portability and safety for use in a teaching environment, I concentrated on making a powerful small Pascal environment for microprocessors, which was able to control memory mapped IO if necessary directly with Pascal. Programs where therefore not portable.

During all this time, I did not follow anymore the very first idea to build a digital controller for my model railroad. Only 30 years later I returned to that project. The R65 was by itself a more interesting and also very time consuming project, especially writing all the software. It was also the root of my later professional work in Silicon Valley in California in the second half of the 1980s.

How the R65 became obsolete

I used the R65 until 1983, when Turbo Pascal and the PC became available and were inexpensive and much more powerful than the R65. I bought one of the first PCs available to replace the beloved R65. During that time, development of better hardware and software was very rapid, and I thought that the R65 was obsolete and gone forever.

A R65 replica

Only about 35 years later I started to work on some projects to keep computer history alive, such as for example a software to emulate a Tektronix 4010 storage tube terminal emulator, see https://github.com/rricharz/Tek4010. I used tek4010 together with a PDP-11 replica called PiDP-11. I started to think that the R65 software also deserved to be kept alive and wrote a R65 emulator. Fortunately, my collaborator Ruedi Baumann had kept paper printouts of his copy of the R65 software for his hardware, called the JOB system. I was able to digitize his printouts. The JOB computer lacked a few capabilities of the R65, but the software was essentially identical. What was lost unfortunately was for example background printing to the ball type printer, and the pretty printing software.

The R65 emulator emulates a slightly improved graphics board as compared to original R65 system. The graphics are displayed with the proper pixel aspect ratio, but still use the R65 graphics drivers and 4K video memory. There has been a graphics overlay mode added, in which the graphics are displayed on the top right of the screen while the text display is still shown. I wanted to add this feature to the original R65 system, but that never happened.

Since then I had fun writing quite a few R65 Pascal programs. I still like R65 Pascal as a nice little software development system, and am amazed by what could be done in 1979 on an 8-bit, 64 KB memory microprocessor system with software built almost exclusively by one person. Using a Raspberry Pi 5 as the host, the Pascal compiler and editor (pedit) are very fast. Games still run at the original speed, if their code uses a properly timed waiting function.

2. General description of the basic R65 software

This manual describes the system software for the R65 computer system. The system software consists of 4 separate modules, which are stored in EPROM.

R65 CRT Controller:

Handles the Video interface and the keyboard.

Adress space: E000-E7FF

R65 Input/Output Controller:

Routines for printing on a RS-232 needle printer, interrupt handling and audio tape io routines. The audio tape is not emulated in the emulator. Printing to a printer is emulated by printing to a text file printout.txt in the working directory of the emulator.

Address space: E800-Efff

R65 Floppy Disk Controller:

Handles 2 floppy disk drives. 80 tracks, 10 sectors of 256 bytes per track. Includes the subroutines for sequential file i/o.

Address space: F000-F7FF

The R65 System Monitor:

Includes all necessary commands to work with Machinge Language Programs using hex numbers.

Address space: F800-FFFF

3. STOP, QUIT and SHUTDOWN

The following keyboard keys are recognized by the R65 Emulator and executed immediately. The buttons on the top left of the screen can also be touched or clicked.

<shift>ESC or STOP button:

Emulate a 6502 non mascable interrupt (NMI)

<shift>MENU or <shift>WINDOWS or QUIT button or x(close window):

Quit R65 emulator, go to dektop

<shift>ALT SHUTDOWN:

Quit R65 emulator and shut down Linux

Note that open sequential files are not closed if one of these keys or buttons are executed. This is not a problem if a sequential file read is in progress, but in the case of a sequential file write this leaves a large useless file on the disk. Deleting this file afterwards and using PACK on the disk will solve the problem. Try to avoid therefore these actions if a sequential disk write is in progress. Disks are closed properly.

4. Installation

The R65 Emulator for Raspbian can be downloaded from GITHUB and built on the local system. Open a terminal and type:

```
cd
git clone --depth 1 git://github.com/rricharz/R65
sudo apt-get install libgtk-3-dev
cd R65
cp R65-Emulator.desktop ~/desktop
cd R65-Emulator
make clean; make
mkdir Files
cd Disks
cp EMPTY.disk WORK.disk
cd ../Tools
make clean; make
```

Installing the mandatory fonts

```
cd
cd R65/R65-Emulator/Fonts
sudo ./install_fonts
```

Starting the emulator

Now you should be able to start the R65 emulator using the R65 Emulator icon on the desktop. Read chapter 5

Floppy Disk Drives before proceeding.

You can also start the emulator using a terminal window:

```
cd
cd R65/R65-emulator
./emulator
```

“emulator” has the following options:

- full	Display the emulator in a full screen window. Looks much more pretty.
- pixelated	Use a pixelated font for the R65 screen. This resembles the original. Works best together with -full.

As the first thing, you should rename your newly created WORK disk.

Type FLOPPY WORK,1 to ensure that the WORK disk is in drive 1.

Type VOLUME 1<return> and then enter WORK.04 <return> to label your work disk properly.

Adding additional disks

Later, you can create additional disks with the name MYNAME (where MYNAME stands for any name you want to use) as you have done above for WORK. In the Raspbian terminal, type

```
cd
cd R65
cd R65-Emulator/Disks
cp EMPTY.disk MYNAME.disk
```

Then, restart R65-Emulator using the Desktop icon.

Type FLOPPY MYNAME,1 to ensure that the MYNAME disk is in drive 1.

Type VOLUME 1<return> and then enter MYNAME.05 <return> to label your new disk properly. Instead of 05 you can give the floppy any number you wish. It is strongly recommended to label your disks with the same names as the name of the disk file in Raspbian in order to avoid confusion. On the top status line of the R65 emulator the name of the Raspbian disk file is displayed.

Upgrading the R65 Emulator

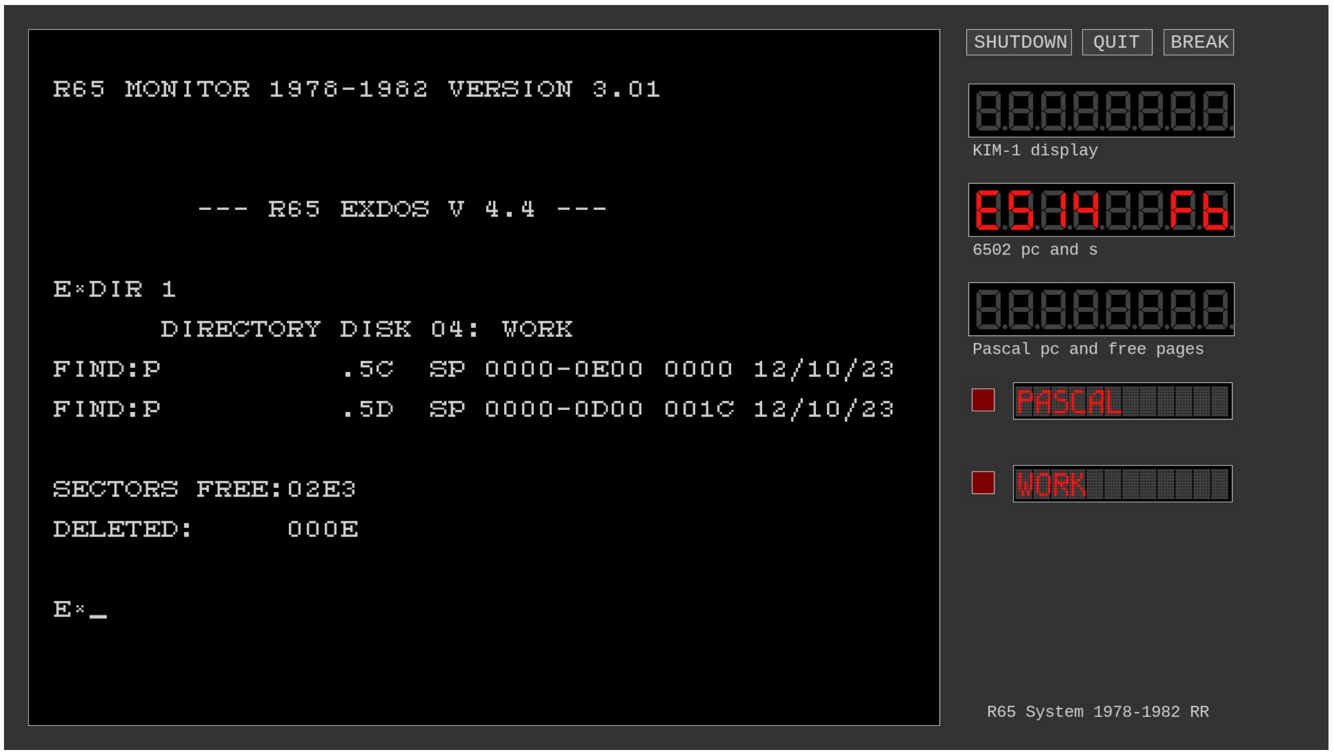
If you have not modified the PASCAL, SOURCE and PROGRAMS disks, upgrading is easy. If you have modified these disks, you need to first make a backup of your modifications on any other disk. It is necessary that the upgrading process upgrades these disks, because most upgrades are likely programs, libraries and source files on these disks. Also, some of these programs might have been modified to work together properly with any upgraded version of the Emulator.

Type

```
cd
cd R65
make clean
cd Tools
make clean
cd ..
git pull
make
cd Tools
```

make

5. R65 Emulator front panel



```
R65 MONITOR 1978-1982 VERSION 3.01


        --- R65 EXDOS V 4.4 ---

E*DIR 1
      DIRECTORY DISK 04: WORK
FIND:P          .5C  SP 0000-0E00 0000 12/10/23
FIND:P          .5D  SP 0000-0D00 001C 12/10/23

SECTORS FREE:02E3
DELETED:     000E

E*_
```

On the left side is the **CRT screen**, using a pixelated font to resemble the original, while still being readable. The original had a 5x7 pixel character set, while the emulator uses a 7x11 matrix to imrove readability on modern high resolution computer screens. It is shown if the emulator is started with the -pixelated option. Without this option, a modern monospace font is used. Do not forget to install the required fonts as described in the installation section of this manual.

On the top right are 3 buttons, which can be clicked with the mouse. “Break” executes a 6502 NMI break as on the original KIM-1 board. “Quit” exits the emulation. “Shutdown” exits the simulation and runs the host Raspberry Pi or Linux system down. On suitable keyboards, <shift><esc> executes the break button, <shift><menu> executes the quit button, and <shift><alt> executes the shutdown button.

The “KIM-1 display” resembles the original KIM-1 seven segment led display. The original had a 6 digit display with a space between the 4th and 5th character, which was driven by the R65 software using a background, interrupt driven program. This code was unfortunately lost, so that the R65 emulator handles this in the emulator code, using a 8 digit 7 segment display. This display is under control of user programs written in 6502 code or Pascal. See the Pascal “ledlib” library for an example. As long as the user hasn’t written anything onto this display, the emulator shows the current time and host CPU temperature.

The “6502 pc and s” seven segment display shows the 6502 program counter (pc) and stack pointer (s) in hex. The program counter is frozen while the program waits for user input, even so on the original this was handled in a short 6502 code loop. But to avoid overheating of the host CPU of the emulator this is now done in the emulator software with the proper Linux sleep functions. The stack pointer can be used to monitor the 256 byte stack of the 6502. A too low number suggests that the stack may soon overflow and the program may crash. Typically this happens if a recursive subroutine is called with too many levels of recursion, or if too much data is pushed on the stack, using the corresponding 6502 instructions such as PHP or PHA. There is a delay of 3 seconds at the currently lowest number to allow to monitor this critical level.

The “Pascal pc and free pages” seven segment led display shows the pascal program counter in decimal and the number of free 256 byte pages in hex. The program counter of each program line is shown in the Pascal program listings as printed by the Pascal compiler. The number of free pages is the free memory between the top of the

Pascal user memory (topmem), or the heap pointer if heaplib is used, and the Pascal stack pointer, which moves upwords. This should never go below approximately 5 pages to allow the Pascal interpreter enough space to breath. As on the display of the 6502 stack there is a delay of 3 seconds at the currently lowest number to allow to monitor this critical level.

Below the seven segment displays are two LEDs, which glow when the two floppy drives are spinning. Because it took some time to spin up a floppy drive on the original, the drives were kept spinning for several seconds after a read or write was executed. Constantly spinning the drives was not possible to minimize wear out of the floppies. To the right of these LEDs are two dot matrix displays showing the names of the floppies in the drive. I always wanted to have these displays, but they were not yet available in the late 1970s.

6. Monitor Commands

The following abbrevations are used:

exp1	A hexadecimal expression
filnam	A file name of up to 16 characters
wfilnam	File name with wild cards (* and ?)
drive	0 or 1 for the two floppy drives
cy	a cyclus number for a file
[ ]	These arguments can be omitted

CLRB exp1

Erase breakpoint at address exp1.

COPY filnam[.cy],from_drive,to_drive

Copy a file from from_drive to to_drive. The drives must both be given and must be different. The wildcard character @ can be used in filnam. Sequential and block files can be copied. The maximal file size to be copied is $9000 (32768) bytes. The memory space $2000-$BFFF is used as a buffer. Extremely large sequential files (such as COMPILE1:P and ASSEMBLER:A) cannot be copied using the copy command. Use EXPORT and IMPORT for these very large files. Do not use the combination of EXPORT and IMPORT for any files except :A and :P files.

DATE

Display actual date. Date can be changed once displayed. <return> returns to monitor. The emulator reads the Linux system dates at startup and stores it in the R65 date memory. Afterwards it can be changed if necessary.

DELETE filnam[.cy][,drive]

Delete a file. The wild card letter @ can be used in filnam.

DFORMAT diskname,drive

Erase the disk directory completey and create a new directory

DIR drive

Display directory of a disk drive.

DIS [exp1][,exp2]

Disassembles machine language program at address exp1 and following. exp2 (default 10) commands are decoded. Use <esc> to return to the monitor.

EDIT filnam[.cy][,drive]

Edit a file with the Linux mousepad editor. Use the command NEW (see below) if you want to edit a new file.

EXPORT filnam[.cy][,drive]

Export a sequential file to the Linux Files directory

FDIR [drive]

Full disk directory, including deleted files which can be recovered with the revive command. Also shows how many sectors can be recovered with the pack command.

FLOPPY disknam[,drive]

Change the floppy disk in drive (default 0) to the one with the name disknam. Floppy disks are stored in the Linux Disks folder with the extension .disk, but disknam must be given without this extension. Additional floppies can be made there by using Linux to make copies of existing disks. The emulator can handle an unlimited number of floppies, but only 2 floppies can be in the 2 drives at the same time.

GO [exp1][,register definitions]

Start executing the machine language code at address exp1. See command REG for the register definitions.

GSB [exp1][,register definitions]

Same as GO, but instead of a jmp to the start address a jsr (subroutine call) starts the execution.

IMPORT filnam[,drive]

Import a Linux file from the Linux Files directory. The Linux file must have the extension .asm or .txt, but filnam must be given without this extension.

LOAD filnam[.cy][,drive][,exp1]

Load a disk file with name filnam (and cyclus) from drive to the stored memory addess. If exp1 is given, the file is loaded to the address exp1.

NEW filnam[.cy][,drive]

Create a new empty sequential file.

PACK [drive]

Permanently erase all deleted files and recover the disk space. FDIR displays how much space can be recoverd with the PACK command.

PROTECT filnam[.cy][,drive]

Protect a file. Delete will ask for permission before deleting protected files.

PRTB

Display the active breakpoints.

REG [register definitions]

Display saved CPU registers, allow to change them. Register definitions are name=exp[,...], where the register names are P,S,F,A,X,Y. Use <esc> to return to the monitor.

RENAME filnam[.cy][,drive]

Rename an existing file. The command asks for the new name and subtype. The file is protected if subtype is preceded with !.

REVIVE entry number[, drive]

Recover a deleted file. Use FDIR to get the entry number.

RESETB

Clear all active break points.

RUN filnam[.cy][,drive],[,exps1]

Loads a file like load, but starts it afterwards automatically. The file type must be "M" (machine language).

SAVE [drive]

Make a backup copy of the disk directory on the last track of the disk.

SETB exp1

Set a breakpoint at address exp1

STEP [exp1]

Execute one machine language instruction. Use <esc> to return to the monitor.

STORE filnam[.cy],[drive],exp1-exp2,[P]S[,exp3]

Store a file from memory exp1-exp2, file type S. If exp3 is not given, the address for LOAD is exp1, otherwise exp3.

SWAP [dive]

Exchange the disk directory with the one backed up with save before.

TRACE [exp1,exp2]

Like STEP, but executes exp2 (default 6) instuctions

VOLUME [drive]

Set the internal name of a disk displayed if DIR or FDIR is run in EXDOS. Does not change the file name of the floppy disk file, which is used in the FLOPPY command. It is therefore recommended to setthis name to the name of the disk file.

/exp1

open memory address at exp1

/exp2	open new address exp2
<return>	open next address
L	open previous address
'C	set memory cell to Ascii code
exp2	set memory cell to exp2

7. PASCAL commands

The PASCAL envirement is started with the command

```
RUN PASCAL
```

The Pascal system tries first to read the command from drive 1, if that fails from drive 0. Drive 0 is the drive where most of the programs should be (normally on the disk PASCAL. Commands, which are under ongoing work, should be on drive 1 (normally on disk WORK).

In the PASCAL environment, the following commands are available, if they can be found on disk 0. In their PASCAL version, most commands have drive 1 as the default drive for their argument in order to facilitate Pascal development on the WORK disk on drive 1. The source of the Pascal programs should be on the disk SOURCEPASCAL.

ARGLIST arguments

Displays the list of arguments as they are forwarded to Pascal programs.

BOUNCE

Bouncing ball on graphics display.

CALC

A small calculator. Makes basic calculations and displays result in decimal, hexadecimal and binary, when possible.

CHKDSK

Check integrity of disk directory.

CLEAN [drive]

Automatically deletes unnecessary files on the disk drive (default 0).

- All files with the type :Q (compiler temporary files)
- All but the latest cyclus of any file

The command CLEAN does not PACK the disk, in case you decide that you need to REVIVE a file which has been deleted by CLEAN. Use PACK after CLEAN if you want to recover the space.

COMPILE filnam[.cy][,drive] [/options]

To compile a Pascal source file (:P). Give the file name without the :P. Options can be several letters stringed together. The Pascal compiler is a 2-pass compiler. The second pass is only executed if the first one is successful.

P	Print hard copy (into printout.txt)
I or R	Compile with runtime index checking (slower and larger file)
N	Do not produce a loader file for pass 2
L	Insert line numbers in object file, which are displayed, if a Pascal runtime error occurs. They are also required for the PCODES, and SETB commands

Use COMPILE1 instead of COMPILE to compile libraries. Libraries do not need the second pass COMPILE2, and an error message will be produced when COMPILE2 wants to load the loader file :Q of a library. The default drive is drive 1.

COMPILE 1 filnam[.cy][,drive] [/options]

Execute only pass 1 of the compiler. The default drive 1 drive 1. Same options in compile.

COMPILE 2 filnam[.cy][,drive]

Execute only pass 2 of the compiler. Libraries do not need the second pass of the compiler and cannot be loaded with COMPILE2. The default drive 1 drive 1. Options are ignored.

COPY wfilnam[.cy][,sourcedrive] desdrive

Copy a file file from thesource drive to the destination. Copies all types of sequential files and Pascal binary runtime files. Cannot be used for machine language programs. The lenght of sequential files is only limited by available floppy disk space.

DELETE wfilnam[.cy][,drive]

Delete a file. The wild card letter @ can is not available in Pascal. Use the command CLEAN instead to delete multiple unwanted files. Drive 1 is the default drive in the Pascal version of the command.

DIR [drive]

Display directory of a disk drive. The entries are displayed in several columns, depending on the longest name and the number of files on the disk. The free space and the space occupied by deleted files is displayed in decimal. The default drive is remains drive 0 as in EXDOS.

EDIT filnam[.cy][,drive]

Edit a file with the Linux mousepad editor. Use the command NEW (see below) if you want to edit a new file. The default drive is drive 1, and the command automatically adds :P to filnam, if no other type is given. Use pedit instead of edit to use the internal Pascal editor.

ERROR errorcode

Display Pascal error code as text.

EXTIME pname,drive arguments

Measure and diplay the execution time of a Pascal program pname, called with its arguments.

FIND filnam

Find files on drive 0 and 1. Wildcards * and ? are allowed, except as the first character.
Examples: find name*, find name:?, find name:p

FLOPPY fname[,drive]

Change the floppy disk in drive (default drive 0) to the one with the name fname. Floppy disks are stored in the Linux Disks folder with the extension .disk, but fname must be given without this extension. Additional floppies can be made there by using Linux to make copies of existing disks. The emulator can handle an unlimited number of floppies, but only 2 floppies can be in the 2 drives at the same time.

GETSOURCE pname

Copy a source file from SOURCEPASCAL or SOURCECOMPIL to WORK.

GRAPH

Example of displaying a graph of a table of a real numbers (TABLE.X on disk 1). Create first the table with MAKETABLE. The example demonstrates how to read from a binary random access file.

GR3D

Create a 3D plot on the R65 graphics display. The same plot is available as TEK3D on an attached tek4010 emulator (see chapter 9).

GREND

Release graphics memory and turn graphics overlay off.

GRTEST

Display a test pattern on the R65 graphics window.

GRTEST2

Create a test plot on the R65 graphics display. The same plot is available as TEKTEST on an attached tek4010 emulator (see chapter 9).

HELP topic

Display help topics. For a list type help without a topic

KEYS

Display ASCII code of keys typed.

LEDTEST

Create test patterns on the 7-segment display of the R65 replica, or on the top line of the R65 window.

MAKETABLE

Example of creating a binary random access file holding a table of real numbers. The file is created on disk1 and is called TABLE:X. The example created a fading sine wave. A graph of the file can be displayed with GRAPH.

NEW filnam[.cy][,drive]

Create a new empty sequential file. The Pascal version of the command has drive 1 as the default drive, and sets the file type automatically to :P.

PACK [drive]

Permanently erase all deleted files and recover the disk space. DIR displays how much space can be recoverd with the PACK command, and CLEAN automatically deletes certain files (see details in the description of the command). The default drive is remains drive 0 as in EXDOS.

PAINT filnam[.cy][,drive]

Paint in graphics canvas, save canvas on exit

PEDIT filnam[.cy][,drive]

Edit a file with the internal Pascal editor. Use the command NEW (see below) if you want to edit a new file. The default drive is drive 1, and the command automatically adds :P to filnam, if no other type is given. Use edit instead of pedit to use the linux mousepad editor.

PUTSOURCE pname

Move a source file from WORK to SOURCEPASCAL or SOURCECOMPIL.

PUTOBJECT pname

Move a Pacal object file from WORK to PASCAL.

RESETB

Clears a breakpoint set with SETB.

SHOW filename

Show a graphics picture saved with “snap”:

SETB n

Set a breakpoint at line n. Stops execution of any command compiled with the /L option at line n. Use RESETB to clear the breakpoint.

SINETST

Create a sine and cosine graph on the R65 graphics screen.

SNAP filename

Save a snapshot of the graphics display. Display it later with “show”.

TEKTEST

Create a test plot on an attached tek4010 emulator (see chapter 9). The same plot is available as GRTEST2 for the R65 graphics display.

TEK3D

Create a 3D plot on an attached tek4010 emulator (see chapter 9). The same plot is available as GR3D for the R65 graphics display.

TGRAPH

Example of displaying a graph of a table of a real numbers (TABLE.X on disk 1) on an attached Tektronix 4010 terminal. Create first the table with MAKETABLE.

THROWSIM

Display a flight path of a thrown object on the R65 graphics screen.

WATCH

A small digital time of day watch and stop watch using the simulated KIM-1 display.

8. Pascal libraries

Look at the source of these libraries on SOURCEPASCAL for details

ARGLIB

Handles arguments given by the pascal system to a program

DISKLIB

Provide some disk related procedures, e.g. calculate free space on disk

LEDLIB

Handles output to the 7-segment display

MATHLIB

Handles math functions and the output of floating point numbers

PCODES filename [first]

Displays a Pascal source file (:P) together with the pcodes generated for each line, starting from line

first (default 1),

PLOTLIB

Plot library for the R65 graphics screen

RALIB

Random access (binary) file access library

SHOW filename[,drive] [first]

Display a text file (default :P), starting from line first (default 1). The display is paused every 12 lines and waiting for any key to be typed. <ESC> stops the display

STRLIB

String handling with cpnt pointers

SYSLIB

Main R65 Pascal system library

TEKLIB

Plot library for an attach tek4010 plotter

TIMELIB

Handles system and execution time

WILDLIB

Handles wild cards

9. Pascal games

ALIEN

Save the earth from an alien invasion.

PONG

Play the game of Pong.

REVERSI

Play the game of Reversi. The rules can be found at https://en.wikipedia.org/wiki/Reversi. Please note that the first two moves need to be placed in the center.

SCRAMBLE

Climb up on ladders and exit through the top right door. This is the most complex game that I have written for the R65 system. It has not been optimized for minimal processor usage and needs a Raspberry Pi 5 or similar hardware to run smoothly.

10.R65 Graphic Basic

The R65 Graphic Basic includes the basic commands found in Basic Interpreters of that time. The following commands have been added in the R65 version or are working slightly different:

CLOSE [D1]

Close a sequential file (if D1 is given) or all open sequential files

DIR [D]

Dispay the directory of floppy disk drive D (default 0).

FILES

Displays information about all currently open sequential files

GET [#D1;] V1 [,V2….]

Read one character from device D1. If V1 is a string variable, one character is read, otherwise one digit of a number. The following devices are supported:

D1=0:	Read from keyboard, wait until a key is ready
D1=1:	Read from keyboars, return with empty string if no character available
D1=2:	Not implemented in Emulator

D1=3:	Read from keyboard and display immediately on Graphics Display
D1>=4:	Read from a open disk file

INPUT [#D1] V1 [,V2…]

Read from device D1 (see GET for details). Reads one full string or one number)

LOAD S1[.D1] [,D2}

Load a Basic program from floppy drive D2. Example LOAD “LIFE:B”

LOAD #D1;

Import a basic program from a text file which has previously been opened with OPEN D1

MOVE N1,N2

Move graphic cursor to n1,n2

OPEN D1,S1[.D2][,D3,[D4]]

Open a sequential file (D1>=4). D3 is the floppy drive number, D4=0 for read and 1 for write.

PLOT N1,N2[,D1]

Plot a dot on the graphics display at N1,N2. D1=0 in white, D1=1 in black, D1=2 inverted.

PLOT NEW

Initialize graphics display, Switch alphanumeric display buffer to 16 lines

PLOT CLR

Clear graphics display

PLOT STOP

Go back to alphanumeric display without loosing graphics display buffer

PLOT CONT

Go back to graphics display after a PLOT STOP

PLOT END

Erase graphics display area and go back to alphanumeric display. Switch back alphanumeric display buffer to 42 lines.

PRINT [D1;] ….

Basic print statement. Devices are:

D1=0:	Standard display
D1=1:	Not implemented in emulator
D1=2:	Not implemented in emulator
D1=3:	Graphics display at current position of graphic cursor
D1>=4:	Open sequential file

STORE S1[.D1] [,D2]

Store Basic program on drive D2.

SYS N1

Execute 6502 code at N1

WAIT N1

Wait N1 times 10 msec. The emulator process is suspended in intervals of 10 msec to avoid overheating of the Raspberry Pi CPU as during character input.

11. Editing files

Editing on a system without a mouse is quite cumbersome by modern standards. Therefore the adaption for the virtual R65 system includes the EDIT command in exdos and Pascal to use the external mousepad editor.

There is also an internal, Pascal based editor available by using the PEDIT command. It is limited to 400 lines and slow for certain operations. Nevertheless it is quite usefull. On the original system, a stream oriented text editor was used due to memory limitations on the first version of R65.

The following escape sequencies are available in PEDIT:

t	Move to top of text. Available also with the HOME key where available.
b	Move to bottom of text. Available also with the END key where available.
l*n*	Put the cursor on line *n* and scroll if necessary.
f *xxx*	Find string *xxx*. Entering an empty string clears all marked fields.
a	Find again using the previous string.
c*n*	Copy marks *n* lines for copying. The copying will be done when the paste command is executed. It is therefore possible to modify the marked lines before the paste command is executed.
p	Paste the marked (copied) lines to the cursor line. This duplicates the lines.
m	Move the marked (copied) lines to the cursor line. The originals are deleted.
d*n*	Delete *n* lines
w	Write the file to disk. The next cyclus number is used. It is therefore possible to write multiple times to disk.
q	Write file to disk and quit.
k	Kill editor without writing to disk.
h or ?	Help: Show available escape characters.

12. Using Pascal

R65 Pascal is a very powerful programming environment for an 8-bit processor. Originally developed for 32 KB memory, it has later been expended and runs now in the 46 KB RAM memory available in the R65 system. In this memory area, the Pascal interpreter (PASCAL) written in assembler, the Pascal program SYSTEM:R written in Pascal, the user program name:R written in Pascal, the Pascal stack and the heap have to fit. It is even possible to run a further program in a user program, see for example COMPILE, GETSOURCE, PUTSOURCE and PUTOBJECT.

The R65 Pascal is a very limited subset of Pascal. It does not support multidimensional arrays (expect an array of cpnt strings), types and records. But is supports low level access to memory with the extremely powerful mem declaration, and pointers to cpnt string on the heap delimited with the ENDMARK character chr(0). Programs are therefore not portable, but can handle basic R65 memory mapped hardware functions very well. The heap is quite limited. It does not support a *release* except of the last *new*, because it does not support memory management (pedit uses a special type of a more powerful release).

If you want to use Pascal, but the disk PASCAL in drive 0, and normally WORK in drive one. The Pascal programming environment requires this configuration.

Pascal programs are executed with *progname*. The Pascal system first searches for progname on drive 1, and then on drive 0. It is therefore possible to work with new or modified programs on drive 1.

Pascal source files are edited with *edit filename* or *pedit filename (called without the default extension :P)* on drive 1. A new source file is generated with *new filename*, and then edited.

Pascal source files :P are compiled with *compile filename*, and the possible options /*xxx*. Important is the option /*L*, which generates source line number information in the object file, as well as *R,* which generates index range checking at runtime. Both options as well as others can be combined with /*LR*, for example.

Pascal source files are saved on the drive PSOURCE, which does not have to be mounted by the user. The commands *getsource name* copies a source file from this disk, which is mounted temporarily on drive 0. *putsource name* puts the source back on psource, which is cleaned and packed. Any remaining source files *name* on the dirs WORK are deleted during putsource, but WORK is not packed to make it possible to REVIVE earlier versions of the source, if required. Some programs like *getsource, putsource, putobject, edit, pedit* and *compile* display the remaining disk space on

WORK. Use *clean* and *pack* if the free disk space goes below 20%.

Compiled object files :R are copied from WORK to PASCAL with the command *putobject*. Thea are then deleted from drive WORK, as well as any remaining temporary files :Q generated by *compile*.

Debugging Pascal programs

The traditional way to debug Pascal programs is by using the *abort;* statement in the source, which stops execution at that location immediately. The statement *exit*; does the same thing, if called in the main Pascal block between *begin* and *end.* But it only returns from a procedure or function immediately to the calling code, if executed in a procedure or function. Also, *printf* can be used to examine variables and parameters, but the program needs to be compiled again after inserting these statements. Usually this is done as follows:

```
program progname;
uses libname, ….;

const debug=true;

….

if debug then printf(….);
```

Further runtime debug tools (program compiled with the */L* option)

abort; or any runtime error display the line number of the current source line, rather than the program counter of the pcode interpreter.

pcodes progname displays the pcodes for each line to better understand what the code does.

setb linenumber inserts a break point at runtime at line number *linenumber*. Afterwards, the p-code interpreter goes into a **trace** mode, where each <return> entered executes one additional line consisting of several p-codes. This allows to find out in which order the code is executed, for example in *if, while* and *repeat* statements, as well as procedure and function calls. Calls to library procedures and functions are executes without tracing. Libraries should not be compiled with */L*. Only one breakpoint can be set. r*esetb* deleted that breakpoint. Compiling again is not required each time *setb* and *resetb* are used.

13. Floppy disk drives (tapes are not emulated anymore)

Drive 0	disk drive A
drive 1	disk drive B

Floppy disks can be changed with the EXDOS command:

```
FLOPPY disknam[,drive].
```

The recommended setup is to use PROGRAMS, PASCAL or SOURCE in drive 0, and WORK in drive 1. The defaults are set for this setup. The disks PROGRAMS, PASCAL and SOURCE are distribution disks. They are replaced with their standard version during a software upgrade. Use WORK for your own work and make backup copies on another disk before upgrading the software. The latest version of the R65 emulator supports disks with 256 file entries and 160 tracks of 16 sectors, which results in a capacity of 2560 sectors.

14. Attaching a printer/plotter

The R65 emulator emulates a printer by printing to the Linux file “printout.txt” in the R65 directory. This file is cleared or created each time the emulator starts. The text printed is formatted for Linux, and page headers are added. It is possible to print raw R65 text (without any formatting) by switching the printer driver to raw mode:

Device control 1	hex 11	switch to raw mode
Device control 2	hex 12	switch back to normal formatted mode

If you want to see the printed text while it is generated, open a shell window, go to your R65 directory and type

```
tail -f printout.txt
```

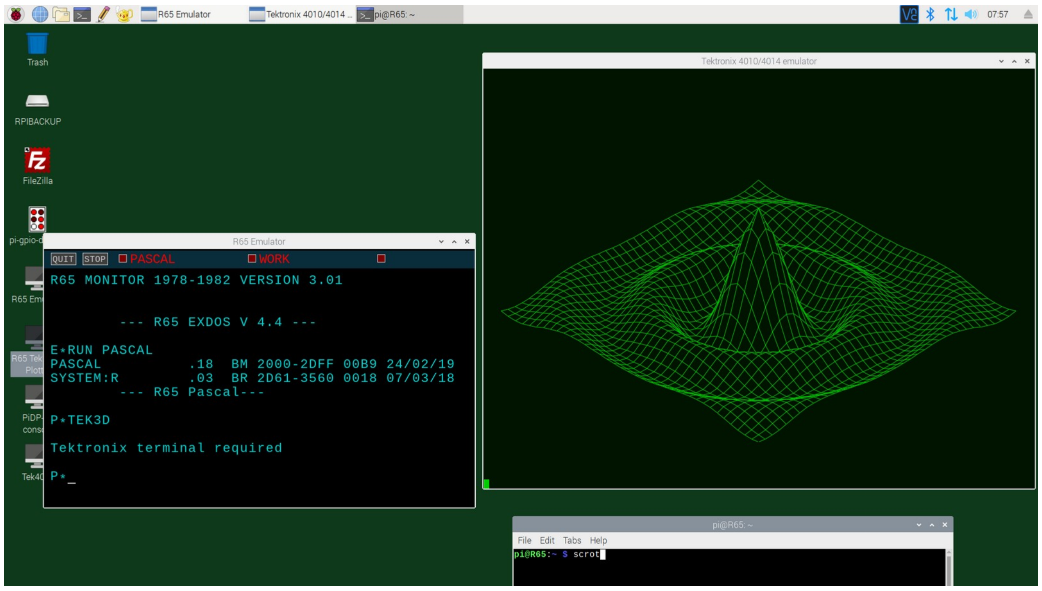
It is also prossible to use my Tektronix 4010 emulator as a plotter window. Download tek4010 from https://github.com/rricharz/Tek4010. After having tested your installation, put a copy of the executable “tek4010” in your R65 directory.

Now start R65 and afterwards tek4010 by executing the command

```
./tek4010 -autoClear tail -f printout.txt
```

You can now see any printed text in the tek4010 window.

With R65



```
R65 MONITOR 1978-1982 VERSION 3.01


        --- R65 EXDOS V 4.4 ---

E*RUN PASCAL
PASCAL          .18  BM 2000-2DFF 00B9 24/02/19
SYSTEM:R        .03  BR 2D61-3560 0018 07/03/18
        --- R65 Pascal---

P*TEK3D

Tektronix terminal required

P*_
```

```
pi@R65:~ $ scrot
```

PASCAL, you can also plot in the tek4010 window. The “teklib” library is on the PASCAL disk and its source is on the SOURCECOMPIL disk. There are also examples on the SOURCEPASCAL and PASCAL disks: “tektest”, “tek3D” and “tgraph”. Please note that R65 floating point calculations using the 8-bit R65 Pascal are rather slow as compared to modern standards, but the examples show that it was possible to produce high resolution graphics with a Tektronix 4010 and a 6502 8-bit computer in the late 1970s.

15. File Type

"Block" files (binary files)

M	Machine language program
R :R	Pascal binary runtime file
X :X	Other binary file

Sequential files (text files)

A :A	Assembler source file
P :P	Pascal program or library source file
Q :Q	Pascal loader program file
T :T	Pascal loader library file
L :L	Pascal loader ident table
W :W	Pascal compiler reserved words table
B	Other text file

16. Error Codes

System error codes

01 READ/WRITE ERROR

02 CHECKSUM ERROR
03 ESCAPE EXIT DURING READ/WRITE
04 RECORD NUMBER ERROR
05 FILE TYPE ERROR
06 FILE NOT FOUND
07 DISK NOT READY
08 DIRECTORY FULL, FILE NOT STORED
09 ILLEGAL IRQ
10 EXPRESSION MISSING
11 MEMORY CELL CANNOT BE CHANGED
12 BREAK TABLE FULL, NOT INSERTED
13 ILLEGAL MEMORY CELL FOR BREAK
14 DOUBLE BRAK POINT SETTING
15 END OF LINE EXPECTED
16 SYNTAX WRONG IN REGISTER NAME OR =
17 BREAKPOINT NOT FOUND IN TABLE
18 SYNTAX FRONG IN STORE
19 FILE SUBTYPE WRONG OR MISSING
20 WRONG FILE TYPE NOT RUN
21 UNKNOWN MONITOR COMMAND
22 ILLEGAL OPCODE FOR STEP/TRACE
23 TOO MANY OPEN FILES, NOT OPENED
24 DIRECTION ERROR IN SEQUENTIAL R/W
25 WRONG FILE NUMBER, FILE NOT OPEN
26 DISK FULL, FILE NOT STORED
27 RANDOM ACCESS INDEX OUT OF RANGE
28 ILLEGAL DRIVE FOR RANDOM ACCESS
29 FILE NOT OPEN FOR RANDOM ACCESS WRITE

ASSEMBLER error codes

31 CLOSING) EXPECTED IN EXPRESSION
32 SYNTAX ERROR IN LABEL
33 HEX CHAR EXPECTED AFTER $
34 LABEL TABLE OVERFLOW
35 LOGICAL CHAR EXPECTED AFTER #
36 EXPRESSION NOT RESOLVED (PASS 2)
37 SYNTAX ERROR IN OPCODE
38 MNEMONIC OR ADDRESSING NOT ALLOWED
39 ADDRESSING MODE NOT ALLOWED
40 SYNTAX ERROR IN OPERAND
41 ABSOLUTE ADDRESS NOT ALLOWED
42 MORE THAN 1 UNRESOLVED LABEL IN FORWARD BRANCH
43 BRANCH EXCEEDS BOUNDS
44 FORWARD BRANCH TO THIS LABEL EXCEEDS BOUNDS
45 DOUBLE LABELDEFINITION
46 MISSMATCH IN SECOND PASS
47 LABEL MISSING IN EQU
48 OPERAND OF BYT TOO LONG
49 EXPRESSION MUST BE RESOLVED
50 LINE TOO LONG
51 CHAR FOLLOWS LOGICAL END OF OPERAND
52 TOO MANY UNRESOLVED BRANCHES (NOT INSERTED INTO TEST TABLE)

EXDOS error codes

61: WILD CARD NOT ALLOWED
62: ONLY FOR DISK, NOT TAPE
63: ILLEGAL COPY
64: FILE TOO LARGE
65: WRITE ERROR

66: IMPORT ERROR
67: UNKNOWN EMU COMMAND
68: UNABLE TO RUN MOUSEPAD

PASCAL runtime error codes

81: DIVISION BY ZERO
82: STACK OVERFLOW
83: INDEX OUT OF BOUNDS
84: PROGRAM NOT FOUND, OR NOT PASCAL PROGRAM
85: ILLEGAL P-CODE
86: ESCAPE DURING EXECUTION
87: NO LOADER FILE MADE BY COMPILER
88: HEAP OVERFLOW
89: POINTER NOT ALLOCATED (NIL)
90: WRITING TO CONSTANT STRING NOT ALLOWED
91: STRING TOO LONG (MORE THAN 63 CHARACTERS)
92: ONLY THE LAST ALLOCATED STRING CAN BE RELEASED

PASCAL argument error codes

101: ARGUMENT IS NOT STRING OR DEFAULT
102: ARGUMENT IS NOT NUMBER OR DEFAULT
103: ARGUMENT IS NOT STARTING WITH /
104: UNKNOWN ARGUMENT
105: DRIVE IS NOT 0 OR 1
106: ARGUMENT SYNTAX ERROR
107: TOO MANY ARGUMENTS

17. R65 Control keys

See also capture 2: STOP, QUIT and SHUTDOWN

Video control functions

	CTRL	VCOD		ASCII	
CURSOR DOWN	CDOWN	E9	X	18	
CURSOR RIGHT	CRIGHT	E8	V	16	
CURSOR LEFT	CLEFT	E7	C	03	
CURSOR UP	CUP	E6	Z	1A	
CURSOR HOME	CHOME	E5	A	0	
INSERT CHAR	INSCHR	E4	U	15	CTRL-RIGH TARROW
DELETE CHAR	DELCHR	E3	Y	19	CTRL-LEFT ARROW
CLEAR LINE	CLRLIN	E2	E	17	
CLEAR TO END OF SCREEN	CLRDSP	E1	Q	11	CTRL-RETURN
INSERT LINE	INSLIN	C4	D	04	
DELETE LINE	DELLIN	C3	F	06	
TOGGLE ALPHA/GRAPHICS	TALGRA	E9	L	0C	
ESCAPE	ESCAPE	91		1B	
SET TABULATOR	SETTAB	8B			
ROLL DOWN	RDOWN	89	B	02	CTRL-DOWN ARROW
TO RIGHT MARGIN	CRMARG	88			
TO LEFT MARGIN	CLMARG	87			
ROLL UP	RUP	86	H	08	CRTL-UP ARROW
REVERSE HOME	RVHOME	85	P	10	
RESUME WRITING	RWRITE	84			
QUIET WRITING	QWRITE	83			
TOGGLE BLACK/WHITE	TBLWHI	82	E	05	
CLEAR GRAPHIC DISPLAY	CLRGRA	81			

Other control functions

PRINT ALL ON		PRTON	R	12
PRINT ALL OFF		PRTOFF	T	14
DISPLAY CONTROL CHAR		DSPCC	S	13
CLEAR TABULATOR		CLRTAB	O	0F
INVERSE VIDEO	I	NVVID	N	0E
NORMAL VIDEO		NORVID	K	0B
CARRIAGE RETURN		EXCR		0D
LINE FEED		EXLF		0A
TABULATOR (8)		TAB	I	09
BELL		BELL	G	07
PAD CHARACTER		PADCH	@	00

R65 emulator control functions

MAKE SCREEN SHOT (WAYLAND only)	DEL	del	FF

18. “Burning” new PROMS/EPROMS

The System software on the original system was burned in EPROMS and used the KIM-1 ROM. The corresponding Assembler source files are:

- KIM1.asm
- CRT.asm
- DISK.asm
- IOCONTROL.asm
- MONITOR.asm
- EXDOS.asm

With the exception of EXDOS, the corresponding object code was read only in the original system, and the R65 Emulator does not allow to write into these memory areas. Nevertheless, it is quite easy to patch these programs (in fact much easier as it was to burn new EPROMS on the original system).

If you just want to make a simple patch (just change a few locations, but not move anything!), or look at the programs, go to RASPIAN and there to R65/R65-Emulator/Assembler. The source files are there with the extension .asm, and the Assembler listings with identical line numbers with the extension txt. The emulator just reads the the object code from the .txt files at startup. This sounds crasy at first, but it is very fast and eliminates the additional step of making binary files. You can therefore just patch those files by editing them carefully. Write down somewhere what you have patched in case you later want to upgrade the software, and make a backup copy before starting to patch, so that you can go back if things don’t work anymore.

19. The usual disclaimer

Please report any problems or suggestions for improvements to r77@bluewin.ch