TRS80Model1

COLLABORATORS

	TITLE : TRS80Model1	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 8, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

TRS80Model1

1.1 TRS80Model1

```
                                    TRaSh 80
The only (?) Model 1 emulator for the Amiga
```

1.2 Requirements

REQUIREMENTS:

- o Amiga computer with Kickstart 2.0 or newer
- o A 68020+ CPU. Emulation WILL NOT WORK on a 68000 system.
- o About 512k free RAM (preferably most of it FAST RAM)
- o ASL.library is used for the File Requesters

1.3 Description

DESCRIPTION:

"TRaSh 80" is currently the only TRS80 Model 1 emulator (as far as I am aware) for the Amiga computer. At present it emulates the following configuration:-

- o Z-80 CPU (including a number of Undocumented Instructions)
- o Model 1 Keyboard at the hardware level.
- o 48k RAM
- o Load "/CMD" and execute directly from AmigaDOS
- o Write protection for the ROM area
- o Virtual Cassette Support

Coming in the next version: (if there is sufficient interest in the project)

- o Load/Save Snapshot
- o 4 80Tk Single Sided Disk Drives (MAYBE DOUBLE/DEN)
- o Serial Port Support.
- o Printer Port Support. (print to an amiga disk file)

The emulation also runs in a reasonably system friendly manner, multitasking properly with other programs. It does, however, bypass the graphics library calls in favour of directly writing to it's own screens Bitmap.
This was done primarily for speed reasons.This may or may not be compatible with third party graphics cards.

It has been tried on a PICASSO 2 and worked fine.

1.4 Why A TRS80 Model 1 Emulator?

WHY A TRS 80 Model 1 EMULATOR?

Primarily it was the first computer that I owned. I remember the thrill of getting my Level1 4k machine home for the first time early in 1978. It was miles ahead of the single board 6502 machine I had been playing with up to that point.

Secondly I feel that it is much easier for a beginner to create a working program on one of these old 8-bit machines than on one of todays super powered (by comparison) home computers. Just look at myriad compile options on any of todays C-compilers!

Thirdly a lot of the old software is really worthwile, despite the graphics limitations, and addictive to play.

Last but not least because I envied the IBM-PC crowd having a Model 1 emulator!

1.5 Running the Emulation

RUNNING THE EMULATION

Simply Click on the TRaSh-80 icon or run it from a CLI.

The gadgets are pretty self-explanatory.

1.6 Keyboard Mapping

The Keyboard of TRaSh 80 corresponds (pretty much) to the keys of the AMIGA

The <HELP> key is the TRaSh 80 <CLR> key.
The <DEL> key is the TRaSh 80 <BREAK> key.

Both Left and Right <SHIFT> keys are read as a LEFT <SHIFT> by the Z-80 routines.

One slight problem exists with selection of shifted key combinations.

If a shifted key combination is pressed and the <SHIFT> key is released FIRST, then that key's ordinary function cannot be accessed until the previously shifted combination is again selected and the <SHIFT> key released LAST.

This shouldn't cause any problems, just be aware if certain key combinations don't seem to work this is probably why.

1.7 Speed Regulation

SPEED REGULATION

Come on! This is only a pre-release Beta version.

However, The debug version runs pretty close to normal TRS-80 speed on my
030/40 A1200.

1.8 Rom Image?

Obtaining a ROM IMAGE..

We dont need no steenking Rom Images...

TRaSh80 has it's own MODIFIED inbuilt image.

1.9 Virtual Cassette Operation

VIRTUAL CASSETTE

A 64k buffer is set aside for the Virtual Cassette. Virtual Cassette files need
to be brought into the buffer before they are visible to TRaSh80, just like ↩
putting
a cassette into the recorder.

The LOADCASS gadget allows you to load a cassette file into the buffer, ↩
effectively
destroying the previous contents. Then use the CLOAD or SYSTEM commands from BASIC
to "read" the cassette.

Similarly SAVECASS saves the currently used portion of the buffer out to disk.
Using the same filename as a previously saved cassfile will over-write that file.

By "currently used" I mean that if you have CSAVE'D a program that would only be 6 ↩
k
long then only that portion of the buffer is saved.

If you CSAVE twice before saving the buffer out to AMIGADOS, then the first ↩
program
CSAVE'D will be LOST!

TRaSh80 operates the cassette by using three currently undefined Z80 instructions.

ED01 - writes the A register to the cass buffer and increments the buffer ↩
pointer.

```
  ED02 - reads the A register from the current buffer location and increments
         the buffer pointer.
  ED03 - rewinds the tape (resets the buffer pointer to 0000).

The relevent cassette routines in the inbuilt ROM image have been patched to use  ↩
   these
instructions:

  $0235 - CSIN  - read the A reg from cassette.
  $0264 - CSOUT - write the A reg to cassette.
  $0287 - CSHWR - write the cass header
  $0296 - CSHIN - find the header info on the tape.

All programs that use the ROM routines to read/write cassette data should work  ↩
   fine.

Unpredictable results will occur if you Load/Save a cassfile while the TRS-80
side is in the process of a CLOAD or CSAVE.
```

1.10 Program Information

```
TRasH 80 is written in a mixture of C and 68020 Assembler

SAS 6.58 (Yep! still being supported, you can't keep a good program team down)
and PHXass (freeware) were used respectively in development, with
CED V3.5 the editor of choice.

Gadget Handling, program loading/snapshot , keyboard emulation are in C.
The Z-80 emulator and the Screen write routines are written in assembler and
run as a seperate process on their own.

The Z-80 core uses an "instruction pipelining" system, not dissimilar to that
used in the excellent "APPLE2000" by Kevin Kralian. While tending to be rather
memory hungry, speed wise it is very efficient.

ASL.library is used for the file requesters as nothing elaborate was required
on the 1 Bitplane TRaSh 80 screen.

The Z-80 HALT instruction will currently terminate program operation.


Compiling your own version from the included source:

All you need to compile from the SAS/C editor is included in the SRC directory.

Once you have compiled the executable you must make one slight ZAP.

Load up the executable with your favourite binary editor and ZAP the longword
at file offset $18

from 00000C00  -> 00008000

This sets aside some DX.L space for the cassette buffer.
```

If you wan't to create a Model3 version just change the SCOPTIONS to include "rom3.o" instead of "rom.o"

Then zap offset $18 00000E00 -> 00008000

I haven't patched the model3 rom for virtual cassette operation yet. That will appear in the next release.

1.11 Planned Improvements

PLANNED IMPROVEMENTS

1.12 Bugs? What Bugs?

Lets call them Program limitations.

1. Stack Pointer (SP).

  Due to the way the Z-80 is emulated the STACK PTR (SP) can point to mem locations outside of the 64k address space. Consider the following fragment.

```
      31 FE FF  ; LD SP,$FFFE
      E1    ; POP HL  (SP NOW $10000)
      E1    ; POP HL  (SP NOW $10002)
```

  Unfortunately SP is now pointing outside memory that TRaSh 80 has AllocMem'ed from the EXEC memory pool. The GURU awaits.....

  The SP cannot however be explicitly loaded with a value outside of the 64k address space. Any instructions like INC SP will NOT send it out of the 64k range, only POP's and PUSH's.

  This could on the other hand be a usefull feature. Would anyone like to see support for a stack that sits outside the main 64k memory? For that matter would anyone like support for block moves into and out of the 64k address space?

2. Undocumented Instruction.

  A number of the more popular undocumented instructions are currently implemented. The next couple of versions of TRaSh 80 will have more.

3. Screen Update Routines.

  Currently all instructions that write into the Screen Memory area ($3C00-$3FFF) will cause an on-screen change, with the following exceptions:-

All Stack operations. Let's face it, code that tries to write to the screen by PUSH's and POP's must be CRAP!

4. Memory Protection.

Memory locations < $3C00 (start of the screen) will not be affected by instructions that write into system memory, with the exception of:-

All stack operations. Again only bad code would create such a situation.

This means that the ROM area ($0000-$2FFF) is protected from spurious writes by misbehaving programs.

5. Index Register Operation.

The index register READ instructions (IX+d) and (IY+d) currently do NOT wrap-around when they index past $FFFF or past $0000 in the other direction. This shouldn't cause many problems except for software that is already buggy.

Index register WRITE operations function correctly and will wrap at the 64k boundaries. Indexed writes to the ROM area are intercepted.

6. Several of the Z-80 instructions do not yet operate correctly. Notably the DDCB and FDCB opcodes. However they do not seem to be used by the ROM to run BASIC. This is being worked on as you read this.

Any bug reports or suggestions for improvements please contact me at
<RedSkull@digital-corruption.net>

1.13 Payment

This program is being released as FREEWARE.

That means you are free to COPY, USE, and ENJOY this software for personal use, provided that the distributed archive is copied intact.

I would be pleased to receive any e-mails from people who are using this program and would like to see it further developed.

However if you wish to distribute this program on CD or diskette as part of a collection, I insist on receiving a copy. Also no more than a reasonable fee for duplication must be charged.

The Z-80 emulation core of TRaSh 80 is rather zippy. If you would like to include it as part of another emulator project feel free to contact me at <RedSkull@digital-corruption.net>

If you have a commercial project that you think may benefit from the Z-80 core, likewise feel free to contact me.

Naturally users of this program accept all responsibility for it's use,suitability
for any purpose, etc, etc.......Legal Babble....

1.14 Credits

CREDITS

I owe lots of thanks to lots of people.

1.15 Debug Version

In this archive I have included a Debug version that includes
a trace mode.

Select the "trace" gadget then click on the "Singlestep" gadget.
A window will open on the Workbench screen with the next 40
instructions and information on registers/dissassembly etc.

Load/cmd in the debug version sets the trace flag internally
but doesn't highlight the gadget. If you load a cmd file in
the debug version just select the trace gadget then singlestep
to go through the program just loaded. Or just select trace
then de-select it again to run the loaded program normally.

NOTE the trace routine misses the first instruction of the
loaded cmd file, it is normally just a DI (F3) instruction in
99.999% of cases anyway.

1.16 History

HISTORY:

V0.8 - 14/08/97

  First AMINET release.

V0.9 - xx/xx/97

  Fixed bug with SBC op-codes
  Added virtual cassette support
  Fixed CHAR set , replaced][/^ with the appropriate arrows

V0.9b- 25/12/98

  Finally got back to work on the emulator!
  Fixed a couple of bugs in the Z80 core (still not 100% yet)
  Added the complete source to the archive for this release.
  Tightened up the screen routines even further.

```
  Included a Debug version for testing purposes

V0.9d- 27/12/98

  Fixed the Debug Version so that all the gadgets work properly.
  Added code to show CB,ED,DD,FD opcodes in the trace window
  (A couple of instructions still show as ILLEGAL so don't
  be too alarmed!)

  Fixed a minor bug in LDIR, LDDR instructions
  (Z flag was being reset when it shouldn't)

  Fixed some lame-ass typos in Z80.ASM  that were causing problems with
  16-bit SBC instructions (ie. SBC HL,DE)

  Due to a couple of requests I have added a version using Model3 roms.


V0.9e 01/01/99

  Fixed LDI,LDD instructions. They were sign extending outside the
  Z80 memory map for values of DE > $8000.

  Fixed CPIR,CPDR instructions. V flag was being set incorrectly if
  BC reached 0.

  implemented DD CB xx xx instructions.
  implemented FD CB xx xx instructions.

  implemented LD A,R instruction. It now pulls a random 8 bit value
  from the CIA timer which counts 50/60Hz events.

  implemeted NEG instruction. Don't know how I missed that one?

  implemented RLD,RRD instructions.

  Added support for programs which look at (Z80) $387F to see if a
  key is down. On a real Model 1/3 this location is Non-zero when
  any key is being pressed. Several programs (RSM-2D and the Tandy
  Level1.rom for example) check this location for a keypress.

V0.9f 03/01/99

  Fixed SBC HL,xx and ADC HL,xx instructions. They now correctly
  set the Z flag on a 0000 condition.

  Fixed SBC A,x (DE) wasn't setting Z properly on a zero condition.

  Fixed LD E,(IX+n) and LD E,(IY+n) instructions.

  Fixed port input instructions. (ie. IN A,(n) ) They now return
  (correctly) $FF as if no device was connected. Notably Big 5
  software looks for an "Alpha Joystick" at one of the ports.

  Z80 core seems reasonably stable now. Added a few sample programs
  into the archive.
```

1.17 Finding Software

FINDING YOURSELF SOME SOFTWARE

There a 2 main sources on the NET to the best of my knowledge.

The "TRS-80 Page" at HTTP://WWW.KJSL.COM/ includes a lot of general information as well as a link to their FTP site.

The other is IRA GOLDKLANG's TRS80 pages at HTTP://WWW.TRS-80.COM/

he has many Model 1 and also Model3 programs available for http download.

1.18 Compatibility

COMPATIBILITY

All TRS-80 programs for the Model 1 should work straight away.

Problems are likely in the following areas:

1. Assembler programs using "Undocumented Instructions".

   A number of undocumented instructions that affect the HIGH and LOW bytes of IX and IY exist in the DD and FD ranges.

   TRaSh 80 implements some of these instruction, however the next release will implement more still.

2. Programs trying to "LPRINT". Since printer support is not yet active, programs trying to output to the printer port will simply hang.

3. Likewise the Serial Port.

4. " " the Disk Drives.