feilipu Update README.md 6f846e5 · last year History

Name	Name	Last commit date
..		
Makefile	hexload - relocate & polish	6 years ago
README.md	Update README.md	last year
bin2bas.py	hexload - relocate & polish	6 years ago
helloworld.hex	hexload example HEX files	6 years ago
hexload.asm	hexload - relocate & polish	6 years ago
hexload.bas	BASIC PROGRAMS - use Windows /r/n rather than Basic /r	3 years ago
hexload.lst	APU - larson scanner patch	4 years ago
hexload_map.asm	hexload - relocate & polish	6 years ago
password.hex	hexload example HEX files	6 years ago
slowprint.py	hexload - relocate & polish	6 years ago

README.md

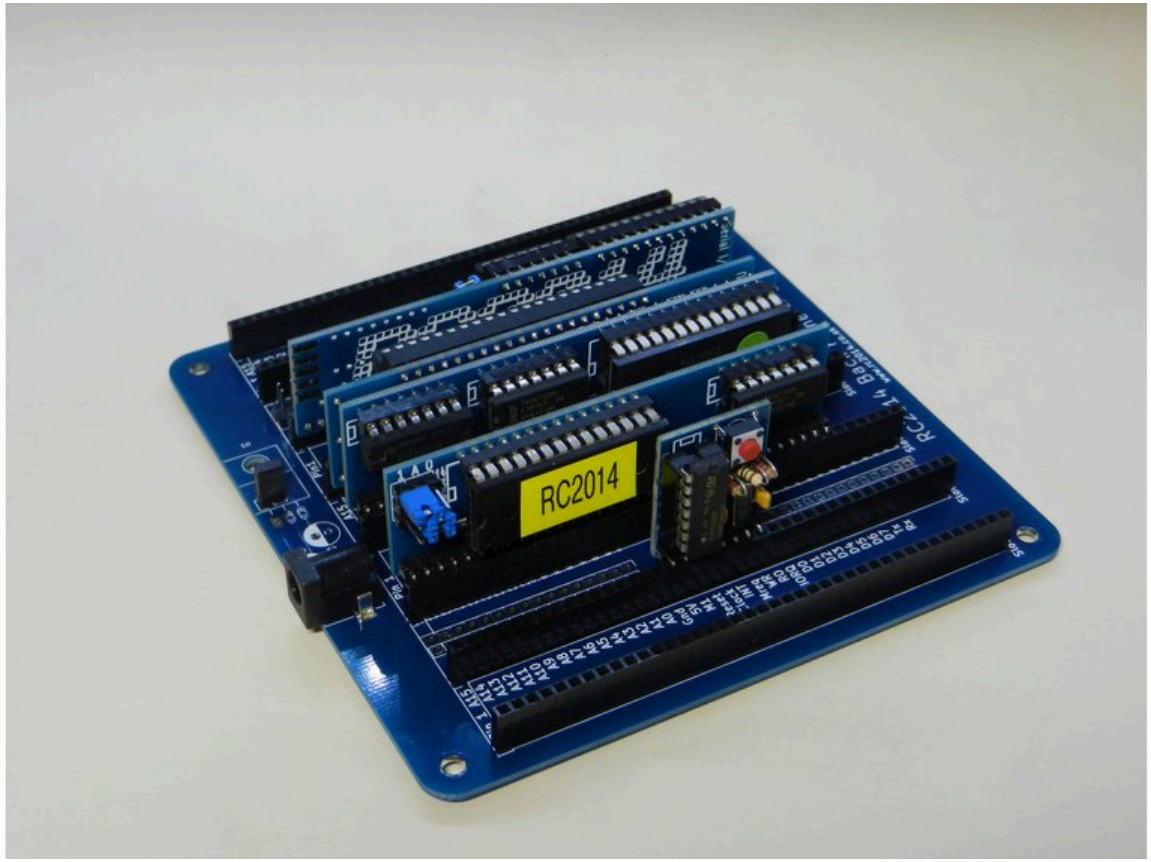
Hexload

Hexload is a MS Basic program, that uploads an assembled binary program to support uploading other Intel HEX formatted programs into the memory of the RC2014.

C and Assembly file upload, no EEPROM programmer required

Just built a Classic or Mini RC2014, and don't yet have an EEPROM burner?

Do you want to do Assembly or C language programming, and then load and run these programs on your RC2014?



RC2014 Classic

RC2014 Mini

`hexload` is the way you can upload and run either assembly or C compiled code on your RC2014.

- Connect to the RC2014 with a serial interface
- Press the Reset button Boot the machine to restart the RC2014
- Select Cold start and set Memory Top? to 35071.
- Set your UART interface program (like minicom or Putty) so that it waits at least 10 msec between each byte and copy-paste the file `hexload.bas` or use `slowprint.py` as directed below.

You should see one of these two messages.

```
80 SBC By Grant Searle

Memory top? 35071
Z80 BASIC Ver 4.7b
Copyright (C) 1978 by Microsoft
1916 Bytes free
Ok
```

or

```
Z80 SBC By Grant Searle

Cold or warm start (C or W)? C
```

```
Memory top? 35071
Z80 BASIC Ver 4.7b
Copyright (C) 1978 by Microsoft
1916 Bytes free
Ok
```

Once you have pasted all of the `hexload.bas` file into your terminal, the following message should appear (following lots of Basic commands):

```
...
...
run
Loading Data
Start Address: 8900
End Address:   89D7
USR(0) -> HexLoad
HEX LOADER by Filippo Bergamasco & feilipu for z88dk

:
```

At that point, copy-paste the desired Intel HEX file, i.e. the Assembly or C language compiled program in HEX format, into the terminal.

An example `helloworld.hex` is provided to check that this process is working. You should see the "RC2014" hello world as below.

```
...
...
9550 data 999
9999 END
run
Loading Data
Start Address: 8900
End Address:   89D7
USR(0) -> HexLoad
HEX LOADER by Filippo Bergamasco & feilipu for z88dk

:################################################################################
################################################################################
################################################################################
#####

Done

 0
USR(0) -> 0x9000, z88dk default
```

```
Hello World R
Hello World C
Hello World 2
Hello World 0
Hello World 1
Hello World 4

 0
Ok
```

Program upload from a Linux system

On a Linux system assuming an USB-to-serial converter attached as `/dev/ttyUSB0` the following commands can be used:

```
$ cd hexload
$ python slowprint.py < hexload.bas > /dev/ttyUSB0
$ cat < helloworld.hex > /dev/ttyUSB0
```

How does hexload work?

from `0x8000` through to `0x8183` to manage itself, and it usually leaves the bytes from there through to `0xFFFF` free for BASIC programs.

The MS Basic language has a special command `USR(x)` which can cause Basic to jump to a user program, when the address of that program is written to a special location. For the default ROM this location is the two bytes from `0x8049`. If we write the address of our program to that location, then when we call the Basic `USR(0)` program, then the RC2014 will begin executing our program.

So with `hexload` we do this twice. Firstly, to run the actual Intel HEX upload program from address `0x8900` after we upload it, and then again to run our own program to the location from `0x9000` after it is uploaded.

We have to let MS Basic know not to use this space for its programs, hence when we cold start the RC2014 we ask it to keep the top memory address to 35071, or in hexadecimal `0x88FF`, just below the intended location for our `hexload` program.

Before we can run the `hexload` program we have to upload and run a Basic program to `poke` the program into the right location, from `0x8900` to `0x89D7`, set the starting address for `USR(0)` correctly, and then set it running. And this program has to fit within the 1916 bytes available to Basic.

Once we have the `hexload` colon prompt then we can use it to upload, and also start, our own program with the origin of `0x9000`.

```
Loading Data
Start Address: 8900
End Address:   89D7
USR(0) -> HexLoad
HEX LOADER by Filippo Bergamasco & feilipu for z88dk

:
```

An additional test program provided by Z88DK is the `password.hex` program also in this directory, which demonstrates the terminal editing capabilities of the Z88DK standard library. It is available in the examples directory of Z88dk.

How to prepare C Programs?

The easiest way to compile C programs for the RC2014 is using the Z88DK.

The simple command line (to get started) looks like this below.

```
zcc +rc2014 -subtype=basic -clib=sdcc_iy helloworld.c -o helloworld -create-app
```

Whilst there are many additional options which can add information about the process, the above provides the intel hex or `ihx` information that can be uploaded to the RC2014 and run.

The `+rc2014` advises that the machine is the RC2014, and the `-subtype=basic` advises that the program should be compiled with `0x9000` as its origin, and that it should use the serial drivers included in the standard MS Basic ROM. The `-clib=sdcc_iy` sets sdcc as the compiler, reserving register pair `iy` for the library usage. There are many options available for the compile process, but initially the above incantation provides a good result.

The Z88DK examples include some classic games including StarTrek, Sudoku, Eliza, and Chess, all written in C, that can be compiled and run on the RC2014.

The Z88DK source code has a (actually two) full standard C libraries, and two alternative compilers (sccz80 and patched sdcc) to chose from. It supports over 50 machine types (including the RC2014), and is very actively maintained.

Full instructions to use the Z88DK are available from the Wiki and support is available on the forum.

Normal usage

Once a program has been uploaded it will be automatically started by the `hexload` program. It can be run as often as needed by typing `print usr(0)` in lowercase or uppercase `PRINT USR(0)` . Also the short form command `? usr(0)` or `? USR(0)` works equally well.

To upload a new program version type `run` or `RUN` and `hexload` will restart, and will wait for your new Intel Hex program to be uploaded from the serial port.

When the RC2014 is warm restarted (select `W` when the RESET button is pressed), all memory contents are preserved. So this allows the user to restart the program without reloading any information.

In this way, if your program crashes you can simply RESET, select `W` for warm boot, and then `RUN` the still resident `hexload` program to download a revised version of your program to the RC2014.

Advanced usage

Integers can be passed into your programs by the parameter in the `USR(x)` command, and can be obtained into the DE registers using the `DEINT` routine.

When the program returns, it will show the `ABPASS` return value (in the example it is 0), and then the `OK` from Basic. Once this happens, the program can be simply restarted by `PRINT USR(x)`, where `x` is the value passed via the `DEINT` routine into the program.

This means that the program can be started as often as needed, and can be (for example) treated as a subroutine from a Basic program you load to replace the `hexload.bas` program, after uploading your subroutine program. This can be useful where you need to write a driver for hardware, or need an optimised calculation written in assembly language.

Your assembly or C program can reference the MS Basic `DEINT` and `ABPASS` routines at these addresses.

```
DEINT   $0A07   ; Get integer value into DE
ACPASS  $117C   ; Return integer AC
ABPASS  $117D   ; Return integer AB
```

Note that your program should jump to `ABPASS` when it exits, as this routine provides the final `RET` instruction.