PCSX-Redux

None

None

None

Table of contents

1. Home

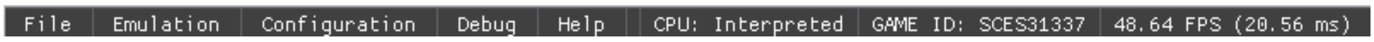
Welcome to the Pcsx-redux emulator documentation.

You can get the emulator for various platforms here : https://github.com/grumpycoders/pcsx-redux#where

You can find a one page version of this site here : One page version

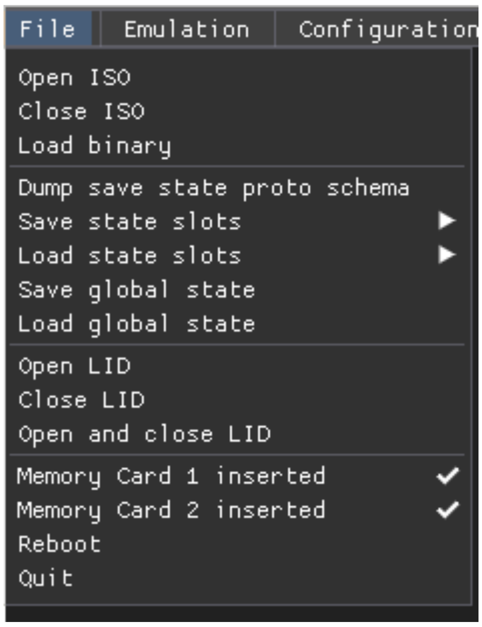
2. Pcsx-redux menus

The menu bar holds some informations :



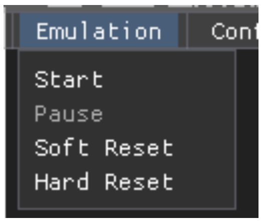
- CPU mode
- Game ID
- ImGui FPS counter (not psx internal fps)

2.1 File



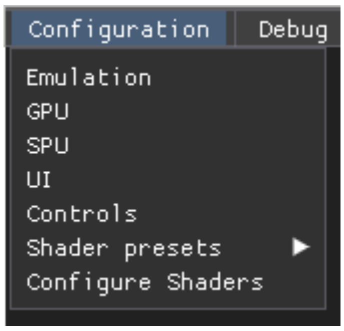
- Open ISO
- Close ISO
- Load Binary
- Dump save state proto schema
- Save state slots
- Load state slots
- Save global state
- Load global state
- Open Lid : Simulate open lid
- Close Lid : Simulate closed lid
- Open and Close Lid : Simulate open then closed lid
- MC1 inserted
- MC2 inserted
- Reboot : Restart emulator
- Quit

2.2 Emulation



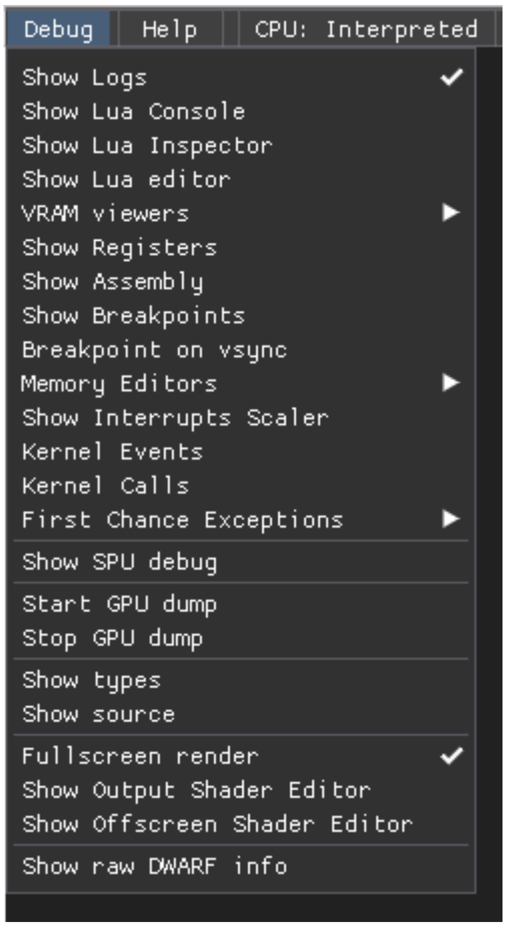
- Start : Start execution
- Pause : Pause execution
- Soft reset : Calls Redux's CPU reset function, which jumps to the BIOS entrypoint (0xBFC00000), resets some COP0 registers and the general purpose registers, and resets some IO. Does not clear vram.
- Hard reset : Similar to a reboot of the PSX.

2.3 Configuration



- Emulation : Emulation settings
- GPU : graphical processor settings
- SPU : Sound processor settings
- UI : Change interface settings
- Controls : Edit KB/Pad controls
- Shader presets : Apply a shader preset
- Configure shaders : show shader editor

2.4 Debug



2.5 Help

- Show Imgui demo
- About

3. Compiling Pcsx-redux

3.1 Getting the sources

The only location for the source is on github. Clone recursively, as the project uses submodules:

`git clone https://github.com/grumpycoders/pcsx-redux.git --recursive`.

3.2 Windows

Install Visual Studio 2019 Community Edition.
Open the file `vsprojects\pcsx-redux.sln`, select `pcsx-redux -> pcsx-redux`, right click, `Set as Startup Project`, and hit `F7` to build.
The project follows the open-and-build paradigm with no extra step, so no specific dependency ought to be needed, as NuGet will take care of downloading them automatically for you on the first build.

Note: If you get an error saying `hresult e_fail has been returned from a call to a com component`, you might need to delete the .suo file in vsproject/vs, restart Visual Studio and retry.

Openbios

Using Visual Studio Code, one can use the task "make_openbios" to compile: CTRL-P then `task make_openbios` to compile.

3.3 Linux

3.3.1 Compiling with Docker

Run `./dockermake.sh`. You need docker for this to work.

```
# Debian derivative; Ubuntu, Mint...
sudo apt install docker
# Arch derivative; Manjaro...
sudo pacman -S docker
```

You will also need a few libraries on your system for this to work. Check the Dockerfile for a list of library packages to install.

3.3.2 Compiling with make

- Debian derivatives (for full emulator compilation):

```
sudo apt-get install -y build-essential git make pkg-config clang g++ g++-mipsel-linux-gnu cpp-mipsel-linux-gnu binutils-mipsel-linux-gnu libfreetype-dev libavcodec-dev libavformat-dev libavutil-dev libglfw3-dev libswresample-dev libuv1-dev zlib1g-dev
```

- Arch derivatives :

```
sudo pacman -S clang git make pkg-config ffmpeg libuv zlib glfw-x11 curl xorg-server-xvfb
```

You can then just enter the 'pcsx-redux' directory and compile without using docker with `make`.

If you have a different mips compiler, you'll need to override some variables, such as `PREFIX=mipsel-none-elf FORMAT=elf32-littlemips`.

Openbios

Building OpenBIOS on Linux can be done with docker : `./dockermake.sh openbios`, or using `make`, with the `g++-mipsel-linux-gnu` package installed ; `make openbios`.

3.3.3 MacOS

You need MacOS Catalina with the latest XCode to build, as well as a few homebrew packages.
Run the brew installation script to get all the necessary dependencies.

Run `make` to build.

Compiling OpenBIOS will require a mips compiler, that you can generate using the following commands:

Openbios

```
brew install ./tools/macos-mips/mipsel-none-elf-binutils.rb
brew install ./tools/macos-mips/mipsel-none-elf-gcc.rb
```

Then, you can compile OpenBIOS using `make -C ./src/mips/openbios`.

3.4 Compiling PSX code

If you're only interested in compiling psx code, you can clone the pcsx-redux repo;

```
git clone https://github.com/grumpycoders/pcsx-redux.git --recursive
```

then install a mips toolchain and get the converted PsyQ libraries in the `pcsx-redux/src/mips/psyq/` folder as per these instructions.

You can also find the pre-compiled converted Psyq libraries online.

3.4.1 Getting the toolchain on Windows

Download the MIPS toolchain here : https://static.grumpycoder.net/pixel/mips/g++-mipsel-none-elf-10.3.0.zip
and add the `bin` folder to your $PATH.
You can test it's working by launching a command prompt and typing `mipsel-none-elf-gcc.exe --version`. If you get a message like `mipsel-none-gnu-gcc (GCC) 10.3.0`, then it's working !

3.4.2 Getting the toolchain on GNU/Linux

Debian derivative; Ubuntu, Mint...

```
sudo apt install g++-mipsel-linux-gnu cpp-mipsel-linux-gnu binutils-mipsel-linux-gnu
```

Arch derivative; Manjaro...

The mipsel environment can be installed from AUR : cross-mipsel-linux-gnu-binutils and cross-mipsel-linux-gnu-gcc using your AURhelper of choice:

```
trizen -S cross-mipsel-linux-gnu-binutils cross-mipsel-linux-gnu-gcc
```

4. Command Line Flags

You can launch `pcsx-redux` with the following command line parameters:

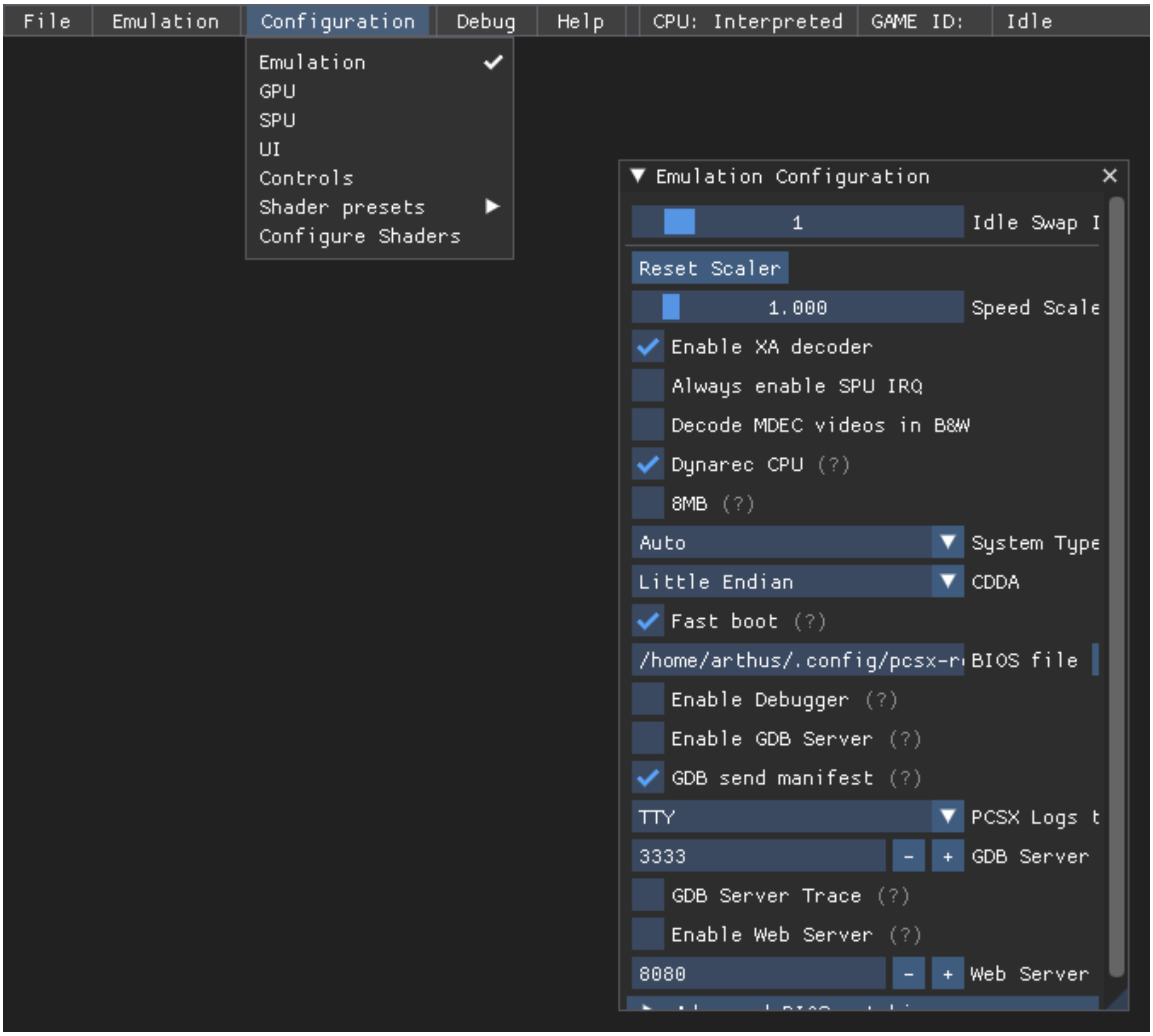
The parsing code doesn't care about the number of dashes in the parameter's flag, so '-' can be used as well as '--', or any number of dashes.

Flag	Meaning
`-run`	Begin execution on startup.
`-stdout`	Redirect log output to stdout.
`-logfile`	Specify a file to log output to.
`-bios`	Specify a BIOS file.
`-testmode`	Interpret internal API's `pcsx_exit()` command and close the emulator.
`-loadexe`	Load a PSX exe.
`-iso`	Load a PSX disk image (iso, bin/cue).
`-memcard1`	Specify a memory card file to use as memory card slot 1.
`-memcard2`	Specify a memory card file to use as memory card slot 2.
`-pcdrv`	Enable the pcdrv: device interface. (Access PC filesystem through SIO)
`-pcdrvbase`	Specify base directory for pcdrv

5. GDB server

The GDB server allows you to set breakpoints and control your PSX program's execution from your gdb compatible IDE.

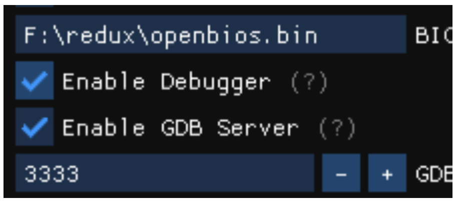
5.1 Enabling the GDB server



1 Idle Swap I
1.000 Speed Scale
✓ Enable XA decoder
Always enable SPU IRQ
Decode MDEC videos in B&W
✓ Dynarec CPU (?)
8MB (?)
Auto ▼ System Type
Little Endian ▼ CDDA
✓ Fast boot (?)
/home/arthus/.config/pcsx-r BIOS file
Enable Debugger (?)
Enable GDB Server (?)
✓ GDB send manifest (?)
TTY ▼ PCSX Logs t
3333 - + GDB Server
GDB Server Trace (?)
Enable Web Server (?)
8080 - + Web Server

In pcsx-redux: `Configuration > Emulation > Enable GDB server`.

Make sure the debugger is also enabled.



F:\redux\openbios.bin BIO

✓ Enable Debugger (?)

✓ Enable GDB Server (?)

3333 GDB

5.2 GDB setup

You need `gdb-multiarch` on your system :

5.2.1 Windows

Download a pre-compiled version from here : https://static.grumpycoder.net/pixel/gdb-multiarch-windows/

5.2.2 GNU/Linux

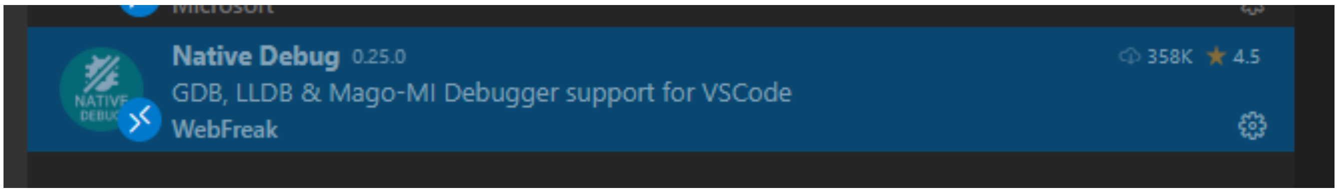
Install via your package manager :

```
# Debian derivative; Ubuntu, Mint...
sudo apt install gdb-multiarch
# Arch derivative; Manjaro
# 'gdb-multiarch' is available in aur : https://aur.archlinux.org/packages/gdb-multiarch/
sudo trizen -S gdb-multiarch
```

5.3 IDE setup

5.3.1 MS VScode

- Install the `Native debug` extension : https://marketplace.visualstudio.com/items?itemName=webfreak.debug



- Adapt your `launch.json` file to your environment :
  A sample `lanuch.json` file is available here.
  This should go in `your-project/.vscode/`.

You need to adapt the values of `"target"`, `"gdbpath"` and `"autorun"` according to your system :

target

This is the path to your `.elf` executable :

```
      "target": "HelloWorld.elf",
```

https://github.com/NDR008/VSCodePSX/blob/d70658b5ad420685367de4f3c18b89d72535631e/get_started/.vscode/launch.json#L9

gdbpath

This the path to the `gdb-multiarch` executable:

```
      "gdbpath": "/usr/bin/gdb-multiarch",
```

https://github.com/NDR008/VSCodePSX/blob/d70658b5ad420685367de4f3c18b89d72535631e/get_started/.vscode/launch.json#L10

autorun

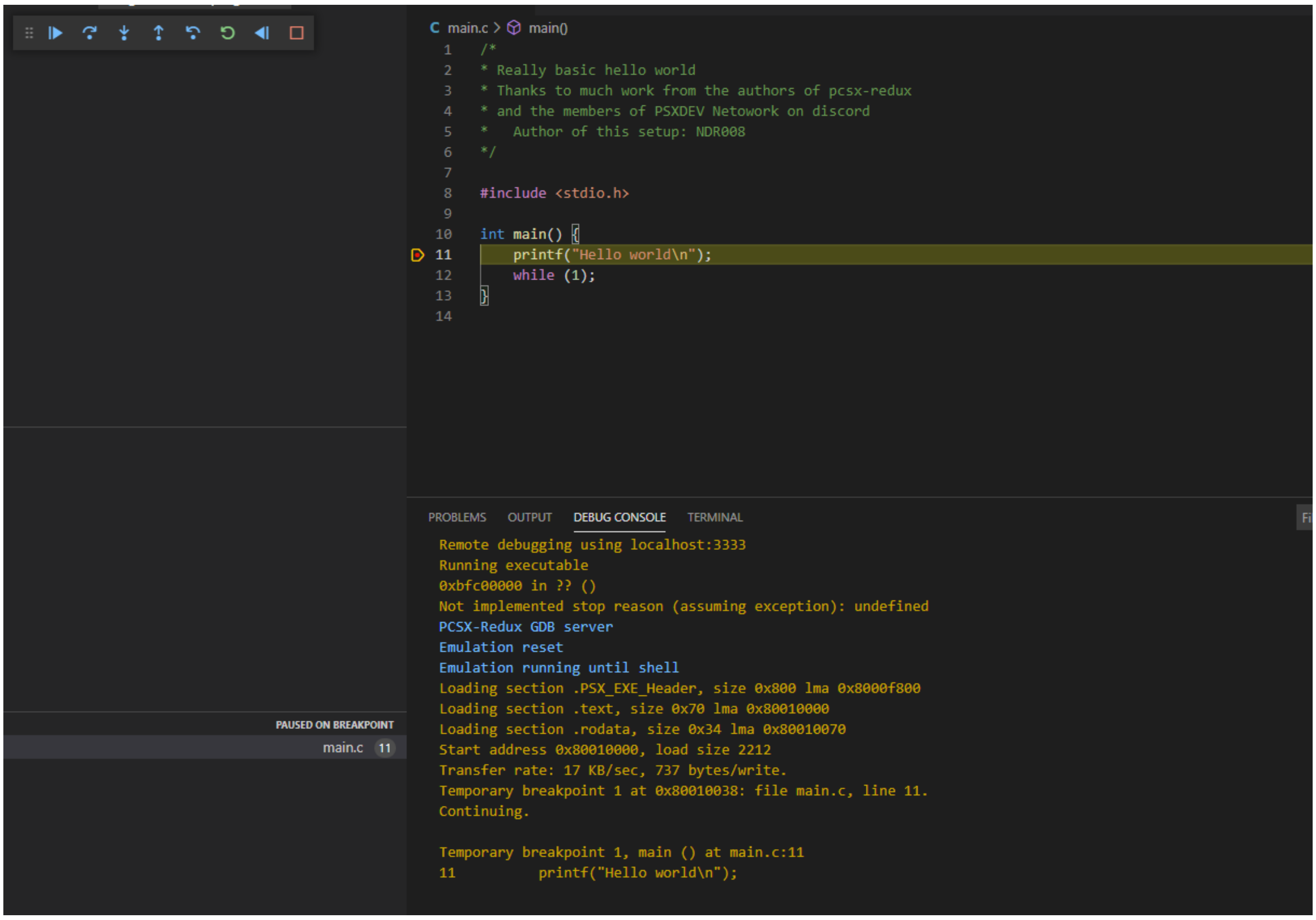
```
      "autorun": [
       "target remote localhost:3333",
       [...]
       "load HelloWorld.elf",
```

Make sure that `"load your-file.elf"` corresponds to the `"target"` value.

https://github.com/NDR008/VSCodePSX/blob/d70658b5ad420685367de4f3c18b89d72535631e/get_started/.vscode/launch.json#L15

By default, using `localhost` should work, but if encountering trouble, try using your computer's local IP (e.g; 192.168.x.x, 10.0.x.x, etc.)

https://github.com/NDR008/VSCodePSX/blob/d70658b5ad420685367de4f3c18b89d72535631e/get_started/.vscode/launch.json#L13



```c
/*
* Really basic hello world
* Thanks to much work from the authors of pcsx-redux
* and the members of PSXDEV Netowork on discord
*   Author of this setup: NDR008
*/

#include <stdio.h>

int main() {
    printf("Hello world\n");
    while (1);
}
```

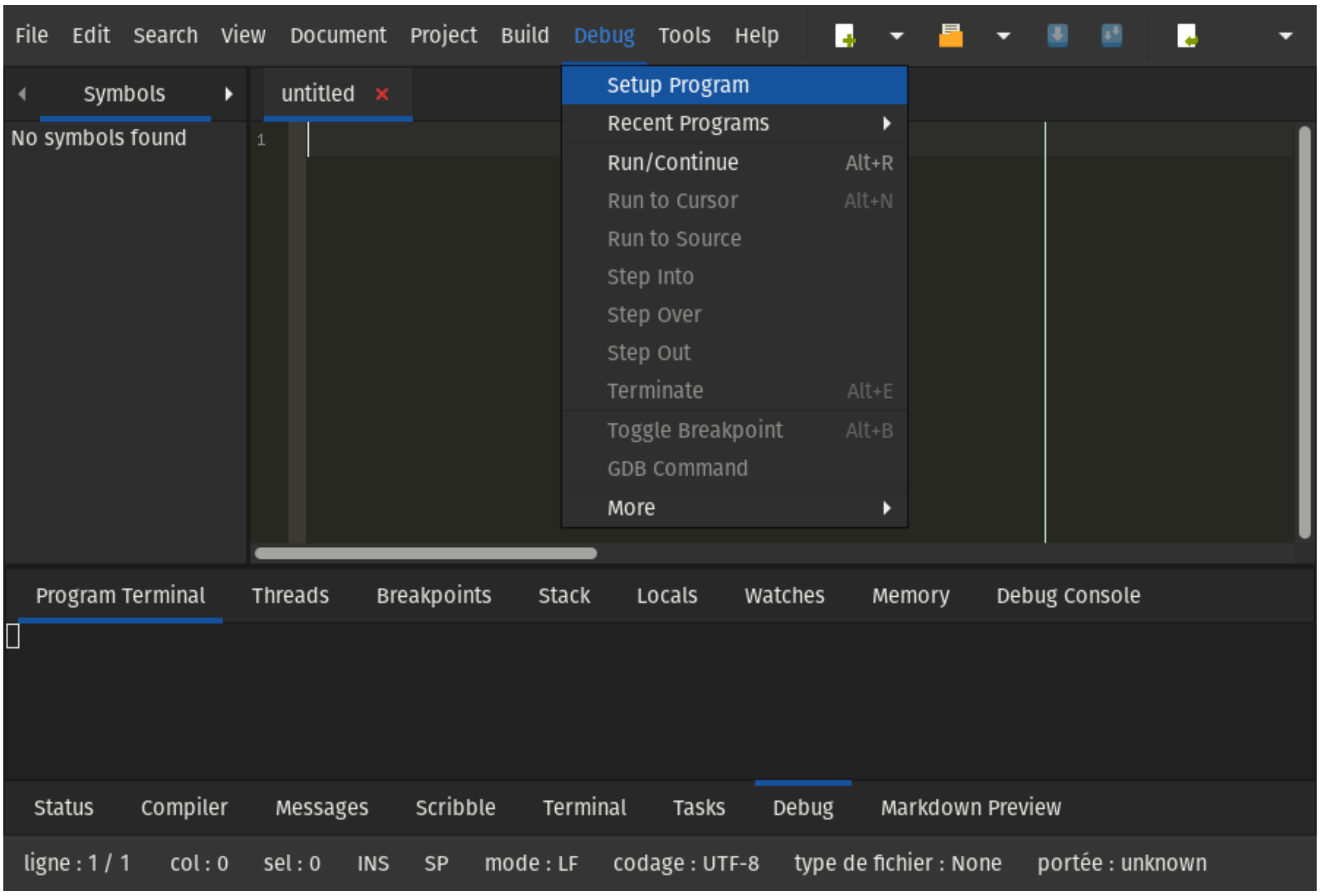
```
Remote debugging using localhost:3333
Running executable
0xbfc00000 in ?? ()
Not implemented stop reason (assuming exception): undefined
PCSX-Redux GDB server
Emulation reset
Emulation running until shell
Loading section .PSX_EXE_Header, size 0x800 lma 0x8000f800
Loading section .text, size 0x70 lma 0x80010000
Loading section .rodata, size 0x34 lma 0x80010070
Start address 0x80010000, load size 2212
Transfer rate: 17 KB/sec, 737 bytes/write.
Temporary breakpoint 1 at 0x80010038: file main.c, line 11.
Continuing.

Temporary breakpoint 1, main () at main.c:11
11          printf("Hello world\n");
```

5.3.2 Geany

Make sure you installed the official plugins and enable the `Scope debugger`.

To enable the plugin, open Geany, go to `Tools > Plugin manager` and enable `Scope Debugger`.

You can find the debugging facilities in the `Debug` menu ;



You can find the plugin's documentation here : https://plugins.geany.org/scope.html

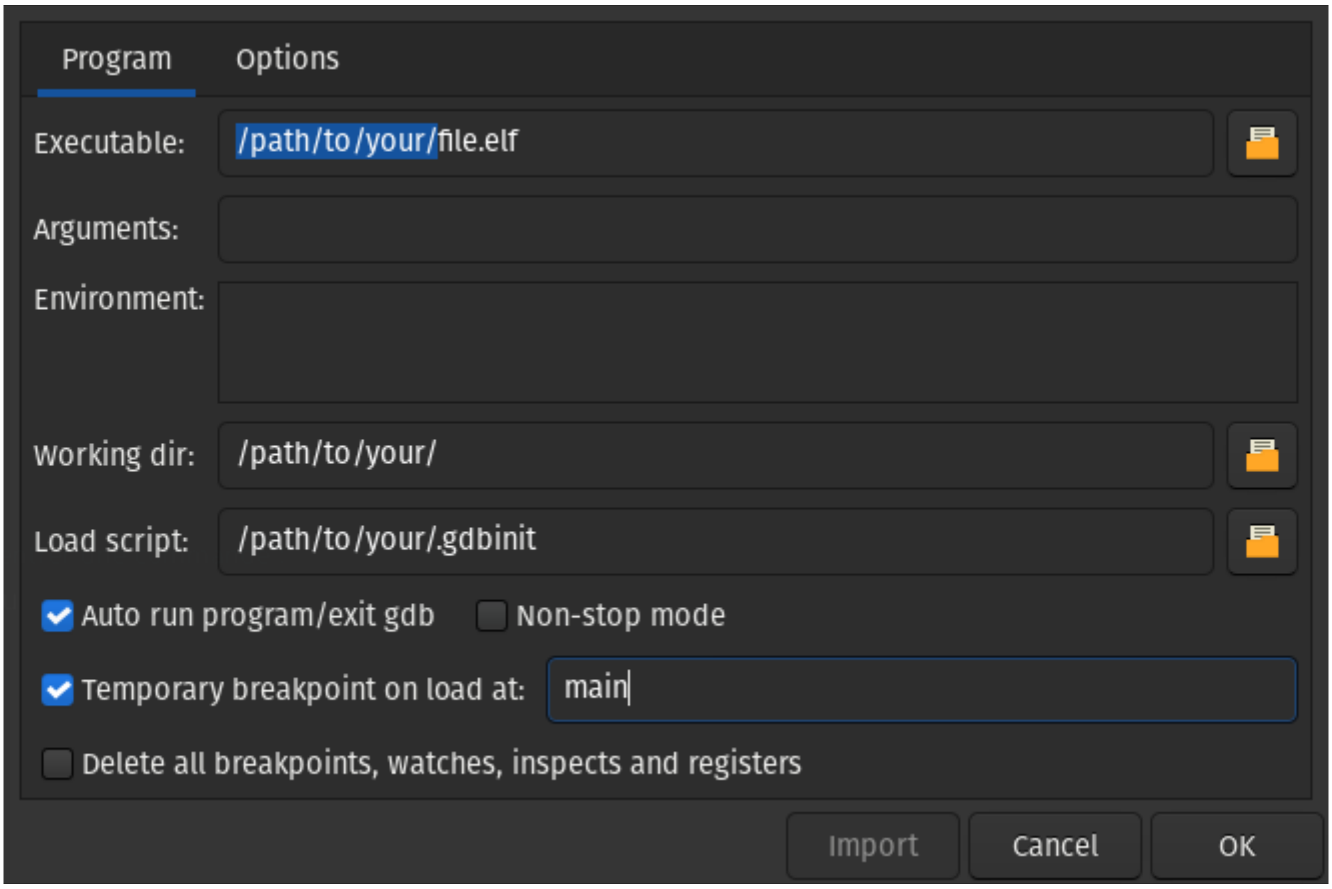
.gdbinit

Create a `.gdbinit` file at the root of your project with the following content, adapting the path to your `elf` file and the gdb server's ip.

```
target remote localhost:3333
symbol-file load /path/to/your/executable.elf
monitor reset shellhalt
load /path/to/your/executable.elf
```

Plugin configuration

In Geany : `Debug > Setup Program` :



Executable: /path/to/your/file.elf

Arguments:

Environment:

Working dir: /path/to/your/

Load script: /path/to/your/.gdbinit

[✓] Auto run program/exit gdb

[ ] Non-stop mode

[✓] Temporary breakpoint on load at: main

[ ] Delete all breakpoints, watches, inspects and registers

5.4 Beginning Debugging

Launch `pcsx-redux`, then run the debugger from your IDE. It should load the `elf` file, and execute until the next breakpoint.

5.4.1 Starting debugging in Geany

Your browser does not support the video tag.

Source : https://archive.org/details/pcsx_redux_geany_gdb

5.5 Additional tools

https://github.com/cyrus-and/gdb-dashboard/

6. Mips API

6.1 Description

Pcsx-redux has a special API that mips binaries can use :

```
static __inline__ void pcsx_putc(int c) { *((volatile char* const)0x1f802080) = c; }
static __inline__ void pcsx_debugbreak() { *((volatile char* const)0x1f802081) = 0; }
static __inline__ void pcsx_exit(int code) { *((volatile int16_t* const)0x1f802082) =
code; }
static __inline__ void pcsx_message(const char* msg) { *((volatile char*
const)0x1f802084) = msg; }

static __inline__ int pcsx_present() { return *((volatile uint32_t* const)0x1f802080)
== 0x58534350; }
```

Source : https://github.com/grumpycoders/pcsx-redux/blob/main/src/mips/common/hardware/pcsxhw.h#L31-L36

The API needs DEV8/EXP2 (1f802000 to 1f80207f), which holds the hardware register for the bios POST status, to be expanded to 1f8020ff.
Thus the need to use a custom `crt0.s` if you plan on running your code on real hardware.
The default file provided with the Nugget+PsyQ development environment does that:

```
_start:
    lw     $t2, SBUS_DEV8_CTRL
    lui    $t0, 8
    lui    $t1, 1
_check_dev8:
    bge    $t2, $t0, _store_dev8
    nop
    b      _check_dev8
    add    $t2, $t1
_store_dev8:
    sw     $t2, SBUS_DEV8_CTRL
```

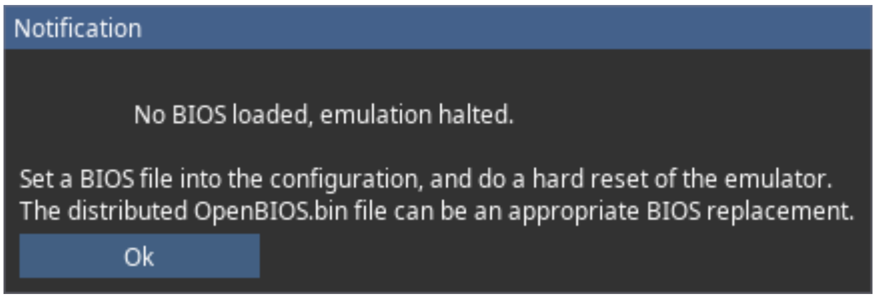
Source : https://github.com/grumpycoders/pcsx-redux/blob/main/src/mips/common/crt0/crt0.s#L36-L46

6.2 Functions

The following functions are available :

Function	Usage
`pcsx_putc(int c)`	Print ASCII character with code `c` to console/stdout.
`pcsx_debugbreak()`	Break execution (Pause emulation).
`pcsx_exit(int code)`	Exit emulator and forward `code` as exit code.
`pcsx_message(const char* msg)`	Create a UI dialog displaying `msg`
`pcsx_present()`	Returns 1 if code is running in pcsx-redux

Example of a UI dialog created with `pcsx_message()` :



Notification

No BIOS loaded, emulation halted.

Set a BIOS file into the configuration, and do a hard reset of the emulator.
The distributed OpenBIOS.bin file can be an appropriate BIOS replacement.

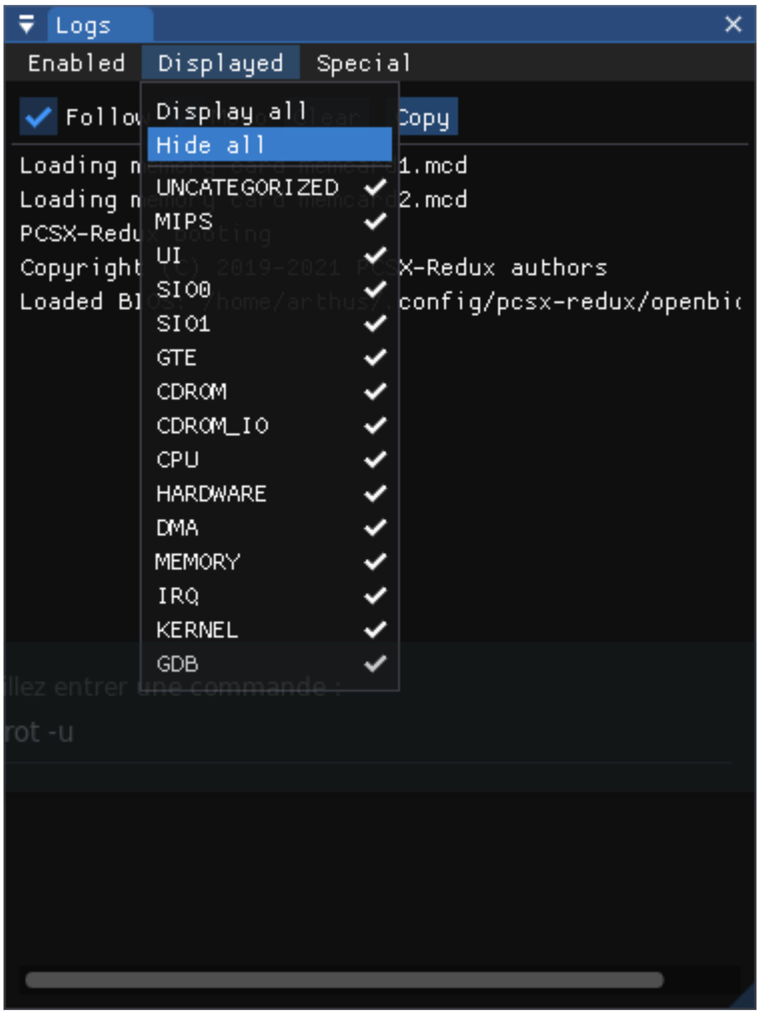
7. Dumping a CPU trace to a file

7.1 Setup

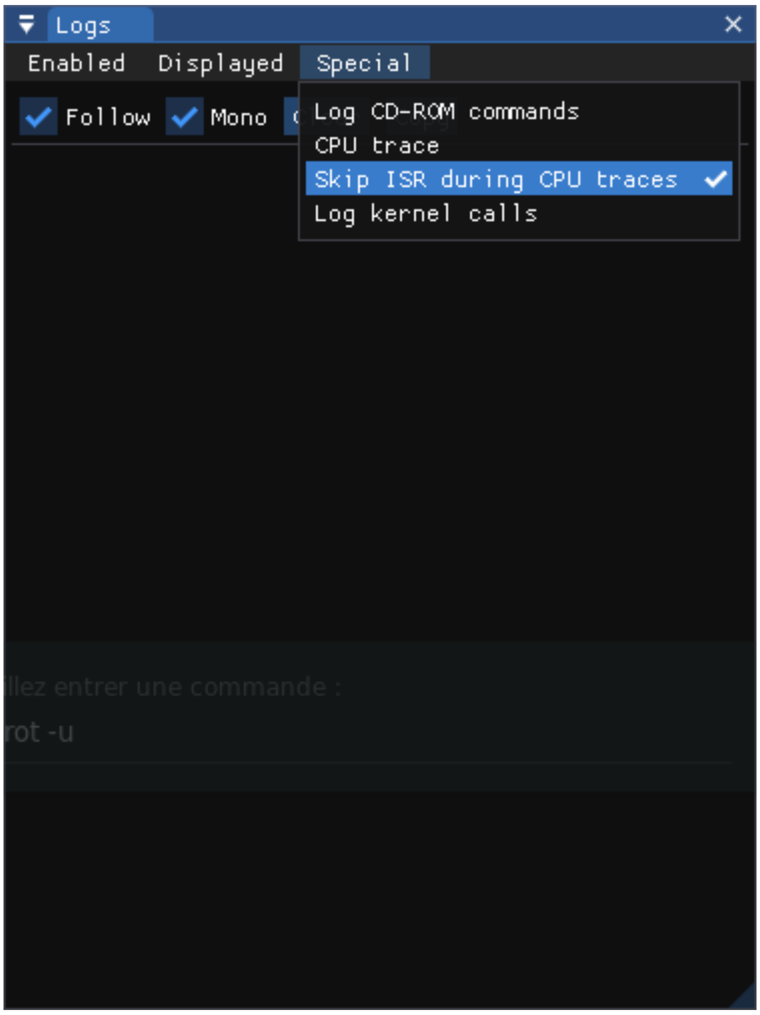
In pcsx-redux, make sure `Debug > Show logs` is enabled.

In the 'Logs' window, hide all logs : `Displayed > Hide all`

To avoid unnecessary noise, you can also skip ISR during CPU traces : `Special > Skip ISR during CPU traces`



```
Loading memory card memcard1.mcd
Loading memory card memcard2.mcd
PCSX-Redux booting
Copyright (C) 2019-2021 PCSX-Redux authors
Loaded BIOS: /home/arthus/.config/pcsx-redux/openbi
```



7.2 Begin dump

To dump the CPU traces, launch pcsx-redux with the following command :

```
pcsx-redux -stdout -logfile log.txt
# Alternatively, you can use -stdout on its own and pipe the output to a file.
pcsx-redux -stdout >> log.txt
```

You can use additional flags to launch an executable/disk image in one go, e.g :

```
pcsx-redux -stdout -logfile tst.log -iso image.cue -run
```

7.3 Source

https://discord.com/channels/642647820683444236/663664210525290507/882608398993063997

8. Web server

A web server can be activated. This allows the use of a REST api to access various features.

8.1 Activation

You can activate the web server by going to `Configuration > Emulation > Enable Web Server`

8.2 REST API

By default, the server listens for incoming connection on `localhost:8080`.

URL	Function
http://localhost:8080/api/v1/gpu/vram/raw	Dump VRAM
http://localhost:8080/api/v1/cpu/ram/raw	Dump RAM

9. Openbios

Openbios is, as it's name imply, an open-source alternative to a retail PSX bios that can be non-trivial to dump.

9.1 Purposes of Openbios

- Educational
- Ease of distribution
- Automated testing

See this page for more details.

9.2 Building

It is compiled together with `pcsx-redux` or can be compiled on it's own.

See the corresponding sections in Compiling for instructions.

The result of the compilation should be a file called `openbios.elf` that contains all useful debugging symbols,
and a file called `openbios.bin` which can be used in emulators or even burned to a chip and placed on a retail console.

9.3 Status

This subproject is still under construction, but is fairly functional and usable. OpenBIOS does almost all the same things as the retail BIOS does when booting, and implements most of its features.
Many games are booting and working properly with this code.
It can be used in emulators or on the real console, either while replacing the rom chip, or by using the "cart" build and programming the flash chip of a cheat cart with the result.

9.4 Organization

The BIOS is split in two major parts: the low level code for the bios itself, and the shell, which is the binary that's being loaded into memory at boot time by the bios, to display the SONY sound and logo, and has a small utility menu for playing audio discs, or shuffling around memory cards.

While the first part is the main one that's being targeted here, the second one isn't currently present. This may change in the future, but this isn't currently the focus of this project.

The original code was most likely chunked into several sub-projects, that were all linked together like a giant patchwork. This approach is less readable, and for this reason, we're not going to do this.
However this will result in the ROM/RAM split to be less obvious, and slower at times than the original. Tuning of the hot functions is eventually required.

9.5 Technicalities

The code has been rewritten based off the reverse engineering of a dump of the BIOS of an american **SCPH-7001** machine. *MD5sum: 1e68c231d0896b7eadcad1d7d8e76129*

The ghidra database for it is currently being hosted on a server, alongside a few other pieces of software being reversed. Contact one of the authors if you want access.

9.6 Commentary

The retail PlayStation BIOS code is a constellation of bugs and bad design.
The fact that the retail console boots at all is nothing short of a miracle. Half of the provided libc in the A0 table is buggy.
The BIOS code is barely able to initialize the CD-Rom, and read the game's binary off of it to boot it; anything beyond that will be crippled with bugs.
And this only is viable if you respect a very strict method to create your CD-Rom. The memory card and gamepad code is a steaming-hot heap of human excrement.
The provided GPU stubs are inefficient at best.

The only sane thing that any software running on the PlayStation ought to do is to immediately disable interrupts, grab the function pointer located at *0x00000310* for

`FlushCache`,
in order put it inside a wrapper that disables interrupts before calling it, and then trash the whole memory to install its own code.
The only reason `FlushCache` is required from the retail code is because since the function will unplug the main memory bus off the CPU in order to work, it HAS to run from the *0xbfc* memory map, which will still be connected.
Anything else from the retail code is virtually useless, and shouldn't be relied upon.

9.7 Legality

Disclaimer: the author is not a lawyer, and the following statement hasn't been reviewed by a professional of the law, so the rest of this document cannot be taken as legal advice.

As explained above, this code has been written using disassembly and reverse engineering of a retail bios the author dumped from a second hand console. The same exact methodology was employed by Connectix for their PS1 bios. The conclusion of their lawsuit, and that of Sega v. Accolade seems to indicate that this project here follows and is impacted by the same doctrine.