bare68k Documentation

Release 0.0.0

Christian Vogelgsang

Jul 30, 2017

Contents:

1 Tutorial 3

2 Change Log 5
2.1 0.1.1 (2017-07-30) 5
2.2 0.1.0 (2017-07-26) 5

3 Features 7

4 Installation 9

5 Quick Start 11

6 Indices and tables 13

bare68k allows you to write **m68k system emulators** in Python 2 or 3. It consists of a **CPU emulation** for 68000/68020/68EC020 provided by the Musashi engine written in native C. A **memory map** with RAM, ROM, special function is added and you can start the CPU emulation of your system. You can intercept the running code with a trap mechanism and use powerful diagnose functions,

written by Christian Vogelgsang <chris@vogelgsang.org>

CHAPTER 1

Tutorial

This section gives you a short tutorial on how to use the *bare68k* package.

CHAPTER 2

Change Log

0.1.1 (2017-07-30)

- Added support for Windows build

0.1.0 (2017-07-26)

- First public release

CHAPTER 3

Features

- all emulation code written in C for fast speed
- runs on Python 2.7 and Python 3.5
- emulates CPU 68000, 68020, and 68EC020
- use a 24 or 32 bit memory map
- define memory regions for RAM and ROM with page granularity (64k)
- special memory regions that call your code for each read/write operation
- intercept m68k code by placing ALINE-opcode based traps to call your code
- event-based CPU emulation frontend does always return to Python first
- provide Python handlers for all CPU emulation events
  - RESET opcode
  - ALINE trap opcode
  - invalid memory access (e.g. write in ROM region)
  - out of memory bounds (e.g. read above memory map)
  - control interrupt acknowledgement
  - watch and break points
  - custom timers based on CPU cycles
- extensive diagnose functions
  - instruction trace
  - memory access for both CPU and Python API
  - register dump
  - memory labels to mark memory regions with arbitrary Python data
  - all bare68k components use Python logging

- rich API to configure memory and CPU state
- store/restore CPU context

CHAPTER 4

Installation

- use pip:

```
$ pip install bare68k
```

- use github repository:

```
$ python setup.py install
```

- use dev setup:

```
$ python setup.py develop --user
```

CHAPTER 5

Quick Start

Here is a small code to see **bare68k** in action:

```
from bare68k import *
from bare68k.consts import *

# configure logging
runtime.log_setup()

# configure CPU: emulate a classic m68k
cpu_cfg = CPUConfig(M68K_CPU_TYPE_68000)

# now define the memory layout of the system
mem_cfg = MemoryConfig()
# let's create a RAM page (64k) starting at address 0
mem_cfg.add_ram_range(0, 1)
# let's create a ROM page (64k) starting at address 0x20000
mem_cfg.add_rom_range(2, 1)

# use a default run configuration (no debugging enabled)
run_cfg = RunConfig()

# combine everythin into a Runtime instance for your system
rt = Runtime(cpu_cfg, mem_cfg, run_cfg)

# fill in some code
PROG_BASE=0x1000
STACK=0x800
mem = rt.get_mem()
mem.w16(PROG_BASE, 0x23c0) # move.l d0,<32b_addr>
mem.w32(PROG_BASE+2, 0)
mem.w16(PROG_BASE+6, 0x4e70) # reset

# setup CPU
cpu = rt.get_cpu()
cpu.w_reg(M68K_REG_D0, 0x42)
```

```
# reset your virtual CPU to start at PROG_BASE and setup initial stack
rt.reset(PROG_BASE, STACK)

# now run the CPU emulation until an event occurrs
# here the RESET opcode is the event we are waiting for
rt.run()

# read back some memory
val = mem.r32(0)
assert val == 0x42

# finally shutdown runtime if its no longer used
# and free resources like the allocated RAM, ROM memory
rt.shutdown()
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search