
Ducky Documentation

Release 3.0

Milos Prchlik

Nov 15, 2017

Contents

1	About	3
1.1	ducky	3
1.2	Getting started	5
1.3	Virtual hardware	9
1.4	Software	17
1.5	Tools	19
1.6	Examples	26
1.7	Glossary	26
2	Code Documentation	27
2.1	ducky.boot module	27
2.2	ducky.config module	30
2.3	ducky.console module	31
2.4	ducky.cpu package	32
2.5	ducky.debugging module	69
2.6	ducky.devices package	72
2.7	ducky.errors module	84
2.8	ducky.hdt module	89
2.9	ducky.interfaces module	91
2.10	ducky.log module	92
2.11	ducky.machine module	93
2.12	ducky.mm package	96
2.13	ducky.patch module	107
2.14	ducky.profiler module	107
2.15	ducky.reactor module	109
2.16	ducky.snapshot module	111
2.17	ducky.streams module	112
2.18	ducky.tools package	115
2.19	ducky.util module	119
3	Indices and tables	123
	Python Module Index	125

Ducky is a simple virtual CPU/machine simulator, with modular design and interesting features.

1.1 ducky

Ducky is a simple virtual CPU/machine simulator, with modular design and interesting features. Ducky was created for learning purposes, no bigger ambitions. The goal was to experiment with CPU and virtual machine simulation, different instruction sets, and later working FORTH kernel become one of the main goals.

1.1.1 Features

Ducky - as in “Ducky, the CPU” - is a 32-bit RISC CPU. Ducky, “the VM”, is a simulator of Ducky CPU, adding few other modules to create the whole virtual machine, with CPUs, peripherals, storages and other components.

RISC instruction set

Instruction set was inspired by RISC CPUs, and sticks to LOAD/STORE aproach, with fixed-width instructions.

Memory model

Flat, paged, with linear addressing. Main memory consists of memory pages, each page supports simple access control - simple MMU is implemented.

Modular architecture

Virtual machine consists of several modules of different classes, and only few of them are necessary (e.g. CPU). Various peripherals are available, and it's extremely easy to develop your own and plug them in.

SMP support

Multiple CPUs with multiple cores per each CPU, with shared memory. Each core can be restricted to its own segment of memory.

Persistent storage

Modular persistent storages are available, and accessible by block IO operations, or by mmap-ing storages directly into memory.

Bytecode files

Compiled programs are stored in bytecode files that are inspired by ELF executable format. These files consist of common sections (`.text`, `.data`, ...), symbols, and their content. Assembler (`ducky-as`) translates assembly sources into object files, and these are then processed by a linker (`ducky-ld`) into the final executable. Both object and executable files use the same format and bytecode for instructions and data.

Snapshots

Virtual machine can be suspended, saved, and later restored. This is also useful for debugging purposes, every bit of memory and CPU registers can be investigated.

Debugging support

Basic support is included - break points, watch points, stack traces, stepping, snapshots, ...

Tools

- `as` for translating assembler sources to bytecode files
- `ld` for linking bytecode files into the final executable
- `objdump` for inspection of bytecode files
- `coredump` for inspection of snapshots
- `vm` for running virtual machine itself
- `img` for converting binaries to images

Planned features

- FORTH kernel - basic functionality but at least ANS compliant
- network support - it would be awesome to have a network stack available for running programs
- functioning C compiler, with simple C library
- and few others...

1.1.2 Need help?

The whole development is tracked on a GitHub [page](#), including source codes and issue tracker.

1.2 Getting started

1.2.1 Installing

The easy way is to use package:

```
pip install ducky
```

Or, you can install Ducky by checking out the sources:

```
git clone https://github.com/happz/ducky.git
cd ducky
python setup.py
```

After this, you should have the `ducky` module on your path:

```
>>> import ducky
>>> ducky.__version__
'3.0'
```

1.2.2 Prerequisites

Ducky runs with both Python 2 *and* 3 - supported versions are 2.7, 3.3, 3.4 and 3.5. There are few other dependencies, installation process (or `setup.py`) should take care of them automatically. PyPy is also supported, though only its implementation of Python2.

1.2.3 “Hello, world!” tutorial

Let’s try out the “Hello, world!” example. It’s a simple program that just prints out the well-known message.

Source code

Source is located in `examples/hello-world` directory. If you check it out, it’s a plain and simple assembler:

```
#include <arch/tty.hs>

.data

.type stack, space, 64
.type message, string, "Hello, world!"

.text

main:
    la sp, stack
    add sp, 64

    la r0, message
    call writesn
    hlt 0x00

writesn:
```

```
// > r0: string address
// ...
// r0: port
// r1: current byte
// r2: string ptr
push r1
push r2
mov r2, r0
li r0, TTY_MMIO_ADDRESS
add r0, TTY_MMIO_DATA

.__writesn_loop:
lb r1, r2
bz __writesn_write_nl
stb r0, r1
inc r2
j __writesn_loop

.__writesn_write_nl:
// \n
li r1, 0x0000000A
stb r0, r1
// \r
li r1, 0x0000000D
stb r0, r1
li r0, 0x00000000
pop r2
pop r1
ret
```

It's a little bit more structured than necessary, just for educational purposes.

Binary

To run this code, we have to create a *binary* of it. Of course, there are tools for this very common goal:

```
ducky-as -i examples/hello-world/hello-world.s -o examples/hello-world/hello-world.o
```

This command will translate source code to an *object file* which contains instructions and other necessary resources for *machine* to run it. You can inspect the object file using `objdump` tool:

```
ducky-objdump -i examples/hello-world/hello-world.o -a
```

This should produce output similar to this one:

```
[INFO] Input file: examples/hello-world/hello-world.o
[INFO]
[INFO] === File header ===
[INFO] Magic: 0xDEAD
[INFO] Version: 1
[INFO] Sections: 4
[INFO]
[INFO] === Sections ===
[INFO]
[INFO] Index Name Type Flags Base Items Size Offset
[INFO] -----
[INFO] 0 .data DATA RW-- (0x03) 0x000000 14 14 104
```

```

[INFO]      1  .text      TEXT      RWX- (0x07) 0x000100      24      96      118
[INFO]      2  .symtab    SYMBOLS    ---- (0x00) 0x000200      6      120     214
[INFO]      3  .strings    STRINGS    ---- (0x00) 0x000000      0      122     334
[INFO]
[INFO] === Symbols ===
[INFO]
[INFO] Name                Section    Address    Type                Size  File
[INFO] -----
[INFO]      Line    Content
[INFO] -----
[INFO] message                .data      0x000000    string (2)         14  examples/
[INFO] ↪hello-world.asm  1      "Hello, world!"
[INFO] main                  .text      0x000100    function (3)         0  examples/
[INFO] ↪hello-world.asm  4
[INFO] outb                  .text      0x000110    function (3)         0  examples/
[INFO] ↪hello-world.asm  10
[INFO] writesn               .text      0x000118    function (3)         0  examples/
[INFO] ↪hello-world.asm  16
[INFO] __fn_writesn_loop      .text      0x00012C    function (3)         0  examples/
[INFO] ↪hello-world.asm  27
[INFO] __fn_writesn_write_n1  .text      0x000140    function (3)         0  examples/
[INFO] ↪hello-world.asm  33
[INFO]
[INFO] === Disassemble ==
[INFO]
[INFO] Section .text
[INFO] 0x000100 (0x00000004) li r0, 0x0000
[INFO] 0x000104 (0x0000800D) call 0x0010
[INFO] 0x000108 (0x00000004) li r0, 0x0000
[INFO] 0x00010C (0x0000000B) int 0x0000
[INFO] 0x000110 (0x000000E3) outb r0, r1
[INFO] 0x000114 (0x0000000E) ret
[INFO] 0x000118 (0x000000D4) push r1
[INFO] 0x00011C (0x00000154) push r2
[INFO] 0x000120 (0x00000054) push r0
[INFO] 0x000124 (0x00000095) pop r2
[INFO] 0x000128 (0x00040004) li r0, 0x0100
[INFO] 0x00012C (0x00000842) lb r1, r2
[INFO] 0x000130 (0x00006029) bz 0x000C
[INFO] 0x000134 (0x0FFEC00D) call -0x0028
[INFO] 0x000138 (0x00000096) inc r2
[INFO] 0x00013C (0x0FFF6026) j -0x0014
[INFO] 0x000140 (0x00002844) li r1, 0x000A
[INFO] 0x000144 (0x0FFE400D) call -0x0038
[INFO] 0x000148 (0x00003444) li r1, 0x000D
[INFO] 0x00014C (0x0FFE000D) call -0x0040
[INFO] 0x000150 (0x00000004) li r0, 0x0000
[INFO] 0x000154 (0x00000095) pop r2
[INFO] 0x000158 (0x00000055) pop r1
[INFO] 0x00015C (0x0000000E) ret
[INFO]

```

As you can see, object file contains instructions, some additional data, list of symbols, and some more, with labels replaced by dummy offsets. Offsets in jump instructions make no sense yet because object file is not the finalized binary - yet. For that, there's yet another tool:

```
ducky-ld -i examples/hello-world/hello-world.o -o examples/hello-world/hello-world
```

This command will take object file (or many of them), and produce one *binary* by merging code, data and other sections from all source object files, and updates addresses used by instructions to retrieve data and to perform jumps. You can inspect the resulting binary file using `objdump` tool as well:

```
ducky-objdump -i examples/hello-world/hello-world -a
```

This should produce output very similar to the one you've already seen - not much had changed, there was only one object file, only offsets used by `call` and `j` instructions are now non-zero, meaning they are now pointing to the correct locations.

Running

Virtual machine configuration can get quite complicated, so I try to avoid too many command line options, and opt for using configuration files. For this example, there's one already prepared. Go ahead and try it:

```
ducky-vm --machine-config=examples/hello-world/hello-world.conf --set-  
↪option=bootloader:file=examples/hello-world/hello-world
```

There are two command-line options:

- `--machine-config` tells VM where to find its configuration file,
- `--set-option` modifies this configuration; this particular instance tells VM to set `file` option in section `bootloader` to path of our freshly built binary, `examples/hello-world/hello-world`. Since I run examples during testing process, their config files lack this option since it changes all the time.

You should get output similar to this:

```
1 1441740855.82 [INFO] Ducky VM, version 1.0
2 1441740855.82 [INFO] mm: 16384.0KiB, 16383.5KiB available
3 1441740855.82 [INFO] hid: basic keyboard controller on [0x0100] as device-1
4 1441740855.83 [INFO] hid: basic tty on [0x0200] as device-2
5 1441740855.83 [INFO] hid: basic terminal (device-1, device-2)
6 1441740855.83 [INFO] snapshot: storage ready, backed by file ducky-snapshot.bin
7 1441740855.83 [INFO] RTC: time 21:34:15, date: 08/09/15
8 1441740855.83 [INFO] irq: loading routines from file interrupts
9 1441740856.02 [INFO] binary: loading from from file examples/hello-world/hello-world
10 1441740856.02 [INFO] #0:#0: CPU core is up
11 1441740856.02 [INFO] #0:#0:  check-frames: yes
12 1441740856.02 [INFO] #0:#0:  coprocessor: math
13 1441740856.02 [INFO] #0: CPU is up
14 Hello, world!
15 1441740856.04 [INFO] #0:#0: CPU core halted
16 1441740856.05 [INFO] #0: CPU halted
17 1441740856.05 [INFO] snapshot: saved in file ducky-snapshot.bin
18 1441740856.05 [INFO] Halted.
19 1441740856.05 [INFO]
20 1441740856.05 [INFO] Exit codes
21 1441740856.05 [INFO] Core      Exit code
22 1441740856.06 [INFO] -----
23 1441740856.06 [INFO] #0:#0      0
24 1441740856.06 [INFO]
25 1441740856.06 [INFO] Instruction caches
26 1441740856.06 [INFO] Core      Reads      Inserts      Hits      Misses      Prunes
27 1441740856.06 [INFO] -----
28 1441740856.06 [INFO] #0:#0      133        34        99        34        0
29 1441740856.06 [INFO]
30 1441740856.06 [INFO] Core      Ticks
```

```

31 1441740856.06 [INFO] -----
32 1441740856.06 [INFO] #0:#0    133
33 1441740856.06 [INFO]
34 1441740856.06 [INFO] Executed instructions: 133 0.028670 (4639.0223/sec)
35 1441740856.06 [INFO]

```

And there, on line 16, between all that funny nonsenses, it is! :) The rest of the output are just various notes about loaded binaries, CPU caches, nothing important right now.

And that's it.

1.3 Virtual hardware

1.3.1 CPU

Ducky VM can have multiple CPUs, each with multiple cores. Each core is a 32-bit microprocessor, with 32 32-bit registers, connected to main memory. It is equiped with MMU, and its own instruction cache.

CPU core can work in privileged and unprivileged modes, allowing use of several protected instructions in privileged mode.

Registers

- 32 32-bit registers - registers `r0` to `r29` are general purpose registers - `r31` is reserved, and used as a stack pointer register, `SP` - contains address of the last value pushed on stack - `r30` is reserved, and used as a frame pointer register, `FP` - contains content of `SP` in time of the last `call` or `int` instruction
- `flags` register

Flags register

`flags` register can be considered as read-only value, since it is not possible to modify it using common bit operations or arithmetic instructions. However, its content reflect outcomes of executed instructions, therefore it is possible e.g. to modify it content using comparison instructions. It is also possible to inspect and modify it in exception service routine, where pre-exception `flags` are stored on the stack (and loaded when `ESR` ends).

Mask	Flags	Usage
0x00	<code>privileged</code>	If set, CPU runs in privileged mode, and usage of protected instructions is allowed
0x01	<code>hwint_allowed</code>	If set, HW interrupts can be delivered to this core
0x04	<code>e</code>	Set if the last two compared registers were equal
0x08	<code>z</code>	Set if the last arithmetic operation produced zero
0x10	<code>o</code>	Set if the last arithmetic operation overflowed
0x20	<code>s</code>	Set if the last arithmetic operation produced negative result

Instruction set

CPU supports multiple instruction sets. The default one, *ducky*, is the main workhorse, suited for general coding, but other instruction sets can exist, e.g. coprocessor may use its own instruction set for its operations.

Ducky instruction set

Design principles

- basic data unit is *a word* - 4 bytes, 32 bits. Other units are *short* - 2 bytes, 16 bits - and *byte* - 1 byte, 8 bits. Instructions often have variants for different data units, distinguished by a suffix (*w* for words, *s* for shorts, and *b* for single bytes)
- load and store operations are performed by dedicated instructions
- memory-register transfers work with addresses that are aligned to the size of their operands (1 byte alignment - so no alignment at all - for byte operands)
- unless said otherwise, destination operand is the first one
- when content of a register is changed by instruction, several flags can be modified subsequently. E.g. when new value of register is zero, *z* flag is set.

Notes on documentation

- *rN* refers to generally any register, from *r0* up to *r29* - special registers are referred to by their common names (e.g. *SP*).
- *rA*, *rB* refer to the first and the second instruction operand respectively and stand for any register.
- *<value>* means immediate, absolute value. This covers both integers, specified as base 10 or base 16 integers, both positive and negative, and labels and addresses, specified as *&label*
- when instruction accepts more than one operand type, it is documented using *|* character, e.g. (*rA|<value>*) means either register or immediate value
- immediate values are encoded in the instructions, therefore such value cannot have full 32-bit width. Each instruction should indicate the maximal width of immediate value that can be safely encoded, should you require greater values, please see *li* and *liu* instructions

Stack frames

Several instructions transfer control to other parts of program, with possibility of returning back to the previous spot. It is done by creating a *stack frame*. When stack frame is created, CPU performs these steps:

- *IP* is pushed onto the stack
- *FP* is pushed onto the stack
- *FP* is loaded with value of *SP*

Destroying stack frame - reverting the steps above - effectively transfers control back to the point where the subroutine was called from.

Arithmetic

All arithmetic instructions take at least one operand, a register. In case of binary operations, the second operand can be a register, or an immediate value (15 bits wide, sign-extended to 32 bits). The result is always stored in the first operand.

```
add rA, (rB|<value>)
```

```
dec rA
inc rA
mul rA, (rB|<value>)
sub rA, (rB|<value>)
```

Bitwise operations

All bitwise operations - with exception of `not` - take two operands, a register, and either another register or an immediate value (15 bits wide, sign-extended to 32 bits). The result is always stored in the first operand.

```
and rA, (rB|<value>)
not rA
or rA, (rB|<value>)
shiftrl rA, (rB|<value>)
shiftr rA, (rB|<value>)
shiftrs rA, (rB|<value>)
xor rA, (rB|<values>)
```

Branching instructions

Branching instructions come in form `<inst> (rA|<address>)`. If certain conditions are met, branching instruction will perform jump by adding value of the operand to the current value of PC (which, when instruction is being executed, points *to the next instruction* already). If the operand is an immediate address, it is encoded in the instruction as an immediate value (16 bit wide, sign-extended to 32 bits). This limits range of addresses that can be reached using this form of branching instructions.

Branching instructions do not create new stack frame.

Unconditional branching

```
j (rA|<value>)
```

Conditional branching

Instruction	Jump when ...
be	e = 1
bne	e = 0
bs	s = 1
bns	s = 0
bz	z = 1
bnz	z = 0
bo	o = 1
bno	o = 0
bg	e = 0 and s = 0
bge	e = 1 or s = 0
bl	e = 0 and s = 1
ble	e = 1 or s = 1

Conditional setting

All conditional setting instructions come in form `<inst> rA`. Depending on relevant flags, `rA` is set to 1 if condition is evaluated to be true, or to 0 otherwise.

For flags relevant for each instruction, see branching instruction with the same suffix (e.g. `setle` evaluates the same flags with the same result as `ble`).

Instruction
sete
setne
setz
setnz
seto
setno
sets
setns
setg
setge
setl
setle

Comparing

Two instructions are available for comparing of values. Compare their operands and sets corresponding flags. The second operand can be either a register or an immediate value (15 bits wide).

`cmp rA, (rB|<value>)` - immediate value is sign-extended to 32 bits.

`cmphu rA, (rB|<value>)` - treat operands as unsigned values, immediate value is zero-extended to 32 bits.

Interrupts

Delivery

If flag `hwint_allowed` is unset, no hardware IRQ can be accepted by CPU and stays queued. All queued IRQs will be delivered as soon as flag is set.

`cli` - clear `hwint` flag

`sti` - set `hwint` flag

In need of waiting for external events it is possible to suspend CPU until the next IRQ is delivered.

`idle` - wait until next IRQ

Invocation

Any interrupt service routine can be invoked by means of special instruction. When invoked several events take place:

- SP is saved in temporary space
- IP and SP are set to values that are stored in EVT in the corresponding entry
- important registers are pushed onto new stack (in this order): old SP, flags
- new stack frame is created
- privileged mode is enabled
- delivery of hardware interrupts is disabled

When routine ends (via `retint`), these steps are undone, and content of saved registers is restored.

`int (rA|<index>)`

`retint` - return from interrupt routine

Inter-processor interrupts (IPI) can be delivered to other processors, via dedicated instruction, similar to `int` but specifying CPUID of target core in the first operand.

`ipi rA, (rB|<index>)`

Routines

When routine is called, new stack frame is created, and CPU continues with instructions pointed to by the first operand. For its meaning (and limitations) see *Branching instructions*.

`call (rA|<address>)`

`ret`

Stack

`pop rA`

`push (rA|<value>)`

Miscellaneous

`nop` - do absolutely nothing

`hlt (rA|<value>)` - Halt CPU and set its exit code to specified value.

`rst` - reset CPU state. All flags cleared, `privileged = 1`, `hwint_allowed = 0`, all registers set to 0

`mov rA, rB` - copy value of `rB` into `rA`

`swp rA, rB` - swap content of two registers

`sis <value>` - switch instruction set to a different one

Memory access

Address operand - `{address}` - can be specified in different ways:

- `rA` - address is stored in register
- `rA[<offset>]` - address is computed by addition of `rA` and `offset`. `offset` can be both positive and negative. `fp` and `sp` can be also used as `rA`. `<offset>` is an immediate value, 15 bits wide, sign-extended to 32 bits.

Read

`lw rA, {address}` - load word from memory

`ls rA, {address}` - load short from memory

`lb rA, {address}` - load byte from memory

Write

`stw {address}, rA`

`sts {address}, rA` - store lower 2 bytes of `rA`

`stb {address}, rA` - store lower byte of `rA`

Constants

Instructions for filling registers with values known in compile time.

`li rA, <constant>` - load constant into register. `constant` is encoded into instruction as an immediate value (20 bits wide, sign-extended to 32 bits)

`liu rA, <constant>` - load constant into the upper half of register. `constant` is encoded into instruction as an immediate value (20 bits wide immediate, only lower 16 bits are used)

`la rA, <constant>` - load constant into the register. `constant` is an immediate value (20 bits wide, sign-extended to 32 bits), and is treated as an offset from the current value of `PC` - register is loaded with the result of `PC + constant`.

Compare-and-swap

`cas rA, rB, rC` - read word from address in register `rA`. Compare it with value in register `rB` - if both are equal, take content of `rC` and store it in memory on address `rA`, else store memory value in `rB`.

Math coprocessor instruction set

This page describes instruction set of math coprocessor.

Manipulating registers

Instructions for moving values between coprocessor stack and memory or CPU registers.

`itol`
`utol`
`ltoi`
`ltoii`
`iitol`
`dupl`

Arithmetic operations

`incl`
`decl`
`addl`
`subl`
`mull`
`divl`
`modl`
`syndivl`
`symmodl`

1.3.2 Control coprocessor

Control coprocessor (*CC*) is a coprocessor dedicated to control various runtime properties of CPU core. Using `ctr` and `ctw` instructions it is possible to inspect and modify these properties.

Control registers

CR0

`cr0` register contains so-called `CPUID` value which intendifies location of this CPU core in global CPU topology.

- upper 16 bits contain index number of CPU this core belongs to
- lower 16 bits contain index number of this core in the context of its parent CPU

There's always core with `CPUID 0x00000000`, since there's always at least one core present.

This register is read-only.

CR1

`cr1` register contains *Exception Vector Table* address for this core. Be aware that *any* address is accepted, no alignment or any other restrictions are applied.

CR2

`cr2` register contains page table address for this core. Be aware that *any* address is accepted, no alignment or any other restrictions are applied.

CR3

`cr3` register provides access to several flags that modify behavior of CPU core.

Mask	Flag	Usage
0x00	pt_enabled	If set, MMU consults all memory accesses with page tables.
0x01	jit	If set, JIT optimizations are enabled.
0x02	vmdebug	If set, VM will produce huge amount of debugging logs.

Note: `jit` flag is read-only. It is controlled by options passed to Ducky when VM was created, and cannot be changed in runtime.

Note: `vmdebug` flag is shared between all existing cores. Changing it on one core affects immediately all other cores.

Note: `vmdebug` flag will not produce any debugging output if debug mode was disabled e.g. by not passing `-d` option to `ducky-vm` tool. If debug mode was allowed, changing this flag will control log level of VM.

1.3.3 Exception Vector Table

Exception vector table (*EVT*) is located in main memory, by default at address `0x00000000`, and provides core with routines that can help resolve some of exceptional states core can run into.

EVT address can be set per CPU core, see *CR1*.

EVT is 256 bytes - 1 memory page - long, providing enough space for 32 entries. Typically, lower 16 entries are reserved for hardware interrupts, provided by devices, and upper 16 entries lead to software routines that deal with exceptions, and provide additional functionality for running code in form of software interrupts.

Entry format

IP - 32 bits	SP - 32 bits
--------------	--------------

When CPU is interrupted - by hardware (device generates IRQ) or software (exception is detected, or program executes `int` instruction) interrupt - corresponding entry is located in EVT, using interrupt ID as an index.

1.3.4 Hardware Description Table

Hardware Description Table (*HDT*) is located in main memory, by default at address `0x00000100`, and hardware setup of the machine.

1.3.5 Memory

Memory model

- the full addressable memory is 4 GB, since the address bus is 32-bit wide, but it is quite unrealistic expectation. I usually stick to 24-bits for addresses, which leads to 16MB of main memory
- memory is organized into pages of 256 bytes - each page can be restricted for read, write and execute operations

Memory layout

Stack

- standard LIFO data structure
- grows from higher addresses to lower
- there is no pre-allocated stack, every bit of code needs to prepare its own if it intends to use instructions that operate with stack
- when push'ing value to stack, SP is decreased by 4 (size of general register), and then value is stored on this address
- each EVT provides its own stack pointer

1.4 Software

1.4.1 Calling convention

I use very simple calling convention in my code:

- all arguments are in registers
- first argument in `r0`, second in `r1`, ... You get the picture.
- if there's too many arguments, refactor your code or use a stack...
- return value is in `r0`
- callee is responsible for save/restore of registers it's using, with exception of:
 - registers that were used for passing arguments - these are expected to have undefined value when callee returns

- `r0` if callee returns value back to caller

All virtual interrupt routines, assembler code, any pieces of software I've written for this virtual machine follows this calling convention - unless stated otherwise...

1.4.2 Software interrupts

Software interrupts provide access to library of common functions, and - in case of virtual interrupts - to internal, complex and otherwise inaccessible resources of virtual machine itself.

For the list for existing interrupts and their numbers, see `ducky.irq.IRQList`. However, by the nature of invoking a software interrupt, this list is not carved into a stone. You may easily provide your own `EVT`, with entries leading to your own routines, and use e.g. the 33th entry, `HALT`, to sing a song.

All values are defined in files in `defs/` directory which you can - and should - include into your assembler sources.

BLOCKIO

EVT entry	33
	Read mode Write mode
Parameters	<code>r0</code> device id
	<code>r1</code> bit #0: 0 for read, 1 for write bit #1: 0 for synchronous, 1 for asynchronous operation
	<code>r2</code> block id src memory address
	<code>r3</code> dst memory address block id
	<code>r4</code> number of blocks
Returns	<code>r0</code> 0 for success

Perform block IO operation - transfer block between memory and storage device. Use the lowest bit of `r1` to specify direction:

- 0 - *read mode*, blocks are transfered from storage into memory
- 1 - *write mode*, blocks are tranfered from memory to storage

Current data segment is used for addressing memory locations.

If everything went fine, 0 is returned, any other value means error happened.

IO operation is a blocking action, interrupt will return back to the caller once the IO is finished. Non-blocking (*DMA-like*) mode is planned but not yet implemented.

This operation is implemented as a virtual interrupt, see `ducky.blockio.BlockIOInterrupt` for details.

VMDEBUG

EVT entry	34
	QUIET mode
Parameters	<code>r0</code> 0
	<code>r1</code> 0 for <i>quiet</i> mode, anything else for <i>full</i> mode
Returns	<code>r0</code> 0 for success

VM interrupt allows control of VM debugging output. Currently, only two levels of verbosity that are available are *quiet* and *full* mode. In *quiet* mode, VM produces no logging output at all.

This interrupt can control level amount of debugging output in case when developer is interested only in debugging only a specific part of his code. Run VM with debugging turned on (`-d` option), turn the debugging off at the beggining of the code, and turn it on again at the beggining of the interesting part to get detailed output.

1.5 Tools

Ducky comes with basic toolchain necessary for development of complex programs. One piece missing is the C cross-compiler with simple C library but this issue will be resolved one day.

1.5.1 Common options

All tools accept few common options:

-q, --quiet

Lower verbosity level by one. By default, it is set to *info*, more quiet levels are *warnings*, *error* and *critical*.

-v, --verbose

Increase verbosity level by one. By default, it is set to *info*, more verbose levels is *debug*. *debug* level is not available for `ducky-vm` unless `-d` option is set.

-d, --debug

Set logging level to *debug* immediately. `ducky-vm` also requires this option to even provide any debugging output - it is not possible to emit *debug* output by setting `-v` enough times if `-d` was not specified on command-line.

When option takes an address as an argument, address can be specified either using decimal or hexadecimal base. Usually, the absolute address is necessary when option is not binary-aware, yet some options are tied closely to particular binary, such options can also accept name of a symbol. Option handling code will try to find corresponding address in binary's symbol table. This is valid for both command-line options and configuration files.

1.5.2 as

Assembler. Translates *assembler files* (`.asm`) to *object files* (`.o`) - files containing bytecode, symbol information, etc.

Options

-i FILE

Take assembly `FILE`, and create an object file from its content. It can be specified multiple times, each input file will be processed.

-o FILE

Write resulting object data into `FILE`.

This options is optional. If no `-o` is specified, `ducky-as` will then create output file for each input one by replacing its suffix by `.o`. If it is specified, number of `-o` options must match the number of `-i` options.

-f

When output file exists already, `ducky-as` will refuse to overwrite it, unless `-f` is set.

-D VAR

Define name, passed to processed assembly sources. User can check for its existence in source by `.ifdef/.ifndef` directives.

-I DIR

Add `DIR` to list of directories that are searched for files, when `.include` directive asks assembler to process additional source file.

-m, --mmapable-sections

Create object file with sections that can be loaded using `mmap()` syscall. This option can save time during VMstartup, when binaries can be simply mmaped into VM's memory space, but it also creates larger binary files because of the alignment of sections in file.

-w, --writable-sections

By default, `.text` and `.rodata` sections are read-only. This option lowers this restriction, allowing binary to e.g. modify its own code.

-b, --blob

Create object file wrapping a binary blob. Such file then contains the data from input file, encoded as ASCII data, with several symbols allowing other code to access this data.

-B BLOB_FLAGS, --blob-flags=BLOB_FLAGS

Flags of blob section. Syntax of assembly `.section` directive is accepted.

1.5.3 ld

Linker. Takes (one or multiple) *object files* (`.o`) and merges them into one, *binary*, which can be executed by VM.

Options

-i FILE

Take object `FILE`, and create binary file out of it. It can be specified multiple times, all input files will be processed into one binary file.

-o FILE

Output file.

-f

When output file exists already, `ducky-ld` will refuse to overwrite it, unless `-f` is set.

--section-base=SECTION=ADDRESS

Linker tries to merge all sections into a binary in a semi-random way - it can be influenced by order of sections in source and object files, and order of input files passed to linker. It is in fact implementation detail and can change in the future. If you need specific section to have its base set to known address, use this option. Be aware that linker may run out of space if you pass conflicting values, or force sections to create too small gaps between each other so other sections would not fit in.

1.5.4 coredump

Prints information stored in a saved VM snapshot.

1.5.5 objdump

Prints information about object and binary files.

1.5.6 profile

Prints information stored in profiling data, created by VM. Used for profiling running binaries.

1.5.7 vm

Stand-alone virtual machine - takes binary, configuration files, and other resources, and executes binaries.

VM Configuration file

Number of available options can easily get quite high, especially when different devices come into play, and setting all of them on command line is not very clear. To ease this part of VM processes, user can create a configuration file. Syntax is based on Python's `ConfigParser` (or Windows `.ini`) configuration files. It consists of sections, setting options for different subsystems (VM, CPUs, ...). You can find few configuration files in `examples/` directory.

When option takes an address as an argument, address can be specified either using decimal or hexadecimal base. Usually, the absolute address is necessary when option is not binary-aware, yet some options are tied closely to particular binary, such options can also accept name of a symbol. Option handling code will try to find corresponding address in binary's symbol table. This is valid for both command-line options and configuration files.

[machine]

cpus

Number of separate CPUs.

int, default 1

cores

Number of cores per CPU.

int, default 1

[memory]

size

Memory size in bytes.

int, default 0x1000000

force-aligned-access

When set, unaligned memory access will lead to exception.

bool, default yes

[cpu]

math-coprocessor

When set, each CPU core will have its own math coprocessor.

bool, default no

control-coprocessor

When set, each CPU core will have its own control coprocessor.

bool, default yes

inst-cache

Number of slots in instruction cache.

int, default 256

check-frame

When set, CPU cores will check if stack frames were cleaned properly when `ret` or `iret` is executed.

`bool`, default `yes`

evt-address

Address of interrupt vector table.

`int`, default `0x000000`

[bootloader]**file**

Path to bootloader file.

`str`, required

[device-N]

Each section starting with `device-` tells VM to create virtual device, and provide it to running binaries. Device sections have few options, common for all kinds of devices, and a set of options, specific for each different device or driver.

klass

Device class - arbitrary string, describing family of devices. E.g. I use `input` for devices processing user input (e.g. keyboard controllers).

`str`, required

driver

Python class that *is* the device driver.

`str`, required

master

If set, `master` is superior device, with some responsibilities over its subordinates.

`str`, optional

Options

`--machine-config=PATH`

Specify `PATH` to VM configuration file. For its content, see *VM Configuration file*.

`--set-option=SECTION:OPTION=VALUE`

`--add-option=SECTION:OPTION=VALUE`

These two options allow user to modify the content of configuration file, by adding of new options or by changing the existing ones.

Lets have (incomplete) config file `vm.conf`:

```
[machine]
cpu = 1
core = 1

[binary-0]
```

You can use it to run different binaries without having separate config file for each of them, just by telling `ducky-vm` to load configuration file, and then change one option:

```
$ ducky-vm --machine-config=vm.conf --add-option=binary-0:file=<path to binary of_
↪your choice>
```

Similarly, you can modify existing options. Lets have (incomplete) config file `vm.conf`:

```
[machine]
cpus = 1
cores = 1

[binary-0]
file = some/terminal/app

[device-1]
klass = input
driver = ducky.devices.keyboard.KeyboardController
master = device-3

[device-2]
klass = output
driver = ducky.devices.tty.TTY
master = device-3

[device-3]
klass = terminal
driver = ducky.devices.terminal.StandardIOTerminal
input = device-1
output = device-2
```

Your app will run using VM's standard IO streams for input and out. But you may want to start it with a different kind of terminal, e.g. PTY one, and attach to it using `screen`:

```
$ ducky-vm --machine-config=vm.conf --set-option=device-3:driver=ducky.devices.
↪terminal.StandalonePTYTerminal
```

--enable-device=DEVICE

--disable-device=DEVICE

Shortcuts for `--set-option=DEVICE:enabled=yes` and `--set-option=DEVICE:enabled=no` respectively.

--poke=ADDRESS:VALUE:LENGTH

`poke` option allows modification of VM's memory after all binaries and resources are loaded, just before the VM starts execution of binaries. It can be used for setting runtime-specific values, e.g. argument for a binary.

Consider a simple binary, running a loop for specified number of iterations:

By default, 10 iterations are hard-coded into binary. If you want to temporarily change number of iterations, it's not necessary to recompile binary. By default, this binary, being the only one running, would get segment `0x02`, it's `.data` section was mentioned first, therefore its base address will be `0x0000`, leading to `loops` having absolute address `0x020000`. Then:

```
$ ducky-vm --machine-config=vm.conf --poke=0x020000:100:2
```

will load binary, then modify its `.data` section by changing value at address `0x020000` to `100`, which is new number of iterations. Meta variable `LENGTH` specifies number of bytes to overwrite by `poke` value, and `poke` will change exactly `LENGTH` bytes - if `VALUE` cannot fit into available bits, exceeding bits of `VALUE` are masked out, and `VALUE` that can fit is zero-extended to use all `LENGTH` bytes.

--jit

Enable *JIT* - more dense implementation of Ducky instructions is used. Result is higher execution speed of each instruction, however it removes many debugging code. It may be difficult to debug instruction execution even with `-d` option enabled.

1.5.8 img

Converts binaries to binary images that can be loaded by boot loader.

Options

-i FILE

Take binary `FILE`, and create a binary image from its content.

-o FILE

Write resulting object data into `FILE`.

-f

When output file exists already, `ducky-img` will refuse to overwrite it, unless `-f` is set.

1.6 Examples

1.6.1 “Hello, world!”

1.6.2 “Hello, world!” using library

1.6.3 VGA

1.6.4 Clock

1.7 Glossary

binary Binary is the final piece in the process

bootloader *Bootloader*, for purposes of this documentation, means virtually any piece of bytecode. It gains its “bootloader-like” property by the fact that it’s the first piece of code that’s executed by CPU core.

EVT *Exception Vector Table*

HDT *Hardware Description Table*

linker *Linker* takes an *object file* (or more, or even an archive), and creates a *binary* by merging relevant sections and by replacing symbolic references with final offsets.

ld provides this functionality.

machine For a long time, Ducky existed only as an software simulator. But, since I got that great idea about getting me a simple FPGA and learn VHDL, this may no longer be true. So, when I write about *machine*, I mean both software simulator (*VM*) and hardware materialization of Ducky SoC.

object file *Object file* is a file containing compiled code in a form of distinct sections of instructions, data and other necessary resources. Despite sharing their format with *binary* file, object files are usually *not* executable because pretty much no instructions that address memory contain correct offsets, and refer to the locations using symbols. Final offsets are calculated and fixed by a *linker*.

VM *Virtual Machine*. For a long time, the only existing Ducky implementation.

2.1 ducky.boot module

This file provides necessary code to allow boot up of a virtual machine with the correct program running. This code may provide slightly different environment when compared to real hardware process, since e.g. external files can be mmap-ed into VM's memory for writing.

`ducky.boot.DEFAULT_BOOTLOADER_ADDRESS = 131072`

By default, CPU starts executing instructions at this address after boot.

`ducky.boot.DEFAULT_HDT_ADDRESS = 256`

By default, Hardware Description Table starts at this address after boot.

class `ducky.boot.MMapArea` (*ptr, address, size, file_path, offset, pages_start, pages_cnt, flags*)

Bases: `object`

Objects of this class represent one mmaped memory area each, to track this information for later use.

Parameters

- **ptr** – `mmap` object, as returned by `mmap.mmap()` function.
- **address** (*u32_t*) – address of the first byte of an area in the memory.
- **size** (*u32_t*) – length of the area, in bytes.
- **file_path** – path to a source file.
- **offset** (*u32_t*) – offset of the first byte in the source file.
- **pages_start** (*int*) – first page of the area.
- **pages_cnt** (*int*) – number of pages in the area.
- **flags** (`mm.binary.SectionFlags`) – flags applied to this area.

load_state (*state*)

save_state (*parent*)

class `ducky.boot.MMapAreaState`

Bases: `ducky.snapshot.SnapshotNode`

class `ducky.boot.MMapMemoryPage` (*area*, **args*, ***kwargs*)

Bases: `ducky.mm.ExternalMemoryPage`

Memory page backed by an external file that is accessible via `mmap()` call. It's a part of one of `ducky.boot.MMapArea` instances, and if such area was opened as *shared*, every change in the content of its pages will reflect onto the content of an external file, and vice versa, every change of external file will be reflected in content of this page (if this page lies in affected area).

Parameters *area* (`MMapArea`) – area this page belongs to.

`__get_py2` (*offset*)

Read one byte from page.

Parameters *offset* (*int*) – offset of the requested byte.

Return type *int*

`__get_py3` (*offset*)

Read one byte from page.

Parameters *offset* (*int*) – offset of the requested byte.

Return type *int*

`__put_py2` (*offset*, *b*)

Write one byte to page.

Parameters

- **`offset`** (*int*) – offset of the modified byte.
- **`b`** (*int*) – new value of the modified byte.

`__put_py3` (*offset*, *b*)

Write one byte to page.

Parameters

- **`offset`** (*int*) – offset of the modified byte.
- **`b`** (*int*) – new value of the modified byte.

`get` (*offset*)

Read one byte from page.

This is an abstract method, `__init__` is expected to replace it with a method, tailored for the Python version used.

Parameters *offset* (*int*) – offset of the requested byte.

Return type *int*

`put` (*offset*, *b*)

Write one byte to page.

This is an abstract method, `__init__` is expected to replace it with a method, tailored for the Python version used.

Parameters

- **`offset`** (*int*) – offset of the modified byte.
- **`b`** (*int*) – new value of the modified byte.

class `ducky.boot.ROMLoader` (*machine*)

Bases: `ducky.interfaces.IMachineWorker`

This class provides methods for loading all necessary pieces into VM's memory. These methods are called in VM's *boot* phase.

`_get_mmap_fileno` (*file_path*)

`_put_mmap_fileno` (*file_path*)

`boot` ()

`halt` ()

`mmap_area` (*file_path*, *address*, *size*, *offset=0*, *flags=None*, *shared=False*)

Assign set of memory pages to mirror external file, mapped into memory.

Parameters

- **file_path** (*string*) – path of external file, whose content new area should reflect.
- **address** (*u24*) – address where new area should start.
- **size** (*u24*) – length of area, in bytes.
- **offset** (*int*) – starting point of the area in mmaped file.
- **flags** (`ducky.mm.binary.SectionFlags`) – specifies required flags for mmaped pages.
- **shared** (*bool*) – if `True`, content of external file is mmaped as shared, i.e. all changes are visible to all processes, not only to the current ducky virtual machine.

Returns newly created mmap area.

Return type `ducky.mm.MMapArea`

Raises `ducky.errors.InvalidResourceError` – when *size* is not multiply of `ducky.mm.PAGE_SIZE`, or when *address* is not multiply of `ducky.mm.PAGE_SIZE`, or when any of pages in the affected area is already allocated.

`poke` (*address*, *value*, *length*)

`setup_bootloader` (*filepath*, *base=None*)

Load *bootloader* into main memory.

In the world of a real hardware, bootloader binary would be transformed into an image, and then “burned” in some form into the memory - main, or some kind of ROM from which it'd be loaded into main memory at the very beginning of boot process.

Parameters

- **filepath** (*str*) – path to bootloader binary.
- **base** (*u32_t*) – address of the first byte of bootloader in memory. By default, `ducky.boot.DEFAULT_BOOTLOADER_ADDRESS` is used.

`setup_debugging` ()

`setup_hdt` ()

Initialize memory area containing *Hardware Description Table*.

If VM config file specifies HDT image file, it is loaded, otherwise HDT is constructed for the actual configuration, and then it's copied into memory.

Parameters

- **machine.hdt-address** (*u32_t*) – Base address of HDT in memory. If not set, `ducky.boot.DEFAULT_HDT_ADDRESS` is used.
- **machine.hdt-image** (*str*) – HDT image to load. If not set, HDT is constructed for the actual VM's configuration.

setup_mmmaps ()

unmmap_area (*mmap_area*)

2.2 ducky.config module

VM configuration management

class `ducky.config.MachineConfig` (**args, **kwargs*)

Bases: `ConfigParser.ConfigParser`

Contains configuration of the whole VM, and provides methods for parsing, inspection and extending this configuration.

_MachineConfig__count (*prefix*)

_MachineConfig__count_breakpoints ()

_MachineConfig__count_devices ()

_MachineConfig__count_mmmaps ()

_MachineConfig__sections_with_prefix (*prefix*)

_boolean_states = {'1': True, 'on': True, 'false': False, '0': False, 'off': False, 'yes': True, 'no': False, 'true': True}

add_breakpoint (*core, address, active=None, flip=None, ephemeral=None, countdown=None*)

add_device (*klass, driver, **kwargs*)

Add another device to the configuration.

Parameters

- **klass** (*string*) – class of the device (`klass` option).
- **driver** (*string*) – device driver - dot-separated path to class (`driver` option).
- **kwargs** – all keyword arguments will be added to the section as device options.

add_mmap (*filepath, address, size, offset=None, access=None, shared=None*)

add_storage (*driver, sid, filepath=None*)

Add another storage to the configuration.

Parameters

- **driver** (*string*) – storage's driver - dot-separated path to class (`driver` option).
- **sid** (*int*) – storage's SID (`sid` options).
- **filepath** (*string*) – path to backend file, if there's any (`filepath` option).

create_getters (*section*)

dumps ()

get (*section, option, default=None, **kwargs*)

Get value for an option.

Parameters

- **section** (*string*) – config section.
- **option** (*string*) – option name,
- **default** – this value will be returned, if no such option exists.

Return type string

Returns value of config option.

getbool (*section, option, default=None*)

getfloat (*section, option, default=None*)

getint (*section, option, default=None*)

iter_breakpoints ()

iter_devices ()

iter_mmmaps ()

iter_storages ()

read (**args, **kwargs*)

set (*section, option, value, *args, **kwargs*)

`ducky.config.bool2option(b)`

Get config-file-usable string representation of boolean value.

Parameters **b** (*bool*) – value to convert.

Return type string

Returns yes if input is True, no otherwise.

2.3 ducky.console module

class `ducky.console.ConsoleConnection` (*cid, master, stream_in, stream_out*)

Bases: object

boot ()

die (*exc*)

execute (*cmd*)

halt ()

log (*logger, msg, *args*)

prompt ()

read_input ()

table (*table, **kwargs*)

write (*buff, *args*)

writeln (*buff, *args*)

class `ducky.console.ConsoleMaster` (*machine*)

Bases: object

boot ()

```
connect (slave)
console_id = 0
halt ()
is_registered_command (name)
register_command (name, callback, *args, **kwargs)
register_commands (commands, *args, **kwargs)
unregister_command (name)
class ducky.console.TerminalConsoleConnection (cid, master)
    Bases: ducky.console.ConsoleConnection
    halt ()
ducky.console.cmd_help (console, cmd)
    List all available command and their descriptions
```

2.4 ducky.cpu package

2.4.1 Subpackages

ducky.cpu.coprocessor package

Submodules

ducky.cpu.coprocessor.control module

```
class ducky.cpu.coprocessor.control.ControlCoproprocessor (core)
    Bases: ducky.interfaces.ISnapshotable, ducky.cpu.coprocessor.Coproprocessor
    read (r)
    read_cr0 ()
    read_cr1 ()
    read_cr2 ()
    read_cr3 ()
    write (r, value)
    write_cr1 (address)
    write_cr2 (address)
    write_cr3 (value)
class ducky.cpu.coprocessor.control.ControlRegisters
    Bases: enum.IntEnum
    CR0 = 0
    CR1 = 1
    CR2 = 2
    CR3 = 3
```

```

    _member_map_ = OrderedDict([(‘CR0’, <ControlRegisters.CR0: 0>), (‘CR1’, <ControlRegisters.CR1: 1>), (‘CR2’, <ControlRegisters.CR2: 2>), (‘CR3’, <ControlRegisters.CR3: 3>)])
    _member_names_ = [‘CR0’, ‘CR1’, ‘CR2’, ‘CR3’]
    _member_type_ = int
    alias of int
    _value2member_map_ = {0: <ControlRegisters.CR0: 0>, 1: <ControlRegisters.CR1: 1>, 2: <ControlRegisters.CR2: 2>, 3: <ControlRegisters.CR3: 3>}
class ducky.cpu.coprocessor.control.CoreFlags
    Bases: ducky.util.Flags
    _flags = [‘pt_enabled’, ‘jit’, ‘vmdebug’]
    _labels = ‘PJV’

```

ducky.cpu.coprocessor.math_copro module

Stack-based coprocessor, providing several arithmetic operations with “long” numbers.

Coprocessor’s instructions operates on a stack of (by default) 8 slots. Operations to move values between math stack and registers/data stack are also available.

In the following documentation several different data types are used:

- `int` - standard *word*, 32-bit wide integer
- `long` - long integer, 64-bit wide

Unless said otherwise, instruction takes its arguments from the stack, removing the values in the process, and pushes the result - if any - back on the stack.

```

class ducky.cpu.coprocessor.math_copro.ADDL(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH
    static execute(core, inst)
    mnemonic = ‘addl’
    opcode = 32
    operands = [‘]
class ducky.cpu.coprocessor.math_copro.DECL(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH
    Decrement top of the stack by one.
    static execute(core, inst)
    mnemonic = ‘decl’
    opcode = 31
    operands = [‘]
class ducky.cpu.coprocessor.math_copro.DIVL(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH
    Divide the value below the top of the math stack by the topmost value.
    static execute(core, inst)
    mnemonic = ‘divl’
    opcode = 11

```

```
    operands = ['']

class ducky.cpu.coprocessor.math_copro.DROP (instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    static execute (core, inst)

    mnemonic = 'drop'

    opcode = 23

    operands = ['']

class ducky.cpu.coprocessor.math_copro.DUP (instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    static execute (core, inst)

    mnemonic = 'dup'

    opcode = 20

    operands = ['']

class ducky.cpu.coprocessor.math_copro.DUP2 (instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    static execute (core, inst)

    mnemonic = 'dup2'

    opcode = 21

    operands = ['']

class ducky.cpu.coprocessor.math_copro.Descriptor_MATH (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_R

    static assemble_operands (ctx, inst, operands)

    static disassemble_operands (logger, inst)

    operands = ''

exception ducky.cpu.coprocessor.math_copro.EmptyMathStackError (*args, **kwargs)
    Bases: ducky.errors.CoprocessorError

    Raised when operation expects at least one value on math stack but stack is empty.

exception ducky.cpu.coprocessor.math_copro.FullMathStackError (*args, **kwargs)
    Bases: ducky.errors.CoprocessorError

    Raised when operation tries to put value on math stack but there is no empty spot available.

class ducky.cpu.coprocessor.math_copro.INCL (instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    Increment top of the stack by one.

    static execute (core, inst)

    mnemonic = 'incl'

    opcode = 30

    operands = ['']
```

```

class ducky.cpu.coprocessor.math_copro.LOAD (instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    Merge two registers together, and make the result new TOS.

    static assemble_operands (ctx, inst, operands)

    static disassemble_operands (logger, inst)

    static execute (core, inst)

    mnemonic = 'load'

    opcode = 9

    operands = ['r', 'r']

class ducky.cpu.coprocessor.math_copro.LOADUW (instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    Take a value from register, extend it to long, and make the result TOS.

    static assemble_operands (ctx, inst, operands)

    static disassemble_operands (logger, inst)

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'loaduw'

    opcode = 5

    operands = ['r']

class ducky.cpu.coprocessor.math_copro.LOADW (instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    Take a value from register, extend it to long, and make the result TOS.

    static assemble_operands (ctx, inst, operands)

    static disassemble_operands (logger, inst)

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'loadw'

    opcode = 4

    operands = ['r']

class ducky.cpu.coprocessor.math_copro.MODL (instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    static execute (core, inst)

    mnemonic = 'modl'

    opcode = 12

    operands = ['']

class ducky.cpu.coprocessor.math_copro.MULL (instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    Multiply two top-most numbers on the stack.

```

```

static execute (core, inst)

static jit (core, inst)

mnemonic = 'mull'

opcode = 10

operands = ['']

class ducky.cpu.coprocessor.math_copro.MathCoproprocessor (core, *args, **kwargs)
    Bases: ducky.interfaces.ISnapshotable, ducky.cpu.coprocessor.Coprocessor
    Coprocessor itself, includes its register set ("math stack").

        Parameters core (ducky.cpu.CPUCore) – CPU core coprocessor belongs to.

    dump_stack ()
        Log content of the stack using parent's DEBUG method.

    extend_with_push (u32)

    load_state (state)

    save_state (parent)

    sign_extend_with_push (i32)

class ducky.cpu.coprocessor.math_copro.MathCoproprocessorInstructionSet
    Bases: ducky.cpu.instructions.InstructionSet
    Math coprocessor's instruction set.

    instruction_set_id = 1

    instructions = [<ducky.cpu.coprocessor.math_copro.ADDL object>, <ducky.cpu.coprocessor.

    opcode_desc_map = {<MathCoproprocessorOpcodes.POPW: 0>: <ducky.cpu.coprocessor.math_c

    opcode_encoding_map = {<MathCoproprocessorOpcodes.POPW: 0>: <class 'ducky.cpu.instruct

    opcodes
        alias of MathCoproprocessorOpcodes

class ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes
    Bases: enum.IntEnum
    Math coprocessor's instruction opcodes.

    ADDL = 32

    DECL = 31

    DIVL = 11

    DROP = 23

    DUP = 20

    DUP2 = 21

    INCL = 30

    LOAD = 9

    LOADUW = 5

    LOADW = 4

    MODL = 12

```

2.4. ducky.cpu package 37

```
class ducky.cpu.coprocessor.math_copro.POPW(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    Pop the int from data stack, extend it to long, and make the result TOS.

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'popw'

    opcode = 0

    operands = ['']

class ducky.cpu.coprocessor.math_copro.PUSHL(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    Push the TOS on the data stack.

    static execute (core, inst)

    mnemonic = 'pushl'

    opcode = 8

    operands = ['']

class ducky.cpu.coprocessor.math_copro.PUSHW(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    Downsize TOS to int, and push the result on the data stack.

    static execute (core, inst)

    mnemonic = 'pushw'

    opcode = 2

    operands = ['']

class ducky.cpu.coprocessor.math_copro.RegisterSet(core)
    Bases: ducky.interfaces.ISnapshotable

    Math stack wrapping class. Provides basic push/pop access, and direct access to a top of the stack.

    Parameters core (ducky.cpu.CPUCore) – CPU core registers belong to.

    load_state (state)

    pop ()
        Pop the top value from stack and return it.

        Raises ducky.cpu.coprocessor.math_copro.EmptyMathStackError – if there
            are no values on the stack.

    push (v)
        Push new value on top of the stack.

        Raises ducky.cpu.coprocessor.math_copro.FullMathStackError – if there is
            no space available on the stack.

    save_state (parent)

    tos ()
        Return the top of the stack, without removing it from a stack.
```

Raises `ducky.cpu.coprocessor.math_copro.EmptyMathStackError` – if there are no values on the stack.

tos1()

Return the item below the top of the stack, without removing it from a stack.

Raises `ducky.cpu.coprocessor.math_copro.EmptyMathStackError` – if there are no values on the stack.

```
class ducky.cpu.coprocessor.math_copro.SAVE(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH
    Store TOS in two registers.

    static assemble_operands (ctx, inst, operands)
    static disassemble_operands (logger, inst)
    static execute (core, inst)
    static jit (core, inst)
    mnemonic = 'save'
    opcode = 7
    operands = ['r', 'r']

class ducky.cpu.coprocessor.math_copro.SAVEW(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH
    Downsize TOS to int, and store the result in register.

    static assemble_operands (ctx, inst, operands)
    static disassemble_operands (logger, inst)
    static execute (core, inst)
    static jit (core, inst)
    mnemonic = 'savew'
    opcode = 3
    operands = ['r']

ducky.cpu.coprocessor.math_copro.STACK_DEPTH = 8
    Number of available spots on the math stack.

class ducky.cpu.coprocessor.math_copro.SWP(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    static execute (core, inst)
    mnemonic = 'swpl'
    opcode = 22
    operands = ['']

class ducky.cpu.coprocessor.math_copro.SYMDIVL(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH
    The same operation like DIVL but provides symmetric results.

    static execute (core, inst)
    mnemonic = 'symdivl'
```

```
opcode = 13
operands = ['']

class ducky.cpu.coprocessor.math_copro.SYMMODL(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    static execute (core, inst)

    mnemonic = 'symmodl'

    opcode = 14
    operands = ['']

class ducky.cpu.coprocessor.math_copro.UDIVL(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    Divide the value below the top of the math stack by the topmost value.

    static execute (core, inst)

    mnemonic = 'udivl'

    opcode = 15
    operands = ['']

class ducky.cpu.coprocessor.math_copro.UMODL(instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH

    static execute (core, inst)

    mnemonic = 'umodl'

    opcode = 16
    operands = ['']
```

Module contents

Coprocessors are intended to extend operations of CPUs. They are optional, and can cover wide range of operations, e.g. floating point arithmetic, encryption, or graphics. They are always attached to a CPU core, and may contain and use internal resources, e.g. their very own register sets, machine's memory, or their parent's caches.

```
class ducky.cpu.coprocessor.Coprocessor(core)
    Bases: object

    Base class for per-core coprocessors.
```

2.4.2 Submodules

ducky.cpu.instructions module

```
class ducky.cpu.instructions.ADD(instruction_set)
    Bases: ducky.cpu.instructions._BINOP

    static jit (core, inst)

    mnemonic = 'add'

    opcode = 28
```

```

    operands = ['r', 'ri']

class ducky.cpu.instructions.AND(instruction_set)
    Bases: ducky.cpu.instructions._BITOP

    static jit (core, inst)

    mnemonic = 'and'

    opcode = 34

    operands = ['r', 'ri']

class ducky.cpu.instructions.BE(instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'be'

    operands = ['ri']

class ducky.cpu.instructions.BG(instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'bg'

    operands = ['ri']

class ducky.cpu.instructions.BGE(instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'bge'

    operands = ['ri']

class ducky.cpu.instructions.BL(instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'bl'

    operands = ['ri']

class ducky.cpu.instructions.BLE(instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'ble'

    operands = ['ri']

class ducky.cpu.instructions.BNE(instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'bne'

    operands = ['ri']

class ducky.cpu.instructions.BNO(instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'bno'

    operands = ['ri']

class ducky.cpu.instructions.BNS(instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'bns'

    operands = ['ri']

```

```
class ducky.cpu.instructions.BNZ (instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'bnz'
    operands = ['ri']

class ducky.cpu.instructions.BO (instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'bo'
    operands = ['ri']

class ducky.cpu.instructions.BS (instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'bs'
    operands = ['ri']

class ducky.cpu.instructions.BZ (instruction_set)
    Bases: ducky.cpu.instructions._BRANCH

    mnemonic = 'bz'
    operands = ['ri']

class ducky.cpu.instructions.CALL (instruction_set)
    Bases: ducky.cpu.instructions._JUMP

    static execute (core, inst)
    static jit (core, inst)
    mnemonic = 'call'
    opcode = 15
    operands = ['ri']

class ducky.cpu.instructions.CAS (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    static assemble_operands (ctx, inst, operands)
    static disassemble_operands (logger, inst)
    encoding
        alias of EncodingA
    static execute (core, inst)
    mnemonic = 'cas'
    opcode = 7
    operands = ['r', 'r', 'r']

class ducky.cpu.instructions.CLI (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    encoding
        alias of EncodingI
    static execute (core, inst)
    static jit (core, inst)
```

```

    mnemonic = 'cli'
    opcode = 17
    operands = []
class ducky.cpu.instructions.CMP(instruction_set)
    Bases: ducky.cpu.instructions._CMP
    static execute(core, inst)
    static jit(core, inst)
    mnemonic = 'cmp'
    opcode = 47
    operands = ['r', 'ri']
class ducky.cpu.instructions.CMPU(instruction_set)
    Bases: ducky.cpu.instructions._CMP
    static execute(core, inst)
    static jit(core, inst)
    mnemonic = 'cmpu'
    opcode = 48
    operands = ['r', 'ri']
class ducky.cpu.instructions.CTR(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_R
    encoding
        alias of EncodingR
    static execute(core, inst)
    static jit(core, inst)
    mnemonic = 'ctr'
    opcode = 60
    operands = ['r', 'r']
class ducky.cpu.instructions.CTW(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_R
    encoding
        alias of EncodingR
    static execute(core, inst)
    static jit(core, inst)
    mnemonic = 'ctw'
    opcode = 61
    operands = ['r', 'r']
class ducky.cpu.instructions.DEC(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R
    static execute(core, inst)

```

```
    static jit (core, inst)
    mnemonic = 'dec'
    opcode = 27
    operands = ['r']
class ducky.cpu.instructions.DIV (instruction_set)
    Bases: ducky.cpu.instructions._BINOP
    static jit (core, inst)
    mnemonic = 'div'
    opcode = 31
    operands = ['r', 'ri']
class ducky.cpu.instructions.Descriptor (instruction_set)
    Bases: object
    _expand_operands ()
    static _match_operand_type (allowed, operand)
    static assemble_operands (ctx, inst, operands)
    classmethod disassemble_mnemonic (inst)
    static disassemble_operands (logger, inst)
    static emit_instruction (ctx, desc, operands)
    encoding
        alias of EncodingR
    static execute (core, inst)
    static fill_reloc_slot (inst, slot)
    inst_aligned = False
    static jit (core, inst)
    mnemonic = None
    opcode = None
    operands = None
    relative_address = False
class ducky.cpu.instructions.Descriptor_R (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor
    static assemble_operands (ctx, inst, operands)
    static disassemble_operands (logger, inst)
    encoding
        alias of EncodingR
    operands = 'r'
class ducky.cpu.instructions.Descriptor_RI (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor
    static assemble_operands (ctx, inst, operands)
```

```

    static disassemble_operands (logger, inst)

    encoding
        alias of EncodingI

    operands = 'ri'

class ducky.cpu.instructions.Descriptor_R_I (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    static assemble_operands (ctx, inst, operands)

    static disassemble_operands (logger, inst)

    encoding
        alias of EncodingI

    operands = 'r,i'

class ducky.cpu.instructions.Descriptor_R_R (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    static assemble_operands (ctx, inst, operands)

    static disassemble_operands (logger, inst)

    encoding
        alias of EncodingR

    operands = 'r,r'

class ducky.cpu.instructions.Descriptor_R_RI (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    static assemble_operands (ctx, inst, operands)

    static disassemble_operands (logger, inst)

    encoding
        alias of EncodingR

    operands = 'r,ri'

class ducky.cpu.instructions.DuckyInstructionSet
    Bases: ducky.cpu.instructions.InstructionSet

    instruction_set_id = 0

    instructions = [<ducky.cpu.instructions.NOP object>, <ducky.cpu.instructions.INT object>, <ducky.cpu.instructions.
opcode_desc_map = {<DuckyOpcodes.NOP: 0>: <ducky.cpu.instructions.NOP object>, <DuckyOpcodes.LW: 1>: <duc
opcode_encoding_map = {<DuckyOpcodes.NOP: 0>: <class 'ducky.cpu.instructions.EncodingI'>, <DuckyOpcodes.LV
opcodes
    alias of DuckyOpcodes

class ducky.cpu.instructions.DuckyOpcodes
    Bases: enum.IntEnum

    ADD = 28

    AND = 34

    BRANCH = 50

    CALL = 15

```

CAS = 7
CLI = 17
CMP = 47
CMPU = 48
CTR = 60
CTW = 61
DEC = 27
DIV = 31
FPTC = 62
HLT = 20
IDLE = 21
INC = 26
INT = 13
IPI = 23
J = 46
LA = 8
LB = 3
LI = 9
LIU = 10
LPM = 22
LS = 2
LW = 1
MOD = 33
MOV = 11
MUL = 30
NOP = 0
NOT = 37
OR = 35
POP = 25
PUSH = 24
RET = 16
RETINT = 14
RST = 19
SELECT = 51
SET = 49
SHL = 38


```
immediate
    Structure/Union member

immediate_flag
    Structure/Union member

opcode
    Structure/Union member

reg
    Structure/Union member

static sign_extend_immediate (logger, inst)

value
    Structure/Union member

class ducky.cpu.instructions.EncodingContext (logger)
    Bases: ducky.util.LoggingCapable, object
    _u32_to_encoding_pypy (u, encoding)
    _u32_to_encoding_python (u, encoding)
    decode (instr_set, inst, core=None)
    encode (inst, field, size, value, raise_on_large_value=False)

class ducky.cpu.instructions.EncodingI
    Bases: _ctypes.Structure
    _fields_ = [('opcode', <class 'ctypes.c_uint'>, 6), ('reg', <class 'ctypes.c_uint'>, 5), ('immediate_flag', <class 'ctypes.c_
_pack_ = 0
    static fill_reloc_slot (inst, slot)
    immediate
        Structure/Union member
    immediate_flag
        Structure/Union member
    opcode
        Structure/Union member
    reg
        Structure/Union member
    static sign_extend_immediate (logger, inst)

class ducky.cpu.instructions.EncodingR
    Bases: _ctypes.Structure
    _fields_ = [('opcode', <class 'ctypes.c_uint'>, 6), ('reg1', <class 'ctypes.c_uint'>, 5), ('reg2', <class 'ctypes.c_uint'>, 5),
_pack_ = 0
    static fill_reloc_slot (inst, slot)
    immediate
        Structure/Union member
    immediate_flag
        Structure/Union member
```

```

    opcode
        Structure/Union member

    reg1
        Structure/Union member

    reg2
        Structure/Union member

    static sign_extend_immediate (logger, inst)
class ducky.cpu.instructions.EncodingS
    Bases: _ctypes.Structure

    _fields_ = [('opcode', <class 'ctypes.c_uint'>, 6), ('reg1', <class 'ctypes.c_uint'>, 5), ('reg2', <class 'ctypes.c_uint'>, 5),
    _pack_ = 0

    static fill_reloc_slot (inst, slot)

    flag
        Structure/Union member

    immediate
        Structure/Union member

    immediate_flag
        Structure/Union member

    opcode
        Structure/Union member

    reg1
        Structure/Union member

    reg2
        Structure/Union member

    static sign_extend_immediate (logger, inst)

    value
        Structure/Union member

class ducky.cpu.instructions.FPTC (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    encoding
        alias of EncodingI

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'fptc'

    opcode = 62

    operands = []

class ducky.cpu.instructions.HLT (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_RI

    static execute (core, inst)

    mnemonic = 'hlt'

    opcode = 20

```

```
    operands = ['ri']

class ducky.cpu.instructions.IDLE(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    encoding
        alias of EncodingI

    static execute (core, inst)

    mnemonic = 'idle'

    opcode = 21

    operands = []

ducky.cpu.instructions.IE_FLAG(n)
ducky.cpu.instructions.IE_IMM(n, l)
ducky.cpu.instructions.IE_OPCODE()
ducky.cpu.instructions.IE_REG(n)

class ducky.cpu.instructions.INC(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'inc'

    opcode = 26

    operands = ['r']

class ducky.cpu.instructions.INT(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_RI

    static execute (core, inst)

    mnemonic = 'int'

    opcode = 13

    operands = ['ri']

class ducky.cpu.instructions.IPI(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_RI

    static execute (core, inst)

    mnemonic = 'ipi'

    opcode = 23

    operands = ['r', 'ri']

class ducky.cpu.instructions.InstructionSet
    Bases: object

    classmethod disassemble_instruction (logger, inst)

    classmethod init ()

    instruction_set_id = None

    instructions = []
```

```

    opcodes = None

class ducky.cpu.instructions.InstructionSetMetaclass (name, bases, dict)
    Bases: type

class ducky.cpu.instructions.J (instruction_set)
    Bases: ducky.cpu.instructions._JUMP

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'j'

    opcode = 46

    operands = ['ri']

ducky.cpu.instructions.JUMP (core, inst, reg)

class ducky.cpu.instructions.LA (instruction_set)
    Bases: ducky.cpu.instructions._LOAD_IMM

    static jit (core, inst)

    classmethod load (core, inst)

    mnemonic = 'la'

    opcode = 8

    operands = ['r', 'i']

    relative_address = True

class ducky.cpu.instructions.LB (instruction_set)
    Bases: ducky.cpu.instructions._LOAD

    mnemonic = 'lb'

    opcode = 3

    operands = ['r', 'a']

class ducky.cpu.instructions.LI (instruction_set)
    Bases: ducky.cpu.instructions._LOAD_IMM

    static jit (core, inst)

    classmethod load (core, inst)

    mnemonic = 'li'

    opcode = 9

    operands = ['r', 'i']

class ducky.cpu.instructions.LIU (instruction_set)
    Bases: ducky.cpu.instructions._LOAD_IMM

    static jit (core, inst)

    classmethod load (core, inst)

    mnemonic = 'liu'

    opcode = 10

    operands = ['r', 'i']

```

```
class ducky.cpu.instructions.LPM(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    encoding
        alias of EncodingI

    static execute (core, inst)

    mnemonic = 'lpm'

    opcode = 22

    operands = []

class ducky.cpu.instructions.LS(instruction_set)
    Bases: ducky.cpu.instructions._LOAD

    mnemonic = 'ls'

    opcode = 2

    operands = ['r', 'a']

class ducky.cpu.instructions.LW(instruction_set)
    Bases: ducky.cpu.instructions._LOAD

    mnemonic = 'lw'

    opcode = 1

    operands = ['r', 'a']

class ducky.cpu.instructions.MOD(instruction_set)
    Bases: ducky.cpu.instructions._BINOP

    static jit (core, inst)

    mnemonic = 'mod'

    opcode = 33

    operands = ['r', 'ri']

class ducky.cpu.instructions.MOV(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_R

    encoding
        alias of EncodingR

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'mov'

    opcode = 11

    operands = ['r', 'r']

class ducky.cpu.instructions.MUL(instruction_set)
    Bases: ducky.cpu.instructions._BINOP

    static jit (core, inst)

    mnemonic = 'mul'

    opcode = 30

    operands = ['r', 'ri']
```

```

class ducky.cpu.instructions.NOP(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    encoding
        alias of EncodingI

    static execute(core, inst)

    mnemonic = 'nop'

    opcode = 0

    operands = []

class ducky.cpu.instructions.NOT(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R

    encoding
        alias of EncodingR

    static execute(core, inst)

    mnemonic = 'not'

    opcode = 37

    operands = ['r']

class ducky.cpu.instructions.OR(instruction_set)
    Bases: ducky.cpu.instructions._BITOP

    static jit(core, inst)

    mnemonic = 'or'

    opcode = 35

    operands = ['r', 'ri']

class ducky.cpu.instructions.POP(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R

    static execute(core, inst)

    static jit(core, inst)

    mnemonic = 'pop'

    opcode = 25

    operands = ['r']

class ducky.cpu.instructions.PUSH(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_RI

    static execute(core, inst)

    static jit(core, inst)

    mnemonic = 'push'

    opcode = 24

    operands = ['ri']

class ducky.cpu.instructions.RET(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

```

```
    encoding
        alias of EncodingI

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'ret'

    opcode = 16

    operands = []

class ducky.cpu.instructions.RETINT (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    encoding
        alias of EncodingI

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'retint'

    opcode = 14

    operands = []

ducky.cpu.instructions.RI_ADDR (core, inst, reg)

ducky.cpu.instructions.RI_VAL (core, inst, reg, sign_extend=True)

class ducky.cpu.instructions.RST (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    encoding
        alias of EncodingI

    static execute (core, inst)

    mnemonic = 'rst'

    opcode = 19

    operands = []

class ducky.cpu.instructions.SELE (instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'sele'

    operands = ['r', 'ri']

class ducky.cpu.instructions.SELG (instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'selg'

    operands = ['r', 'ri']

class ducky.cpu.instructions.SELGE (instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'selge'

    operands = ['r', 'ri']
```

```

class ducky.cpu.instructions.SELL(instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'sell'
    operands = ['r', 'ri']

class ducky.cpu.instructions.SELLE(instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'selle'
    operands = ['r', 'ri']

class ducky.cpu.instructions.SELNE(instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'selne'
    operands = ['r', 'ri']

class ducky.cpu.instructions.SELNO(instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'selno'
    operands = ['r', 'ri']

class ducky.cpu.instructions.SELNS(instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'selns'
    operands = ['r', 'ri']

class ducky.cpu.instructions.SELNZ(instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'selnz'
    operands = ['r', 'ri']

class ducky.cpu.instructions.SELO(instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'selo'
    operands = ['r', 'ri']

class ducky.cpu.instructions.SELS(instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'sels'
    operands = ['r', 'ri']

class ducky.cpu.instructions.SELZ(instruction_set)
    Bases: ducky.cpu.instructions._SELECT

    mnemonic = 'selz'
    operands = ['r', 'ri']

class ducky.cpu.instructions.SETE(instruction_set)
    Bases: ducky.cpu.instructions._SET

    mnemonic = 'sete'

```

```
    operands = ['r']  
  
class ducky.cpu.instructions.SETG(instruction_set)  
    Bases: ducky.cpu.instructions._SET  
  
    mnemonic = 'setg'  
    operands = ['r']  
  
class ducky.cpu.instructions.SETGE(instruction_set)  
    Bases: ducky.cpu.instructions._SET  
  
    mnemonic = 'setge'  
    operands = ['r']  
  
class ducky.cpu.instructions.SETL(instruction_set)  
    Bases: ducky.cpu.instructions._SET  
  
    mnemonic = 'setl'  
    operands = ['r']  
  
class ducky.cpu.instructions.SETLE(instruction_set)  
    Bases: ducky.cpu.instructions._SET  
  
    mnemonic = 'setle'  
    operands = ['r']  
  
class ducky.cpu.instructions.SETNE(instruction_set)  
    Bases: ducky.cpu.instructions._SET  
  
    mnemonic = 'setne'  
    operands = ['r']  
  
class ducky.cpu.instructions.SETNO(instruction_set)  
    Bases: ducky.cpu.instructions._SET  
  
    mnemonic = 'setno'  
    operands = ['r']  
  
class ducky.cpu.instructions.SETNS(instruction_set)  
    Bases: ducky.cpu.instructions._SET  
  
    mnemonic = 'setns'  
    operands = ['r']  
  
class ducky.cpu.instructions.SETNZ(instruction_set)  
    Bases: ducky.cpu.instructions._SET  
  
    mnemonic = 'setnz'  
    operands = ['r']  
  
class ducky.cpu.instructions.SETO(instruction_set)  
    Bases: ducky.cpu.instructions._SET  
  
    mnemonic = 'seto'  
    operands = ['r']  
  
class ducky.cpu.instructions.SETS(instruction_set)  
    Bases: ducky.cpu.instructions._SET
```

```

    mnemonic = 'sets'
    operands = ['r']
class ducky.cpu.instructions.SETZ(instruction_set)
    Bases: ducky.cpu.instructions._SET
    mnemonic = 'setz'
    operands = ['r']
class ducky.cpu.instructions.SHL(instruction_set)
    Bases: ducky.cpu.instructions._BITOP
    static jit (core, inst)
    mnemonic = 'shifl'
    opcode = 38
    operands = ['r', 'ri']
class ducky.cpu.instructions.SHR(instruction_set)
    Bases: ducky.cpu.instructions._BITOP
    static jit (core, inst)
    mnemonic = 'shiftr'
    opcode = 39
    operands = ['r', 'ri']
class ducky.cpu.instructions.SHRS(instruction_set)
    Bases: ducky.cpu.instructions._BITOP
    static jit (core, inst)
    mnemonic = 'shiftrs'
    opcode = 40
    operands = ['r', 'ri']
class ducky.cpu.instructions.SIS(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_RI
    static execute (core, inst)
    static jit (core, inst)
    mnemonic = 'sis'
    opcode = 63
    operands = ['ri']
class ducky.cpu.instructions.STB(instruction_set)
    Bases: ducky.cpu.instructions._STORE
    mnemonic = 'stb'
    opcode = 6
    operands = ['a', 'r']
class ducky.cpu.instructions.STI(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

```

```
    encoding
        alias of EncodingI

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'sti'

    opcode = 18

    operands = []

class ducky.cpu.instructions.STS (instruction_set)
    Bases: ducky.cpu.instructions._STORE

    mnemonic = 'sts'

    opcode = 5

    operands = ['a', 'r']

class ducky.cpu.instructions.STW (instruction_set)
    Bases: ducky.cpu.instructions._STORE

    mnemonic = 'stw'

    opcode = 4

    operands = ['a', 'r']

class ducky.cpu.instructions.SUB (instruction_set)
    Bases: ducky.cpu.instructions._BINOP

    static jit (core, inst)

    mnemonic = 'sub'

    opcode = 29

    operands = ['r', 'ri']

class ducky.cpu.instructions.SWP (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_R

    encoding
        alias of EncodingR

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'swp'

    opcode = 12

    operands = ['r', 'r']

class ducky.cpu.instructions.UDIV (instruction_set)
    Bases: ducky.cpu.instructions._BINOP

    static jit (core, inst)

    mnemonic = 'udiv'

    opcode = 32

    operands = ['r', 'ri']
```

```

ducky.cpu.instructions.UINT20_FMT(i)

class ducky.cpu.instructions.XOR(instruction_set)
    Bases: ducky.cpu.instructions._BITOP

    static jit(core, inst)

    mnemonic = 'xor'

    opcode = 36

    operands = ['r', 'ri']

class ducky.cpu.instructions._BINOP(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_RI

    encoding
        alias of EncodingR

    static execute(core, inst)

class ducky.cpu.instructions._BITOP(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_RI

    encoding
        alias of EncodingR

    static execute(core, inst)

class ducky.cpu.instructions._BRANCH(instruction_set)
    Bases: ducky.cpu.instructions._COND

    classmethod assemble_operands(ctx, inst, operands)

    static disassemble_mnemonic(inst)

    static disassemble_operands(logger, inst)

    encoding
        alias of EncodingC

    static execute(core, inst)

    static fill_reloc_slot(inst, slot)

    inst_aligned = True

    static jit(core, inst)

    opcode = 50

    operands = 'ri'

    relative_address = True

class ducky.cpu.instructions._CMP(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_RI

    encoding
        alias of EncodingR

    static evaluate(core, x, y, signed=True)
        Compare two numbers, and update relevant flags. Signed comparison is used unless signed is False.
        All arithmetic flags are set to zero before the relevant ones are set.

        ○ flag is reset like the others, therefore caller has to take care of it's setting if it's required to set it.

```

Parameters

- **x** (*u32*) – left hand number
- **y** (*u32*) – right hand number
- **signed** (*bool*) – use signed, defaults to `True`

```
class ducky.cpu.instructions._COND (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    FLAGS = ['arith_equal', 'arith_zero', 'arith_overflow', 'arith_sign', 'l', 'g']
    GFLAGS = [0, 1, 2, 3]
    MNEMONICS = ['e', 'z', 'o', 's', 'g', 'l']

    static evaluate (core, inst)

    static set_condition (ctx, inst, flag, value)

class ducky.cpu.instructions._JUMP (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    static assemble_operands (ctx, inst, operands)

    static disassemble_operands (logger, inst)

    encoding
        alias of EncodingI

    inst_aligned = True

    operands = 'ri'

    relative_address = True

class ducky.cpu.instructions._LOAD (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    static assemble_operands (ctx, inst, operands)

    static disassemble_operands (logger, inst)

    encoding
        alias of EncodingR

    static execute (core, inst)

    static jit (core, inst)

    operands = 'ra'

class ducky.cpu.instructions._LOAD_IMM (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_I

    classmethod execute (core, inst)

    classmethod load (core, inst)

class ducky.cpu.instructions._SELECT (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor

    classmethod assemble_operands (ctx, inst, operands)

    static disassemble_mnemonic (inst)

    static disassemble_operands (logger, inst)

    encoding
        alias of EncodingS
```

```

    static execute (core, inst)
    static jit (core, inst)
    opcode = 51
    operands = 'r,ri'
class ducky.cpu.instructions._SET (instruction_set)
    Bases: ducky.cpu.instructions._COND
    classmethod assemble_operands (ctx, inst, operands)
    static disassemble_mnemonic (inst)
    static disassemble_operands (logger, inst)
    encoding
        alias of EncodingS
    static execute (core, inst)
    opcode = 49
    operands = 'r'
class ducky.cpu.instructions._STORE (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor
    static assemble_operands (ctx, inst, operands)
    static disassemble_operands (logger, inst)
    encoding
        alias of EncodingR
    static execute (core, inst)
    static jit (core, inst)
    operands = 'a,r'
ducky.cpu.instructions.encoding_to_u32 (inst)
ducky.cpu.instructions.get_instruction_set (i, exc=None)
ducky.cpu.instructions.update_arith_flags (core, reg)
    Set relevant arithmetic flags according to content of registers. Flags are set to zero at the beginning, then content
    of each register is examined, and S and Z flags are set.
    E flag is not touched, O flag is set to zero.

```

Parameters `reg` (`u32_t`) – register

ducky.cpu.registers module

```

class ducky.cpu.registers.RegisterSet
    Bases: list
class ducky.cpu.registers.Registers
    Bases: enum.IntEnum
    CNT = 33
    FP = 30
    IP = 32

```


`_member_type_`
alias of `int`

`_value2member_map_` = {0: <Registers.R00: 0>, 1: <Registers.R01: 1>, 2: <Registers.R02: 2>, 3: <Registers.R03: 3>},

2.4.3 Module contents

class `ducky.cpu.CPU` (*machine, cpuid, memory_controller, cores=1*)

Bases: `ducky.interfaces.ISnapshotable`, `ducky.interfaces.IMachineWorker`

`boot` ()

`die` (*exc*)

`halt` ()

`load_state` (*state*)

`on_core_alive` (*core*)

Triggered when one of cores goes alive.

`on_core_halted` (*core*)

Signal CPU that one of cores is no longer alive.

`on_core_running` (*core*)

Signal CPU that one of cores is now running.

`on_core_suspended` (*core*)

Signal CPU that one of cores is now suspended.

`save_state` (*parent*)

`suspend` ()

`wake_up` ()

class `ducky.cpu.CPUCore` (*coreid, cpu, memory_controller*)

Bases: `ducky.interfaces.ISnapshotable`, `ducky.interfaces.IMachineWorker`

This class represents the main workhorse, one of CPU cores. Reads instructions, executes them, has registers, caches, handles interrupts, ...

Parameters

- **`coreid`** (*int*) – id of this core. Usually, it's its serial number but it has no special meaning.
- **`cpu`** (`ducky.cpu.CPU`) – CPU that owns this core.
- **`memory_controller`** (`ducky.mm.MemoryController`) – use this controller to access main memory.

`FP` ()

`IP` ()

`REG` (*reg*)

`SP` ()

`__CPUCore__get_flags` ()

`__CPUCore__set_flags` (*flags*)

`_enter_exception (index, *args)`

Prepare CPU for handling exception routine. CPU core loads new IP and SP from proper entry of EVT. Old SP and FLAGS are saved on the exception stack, and new call frame is created. Privileged mode flag is set, hardware interrupt flag is cleared.

Then, if exception provides its routine with some arguments, these arguments are pushed on the stack.

Exception stack layout then looks like this (original stack is left untouched):



Parameters

- **index** (*int*) – exception ID - EVT index.
- **args** (*u32_t*) – if present, these values will be pushed onto the stack.

`_exit_exception ()`

Restore CPU state after running an exception routine. Call frame is destroyed, registers are restored. Clearing routine arguments is responsibility of the routine.

`_get_instruction_set ()`

`_handle_exception (exc, index, *args)`

This method provides CPU exception classes with a simple recipe on how to deal with the exception:

- tell processor to start exception dance,
- if the exception is raised again, tell processor to plan double fault routine,
- and if yet another exception is raised, halt the core.

`_handle_python_exception (exc)`

`_raw_pop ()`

Pop value from stack. 4 byte number is read from address in SP, then SP is incremented by four.

Returns popped value

Return type *u32*

`_raw_push (val)`

Push value on stack. SP is decremented by four, and value is written at this new address.

Parameters **val** (*u32*) – value to be pushed

`_set_instruction_set (instr_set)`

boot()

change_runnable_state (*alive=None, running=None, idle=None*)

check_protected_ins()

Raise `AccessViolationError` if core is not running in privileged mode.

This method should be used by instruction handlers that require privileged mode, e.g. protected instructions.

Raises `ducky.errors.PrivilegedInstructionError` – if the core is not in privileged mode

create_frame()

Creates new call stack frame, by performing the following operations:

- push IP
- push FP
- set FP to SP (i.e. SP before this method + 2 pushes)

Stack layout then looks like this:

```
+-----+
|  IPx  |
+-----+
|  FPx  |
+-----+ <= FPy
|  ...  |
+-----+ <= original SP
|  IPy  |
+-----+
|  FPy  |
+-----+ <= SP, FP
|  ...  |
+-----+
```

FP then points to the newly created frame, to the saved FP in particular, and this saved FP points to its predecessor, thus forming a chain.

destroy_frame()

Destroys current call stack frame by popping values from the stack, reversing the list of operations performed by `ducky.cpu.CPUCore.create_frame()`:

- pop FP
- pop IP

After this, FP points to the frame from which the instruction that created the currently destroyed frame was executed, and restored IP points to the next instruction.

Raises `InvalidFrameError` – if frame checking is enabled, current SP is compared with saved FP to see, if the stack was clean before leaving the frame. This error indicated there is some value left on stack when `ret` or `retint` were executed. Usually, this signals missing `pop` to complement one of previous “push”es.

die (*exc*)

do_step (*ip, regset*)

flags

halt()

has_coprocessor (*name*)

init_debug_set ()

instruction_set

irq (*index*)

This is a wrapper for `_enter_exception`, for device drivers to call when hardware interrupt arrives.

Parameters *index* (*int*) – exception ID - EVT index

load_state (*state*)

pop (**regs*)

pop_flags ()

pop_frame ()

push (**regs*)

push_flags ()

reset (*new_ip=0*)

Reset core's state. All registers are set to zero, all flags are set to zero, except HWINT flag which is set to one, and IP is set to requested value.

Parameters *new_ip* (*u32_t*) – new IP value, defaults to zero

run ()

save_state (*parent*)

step ()

Perform one “step” - fetch next instruction, increment IP, and execute instruction's code (see `inst_*` methods)

suspend ()

wake_up ()

class `ducky.cpu.CPUCoreState`

Bases: `ducky.snapshot.SnapshotNode`

class `ducky.cpu.CPUState` (**fields*)

Bases: `ducky.snapshot.SnapshotNode`

get_core_state_by_id (*coreid*)

get_core_states ()

class `ducky.cpu.CoreFlags`

Bases: `ducky.util.Flags`

_flags = ['privileged', 'hwint_allowed', 'equal', 'zero', 'overflow', 'sign']

_labels = 'PHEZOS'

`ducky.cpu.DEFAULT_CORE_INST_CACHE_SIZE = 256`

Default size of core instruction cache, in instructions.

`ducky.cpu.DEFAULT_EVT_ADDRESS = 0`

Default EVT address

`ducky.cpu.DEFAULT_PT_ADDRESS = 65536`

Default PT address

```
class ducky.cpu.InstructionCache_Base (mmu, *args, **kwargs)
```

Bases: `ducky.util.LoggingCapable`, `dict`

Simple instruction cache class, based on a dictionary, with a limited size.

Parameters `core` (`ducky.cpu.CPUCore`) – CPU core that owns this cache.

```
class ducky.cpu.InstructionCache_Full (mmu, *args, **kwargs)
```

Bases: `ducky.util.LoggingCapable`, `list`

Simple instruction cache class, based on a list, with unlimited size.

Parameters `core` (`ducky.cpu.CPUCore`) – CPU core that owns this cache.

```
clear()
```

```
class ducky.cpu.InterruptVector (ip=0, sp=0)
```

Bases: `object`

Interrupt vector table entry.

SIZE = 8

```
static load (core, addr)
```

```
class ducky.cpu.MMU (core, memory_controller)
```

Bases: `ducky.interfaces.ISnapshotable`

Memory management unit (aka MMU) provides a single point handling all core's memory operations. All memory reads and writes must go through this unit, which is then responsible for all translations, access control, and caching.

Parameters

- **core** (`ducky.cpu.CPUCore`) – parent core.
- **memory_controller** (`ducky.mm.MemoryController`) – memory controller that provides access to the main memory.
- **memory.force-aligned-access** (`bool`) – if set, MMU will disallow unaligned reads and writes. `False` by default.
- **cpu.pt-address** (`int`) – base address of page table. `ducky.cpu.DEFAULT_PT_ADDRESS` by default.
- **cpu.pt-enabled** (`bool`) – if set, CPU core will start with page table enabled. `False` by default.

```
_check_access (access, addr, align=None)
```

Check attempted access against several criteria:

- PT is enabled - disabled PT implies different set of read/write methods that don't use this method to check access
- access alignment if correct alignment is required
- privileged access implies granted access
- corresponding PTE settings

Parameters

- **access** – read, write or execute.
- **addr** (`int`) – memory address.
- **align** (`int`) – if set, operation is expected to be aligned to this boundary.

Raises

- `ducky.errors.UnalignedAccessError` – when unaligned access is not allowed, but requested.
- `ducky.errors.MemoryAccessError` – when access is denied.

`_debug_wrapper_read(reader, *args, **kwargs)`

`_debug_wrapper_write(writer, *args, **kwargs)`

`_fetch_instr(addr)`

Read instruction from memory. This method is responsible for the real job of fetching instructions and filling the cache.

Parameters `addr` (`u24`) – absolute address to read from

Returns instruction

Return type `InstBinaryFormat_Master`

`_fetch_instr_jit(addr)`

Read instruction from memory. This method is responsible for the real job of fetching instructions and filling the cache.

Parameters `addr` (`u24`) – absolute address to read from

Returns instruction

Return type `InstBinaryFormat_Master`

`_get_pg_ops_dict(address)`

`_get_pg_ops_list(address)`

`_get_pt_enabled()`

`_get_pte(addr)`

Find out PTE for particular physical address. If PTE is not in internal PTE cache, it is fetched from PTE table.

Parameters `addr` (`int`) – memory address.

`_nopt_read_u16(addr)`

`_nopt_read_u32(addr, not_execute=True)`

`_nopt_read_u8(addr)`

`_nopt_write_u16(addr, value)`

`_nopt_write_u32(addr, value)`

`_nopt_write_u8(addr, value)`

`_pt_read_u16(addr)`

`_pt_read_u32(addr, not_execute=True)`

`_pt_read_u8(addr)`

`_pt_write_u16(addr, value)`

`_pt_write_u32(addr, value)`

`_pt_write_u8(addr, value)`

`_set_access_methods()`
Set parent core's memory-access methods to proper shortcuts. Methods named `MEM_{IN, OUT}` {8, 16, 32} will be set to corresponding MMU methods.

`_set_pt_enabled(value)`

`halt()`

`pt_enabled`

`release_ptes()`
Clear internal PTE cache.

`reset()`
Reset MMU. PT will be disabled, and all internal caches will be flushed.

`class ducky.cpu.StackFrame(sp, ip)`
Bases: `object`

`ducky.cpu.do_log_cpu_core_state(core, logger=None, disassemble=True, inst_set=None)`
Log state of a CPU core. Content of its registers, and other interesting or useful internal variables are logged.

Parameters

- **`core`** (`ducky.cpu.CPUCore`) – core whose state should be logged.
- **`logger`** – called for each line of output to actually log it. By default, core's `ducky.cpu.CPUCore.DEBUG()` method is used.

`ducky.cpu.log_cpu_core_state(*args, **kwargs)`
This is a wrapper for `ducky.cpu.do_log_cpu_core_state` function. Its main purpose is to be removed when debug mode is not set, therefore all debug calls of `ducky.cpu.do_log_cpu_core_state` will disappear from code, making such code effectively “quiet”.

2.5 ducky.debugging module

Virtual machine debugging tools - break points, watch points, etc.

Create “point” that's triggered when a condition is satisfied (e.g. processor executes instruction on specified address, memory at specified address was modified, etc. Then, create “action” (e.g. suspend core), and bind both pieces together - when point gets triggered, execute list of actions.

`class ducky.debugging.Action(logger)`
Bases: `object`

Base class of all debugging actions.

Parameters **`logger`** (`logging.Logger`) – logger instance used for logging.

`act(core, point)`
This method is called when “action” is executed. Implement it in child classes to give child actions a functionality.

Parameters

- **`core`** (`ducky.cpu.CPUCore`) – CPU core where point was triggered.
- **`point`** (`ducky.debugging.Point`) – point that was triggered.

`class ducky.debugging.BreakPoint(debugging_set, ip, *args, **kwargs)`
Bases: `ducky.debugging.Point`

`static create_from_config(debugging_set, config, section)`

is_triggered (*core*)

class `ducky.debugging.DebuggingSet` (*core*)

Bases: `object`

__DebuggingSet__check_chain (*stage, chain, clean_triggered=False, *args, **kwargs*)

add_point (*p, chain*)

post_memory (*address=None, read=None*)

post_step ()

pre_memory (*address=None, read=None*)

pre_step ()

remove_point (*p, chain*)

class `ducky.debugging.LogMemoryContentAction` (*logger, address, size*)

Bases: `ducky.debugging.LogValueAction`

When triggered, logs content of a specified location in memory.

Parameters

- **logger** (*logging.Logger*) – logger instance used for logging.
- **address** (*u32_t*) – memory location.
- **size** (*int*) – size of logged number, in bytes.

static create_from_config (*debugging_set, config, section*)

get_message (*core, point*)

get_values (*core, point*)

class `ducky.debugging.LogRegisterContentAction` (*logger, registers*)

Bases: `ducky.debugging.LogValueAction`

When triggered, logs content of a specified register.

Parameters

- **logger** (*logging.Logger*) – logger instance used for logging.
- **registers** (*list*) – list of register names.

static create_from_config (*debugging_set, config, section*)

get_message (*core, point*)

get_values (*core, point*)

class `ducky.debugging.LogValueAction` (*logger, size*)

Bases: `ducky.debugging.Action`

This is the base class for actions that log a numerical values.

Parameters

- **logger** (*logging.Logger*) – logger instance used for logging.
- **size** (*int*) – size of logged number, in bytes.

act (*core, point*)

get_message (*core, point*)

Return message that, formatted with output of `get_values()`, will be shown to user.

Parameters

- **core** (`ducky.cpu.CPUCore`) – core point was triggered on.
- **point** (`ducky.debugging.Point`) – triggered point.

Return type string**Returns** information message.**get_values** (*core, point*)

Prepare dictionary with values for message that will be shown to the user.

Parameters

- **core** (`ducky.cpu.CPUCore`) – core point was triggered on.
- **point** (`ducky.debugging.Point`) – triggered point.

Return type dict**Returns** dictionary that will be passed to message `format()` method.**class** `ducky.debugging.MemoryWatchPoint` (*debugging_set, address, read, *args, **kwargs*)Bases: `ducky.debugging.Point`**static create_from_config** (*debugging_set, config, section*)**is_triggered** (*core, address=None, read=None*)**class** `ducky.debugging.Point` (*debugging_set, active=True, countdown=0*)Bases: `object`

Base class of all debugging points.

Parameters

- **debugging_set** (`ducky.debugging.DebuggingSet`) – debugging set this point belongs to.
- **active** (*bool*) – if not `True`, point is not active and will not trigger.
- **countdown** (*int*) – if greater than zero, point has to trigger `countdown` times before its actions are executed for the first time.

is_triggered (*core, *args, **kwargs*)

Test point's condition.

Parameters **core** (`ducky.cpu.CPUCore`) – core requesting the test.**Return type** bool**Returns** `True` if condition is satisfied.**class** `ducky.debugging.SuspendCoreAction` (*logger*)Bases: `ducky.debugging.Action`

If executed, this action will suspend the CPU core that triggered its parent point.

act (*core, point*)**static create_from_config** (*debugging_set, config, section*)`ducky.debugging.cmd_bp_active` (*console, cmd*)Toggle “active” flag for a breakpoint: `bp-active <id>``ducky.debugging.cmd_bp_add_breakpoint` (*console, cmd*)Create new breakpoint: `bp-break <#cpuid:#coreid> <address> [active] [countdown]`

```
ducky.debugging.cmd_bp_add_memory_watchpoint (console, cmd)
    Create new memory watchpoint: bp-mwatch <#cpuid:#coreid> <address> [rw] [active] [countdown]

ducky.debugging.cmd_bp_list (console, cmd)
    List existing breakpoints

ducky.debugging.cmd_bp_remove (console, cmd)
    Remove breakpoint: bp-remove <id>
```

2.6 ducky.devices package

2.6.1 Submodules

ducky.devices.keyboard module

Keyboard controller - provides events for pressed and released keys.

```
class ducky.devices.keyboard.Backend (machine, name, mmio_address=None, irq=None)
    Bases: ducky.devices.DeviceBackend
    _process_input_events ()
    _read_char ()
    boot ()
    static create_from_config (machine, config, section)
    static create_hdt_entries (logger, config, section)
    halt ()

class ducky.devices.keyboard.ControlMessages
    Bases: enum.IntEnum
    CONTROL_MESSAGE_FIRST = 1024
    HALT = 1025
    _member_map_ = OrderedDict([('CONTROL_MESSAGE_FIRST', <ControlMessages.CONTROL_MESSAGE_FIRST: 1024>), ('HALT', <ControlMessages.HALT: 1025>)])
    _member_names_ = ['CONTROL_MESSAGE_FIRST', 'HALT']
    _member_type_
        alias of int
    _value2member_map_ = {1024: <ControlMessages.CONTROL_MESSAGE_FIRST: 1024>, 1025: <ControlMessages.HALT: 1025>}

class ducky.devices.keyboard.Frontend (machine, name)
    Bases: ducky.devices.DeviceFrontend
    _close_input ()
    _handle_input_error ()
    _handle_raw_input ()
    _open_input ()
    boot ()
    static create_from_config (machine, config, section)
    enqueue_stream (stream)
```

```

    halt ()

class ducky.devices.keyboard.HDTEEntry_Keyboard (logger, config, section)
    Bases: ducky.hdt.HDTEEntry_Device

    _fields_ = [('type', <class 'ctypes.c_ushort'>), ('length', <class 'ctypes.c_ushort'>), ('name_length', <class 'ctypes.c_ushort'>)]

    flags
        Structure/Union member

    ident
        Structure/Union member

    length
        Structure/Union member

    mmio_address
        Structure/Union member

    name
        Structure/Union member

    name_length
        Structure/Union member

    type
        Structure/Union member

class ducky.devices.keyboard.KeyboardMMIOMemoryPage (device, *args, **kwargs)
    Bases: ducky.devices.MMIOMemoryPage

    read_u8 (offset)

class ducky.devices.keyboard.KeyboardPorts
    Bases: enum.IntEnum

    DATA = 1
    LAST = 1
    STATUS = 0

    _member_map_ = OrderedDict([('STATUS', <KeyboardPorts.STATUS: 0>), ('DATA', <KeyboardPorts.DATA: 1>), ('LAST', <KeyboardPorts.LAST: 1>)])
    _member_names_ = ['STATUS', 'DATA', 'LAST']
    _member_type_ = int
    _member_type_.__doc__ = 'alias of int'
    _value2member_map_ = {0: <KeyboardPorts.STATUS: 0>, 1: <KeyboardPorts.DATA: 1>, 2: <KeyboardPorts.LAST: 1>}

```

ducky.devices.rtc module

```

class ducky.devices.rtc.HDTEEntry_RTC (logger, config, section)
    Bases: ducky.hdt.HDTEEntry_Device

    _fields_ = [('type', <class 'ctypes.c_ushort'>), ('length', <class 'ctypes.c_ushort'>), ('name_length', <class 'ctypes.c_ushort'>)]

    flags
        Structure/Union member

    ident
        Structure/Union member

```

```

    length
        Structure/Union member

    mmio_address
        Structure/Union member

    name
        Structure/Union member

    name_length
        Structure/Union member

    type
        Structure/Union member

class ducky.devices.rtc.RTC(machine, name, frequency=None, mmio_address=None, irq=None,
                             *args, **kwargs)
    Bases: ducky.devices.Device

    boot()

    static create_from_config(machine, config, section)

    static create_hdt_entries(logger, config, section)

    frequency

    halt()

class ducky.devices.rtc.RTCMMIOMemoryPage(device, *args, **kwargs)
    Bases: ducky.devices.MMIOMemoryPage

    read_u8(offset)

    write_u8(offset, value)

class ducky.devices.rtc.RTCPorts
    Bases: enum.IntEnum

    DAY = 5

    FREQUENCY = 0

    HOUR = 4

    MINUTE = 2

    MONTH = 6

    SECOND = 1

    YEAR = 6

    _member_map_ = OrderedDict([('FREQUENCY', <RTCPorts.FREQUENCY: 0>), ('SECOND', <RTCPorts.SECOND: 1>), ('MINUTE', <RTCPorts.MINUTE: 2>), ('HOUR', <RTCPorts.HOUR: 4>), ('DAY', <RTCPorts.DAY: 5>), ('MONTH', <RTCPorts.MONTH: 6>), ('YEAR', <RTCPorts.YEAR: 6>)])
    _member_names_ = ['FREQUENCY', 'SECOND', 'MINUTE', 'HOUR', 'DAY', 'MONTH']
    _member_type_ = int
    _member_type_ = int
    alias of int

    _value2member_map_ = {0: <RTCPorts.FREQUENCY: 0>, 1: <RTCPorts.SECOND: 1>, 2: <RTCPorts.MINUTE: 2>, 4: <RTCPorts.HOUR: 4>, 5: <RTCPorts.DAY: 5>, 6: <RTCPorts.MONTH: 6>, 6: <RTCPorts.YEAR: 6>}}

class ducky.devices.rtc.RTCTask(machine, rtc)
    Bases: ducky.reactor.RunInIntervalTask

    on_tick(task)

    update_tick()

```

ducky.devices.snapshot module

```

class ducky.devices.snapshot.DefaultFileSnapshotStorage (machine,          name,
                                                         filepath=None,    *args,
                                                         **kwargs)

    Bases: ducky.devices.snapshot.FileSnapshotStorage

    static create_from_config (machine, config, section)

class ducky.devices.snapshot.FileSnapshotStorage (machine, name, filepath=None, *args,
                                                  **kwargs)

    Bases: ducky.devices.snapshot.SnapshotStorage

    boot ()

    static create_from_config (machine, config, section)

    halt ()

    save_snapshot (snapshot)

class ducky.devices.snapshot.SnapshotStorage (machine, name, *args, **kwargs)

    Bases: ducky.devices.Device

    static create_from_config (machine, config, section)

    halt ()

    save_snapshot (snapshot)

```

ducky.devices.storage module

Persistent storage support.

Several different persistent storages can be attached to a virtual machine, each with its own id. This module provides methods for manipulating their content. Storages operate with blocks of constant, standard size, though this is not a mandatory requirement - storage with different block size, or even with variable block size can be implemented.

Block IO subsystem transfers blocks between storages and VM,

```
ducky.devices.storage.BIO_BUSY = 16
```

Data transfer in progress.

```
ducky.devices.storage.BIO_DMA = 32
```

Request direct memory access - data will be transfered directly between storage and RAM..

```
ducky.devices.storage.BIO_ERR = 2
```

Error happened while performing the operation.

```
ducky.devices.storage.BIO_RDY = 1
```

Operation is completed, user can access data and/or request another operation

```
ducky.devices.storage.BIO_READ = 4
```

Request data read - transfer data from storage to memory.

```
ducky.devices.storage.BIO_SRST = 64
```

Reset BIO.

```
ducky.devices.storage.BIO_USER = 108
```

Flags that user can set - others are read-only.

```
ducky.devices.storage.BIO_WRITE = 8
```

Request data write - transfer data from memory to storage.

`ducky.devices.storage.BLOCK_SIZE = 1024`

Size of block, in bytes.

class `ducky.devices.storage.BlockIO` (*machine, name, mmio_address=None, irq=None, *args, **kwargs*)

Bases: `ducky.devices.Device`

`_flag_busy()`

Signals BIO is running an operation: *BIO_BUSY* is set, and *BIO_RDY* is cleared.

`_flag_error()`

Signals BIO request failed: *BIO_ERR* is set, and both *BIO_RDY* and *BIO_BUSY* are cleared.

`_flag_finished()`

Signals BIO is ready to accept new request: *BIO_RDY* is set, and *BIO_BUSY* is cleared.

If there was an request running, it is finished now. User can queue another request, or access data in case read by the last request.

`boot()`

`buff_to_memory(addr, buff)`

`static create_from_config(machine, config, section)`

`halt()`

`memory_to_buff(addr, length)`

`read_data()`

`reset()`

`select_storage(sid)`

`status_write(value)`

Handles writes to *STATUS* register. Starts the IO requested when *BIO_READ* or *BIO_WRITE* were set.

`write_data(value)`

class `ducky.devices.storage.BlockIOMMIOMemoryPage` (*device, *args, **kwargs*)

Bases: `ducky.devices.MMIOMemoryPage`

`read_u32(offset)`

`write_u32(offset, value)`

class `ducky.devices.storage.BlockIOPorts`

Bases: `enum.IntEnum`

MMIO ports, in form of offsets from a base MMIO address.

`ADDR = 16`

Address of a memory buffer

`BLOCK = 8`

Block ID

`COUNT = 12`

Number of blocks

`DATA = 20`

Data port, for non-DMA access

`SID = 4`

ID of selected storage device

STATUS = 0

Status port - query BIO status, and submit commands by setting flags

_member_map_ = OrderedDict([('STATUS', <BlockIOPorts.STATUS: 0>), ('SID', <BlockIOPorts.SID: 4>), ('BLOCK',

_member_names_ = ['STATUS', 'SID', 'BLOCK', 'COUNT', 'ADDR', 'DATA']

_member_type_

alias of int

_value2member_map_ = {0: <BlockIOPorts.STATUS: 0>, 4: <BlockIOPorts.SID: 4>, 8: <BlockIOPorts.BLOCK: 8>, 1

class ducky.devices.storage.**FileBackedStorage** (*machine, name, filepath=None, *args, **kwargs*)

Bases: *ducky.devices.storage.Storage*

Storage that saves its content into a regular file.

_read (*cnt*)

_write (*buff*)

boot ()

static create_from_config (*machine, config, section*)

do_read_blocks (*start, cnt*)

do_write_blocks (*start, cnt, buff*)

halt ()

class ducky.devices.storage.**Storage** (*machine, name, sid=None, size=None, *args, **kwargs*)

Bases: *ducky.devices.Device*

Base class for all block storages.

Parameters

- **machine** (*ducky.machine.Machine*) – machine storage is attached to.
- **sid** (*int*) – id of storage.
- **size** (*int*) – size of storage, in bytes.

do_read_blocks (*start, cnt*)

Read one or more blocks from device to internal buffer.

Child classes are supposed to reimplement this particular method.

Parameters

- **start** (*u32_t*) – index of the first requested block.
- **cnt** (*u32_t*) – number of blocks to read.

do_write_blocks (*start, cnt, buff*)

Write one or more blocks from internal buffer to device.

Child classes are supposed to reimplement this particular method.

Parameters

- **start** (*u32_t*) – index of the first requested block.
- **cnt** (*u32_t*) – number of blocks to write.

read_blocks (*start, cnt*)

Read one or more blocks from device to internal buffer.

Child classes should not reimplement this method, as it provides checks common for (probably) all child classes.

Parameters

- **start** (*u32_t*) – index of the first requested block.
- **cnt** (*u32_t*) – number of blocks to read.

write_blocks (*start, cnt, buff*)

Write one or more blocks from internal buffer to device.

Child classes should not reimplement this method, as it provides checks common for (probably) all child classes.

Parameters

- **start** (*u32_t*) – index of the first requested block.
- **cnt** (*u32_t*) – number of blocks to write.

exception `ducky.devices.storage.StorageAccessError`

Bases: `exceptions.Exception`

Base class for storage-related exceptions.

ducky.devices.terminal module

Terminal is a device that groups together character two input and output devices, thus forming a simple channel for bidirectional communication between VM and user.

Terminal has two slave frontends:

- *input*, usually a keyboard
- *output*, from simple TTY to more powerful devices

Terminal then manages input and input streams, passing them to its slave devices, which then transports events between streams and VM's comm channel.

class `ducky.devices.terminal.StandalonePTYTerminal` (**args, **kwargs*)

Bases: `ducky.devices.terminal.StreamIOTerminal`

boot ()

static create_from_config (*machine, config, section*)

halt ()

class `ducky.devices.terminal.StandardIOTerminal` (*machine, name, input_device=None, output_device=None, *args, **kwargs*)

Bases: `ducky.devices.terminal.StreamIOTerminal`

static create_from_config (*machine, config, section*)

class `ducky.devices.terminal.StreamIOTerminal` (*machine, name, input_device=None, output_device=None, *args, **kwargs*)

Bases: `ducky.devices.terminal.Terminal`

boot ()

static create_from_config (*machine, config, section*)

```

    enqueue_input_stream (stream)

    enqueue_streams (streams_in=None, stream_out=None)

    halt ()

class ducky.devices.terminal.Terminal (machine, name, echo=False, *args, **kwargs)
    Bases: ducky.devices.DeviceFrontend

    _input_read_u8_echo (*args, **kwargs)

    _patch_echo (restore=False)

    boot ()

    halt ()

ducky.devices.terminal.get_slave_devices (machine, config, section)

ducky.devices.terminal.parse_io_streams (machine, config, section)

```

ducky.devices.tty module

Very simple character device that just “prints” characters on the screen. It does not care about dimensions of the display, it know only how to “print” characters. Suited for the most basic output possible - just “print” chars by writing to this device, and you’ll get this written into a stream attached to the frontend (stdout, file, ...).

```

class ducky.devices.tty.Backend (machine, name, stream=None, mmio_address=None, *args,
                                **kwargs)
    Bases: ducky.devices.DeviceBackend

    boot ()

    static create_from_config (machine, config, section)

    static create_hdt_entries (logger, config, section)

    halt ()

    tenh (s, *args)

    tenh_close_stream ()

    tenh_enable ()

    tenh_flush_stream ()

class ducky.devices.tty.Frontend (machine, name)
    Bases: ducky.devices.DeviceFrontend

    boot ()

    close (allow=False)

    static create_from_config (machine, config, section)

    flush ()

    halt ()

    set_output (stream)

    sleep_flush ()

    tenh_enable ()

    wakeup_flush ()

```

```
class ducky.devices.tty.FrontendFlushTask (frontend, queue, stream)
    Bases: ducky.interfaces.IReactorTask
```

```
    run ()
```

```
    set_output (stream)
```

```
class ducky.devices.tty.HDTEntry_TTY (logger, config, section)
```

```
    Bases: ducky.hdt.HDTEntry_Device
```

```
    _fields_ = [('type', <class 'ctypes.c_ushort'>), ('length', <class 'ctypes.c_ushort'>), ('name_length', <class 'ctypes.c_ushort'>)]
```

```
    flags
```

```
        Structure/Union member
```

```
    ident
```

```
        Structure/Union member
```

```
    length
```

```
        Structure/Union member
```

```
    mmio_address
```

```
        Structure/Union member
```

```
    name
```

```
        Structure/Union member
```

```
    name_length
```

```
        Structure/Union member
```

```
    type
```

```
        Structure/Union member
```

```
class ducky.devices.tty.TTYMMIOMemoryPage (device, *args, **kwargs)
```

```
    Bases: ducky.devices.MMIOMemoryPage
```

```
    write_u8 (offset, value)
```

```
class ducky.devices.tty.TTYPorts
```

```
    Bases: enum.IntEnum
```

```
    DATA = 0
```

```
    _member_map_ = OrderedDict([('DATA', <TTYPorts.DATA: 0>)])
```

```
    _member_names_ = ['DATA']
```

```
    _member_type_
```

```
        alias of int
```

```
    _value2member_map_ = {0: <TTYPorts.DATA: 0>}
```

ducky.devices.svga module

SimpleVGA is very basic implementation of VGA-like device, with text and graphic modes.

```
class ducky.devices.svga.Char
```

```
    Bases: _ctypes.Structure
```

```
    _fields_ = [('codepoint', <class 'ctypes.c_ushort'>, 7), ('unused', <class 'ctypes.c_ushort'>, 1), ('fg', <class 'ctypes.c_ushort'>, 1)]
```

```
    _pack_ = 0
```

```
    bg
```

```
        Structure/Union member
```

blink
Structure/Union member

codepoint
Structure/Union member

fg
Structure/Union member

static from_u16 (*u*)

static from_u8 (*l*, *h*)

to_u8 ()

unused
Structure/Union member

`ducky.devices.svga.DEFAULT_BOOT_MODE = (t, 80, 25, 1)`
Default boot mode

`ducky.devices.svga.DEFAULT_MEMORY_BANKS = 8`
Default number of memory banks

`ducky.devices.svga.DEFAULT_MEMORY_SIZE = 65536`
Default memory size, in bytes

`ducky.devices.svga.DEFAULT_MMIO_ADDRESS = 33024`
Default MMIO address

`ducky.devices.svga.DEFAULT_MODES = [(g, 320, 200, 1), (t, 80, 25, 2), (t, 80, 25, 1)]`
Default list of available modes

class `ducky.devices.svga.Display` (*machine*, *name*, *gpu=None*, *stream_out=None*, **args*, ***kwargs*)
Bases: `ducky.devices.Device`

boot ()

static create_from_config (*machine*, *config*, *section*)

static get_slave_gpu (*machine*, *config*, *section*)

halt ()

class `ducky.devices.svga.DisplayRefreshTask` (*display*)
Bases: `ducky.reactor.RunInIntervalTask`

on_tick (*task*)

class `ducky.devices.svga.Mode` (*_type*, *width*, *height*, *depth*)
Bases: object

classmethod **from_string** (*s*)

Create Mode object from its string representation. It's a comma-separated list of for items:

	type	width	height	depth
Text mode	t	chars per line	lines on screen	bytes per char
Graphic mode	g	pixels per line	pixel lines on screen	bites per pixel

to_pretty_string ()

to_string ()

```
class ducky.devices.svga.SimpleVGA(machine, name, memory_size=None, mmio_address=None,
                                   memory_address=None, memory_banks=None,
                                   modes=None, boot_mode=None, *args, **kwargs)
```

Bases: `ducky.devices.Device`

SimpleVGA is very basic implementation of VGA-like device, with text and graphic modes.

It has its own graphic memory (“buffer”), split into several banks of the same size. Always only one bank can be directly accessed, by having it mapped into CPU’s address space.

Parameters

- **machine** (`ducky.machine.Machine`) – machine this device belongs to.
- **name** (`string`) – name of this device.
- **memory_size** (`int`) – size of graphic memory.
- **mmio_address** (`u32_t`) – base address of MMIO ports.
- **memory_address** (`u24`) – address of graphic memory - to this address is graphic buffer mapped. Must be specified, there is no default value.
- **memory_banks** (`int`) – number of memory banks.
- **modes** (`list`) – list of `ducky.devices.svga.Mode` objects, list of supported modes.
- **boot_mode** (`tuple`) – this mode will be set when device boots up.

```
boot()
```

```
static create_from_config(machine, config, section)
```

```
halt()
```

```
reset()
```

```
set_mode(mode)
```

```
class ducky.devices.svga.SimpleVGACommands
```

Bases: `enum.IntEnum`

```
COLS = 33
```

```
DEPTH = 35
```

```
GRAPHIC = 32
```

```
MEMORY_BANK_ID = 48
```

```
REFRESH = 2
```

```
RESET = 1
```

```
ROWS = 34
```

```
_member_map_ = OrderedDict([('RESET', <SimpleVGACommands.RESET: 1>), ('REFRESH', <SimpleVGACommands.REFRESH: 2>), ('GRAPHIC', <SimpleVGACommands.GRAPHIC: 32>), ('COLS', <SimpleVGACommands.COLS: 33>), ('ROWS', <SimpleVGACommands.ROWS: 34>), ('DEPTH', <SimpleVGACommands.DEPH: 35>), ('MEMORY_BANK_ID', <SimpleVGACommands.MEMORY_BANK_ID: 48>)])
```

```
_member_names_ = ['RESET', 'REFRESH', 'GRAPHIC', 'COLS', 'ROWS', 'DEPTH', 'MEMORY_BANK_ID']
```

```
_member_type_
alias of int
```

```
_value2member_map_ = {32: <SimpleVGACommands.GRAPHIC: 32>, 1: <SimpleVGACommands.RESET: 1>, 2: <SimpleVGACommands.REFRESH: 2>, 33: <SimpleVGACommands.COLS: 33>, 34: <SimpleVGACommands.ROWS: 34>, 35: <SimpleVGACommands.DEPH: 35>, 48: <SimpleVGACommands.MEMORY_BANK_ID: 48>}
```

```
class ducky.devices.svga.SimpleVGAMMIOMemoryPage(device, *args, **kwargs)
```

Bases: `ducky.devices.MMIOMemoryPage`

```
read_u16(offset)
```

write_u16 (*offset*, *value*)

class `ducky.devices.svga.SimpleVGAMemoryPage` (*dev*, **args*, ***kwargs*)

Bases: `ducky.mm.ExternalMemoryPage`

Memory page handling MMIO of sVGA device.

Parameters **dev** (`ducky.devices.svga.SimpleVGA`) – sVGA device this page belongs to.

get (*offset*)

put (*offset*, *b*)

class `ducky.devices.svga.SimpleVGAPorts`

Bases: `enum.IntEnum`

CONTROL = 0

DATA = 2

_member_map_ = `OrderedDict([('CONTROL', <SimpleVGAPorts.CONTROL: 0>), ('DATA', <SimpleVGAPorts.DATA: 2>)])`

_member_names_ = ['CONTROL', 'DATA']

_member_type_

alias of `int`

_value2member_map_ = {0: <SimpleVGAPorts.CONTROL: 0>, 2: <SimpleVGAPorts.DATA: 2>}

2.6.2 Module contents

Emulating different virtual devices like keyboard, display, disks...

class `ducky.devices.Device` (*machine*, *class*, *name*)

Bases: `ducky.interfaces.IMachineWorker`

Base class for all devices. Serves more like an API description.

Parameters

- **machine** (`ducky.machine.Machine`) – VM this device belongs to.
- **class** (*str*) – device family (input, output, snapshot, ...)
- **name** (*str*) – device name. Maps directly to a section of config file that hosts setup for this device.

boot ()

static create_from_config (*machine*, *config*, *section*)

Create new instance, configured exactly as requested by configuration file.

Parameters

- **machine** (`ducky.machine.Machine`) – VM this device belongs to.
- **config** (`ducky.config.MachineConfig`) – configuration file.
- **section** (*str*) – name of config section with this device's setup.

static create_hdt_entries (*logger*, *config*, *section*)

Create HDT entries for this device, based on configuration file.

Parameters

- **logger** (`logging.Logger`) – logger to use for logging.

- **config** (`ducky.config.MachineConfig`) – configuration file.
- **section** (`str`) – name of config section with this device’s setup.

Return type list of `ducky.hdt.HDTEntry`

Returns list of HDT entries. This list is then appended to entries of other devices.

get_master ()

halt ()

is_slave ()

class `ducky.devices.DeviceBackend` (`machine, klass, name`)

Bases: `ducky.devices.Device`

Backend is a special component of a device, which is connected to internal queues, and processes events in the queue, originating from user via its frontend counterpart.

set_frontend (`device`)

Set frontend counterpart.

Parameters `device` (`ducky.device.DeviceFrontend`) – frontend part.

class `ducky.devices.DeviceFrontend` (`machine, klass, name`)

Bases: `ducky.devices.Device`

Frontend is a special component of a device, that interfaces communication channels from user and internal queues.

set_backend (`device`)

Set backend counterpart.

Parameters `device` (`ducky.devices.DeviceFrontend`) – backend part.

class `ducky.devices.MMIOMemoryPage` (`device, *args, **kwargs`)

Bases: `ducky.mm.VirtualMemoryPage`

Memory page, suited for memory-mapped IO, supported by a device driver.

Parameters `device` (`ducky.devices.Device`) – device instance, backing this memory page.

`ducky.devices.get_driver` (`driver_class`)

Get Python class, implementing device driver, by its name.

Parameters `driver_class` (`str`) – path to a class, e.g. `ducky.devices.rtc.RTC`.

Returns driver class.

2.7 ducky.errors module

exception `ducky.errors.AccessViolationError` (`message=None`)

Bases: `ducky.errors.Error`

Raised when an operation was requested without having adequate permission to do so. `message` attribute will provide better idea about the fault.

exception `ducky.errors.AssemblerError` (`location=None, message=None, line=None, info=None`)

Bases: `ducky.errors.Error`

Base class for all assembler-related exceptions. Provides common properties, helping to locate related input in the source file.

Parameters

- **location** (*ducky.cpu.assembly.SourceLocation*) – if set, points to the location in the source file that was processed when the exception occurred.
- **message** (*str*) – more detailed description.
- **line** (*str*) – input source line.
- **info** – additional details of the exception. This value is usually part of the message, but is stored as well.

create_text ()

log (*logger*)

exception *ducky.errors.AssemblyIllegalCharError* (*c=None, **kwargs*)

Bases: *ducky.errors.AssemblerError*

exception *ducky.errors.AssemblyParseError* (*token=None, **kwargs*)

Bases: *ducky.errors.AssemblerError*

exception *ducky.errors.BadLinkerScriptError* (*script, exc, *args, **kwargs*)

Bases: *ducky.errors.LinkerError*

exception *ducky.errors.ConflictingNamesError* (***kwargs*)

Bases: *ducky.errors.AssemblerError*

exception *ducky.errors.CoprocessorError* (*msg, *args, **kwargs*)

Bases: *ducky.errors.ExecutionException__SimpleESR*, *ducky.errors.ExecutionException*

Base class for coprocessor errors. Raised when coprocessors needs to signal its own exception, when none of already available exceptions would do.

EXCEPTION_INDEX = 25

exception *ducky.errors.DisassembleMismatchError* (***kwargs*)

Bases: *ducky.errors.AssemblerError*

exception *ducky.errors.DivideByZeroError* (**args, **kwargs*)

Bases: *ducky.errors.ExecutionException__SimpleESR*, *ducky.errors.ExecutionException*

Raised when divisor in a mathematical operation was equal to zero.

EXCEPTION_INDEX = 18

exception *ducky.errors.EncodingLargeValueError* (***kwargs*)

Bases: *ducky.errors.AssemblerError*

exception *ducky.errors.Error* (*message=None*)

Bases: *exceptions.Exception*

Base class for all Ducky exceptions.

Parameters **message** (*str*) – optional description.

log (*logger*)

class *ducky.errors.ExceptionList*

Bases: *enum.IntEnum*

List of exception IDs (EVT indices).

COUNT = 32

```
CoprocessorError = 25
```

```
DivideByZero = 18
```

```
DoubleFault = 21
```

```
FIRST_HW = 0
```

```
FIRST_SW = 16
```

```
InvalidException = 24
```

```
InvalidInstSet = 17
```

```
InvalidOpcode = 16
```

```
LAST_HW = 15
```

```
LAST_SW = 31
```

```
MemoryAccess = 22
```

```
PrivilegedInstr = 20
```

```
RegisterAccess = 23
```

```
UnalignedAccess = 19
```

```
_member_map_ = OrderedDict([('FIRST_HW', <ExceptionList.FIRST_HW: 0>), ('LAST_HW', <ExceptionList.LAST_
```

```
_member_names_ = ['FIRST_HW', 'LAST_HW', 'InvalidOpcode', 'InvalidInstSet', 'DivideByZero', 'UnalignedAccess'
```

```
_member_type_  
    alias of int
```

```
_value2member_map_ = {0: <ExceptionList.FIRST_HW: 0>, 32: <ExceptionList.COUNT: 32>, 15: <ExceptionList.LA
```

exception `ducky.errors.ExecutionException` (*msg, core=None, ip=None*)

Bases: `exceptions.Exception`

Base class for all execution-related exceptions, i.e. exceptions raised as a direct result of requests made by running code. Runnign code can then take care of handling these exceptions, usually by preparing service routines and setting up the EVT.

Unless said otherwise, the exception is always raised *before* making any changes in the VM state.

Parameters

- **msg** (*string*) – message describing exceptional state.
- **core** (`ducky.cpu.CPUCore`) – CPU core that raised exception, if any.
- **ip** (*u32_t*) – address of an instruction that caused exception, if any.

`runtime_handle()`

This method is called by CPU code, to find out if it is possible for runtime to handle the exception, and possibly recover from it. If it is possible, this method should make necessary arrangements, and then return `True`. Many exceptions, e.g. when division by zero was requested, will tell CPU to run exception service routine, and by returning `True` signal that it's not necessary to take care of such exception anymore.

Return type `bool`

Returns `True` when it's no longer necessary for VM code to take care of this exception.

class `ducky.errors.ExecutionException__SimpleESR`

Bases: `object`

Helper mixin class - as one of parents, it brings to its children very simple - and most of the time sufficient - implementation of `runtime_handle` method. Such exceptions will tell CPU to run exception service routine with a specific index, specified by class variable `EXCEPTION_INDEX`.

The address of the offensive instruction - or the value of IP when exception was raised, since there may be exceptions not raised in response to the executed instruction - is passed to CPU as the first argument of ESR.

`runtime_handle()`

exception `ducky.errors.IncompatibleSectionFlagsError` (*dst_section*, *src_section*, **args*, ***kwargs*)

Bases: `ducky.errors.LinkerError`

exception `ducky.errors.IncompleteDirectiveError` (***kwargs*)

Bases: `ducky.errors.AssemblerError`

exception `ducky.errors.InvalidExceptionError` (*exc_index*, **args*, ***kwargs*)

Bases: `ducky.errors.ExecutionException__SimpleESR`, `ducky.errors.ExecutionException`

Raised when requested exception index is invalid (out of bounds).

`EXCEPTION_INDEX = 24`

exception `ducky.errors.InvalidFrameError` (*saved_sp*, *current_sp*, **args*, ***kwargs*)

Bases: `ducky.errors.ExecutionException__SimpleESR`, `ducky.errors.ExecutionException`

Raised when VM checks each call stack frame to be cleaned before it's left, and there are some values on stack when `ret` or `retint` are executed. In such case, the actual SP values does not equal to a saved one, and exception is raised.

Parameters

- **`saved_sp`** (*u32_t*) – SP as saved at the moment the frame was created.
- **`current_sp`** (*u32_t*) – current SP

`runtime_handle()`

exception `ducky.errors.InvalidInstructionSetError` (*inst_set*, **args*, ***kwargs*)

Bases: `ducky.errors.ExecutionException__SimpleESR`, `ducky.errors.ExecutionException`

Raised when switch to unknown or invalid instruction set was requested.

Parameters **`inst_set`** (*int*) – instruction set id.

`EXCEPTION_INDEX = 17`

exception `ducky.errors.InvalidOpcodeError` (*opcode*, **args*, ***kwargs*)

Bases: `ducky.errors.ExecutionException__SimpleESR`, `ducky.errors.ExecutionException`

Raised when unknown or invalid opcode is found in instruction.

Parameters **`opcode`** (*int*) – wrong opcode.

`EXCEPTION_INDEX = 16`

exception `ducky.errors.InvalidResourceError` (*message=None*)

Bases: `ducky.errors.Error`

Raised when an operation was requested on somehow invalid resource. `message` attribute will provide better idea about the fault.

exception `ducky.errors.LinkerError` (*message=None*)

Bases: `ducky.errors.Error`

exception `ducky.errors.MemoryAccessError` (*access, address, pte, *args, **kwargs*)

Bases: `ducky.errors.ExecutionException__SimpleESR`, `ducky.errors.ExecutionException`

Raised when MMU decides the requested memory operation is now allowed, e.g. when page tables are enabled, and corresponding PTE denies write access.

Parameters

- **access** (*str*) – read, write or execute.
- **address** (*u32_t*) – address where memory access should have happen.
- **pte** (`ducky.mm.PageTableEntry`) – PTE guarding this particular memory location.

EXCEPTION_INDEX = 22

exception `ducky.errors.OperandMismatchError` (*instr, expected, actual*)

Bases: `ducky.errors.Error`

exception `ducky.errors.PatchTooLargeError` (*message=None*)

Bases: `ducky.errors.LinkerError`

exception `ducky.errors.PrivilegedInstructionError` (**args, **kwargs*)

Bases: `ducky.errors.ExecutionException__SimpleESR`, `ducky.errors.ExecutionException`

Raised when privileged instruction was about to be executed in non-privileged mode.

EXCEPTION_INDEX = 20

exception `ducky.errors.RegisterAccessError` (*access, reg, *args, **kwargs*)

Bases: `ducky.errors.ExecutionException__SimpleESR`, `ducky.errors.ExecutionException`

Raised when instruction tries to access a register which is not available for requested operation, e.g. writing into read-only control register will raise this exception.

Parameters

- **access** (*str*) – read or write.
- **reg** (*str*) – register name.

EXCEPTION_INDEX = 23

exception `ducky.errors.TooManyLabelsError` (***kwargs*)

Bases: `ducky.errors.AssemblerError`

exception `ducky.errors.UnalignedAccessError` (**args, **kwargs*)

Bases: `ducky.errors.ExecutionException__SimpleESR`, `ducky.errors.ExecutionException`

Raised when only properly aligned memory access is allowed, and running code attempts to access memory without honoring this restriction (e.g. LW reading from byte-aligned address).

EXCEPTION_INDEX = 19

exception `ducky.errors.UnalignedJumpTargetError` (***kwargs*)

Bases: `ducky.errors.AssemblerError`

exception `ducky.errors.UnknownDestinationSectionError` (*src_section, *args, **kwargs*)

Bases: `ducky.errors.LinkerError`

```

exception ducky.errors.UnknownFileError (**kwargs)
    Bases: ducky.errors.AssemblerError

exception ducky.errors.UnknownInstructionError (**kwargs)
    Bases: ducky.errors.AssemblerError

exception ducky.errors.UnknownPatternError (**kwargs)
    Bases: ducky.errors.AssemblerError

exception ducky.errors.UnknownSymbolError (message=None)
    Bases: ducky.errors.LinkerError

```

2.8 ducky.hdt module

Hardware Description Table structures.

```

class ducky.hdt.HDT (logger, config=None)
    Bases: object

```

Root of HDT. Provides methods for creating HDT for a given machine configuration.

Parameters

- **logger** (*logging.Logger*) – logger instance used for logging.
- **config** (*ducky.config.MachineConfig*) – configuration file HDT should reflect.

```

create ()
    Create HDT header and entries from config file.

```

```

classes = [<class 'ducky.hdt.HDTEntry_Memory'>, <class 'ducky.hdt.HDTEntry_CPU'>, <class 'ducky.hdt.HDTEntry_...>]
    These HDT entries are added automatically.

```

```

class ducky.hdt.HDTEntry (entry_type, length)
    Bases: ducky.hdt.HDTStructure

```

Base class of all HDT entries.

Each entry has at least two fields, *type* and *length*. Other fields depend on type of entry, and follow immediately after *length* field.

Parameters

- **type** (*u16_t*) – type of entry. See *ducky.hdt.HDTEntryTypes*.
- **length** (*u16_t*) – length of entry, in bytes.

```

ENTRY_HEADER = [('type', <class 'ctypes.c_ushort'>), ('length', <class 'ctypes.c_ushort'>)]

```

```

classmethod create (*args, **kwargs)

```

```

class ducky.hdt.HDTEntryTypes
    Bases: enum.IntEnum

```

Types of different HDT entries.

```

ARGUMENT = 3

```

```

CPU = 1

```

```

DEVICE = 4

```

```

MEMORY = 2

```

```

UNDEFINED = 0

```

```
_member_map_ = OrderedDict([('UNDEFINED', <HDTEntryTypes.UNDEFINED: 0>), ('CPU', <HDTEntryTypes.CPU: 1>), ('MEMORY', <HDTEntryTypes.MEMORY: 2>), ('ARGUMENT', <HDTEntryTypes.ARGUMENT: 3>), ('DEVICE', <HDTEntryTypes.DEVICE: 4>)]  
_member_names_ = ['UNDEFINED', 'CPU', 'MEMORY', 'ARGUMENT', 'DEVICE']  
_member_type_  
    alias of int  
_value2member_map_ = {0: <HDTEntryTypes.UNDEFINED: 0>, 1: <HDTEntryTypes.CPU: 1>, 2: <HDTEntryTypes.MEMORY: 2>, 3: <HDTEntryTypes.ARGUMENT: 3>, 4: <HDTEntryTypes.DEVICE: 4>}  
class ducky.hdt.HDTEntry_Argument (arg_name, arg_type, arg_value)  
    Bases: ducky.hdt.HDTEntry  
    MAX_NAME_LENGTH = 13  
    _fields_ = [('type', <class 'ctypes.c_ushort'>), ('length', <class 'ctypes.c_ushort'>), ('name_length', <class 'ctypes.c_ushort'>), ('value_length', <class 'ctypes.c_ushort'>)]  
    classmethod create (logger, config)  
    length  
        Structure/Union member  
    name  
        Structure/Union member  
    name_length  
        Structure/Union member  
    type  
        Structure/Union member  
    value  
        Structure/Union member  
    value_length  
        Structure/Union member  
class ducky.hdt.HDTEntry_CPU (logger, config)  
    Bases: ducky.hdt.HDTEntry  
    HDT entry describing CPU configuration.  
    Parameters  
    • nr_cpus (u16_t) – number of CPUs.  
    • nr_cores (u16_t) – number of cores per CPU.  
    _fields_ = [('type', <class 'ctypes.c_ushort'>), ('length', <class 'ctypes.c_ushort'>), ('nr_cpus', <class 'ctypes.c_ushort'>), ('nr_cores', <class 'ctypes.c_ushort'>)]  
    length  
        Structure/Union member  
    nr_cores  
        Structure/Union member  
    nr_cpus  
        Structure/Union member  
    type  
        Structure/Union member  
class ducky.hdt.HDTEntry_Device (logger, name, ident)  
    Bases: ducky.hdt.HDTEntry  
    ENTRY_HEADER = [('type', <class 'ctypes.c_ushort'>), ('length', <class 'ctypes.c_ushort'>), ('name_length', <class 'ctypes.c_ushort'>), ('ident_length', <class 'ctypes.c_ushort'>)]  
    MAX_IDENT_LENGTH = 32
```

MAX_NAME_LENGTH = 10

class `ducky.hdt.HDTEntry_Memory` (*logger, config*)

Bases: `ducky.hdt.HDTEntry`

HDT entry describing memory configuration.

Parameters **size** (`u32_t`) – size of memory, in bytes.

fields = [('type', <class 'ctypes.c_ushort'>), ('length', <class 'ctypes.c_ushort'>), ('size', <class 'ctypes.c_uint'>)]

length

Structure/Union member

size

Structure/Union member

type

Structure/Union member

class `ducky.hdt.HDTHeader`

Bases: `ducky.hdt.HDTStructure`

HDT header. Contains magic number, number of HDT entries that immediately follow header.

fields = [('magic', <class 'ctypes.c_uint'>), ('entries', <class 'ctypes.c_uint'>), ('length', <class 'ctypes.c_uint'>)]

entries

Structure/Union member

length

Structure/Union member

magic

Structure/Union member

class `ducky.hdt.HDTStructure`

Bases: `_ctypes.Structure`

Base class of all HDT structures.

pack = 0

`ducky.hdt.HDT_MAGIC = 1298034544`

Magic number present in HDT header

`ducky.hdt.encode_string` (*struct, field, s, max_length*)

Store string in a structure's field, and set the corresponding length field properly.

Parameters

- **struct** (`HDTStructure`) – structure to modify.
- **field** (*str*) – name of field the string should be stored in.
- **s** (*str*) – string to encode.
- **max_length** (*int*) – maximal number of bytes that can fit into the field.

2.9 ducky.interfaces module

class `ducky.interfaces.IMachineWorker`

Bases: `object`

Base class for objects that provide pluggable service to others.

boot (**args*)

Prepare for providing the service. After this call, it may be requested by others.

die (*exc*)

Exceptional state requires immediate termination of service. Probably no object will ever have need to call others' `die` method, it's intended for internal use only.

halt ()

Terminate service. It will never be requested again, object can destroy its internal state, and free allocated resources.

run ()

Called by reactor's loop when this object is enqueued as a reactor task.

suspend ()

Suspend service. Object should somehow conserve its internal state, its service will not be used until the next call of `wake_up` method.

wake_up ()

Wake up service. In this method, object should restore its internal state, and after this call its service can be requested by others again.

class `ducky.interfaces.IReactorTask`

Bases: `object`

Base class for all reactor tasks.

run ()

This method is called by reactor to perform task's actions.

class `ducky.interfaces.ISnapshotable`

Bases: `object`

Base class for objects that can be saved into a snapshot.

load_state (*state*)

Restore state of the object.

Parameters **state** (`ducky.snapshot.SnapshotNode`) – snapshot node containing saved state.

save_state (*parent*)

Create state of the object, and attach it to a parent snapshot node.

Parameters **parent** (`ducky.interfaces.ISnapshotable`) – parent snapshot node.

class `ducky.interfaces.IVirtualInterrupt` (*machine*)

Bases: `object`

run (*core*)

2.10 ducky.log module

class `ducky.log.ColorizedLogFormatter` (*format=None*)

Bases: `ducky.log.LogFormatter`

_default_format = '{color_start}[{level}]{color_end} {message}'

_get_vars (*record*)

```

    blue (s)
    colorize (s, fore='\x1b[39m\x1b[49m\x1b[0m', back='\x1b[39m\x1b[49m\x1b[0m')
    green (s)
    red (s)
    white (s)
class ducky.log.LogFormatter (format=None)
    Bases: logging.Formatter
    _default_format = '[{level}] {message}'
    _get_vars (record)
    blue (s)
    colorize (s, *args, **kwargs)
    double_fault (exc, record)
    format (record)
    green (s)
    red (s)
    white (s)
class ducky.log.StreamHandler (formatter=None, *args, **kwargs)
    Bases: logging.StreamHandler
    ducky.log.create_logger (name=None, handler=None, level=20)

```

2.11 ducky.machine module

ducky.machine.Machine is the virtual machine. Each instance represents self-contained virtual machine, with all its devices, memory, CPUs and other necessary properties.

```

class ducky.machine.CommChannel (machine)
    Bases: object
    create_queue (name)
    get_queue (name)
    unregister_queue (name)
class ducky.machine.CommQueue (channel)
    Bases: object
    is_empty_in ()
    is_empty_out ()
    read_in ()
    read_out ()
    write_in (o)
    write_out (o)

```

class `ducky.machine.EventBus` (*machine*)

Bases: `object`

add_listener (*event, callback, *args, **kwargs*)

remove_listener (*event, callback*)

trigger (*event, *args, **kwargs*)

class `ducky.machine.HaltMachineTask` (*machine*)

Bases: `ducky.interfaces.IReactorTask`

run ()

class `ducky.machine.IRQRouterTask` (*machine*)

Bases: `ducky.interfaces.IReactorTask`

This task is responsible for distributing triggered IRQs between CPU cores. When IRQ is triggered, IRQ source (i.e. device that requires attention) is appended to this tasks queue (`ducky.machine.IRQRouterTask.queue`). As long as this queue is not empty, this task pops IRQ sources, selects free CPU core, and by calling its `ducky.cpu.CPUCore.irq()` method core takes responsibility for executing interrupt routine.

Parameters `machine` (`ducky.machine.Machine`) – machine this task belongs to.

run ()

class `ducky.machine.Machine` (*logger=None, stdin=None, stdout=None, stderr=None*)

Bases: `ducky.interfaces.ISnapshotable`, `ducky.interfaces.IMachineWorker`

Virtual machine itself.

_do_tenh (*printer, s, *args*)

boot ()

capture_state (*suspend=False*)

Capture current state of the VM, and store it in its `last_state` attribute.

Parameters `suspend` (*bool*) – if *True*, suspend VM before taking snapshot.

core (*cid*)

Find CPU core by its string id.

Parameters `cid` (*string*) – id of searched CPU core, in the form `#<cpuid>:#<coreid>`.

Return type `ducky.cpu.CPUCore`

Returns found core

Raises `ducky.errors.InvalidResourceError` – when no such core exists.

cores

Get list of all cores in the machine.

Return type `list`

Returns *list* of `ducky.cpu.CPUCore` instances

die (*exc*)

exit_code

get_device_by_name (*name, klass=None*)

Get device by its name and class.

Parameters

- **name** (*string*) – name of the device.

- **class** (*string*) – if set, search only devices with this class.

Return type `ducky.devices.Device`

Returns found device

Raises `ducky.errors.InvalidResourceError` – when no such device exists

get_storage_by_id (*sid*)

Get storage by its id.

Parameters **sid** (*int*) – id of storage caller is looking for.

Return type `ducky.devices.Device`

Returns found device.

Raises `ducky.errors.InvalidResourceError` – when no such storage exists.

halt ()

hw_setup (*machine_config*)

load_state (*state*)

on_core_alive (*core*)

Signal machine that one of CPU cores is now alive.

on_core_halted (*core*)

Signal machine that one of CPU cores is no longer alive.

run ()

save_state (*parent*)

setup_devices ()

suspend ()

tenh (*s*, **args*)

trigger_irq (*handler*)

wake_up ()

class `ducky.machine.MachineState`

Bases: `ducky.snapshot.SnapshotNode`

get_cpu_state_by_id (*cpuid*)

get_cpu_states ()

`ducky.machine.cmd_boot` (*console*, *cmd*)

Setup HW, load binaries, init everything

`ducky.machine.cmd_halt` (*console*, *cmd*)

Halt execution

`ducky.machine.cmd_run` (*console*, *cmd*)

Start execution of loaded binaries

`ducky.machine.cmd_snapshot` (*console*, *cmd*)

Create snapshot

2.12 ducky.mm package

2.12.1 Submodules

ducky.mm.binary module

```
class ducky.mm.binary.File(logger, stream)
    Bases: ducky.util.BinaryFile

    MAGIC = 57005
    VERSION = 3

    _create_header()
    _read(*args, **kwargs)
    _read_header()
    _write(*args, **kwargs)
    _write_header(header)
    create_section(name=None)
    fix_offsets()
    get_section_by_index(index)
    get_section_by_name(name, dont_create=False)
    get_section_by_type(typ)
    get_strings_section()
    header
    static open(*args, **kwargs)
    save()
    sections
    setup()
    string_table

class ducky.mm.binary.FileFlags
    Bases: ducky.util.Flags

    _encoding
        alias of FileFlagsEncoding
    _flags = ['reserved']
    _labels = 'M'
    field = ('reserved', <class 'ctypes.c_ushort'>, 1)

class ducky.mm.binary.FileFlagsEncoding
    Bases: _ctypes.Structure

    _fields_ = [('reserved', <class 'ctypes.c_ushort'>, 1)]

    reserved
        Structure/Union member
```

```

class ducky.mm.binary.FileHeader
    Bases: _ctypes.Structure

    _fields_ = [('magic', <class 'ctypes.c_ushort'>), ('version', <class 'ctypes.c_ushort'>), ('flags', <class 'ducky.mm.binary.
    _pack_ = 0

    flags
        Structure/Union member

    magic
        Structure/Union member

    sections
        Structure/Union member

    version
        Structure/Union member

class ducky.mm.binary.RelocEntry
    Bases: _ctypes.Structure

    _fields_ = [('flags', <class 'ducky.mm.binary.RelocFlagsEncoding'>), ('name', <class 'ctypes.c_uint'>), ('patch_add', <
    _pack_ = 0

    flags
        Structure/Union member

    name
        Structure/Union member

    patch_add
        Structure/Union member

    patch_address
        Structure/Union member

    patch_offset
        Structure/Union member

    patch_section
        Structure/Union member

    patch_size
        Structure/Union member

class ducky.mm.binary.RelocFlags
    Bases: ducky.util.Flags

    _encoding
        alias of RelocFlagsEncoding

    _flags = ['relative', 'inst_aligned']

    _labels = 'RI'

    field = ('inst_aligned', <class 'ctypes.c_ushort'>, 1)

class ducky.mm.binary.RelocFlagsEncoding
    Bases: _ctypes.Structure

    _fields_ = [('relative', <class 'ctypes.c_ushort'>, 1), ('inst_aligned', <class 'ctypes.c_ushort'>, 1)]

    inst_aligned
        Structure/Union member

```

relative

Structure/Union member

class `ducky.mm.binary.Section` (*parent, index, name=None, header=None, payload=None*)

Bases: `object`

_create_header()

_create_payload()

_get_payload()

_read_header()

_read_payload()

_set_payload(*payload*)

_write_header()

_write_payload()

header

name

payload

prepare_write()

write()

class `ducky.mm.binary.SectionFlags`

Bases: `ducky.util.Flags`

_encoding

alias of `SectionFlagsEncoding`

_flags = ['readable', 'writable', 'executable', 'loadable', 'bss', 'globally_visible']

_labels = 'RWXLBG'

field = ('globally_visible', <class 'ctypes.c_ubyte'>, 1)

class `ducky.mm.binary.SectionFlagsEncoding`

Bases: `_ctypes.Structure`

fields = [('readable', <class 'ctypes.c_ubyte'>, 1), ('writable', <class 'ctypes.c_ubyte'>, 1), ('executable', <class 'ctyp

bss

Structure/Union member

executable

Structure/Union member

globally_visible

Structure/Union member

loadable

Structure/Union member

readable

Structure/Union member

writable

Structure/Union member

```

class ducky.mm.binary.SectionHeader
    Bases: _ctypes.Structure

    _fields_ = [('index', <class 'ctypes.c_ubyte'>), ('name', <class 'ctypes.c_uint'>), ('type', <class 'ctypes.c_ubyte'>), ('flags', <class 'ctypes.c_uint'>), ('padding', <class 'ctypes.c_ubyte'>)]
    _pack_ = 0

    base
        Structure/Union member

    data_size
        Structure/Union member

    file_size
        Structure/Union member

    flags
        Structure/Union member

    index
        Structure/Union member

    name
        Structure/Union member

    offset
        Structure/Union member

    padding
        Structure/Union member

    type
        Structure/Union member

class ducky.mm.binary.SectionTypes
    Bases: enum.IntEnum

    PROGBITS = 1
    RELOC = 4
    STRINGS = 3
    SYMBOLS = 2
    UNKNOWN = 0

    _member_map_ = OrderedDict([('UNKNOWN', <SectionTypes.UNKNOWN: 0>), ('PROGBITS', <SectionTypes.PROGBITS: 1>), ('STRINGS', <SectionTypes.STRINGS: 3>), ('SYMBOLS', <SectionTypes.SYMBOLS: 2>), ('RELOC', <SectionTypes.RELOC: 4>)])
    _member_names_ = ['UNKNOWN', 'PROGBITS', 'SYMBOLS', 'STRINGS', 'RELOC']
    _member_type_
        alias of int

    _value2member_map_ = {0: <SectionTypes.UNKNOWN: 0>, 1: <SectionTypes.PROGBITS: 1>, 2: <SectionTypes.SYMBOLS: 2>, 3: <SectionTypes.STRINGS: 3>, 4: <SectionTypes.RELOC: 4>}

class ducky.mm.binary.SymbolDataTypes
    Bases: enum.IntEnum

    ASCII = 5
    BYTE = 3
    CHAR = 2
    FUNCTION = 6

```

```
INT = 0
SHORT = 1
STRING = 4
UNKNOWN = 7

_member_map_ = OrderedDict([('INT', <SymbolDataTypes.INT: 0>), ('SHORT', <SymbolDataTypes.SHORT: 1>), ('CHAR', <SymbolDataTypes.CHAR: 2>), ('BYTE', <SymbolDataTypes.BYTE: 3>), ('STRING', <SymbolDataTypes.STRING: 4>), ('ASCII', <SymbolDataTypes.ASCII: 5>), ('FUNCTION', <SymbolDataTypes.FUNCTION: 6>), ('UNKNOWN', <SymbolDataTypes.UNKNOWN: 7>)])
_member_names_ = ['INT', 'SHORT', 'CHAR', 'BYTE', 'STRING', 'ASCII', 'FUNCTION', 'UNKNOWN']
_member_type_ =
    alias of int
_value2member_map_ = {0: <SymbolDataTypes.INT: 0>, 1: <SymbolDataTypes.SHORT: 1>, 2: <SymbolDataTypes.CHAR: 2>, 3: <SymbolDataTypes.BYTE: 3>, 4: <SymbolDataTypes.STRING: 4>, 5: <SymbolDataTypes.ASCII: 5>, 6: <SymbolDataTypes.FUNCTION: 6>, 7: <SymbolDataTypes.UNKNOWN: 7>}
```

```
class ducky.mm.binary.SymbolEntry
    Bases: _ctypes.Structure
    _fields_ = [('flags', <class 'ducky.mm.binary.SymbolFlagsEncoding'>), ('name', <class 'ctypes.c_uint'>), ('address', <class 'ctypes.c_uint'>), ('filename', <class 'ctypes.c_char_p'>), ('lineno', <class 'ctypes.c_uint'>), ('size', <class 'ctypes.c_uint'>), ('type', <class 'ctypes.c_uint'>)]
    _pack_ = 0
    address
        Structure/Union member
    filename
        Structure/Union member
    flags
        Structure/Union member
    lineno
        Structure/Union member
    name
        Structure/Union member
    section
        Structure/Union member
    size
        Structure/Union member
    type
        Structure/Union member
```

```
class ducky.mm.binary.SymbolFlags
    Bases: ducky.util.Flags
    _encoding
        alias of SymbolFlagsEncoding
    _flags = ['globally_visible']
    _labels = 'G'
    field = ('globally_visible', <class 'ctypes.c_ushort'>, 1)
```

```
class ducky.mm.binary.SymbolFlagsEncoding
    Bases: _ctypes.Structure
    _fields_ = [('globally_visible', <class 'ctypes.c_ushort'>, 1)]
    globally_visible
        Structure/Union member
```

2.12.2 Module contents

class `ducky.mm.AnonymousMemoryPage` (*controller, index*)

Bases: `ducky.mm.MemoryPage`

“Anonymous” memory page - this page is just a plain array of bytes, and is not backed by any storage. Its content lives only in the memory.

Page is created with all bytes set to zero.

clear ()

read_u16 (*offset*)

read_u32 (*offset*)

read_u8 (*offset*)

write_u16 (*offset, value*)

write_u32 (*offset, value*)

write_u8 (*offset, value*)

class `ducky.mm.ExternalMemoryPage` (*controller, index, data, offset=0*)

Bases: `ducky.mm.MemoryPage`

Memory page backed by an external source. Source is an array of bytes, and can be provided by device driver, mmaped file, or by any other mean.

clear ()

get (*offset*)

Get one byte from page. Override this method in case you need a different offset of requested byte.

Parameters *offset* (*int*) – offset of the requested byte.

Return type *int*

Returns byte at position in page.

put (*offset, b*)

Put one byte into page. Override this method in case you need a different offset of requested byte.

Parameters

- **offset** (*int*) – offset of modified byte.
- **b** (*int*) – new value.

read_u16 (*offset*)

read_u32 (*offset*)

read_u8 (*offset*)

save_state (*parent*)

write_u16 (*offset, value*)

write_u32 (*offset, value*)

write_u8 (*offset, value*)

class `ducky.mm.MMOperationList`

Bases: `enum.IntEnum`

ALLOC = 3

FREE = 4

MMAP = 6

UNMMAP = 7

UNUSED = 5

_member_map_ = `OrderedDict([('ALLOC', <MMOperationList.ALLOC: 3>), ('FREE', <MMOperationList.FREE: 4>), ('UNUSED', <MMOperationList.UNUSED: 5>), ('MMAP', <MMOperationList.MMAP: 6>), ('UNMMAP', <MMOperationList.UNMMAP: 7>)])`

_member_names_ = ['ALLOC', 'FREE', 'UNUSED', 'MMAP', 'UNMMAP']

_member_type_
alias of `int`

_value2member_map_ = {3: <MMOperationList.ALLOC: 3>, 4: <MMOperationList.FREE: 4>, 5: <MMOperationList.UNUSED: 5>, 6: <MMOperationList.MMAP: 6>, 7: <MMOperationList.UNMMAP: 7>}

exception `ducky.mm.MalformedBinaryError`

Bases: `exceptions.Exception`

class `ducky.mm.MemoryController` (*machine*, *size=16777216*)

Bases: `object`

Memory controller handles all operations regarding main memory.

Parameters

- **machine** (`ducky.machine.Machine`) – virtual machine that owns this controller.
- **size** (`int`) – size of memory, in bytes.

Raises `ducky.errors.InvalidResourceError` – when memory size is not multiple of `ducky.mm.PAGE_SIZE`.

`MemoryController.alloc_page` (*index*)

Allocate new anonymous page for usage. The first available index is used.

Be aware that this method does NOT check if page is already allocated. If it is, it is just overwritten by new anonymous page.

Parameters **index** (`int`) – index of requested page.

Returns newly reserved page.

Return type `ducky.mm.AnonymousMemoryPage`

`MemoryController.remove_page` (*pg*)

Removes page object for a specific memory page.

Parameters **pg** (`ducky.mm.MemoryPage`) – page to be removed

`MemoryController.set_page` (*pg*)

Install page object for a specific memory page.

Parameters **pg** (`ducky.mm.MemoryPage`) – page to be installed

Returns installed page

Return type `ducky.mm.MemoryPage`

`alloc_page` (*base=None*)

Allocate new anonymous page for usage. The first available index is used.

Parameters **base** (`int`) – if set, start searching pages from this address.

Returns newly reserved page.

Return type `ducky.mm.AnonymousMemoryPage`

Raises `ducky.errors.InvalidResourceError` – when there is no available page.

alloc_pages (*base=None, count=1*)

Allocate continuous sequence of anonymous pages.

Parameters

- **base** (*u24*) – if set, start searching pages from this address.
- **count** (*int*) – number of requested pages.

Returns list of newly allocated pages.

Return type list of `ducky.mm.AnonymousMemoryPage`

Raises `ducky.errors.InvalidResourceError` – when there is no available sequence of pages.

alloc_specific_page (*index*)

Allocate new anonymous page with specific index for usage.

Parameters **index** (*int*) – allocate page with this particular index.

Returns newly reserved page.

Return type `ducky.mm.AnonymousMemoryPage`

Raises `ducky.errors.AccessViolationError` – when page is already allocated.

boot ()

Prepare memory controller for immediate usage by other components.

free_page (*page*)

Free memory page when it's no longer needed.

Parameters **page** (`ducky.mm.MemoryPage`) – page to be freed.

free_pages (*page, count=1*)

Free a continuous sequence of pages when they are no longer needed.

Parameters

- **page** (`ducky.mm.MemoryPage`) – first page in series.
- **count** (*int*) – number of pages.

get_page (*index*)

Return memory page, specified by its index from the beginning of memory.

Parameters **index** (*int*) – index of requested page.

Return type `ducky.mm.MemoryPage`

Raises `ducky.errors.AccessViolationError` – when requested page is not allocated.

get_pages (*pages_start=0, pages_cnt=None, ignore_missing=False*)

Return list of memory pages.

Parameters

- **pages_start** (*int*) – index of the first page, 0 by default.
- **pages_cnt** (*int*) – number of pages to get, number of all memory pages by default.
- **ignore_missing** (*bool*) – if True, ignore missing pages, False by default.

Raises `ducky.errors.AccessViolationError` – when `ignore_missing == False` and there's a missing page in requested range, this exception is rised.

Returns list of pages in area

Return type *list of `ducky.mm.MemoryPage`*

`halt()`

`load_state(state)`

`pages_in_area(address=0, size=None, ignore_missing=False)`

Return list of memory pages.

Parameters

- **address** (*u24*) – beggining address of the area, by default 0.
- **size** (*u24*) – size of the area, by default the whole memory size.
- **ignore_missing** (*bool*) – if True, ignore missing pages, False by default.

Raises `ducky.errors.AccessViolationError` – when `ignore_missing == False` and there's a missing page in requested range, this exception is rised.

Returns list of pages in area

Return type *list of `ducky.mm.MemoryPage`*

`read_u16(addr)`

`read_u32(addr)`

`read_u8(addr)`

`register_page(pg)`

Install page object for a specific memory page. This method is intended for external objects, e.g. device drivers to install their memory page objects to handle memory-mapped IO.

Parameters **pg** (`ducky.mm.MemoryPage`) – page to be installed

Returns installed page

Return type `ducky.mm.AnonymousMemoryPage`

Raises `ducky.errors.AccessViolationError` – when there is already allocated page

`save_state(parent)`

`unregister_page(pg)`

Remove page object for a specific memory page. This method is intende for external objects, e.g. device drivers to remove their memory page objects handling memory-mapped IO.

Parameters **pg** (`ducky.mm.MemoryPage`) – page to be removed

Raises `ducky.errors.AccessViolationError` – when there is no allocated page

`write_u16(addr, value)`

`write_u32(addr, value)`

`write_u8(addr, value)`

class `ducky.mm.MemoryPage(controller, index)`

Bases: `object`

Base class for all memory pages of any kinds.

Memory page has a set of boolean flags that determine access to and behavior of the page.

Flag	Meaning	Default
read	page is readable by executed instructions	False
write	page is writable by executed instructions	False
execute	content of the page can be used as executable instructions	False
dirty	there have been write access to this page, its content has changed	False

Parameters

- **controller** (`ducky.mm.MemoryController`) – Controller that owns this page.
- **index** (`int`) – Serial number of this page.

clear()

Clear page.

This operation is implemented by child classes.

load_state (`state`)

Restore page from a snapshot.

read_u16 (`offset`)

Read word.

This operation is implemented by child classes.

Parameters **offset** (`int`) – offset of requested word.

Return type `int`

read_u32 (`offset`)

Read longword.

This operation is implemented by child classes.

Parameters **offset** (`int`) – offset of requested longword.

Return type `int`

read_u8 (`offset`)

Read byte.

This operation is implemented by child classes.

Parameters **offset** (`int`) – offset of requested byte.

Return type `int`

save_state (`parent`)

Create state of this page, and attach it to snapshot tree.

Parameters **parent** (`ducky.snapshot.SnapshotNode`) – Parent snapshot node.

write_u16 (`offset, value`)

Write word.

This operation is implemented by child classes.

Parameters

- **offset** (`int`) – offset of requested word.
- **value** (`int`) – value to write into memory.

write_u32 (`offset, value`)

Write longword.

This operation is implemented by child classes.

Parameters

- **offset** (*int*) – offset of requested longword.
- **value** (*int*) – value to write into memory.

write_u8 (*offset, value*)

Write byte.

This operation is implemented by child classes.

Parameters

- **offset** (*int*) – offset of requested byte.
- **value** (*int*) – value to write into memory.

class `ducky.mm.MemoryPageState` (**args, **kwargs*)

Bases: `ducky.snapshot.SnapshotNode`

class `ducky.mm.MemoryRegion` (*mc, name, address, size, flags*)

Bases: `ducky.interfaces.ISnapshotable`, `object`

load_state (*state*)

region_id = 0

save_state (*parent*)

class `ducky.mm.MemoryRegionState`

Bases: `ducky.snapshot.SnapshotNode`

class `ducky.mm.MemoryState`

Bases: `ducky.snapshot.SnapshotNode`

get_page_states ()

`ducky.mm.OFFSET_FMT` (*offset*)

`ducky.mm.PAGE_SIZE` = 256

Size of memory page, in bytes.

class `ducky.mm.PageTableEntry`

Bases: `ducky.util.Flags`

DIRTY = 8

EXECUTE = 4

READ = 1

WRITE = 2

_flags = ['read', 'write', 'execute', 'dirty']

_labels = 'RWXD'

`ducky.mm.SIZE_FMT` (*size*)

class `ducky.mm.VirtualMemoryPage` (*controller, index*)

Bases: `ducky.mm.MemoryPage`

Memory page without any real storage backend.

save_state (*parent*)

`ducky.mm.addr_to_offset` (*addr*)

```
ducky.mm.addr_to_page(addr)
ducky.mm.area_to_pages(addr, size)
```

2.13 ducky.patch module

```
class ducky.patch.Importer
    Bases: object
        find_module(fullname, path=None)
        loader_for_path(directory, fullname)
class ducky.patch.ModuleLoader(fullpath)
    Bases: object
        get_code(fullname)
        get_source(path)
        load_module(fullname)
class ducky.patch.RemoveLoggingVisitor
    Bases: ast.NodeTransformer
        visit_Expr(node)
        visit_For(node)
        visit_If(node)
ducky.patch.debug(s)
ducky.patch.exec_f(object_, globals_=None, locals_=None)
```

2.14 ducky.profiler module

This module provides support for profiling the virtual machine (*machine* profilers) and running programs (*code* profilers). Wrappers for python's deterministic profilers, and simple statistical profiler of running code are available for usage throughout the ducky sources.

cProfile is the preferred choice of machine profiling backend, with profile is used as a backup option.

Beware, each thread needs its own profiler instance. These instances can be acquired from ProfilerStore class which handles saving their data when virtual machine exits.

To simplify the usage of profilers in ducky sources in case when user does not need profiling, I chose to provide classes with the same API but these classes does not capture any data. These are called *dummy* profilers, as opposed to the *real* ones. Both kinds mimic API of profile.Profile - the *real* machine profiler is profile.Profile object.

```
class ducky.profiler.DummyCPUCoreProfiler(core, frequency=17)
    Bases: object
```

Dummy code profiler class. Base class for all code profilers.

Parameters

- **core** (ducky.cpu.CPUCore) – core this profiler captures data from.
- **frequency** (int) – sampling frequency, given as an instruction count.

create_stats()

Preprocess collected data before they can be printed, searched or saved.

disable()

Disable collection of profiling data.

dump_stats(filename)

Save collected data into file.

Parameters filename (*string*) – path to file.

enable()

Enable collection of profiling data.

take_sample()

Take a sample of current state of CPU core, and store any necessary data.

class `ducky.profiler.DummyMachineProfiler(*args, **kwargs)`

Bases: `object`

Dummy machine profiler. Does absolutely nothing.

create_stats()

Preprocess collected data before they can be printed, searched or saved.

disable()

Stop collecting profiling data.

dump_stats(filename)

Save collected data into file.

Parameters filename (*string*) – path to file.

enable()

Start collecting profiling data.

class `ducky.profiler.ProfileRecord`

Bases: `object`

merge(other)

class `ducky.profiler.ProfilerStore`

Bases: `object`

This class manages profiler instances. When in need of a profiler (e.g. in a new thread) get one by calling proper method of ProfilerStore object.

enable_cpu()

Each newly created code profiler will be the real one.

enable_machine()

Each newly created virtual machine profiler will be the real one.

get_core_profiler(core)

Create new code profiler.

Return type *DummyCPUCoreProfiler*

get_machine_profiler()

Create and return new machine profiler.

Returns new machine profiler.

is_cpu_enabled()

Returns `True` when code profiling is enabled.

Return type bool

is_machine_enabled()

Returns True when virtual machine profiling is enabled.

Return type bool

save (*logger*, *directory*)

Save all captured data to files. Each created profiler stores its data in separate file.

Parameters **directory** (*string*) – directory where all files are stored.

class `ducky.profiler.RealCPUCoreProfiler` (*core*)

Bases: `ducky.profiler.DummyCPUCoreProfiler`

Real code profiler. This class actually does collect data.

dump_stats (*filename*)

take_sample ()

`ducky.profiler.STORE` = `<ducky.profiler.ProfilerStore object>`

Main profiler store

2.15 ducky.reactor module

This module provides simple reactor core that runs each of registered tasks at least once during one iteration of its internal loop.

There are two different kinds of objects that reactor manages:

- task - it's called periodically, at least once in each reactor loop iteration
- event - asynchronous events are queued and executed before running any tasks. If there are no runnable tasks, reactor loop waits for incoming events.

class `ducky.reactor.CallInReactorTask` (*fn*, **args*, ***kwargs*)

Bases: `ducky.interfaces.IReactorTask`

This task request running particular function during the reactor loop. Useful for planning future work, and for running tasks in reactor's thread.

Parameters

- **fn** – callback to fire.
- **args** – arguments for callback.
- **kwargs** – keyword arguments for callback.

run ()

class `ducky.reactor.FDCallbacks` (*on_read*, *on_write*, *on_error*)

Bases: tuple

_asdict ()

Return a new OrderedDict which maps field names to their values

_fields = ('on_read', 'on_write', 'on_error')

classmethod **_make** (*iterable*, *new=<built-in method __new__ of type object at 0x906d60>*, *len=<built-in function len>*)

Make a new FDCallbacks object from a sequence or iterable

_replace (*_self*, ***kws*)

Return a new FDCallbacks object replacing specified fields with new values

on_error

Alias for field number 2

on_read

Alias for field number 0

on_write

Alias for field number 1

class `ducky.reactor.Reactor` (*machine*)

Bases: `object`

Main reactor class.

add_call (*fn*, **args*, ***kwargs*)

Enqueue function call. Function will be called in reactor loop.

add_event (*event*)

Enqueue asynchronous event.

add_fd (*fd*, *on_read=None*, *on_write=None*, *on_error=None*)

Register file descriptor with reactor. File descriptor will be checked for read/write/error possibilities, and appropriate callbacks will be fired.

No arguments are passed to callbacks.

Parameters

- **fd** (*int*) – file descriptor.
- **on_read** – function that will be called when file descriptor is available for reading.
- **on_write** – function that will be called when file descriptor is available for write.
- **on_error** – function that will be called when error state raised on file descriptor.

add_task (*task*)

Register task with reactor's main loop.

remove_fd (*fd*)

Unregister file descriptor. It will no longer be checked by its main loop.

Parameters **fd** (*int*) – previously registered file descriptor.

remove_task (*task*)

Unregister task, it will never be ran again.

run ()

Starts reactor loop. Enters endless loop, calling runnable tasks and events, and - in case there are no runnable tasks - waits for new events.

When there are no tasks managed by reactor, loop quits.

task_runnable (*task*)

If not yet marked as such, task is marked as runnable, and its `run()` method will be called every iteration of reactor's main loop.

task_suspended (*task*)

If runnable, task is marked as suspended, not runnable, and it will no longer be ran by reactor. It's still registered, so reactor's main loop will not quit, and task can be later easily re-enabled by calling `ducky.reactor.Reactor.task_runnable()`.

class `ducky.reactor.RunInIntervalTask` (*ticks*, *fn*, **args*, ***kwargs*)

Bases: `ducky.interfaces.IReactorTask`

This task will run its callback every `ticks` iterations of reactor's main loop.

Parameters

- **ticks** (*int*) – number of main loop iterations between two callback calls.
- **args** – arguments for callback.
- **kwargs** – keyword arguments for callback.

run ()

class `ducky.reactor.SelectTask` (*machine*, *fds*, **args*, ***kwargs*)

Bases: `ducky.interfaces.IReactorTask`

Private task, serving as a single point where `select` syscall is being executed. When a subsystem is interested in IO on a file descriptor, such file descriptor should be set as non-blocking, and then registered with reactor - it's not viable to place `select` calls everywhere in different drivers. This task takes list of registered file descriptors, checks for possible IO opportunities, and fires callbacks accordingly.

Parameters

- **machine** (`ducky.machine.Machine`) – VM this task (and reactor) belongs to.
- **fds** (*dict*) – dictionary, where keys are descriptors, and values are lists of their callbacks.

add_fd (*fd*, *on_read=None*, *on_write=None*, *on_error=None*)

Register file descriptor with reactor. File descriptor will be checked for read/write/error possibilities, and appropriate callbacks will be fired.

No arguments are passed to callbacks.

Parameters

- **fd** (*int*) – file descriptor.
- **on_read** – function that will be called when file descriptor is available for reading.
- **on_write** – function that will be called when file descriptor is available for write.
- **on_error** – function that will be called when error state raised on file descriptor.

remove_fd (*fd*)

Unregister file descriptor. It will no longer be checked by its main loop.

Parameters **fd** (*int*) – previously registered file descriptor.

run ()

2.16 ducky.snapshot module

class `ducky.snapshot.CoreDumpFile` (*logger*, *stream*)

Bases: `ducky.util.BinaryFile`

load ()

static open (**args*, ***kwargs*)

save (*state*)

class `ducky.snapshot.SnapshotNode` (**fields*)

Bases: `object`

```
    add_child (name, child)
    get_child (name)
    get_children ()
    print_node (level=0)
class ducky.snapshot.VMState (logger)
    Bases: ducky.snapshot.SnapshotNode
    static capture_vm_state (machine, suspend=True)
    static load_vm_state (logger, filename)
    save (filename)
```

2.17 ducky.streams module

Streams represent basic IO objects, used by devices for reading or writing (streams) of data.

Stream object encapsulates an actual IO object - file-like stream, raw file descriptor, or even something completely different. Stream classes then provide basic IO methods for moving data to and from stream, shielding user from implementation details, like Python2/Python3 differencies.

```
class ducky.streams.FDInputStream (machine, fd, **kwargs)
    Bases: ducky.streams.InputStream
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29
    _abc_registry = <_weakrefset.WeakSet object>
class ducky.streams.FDOutputStream (machine, fd, **kwargs)
    Bases: ducky.streams.OutputStream
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29
    _abc_registry = <_weakrefset.WeakSet object>
class ducky.streams.FileInputStream (machine, f, **kwargs)
    Bases: ducky.streams.InputStream
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29
    _abc_registry = <_weakrefset.WeakSet object>
class ducky.streams.FileOutputStream (machine, f, **kwargs)
    Bases: ducky.streams.OutputStream
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29
```

```

    _abc_registry = <_weakrefset.WeakSet object>
class ducky.streams.InputStream(machine, desc, stream=None, fd=None, close=True, al-
                                low_close=True)
    Bases: ducky.streams.Stream
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29
    _abc_registry = <_weakrefset.WeakSet object>
    static create(machine, desc)
    read(size=None)
    write(b)
class ducky.streams.MethodInputStream(machine, desc, **kwargs)
    Bases: ducky.streams.InputStream
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29
    _abc_registry = <_weakrefset.WeakSet object>
class ducky.streams.MethodOutputStream(machine, desc, **kwargs)
    Bases: ducky.streams.OutputStream
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29
    _abc_registry = <_weakrefset.WeakSet object>
class ducky.streams.OutputStream(machine, desc, stream=None, fd=None, close=True, al-
                                low_close=True)
    Bases: ducky.streams.Stream
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29
    _abc_registry = <_weakrefset.WeakSet object>
    static create(machine, desc)
    flush()
    read(size=None)
    write(buff)
class ducky.streams.StderrStream(machine)
    Bases: ducky.streams.OutputStream
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29

```

```
    _abc_registry = <_weakrefset.WeakSet object>
class ducky.streams.StdinStream(machine, **kwargs)
    Bases: ducky.streams.InputStream
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29
    _abc_registry = <_weakrefset.WeakSet object>
    close()
    get_selectee()
class ducky.streams.StdoutStream(machine)
    Bases: ducky.streams.OutputStream
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29
    _abc_registry = <_weakrefset.WeakSet object>
class ducky.streams.Stream(machine, desc, stream=None, fd=None, close=True, allow_close=True)
    Bases: object
    Abstract base class of all streams.
```

Parameters

- **machine** – parent :py:class:`ducky.machine.Machine` object.
- **desc** – description of stream. This is a short, string representation of the stream.
- **stream** – file-like stream that provides IO method (`read()` or `write()`). If it is set, it is preferred IO object.
- **fd** (*int*) – raw file descriptor. `stream` takes precedence, otherwise this file descriptor is used.
- **close** (*bool*) – if True, and if `stream` has a `close()` method, `stream` will provide `close()` method that will close the underlying file-like object. True by default.
- **allow_close** (*bool*) – if not True, `stream`'s `close()` method will *not* close underlying IO resource. True by default.

```
    _abc_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache = <_weakrefset.WeakSet object>
    _abc_negative_cache_version = 29
    _abc_registry = <_weakrefset.WeakSet object>
    _raw_read_fd(size=None)
    _raw_read_stream(size=None)
    _raw_write_fd(data)
    _raw_write_stream(data)
```

close()

This method will close the stream. If `allow_close` flag is not set to `True`, nothing will happen. If the stream wasn't created with `close` set to `True`, nothing will happen. If the wrapped IO resource does not have `close()` method, nothing will happen.

has_fd()

Check if stream has raw file descriptor. File descriptors can be checked for IO availability by reactor's polling task.

Return type bool

Returns `True` when stream has file descriptor.

has_poll_support()

Streams that can be polled for data should return `True`.

Return type bool

read(size=None)

Read data from stream.

Parameters `size (int)` – if set, read at maximum `size` bytes.

Return type bytearray (Python2), bytes (Python3)

Returns read data, of maximum length of `size`, `None` when there are no available data, or empty string in case of EOF.

register_with_reactor(reactor, **kwargs)

Called by owner to register the stream with reactor's polling service.

See `ducky.reactor.Reactor.add_fd()` for keyword arguments.

Parameters `reactor (ducky.reactor.Reactor)` – reactor instance to register with.

unregister_with_reactor(reactor)

Called by owner to unregister the stream with reactor's polling service, e.g. when stream is about to be closed.

Parameters `reactor (ducky.reactor.Reactor)` – reactor instance to unregister from.

write(buff)

Write data to stream.

Parameters `buff (bytearray)` – data to write. bytearray (Python2), bytes (Python3)

ducky.streams.fd_blocking(fd, block=None)

Query or set blocking mode of file descriptor.

Return type bool

Returns if `block` is `None`, current setting of blocking mode is returned - `True` for blocking, `False` for non-blocking. Otherwise, function returns nothing.

2.18 ducky.tools package

2.18.1 Submodules

ducky.tools.as module

`ducky.tools.as.encode_blob(logger, file_in, options)`

```
ducky.tools.as.get_assembler_process(logger, buffer, file_in, options)
ducky.tools.as.main()
ducky.tools.as.save_object_file(logger, sections, file_out, options)
```

ducky.tools.coredump module

```
ducky.tools.coredump.__load_forth_symbols(logger)
ducky.tools.coredump.__read(state, cnt, address)
ducky.tools.coredump.__show_forth_word(state, symbols, base_address, ending_addresses)
ducky.tools.coredump.main()
ducky.tools.coredump.show_cores(logger, state)
ducky.tools.coredump.show_dump(state, dumps)
ducky.tools.coredump.show_forth_dict(logger, state, last)
ducky.tools.coredump.show_forth_word(logger, state, base_address)
ducky.tools.coredump.show_header(logger, state)
ducky.tools.coredump.show_memory(logger, state)
ducky.tools.coredump.show_pages(logger, state, empty_pages=False)
```

ducky.tools.defs module

```
ducky.tools.defs.main()
ducky.tools.defs.parse_template(file_in, file_out)
```

ducky.tools.img module

```
ducky.tools.img.align_file(f_out, boundary)
ducky.tools.img.create_binary_image(f_in, f_out, bio=False)
ducky.tools.img.create_hdt_image(file_in, f_out, options)
ducky.tools.img.main()
```

ducky.tools.ld module

```
class ducky.tools.ld.LinkerInfo(linker_script)
    Bases: object

class ducky.tools.ld.LinkerScript(filepath=None)
    Bases: object

    section_ordering()

    where_to_base(section)

    where_to_merge(src_section)
```

```

class ducky.tools.ld.RelocationPatcher(re, se, symbol_name, section, original_section=None,
                                     section_offset=0)
    Bases: object
    _apply_patch(value)
    _create_patch()
    patch()

ducky.tools.ld.archive_files(logger, files_in, file_out)
ducky.tools.ld.fix_section_bases(info, f_out)
ducky.tools.ld.link_files(info, files_in, file_out)
ducky.tools.ld.main()
ducky.tools.ld.merge_object_into(info, f_dst, f_src)
ducky.tools.ld.resolve_relocations(info, f_out, f_ins)
ducky.tools.ld.resolve_symbols(info, f_out, f_ins)

```

ducky.tools.objdump module

```

ducky.tools.objdump.main()
ducky.tools.objdump.show_disassemble(f)
ducky.tools.objdump.show_file_header(f)
ducky.tools.objdump.show_reloc(f)
ducky.tools.objdump.show_sections(options, f)
ducky.tools.objdump.show_symbols(options, f)

```

ducky.tools.profile module

```

ducky.tools.profile.main()
ducky.tools.profile.read_profiling_data(logger, files_in)

```

ducky.tools.vm module

```

class ducky.tools.vm.DuckyProtocol(logger, options, config)
    Bases: autobahn.twisted.websocket.WebSocketServerProtocol
    Protocol handling communication between VM and remote terminal emulator.

    Parameters
        • logger (logging.Logger) – Logger instance to use for logging.
        • options – command-line options, as returned by option parser.
        • config (ducky.config.MachineConfig) – VM configuration.

    _machines = []

    onClose(wasClean, code, reason)
        Called when connection ends. Tell VM to halt, and wait for its thread to finish. Then, print some VM stats.

```

onMessage (*payload, isBinary*)

Called when new data arrived from client. Feeds the data to VM's input stream.

See `autobahn.twisted.websocket.WebSocketServerProtocol.onMessage()`.

onOpen (**args, **kwargs*)

Called when new client connects to the server.

This callback will setup and start new VM, connecting it to remote terminal by wrapping this protocol instance in two streams (input/output), and spawn a fresh new thread for it.

run_machine ()

Wraps VM's `run()` method by `try/except` clause, and call protocols `sendClose()` method when VM finishes.

This is the target of VM's thread.

class `ducky.tools.vm.DuckySocketServerFactory` (*logger, options, config, *args, **kwargs*)

Bases: `autobahn.twisted.websocket.WebSocketServerFactory`

buildProtocol (**args, **kwargs*)

class `ducky.tools.vm.WSInputStream` (*protocol, *args, **kwargs*)

Bases: `ducky.streams.InputStream`

Websocket input stream, receiving bytes from opened websocket, and pushing them to keyboard frontend device.

Stream has an internal LIFO buffer that is being fed by protocol, and consumed by VM frontend device.

Parameters `protocol` (`DuckyProtocol`) – protocol instance with opened websocket.

`_abc_cache` = `<_weakrefset.WeakSet object>`

`_abc_negative_cache` = `<_weakrefset.WeakSet object>`

`_abc_negative_cache_version` = 39

`_abc_registry` = `<_weakrefset.WeakSet object>`

enqueue (*buff*)

Called by protocol instance, to add newly received data to stream's buffer.

Parameters `buff` (`bytearray`) – received bytes.

has_poll_support ()

See `ducky.streams.Stream.has_poll_support()`.

read (*size=None*)

See `ducky.streams.Stream.read()`.

register_with_reactor (*reactor, on_read=None, on_error=None*)

See `ducky.streams.Stream.register_with_reactor()`.

unregister_with_reactor (*reactor*)

See `ducky.streams.Stream.unregister_with_reactor()`.

class `ducky.tools.vm.WSOutputStream` (*protocol, *args, **kwargs*)

Bases: `ducky.streams.OutputStream`

Websocket output stream, receiving bytes from TTY frontend, and pushing them to protocol's socket.

Parameters `protocol` (`DuckyProtocol`) – protocol instance with opened websocket.

`_abc_cache` = `<_weakrefset.WeakSet object>`

`_abc_negative_cache` = `<_weakrefset.WeakSet object>`

```
_abc_negative_cache_version = 39
_abc_registry = <_weakrefset.WeakSet object>
```

```
write (buff)
```

Write buffer into the socket.

Called by device from machine thread, therefore this method hands buffer over to the reactor thread.

Parameters *buff* (*bytearray*) – bytes to send to client.

```
ducky.tools.vm.main ()
```

```
ducky.tools.vm.print_machine_stats (logger, M)
```

```
ducky.tools.vm.process_config_options (logger, options, config_file=None, set_options=None,
                                         add_options=None, enable_devices=None, dis-
                                         able_devices=None)
```

Load VM config file, and apply all necessary changes, as requested by command-line options.

Parameters

- **logger** (*logging.Logger*) – Logger instance to use for logging.
- **options** – command-line options, as returned by option parser.
- **config_file** (*string*) – path to configuration file.

Return type *ducky.config.MachineConfig*

Returns VM configuration.

2.18.2 Module contents

```
ducky.tools.add_common_options (parser)
```

```
ducky.tools.parse_options (parser, default_loglevel=20, stream=None)
```

```
ducky.tools.setup_logger (stream=None, debug=False, quiet=None, verbose=None, de-
                           fault_loglevel=20, colorize=None)
```

2.19 ducky.util module

```
class ducky.util.BinaryFile (logger, stream)
```

Bases: object

Base class of all classes that represent “binary” files - binaries, core dumps. It provides basic methods for reading and writing structures.

```
static do_open (logger, path, mode='rb', klass=None)
```

```
static open (*args, **kwargs)
```

```
read_struct (st_class)
```

Read structure from current position in file.

Returns instance of class *st_class* with content read from file

Return type *st_class*

```
setup ()
```

write_struct (*st*)

Write structure into file at the current position.

Parameters **st** (*class*) – ctype-based structure

class `ducky.util.Flags`

Bases: `object`

_encoding = []

_flags = []

_labels = ''

classmethod **create** (***kwargs*)

classmethod **encoding** ()

classmethod **from_encoding** (*encoding*)

classmethod **from_int** (*u*)

classmethod **from_string** (*s*)

load_encoding (*encoding*)

load_int (*u*)

load_string (*s*)

save_encoding (*encoding*)

to_encoding ()

to_int ()

to_string ()

class `ducky.util.Formatter`

Bases: `string.Formatter`

format_field (*value, format_spec*)

format_int (*format_spec, value*)

class `ducky.util.LoggingCapable` (*logger, *args, **kwargs*)

Bases: `object`

class `ducky.util.StringTable` (*buff=None*)

Bases: `object`

Simple string table, used by many classes operating with files (core, binaries, ...). String can be inserted into table and read, each has its starting offset and its end is marked with null byte ().

This is a helper class - it makes working with string, e.g. section and symbol names, much easier.

buff

Serialize internal string table to a stream of bytes.

get_string (*offset*)

Read string from table.

Parameters **offset** (*int*) – offset of the first character from the beginning of the table

Returns `string`

Return type `string`

put_string(*s*)

Insert new string into table. String is appended at the end of internal buffer.

Returns offset of inserted string

Return type int

class ducky.util.**SymbolTable**(*binary*)

Bases: dict

get_symbol(*name*)

ducky.util.**align**(*boundary*, *n*)

ducky.util.**isfile**(*o*)

ducky.util.**sizeof_fmt**(*n*, *suffix*='B', *max_unit*='Zi')

ducky.util.**str2int**(*s*)

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

d

- `ducky.boot`, 27
- `ducky.config`, 30
- `ducky.console`, 31
- `ducky.cpu`, 63
 - `ducky.cpu.coprocessor`, 40
 - `ducky.cpu.coprocessor.control`, 32
 - `ducky.cpu.coprocessor.math_copro`, 33
 - `ducky.cpu.instructions`, 40
 - `ducky.cpu.registers`, 61
- `ducky.debugging`, 69
- `ducky.devices`, 83
 - `ducky.devices.keyboard`, 72
 - `ducky.devices.rtc`, 73
 - `ducky.devices.snapshot`, 75
 - `ducky.devices.storage`, 75
 - `ducky.devices.svga`, 80
 - `ducky.devices.terminal`, 78
 - `ducky.devices.tty`, 79
- `ducky.errors`, 84
- `ducky.hdt`, 89
- `ducky.interfaces`, 91
- `ducky.log`, 92
- `ducky.machine`, 93
- `ducky.mm`, 101
 - `ducky.mm.binary`, 96
- `ducky.patch`, 107
- `ducky.profiler`, 107
- `ducky.reactor`, 109
- `ducky.snapshot`, 111
- `ducky.streams`, 112
- `ducky.tools`, 119
 - `ducky.tools.as`, 115
 - `ducky.tools.coredump`, 116
 - `ducky.tools.defs`, 116
 - `ducky.tools.img`, 116
 - `ducky.tools.ld`, 116
 - `ducky.tools.objdump`, 117
 - `ducky.tools.profile`, 117
 - `ducky.tools.vm`, 117
 - `ducky.util`, 119

Symbols

- `_BINOP` (class in `ducky.cpu.instructions`), 59
- `_BITOP` (class in `ducky.cpu.instructions`), 59
- `_BRANCH` (class in `ducky.cpu.instructions`), 59
- `_CMP` (class in `ducky.cpu.instructions`), 59
- `_COND` (class in `ducky.cpu.instructions`), 60
- `_CPUCore__get_flags()` (`ducky.cpu.CPUCore` method), 63
- `_CPUCore__set_flags()` (`ducky.cpu.CPUCore` method), 63
- `_DebuggingSet__check_chain()`
(`ducky.debugging.DebuggingSet` method), 70
- `_JUMP` (class in `ducky.cpu.instructions`), 60
- `_LOAD` (class in `ducky.cpu.instructions`), 60
- `_LOAD_IMM` (class in `ducky.cpu.instructions`), 60
- `_MachineConfig__count()` (`ducky.config.MachineConfig` method), 30
- `_MachineConfig__count_breakpoints()`
(`ducky.config.MachineConfig` method), 30
- `_MachineConfig__count_devices()`
(`ducky.config.MachineConfig` method), 30
- `_MachineConfig__count_mmaps()`
(`ducky.config.MachineConfig` method), 30
- `_MachineConfig__sections_with_prefix()`
(`ducky.config.MachineConfig` method), 30
- `_MemoryController__alloc_page()`
(`ducky.mm.MemoryController` method), 102
- `_MemoryController__remove_page()`
(`ducky.mm.MemoryController` method), 102
- `_MemoryController__set_page()`
(`ducky.mm.MemoryController` method), 102
- `_SELECT` (class in `ducky.cpu.instructions`), 60
- `_SET` (class in `ducky.cpu.instructions`), 61
- `_STORE` (class in `ducky.cpu.instructions`), 61
- `__load_forth_symbols()` (in module `ducky.tools.coredump`), 116
- `__read()` (in module `ducky.tools.coredump`), 116
- `__show_forth_word()` (in module `ducky.tools.coredump`), 116
- `_abc_cache` (`ducky.streams.FDInputStream` attribute), 112
- `_abc_cache` (`ducky.streams.FDOutputStream` attribute), 112
- `_abc_cache` (`ducky.streams.FileInputStream` attribute), 112
- `_abc_cache` (`ducky.streams.FileOutputStream` attribute), 112
- `_abc_cache` (`ducky.streams.InputStream` attribute), 113
- `_abc_cache` (`ducky.streams.MethodInputStream` attribute), 113
- `_abc_cache` (`ducky.streams.MethodOutputStream` attribute), 113
- `_abc_cache` (`ducky.streams.OutputStream` attribute), 113
- `_abc_cache` (`ducky.streams.StderrStream` attribute), 113
- `_abc_cache` (`ducky.streams.StdinStream` attribute), 114
- `_abc_cache` (`ducky.streams.StdoutStream` attribute), 114
- `_abc_cache` (`ducky.streams.Stream` attribute), 114
- `_abc_cache` (`ducky.tools.vm.WSInputStream` attribute), 118
- `_abc_cache` (`ducky.tools.vm.WSOutputStream` attribute), 118
- `_abc_negative_cache` (`ducky.streams.FDInputStream` attribute), 112
- `_abc_negative_cache` (`ducky.streams.FDOutputStream` attribute), 112
- `_abc_negative_cache` (`ducky.streams.FileInputStream` attribute), 112
- `_abc_negative_cache` (`ducky.streams.FileOutputStream` attribute), 112
- `_abc_negative_cache` (`ducky.streams.InputStream` attribute), 113
- `_abc_negative_cache` (`ducky.streams.MethodInputStream` attribute), 113
- `_abc_negative_cache` (`ducky.streams.MethodOutputStream` attribute), 113

- `_abc_negative_cache` (ducky.streams.OutputStream attribute), 113
- `_abc_negative_cache` (ducky.streams.StderrStream attribute), 113
- `_abc_negative_cache` (ducky.streams.StdinStream attribute), 114
- `_abc_negative_cache` (ducky.streams.StdoutStream attribute), 114
- `_abc_negative_cache` (ducky.streams.Stream attribute), 114
- `_abc_negative_cache` (ducky.tools.vm.WSInputStream attribute), 118
- `_abc_negative_cache` (ducky.tools.vm.WSOutputStream attribute), 118
- `_abc_negative_cache_version` (ducky.streams.FDInputStream attribute), 112
- `_abc_negative_cache_version` (ducky.streams.FDOutputStream attribute), 112
- `_abc_negative_cache_version` (ducky.streams.FileInputStream attribute), 112
- `_abc_negative_cache_version` (ducky.streams.FileOutputStream attribute), 112
- `_abc_negative_cache_version` (ducky.streams.InputStream attribute), 113
- `_abc_negative_cache_version` (ducky.streams.MethodInputStream attribute), 113
- `_abc_negative_cache_version` (ducky.streams.MethodOutputStream attribute), 113
- `_abc_negative_cache_version` (ducky.streams.OutputStream attribute), 113
- `_abc_negative_cache_version` (ducky.streams.StderrStream attribute), 113
- `_abc_negative_cache_version` (ducky.streams.StdinStream attribute), 114
- `_abc_negative_cache_version` (ducky.streams.StdoutStream attribute), 114
- `_abc_negative_cache_version` (ducky.streams.Stream attribute), 114
- `_abc_negative_cache_version` (ducky.tools.vm.WSInputStream attribute), 118
- `_abc_negative_cache_version` (ducky.tools.vm.WSOutputStream attribute), 118
- `_abc_registry` (ducky.streams.FDInputStream attribute), 112
- `_abc_registry` (ducky.streams.FDOutputStream attribute), 112
- `_abc_registry` (ducky.streams.FileInputStream attribute), 112
- `_abc_registry` (ducky.streams.FileOutputStream attribute), 112
- `_abc_registry` (ducky.streams.InputStream attribute), 113
- `_abc_registry` (ducky.streams.MethodInputStream attribute), 113
- `_abc_registry` (ducky.streams.MethodOutputStream attribute), 113
- `_abc_registry` (ducky.streams.OutputStream attribute), 113
- `_abc_registry` (ducky.streams.StderrStream attribute), 113
- `_abc_registry` (ducky.streams.StdinStream attribute), 114
- `_abc_registry` (ducky.streams.StdoutStream attribute), 114
- `_abc_registry` (ducky.streams.Stream attribute), 114
- `_abc_registry` (ducky.tools.vm.WSInputStream attribute), 118
- `_abc_registry` (ducky.tools.vm.WSOutputStream attribute), 119
- `_apply_patch()` (ducky.tools.ld.RelocationPatcher method), 117
- `_asdict()` (ducky.reactor.FDCallbacks method), 109
- `_boolean_states` (ducky.config.MachineConfig attribute), 30
- `_check_access()` (ducky.cpu.MMU method), 67
- `_close_input()` (ducky.devices.keyboard.Frontend method), 72
- `_create_header()` (ducky.mm.binary.File method), 96
- `_create_header()` (ducky.mm.binary.Section method), 98
- `_create_patch()` (ducky.tools.ld.RelocationPatcher method), 117
- `_create_payload()` (ducky.mm.binary.Section method), 98
- `_debug_wrapper_read()` (ducky.cpu.MMU method), 68
- `_debug_wrapper_write()` (ducky.cpu.MMU method), 68
- `_default_format` (ducky.log.ColorizedLogFormatter attribute), 92
- `_default_format` (ducky.log.LogFormatter attribute), 93
- `_do_tenh()` (ducky.machine.Machine method), 94
- `_encoding` (ducky.mm.binary.FileFlags attribute), 96
- `_encoding` (ducky.mm.binary.RelocFlags attribute), 97
- `_encoding` (ducky.mm.binary.SectionFlags attribute), 98
- `_encoding` (ducky.mm.binary.SymbolFlags attribute), 100
- `_encoding` (ducky.util.Flags attribute), 120
- `_enter_exception()` (ducky.cpu.CPUCore method), 63
- `_exit_exception()` (ducky.cpu.CPUCore method), 64
- `_expand_operands()` (ducky.cpu.instructions.Descriptor method), 44
- `_fetch_instr()` (ducky.cpu.MMU method), 68
- `_fetch_instr_jit()` (ducky.cpu.MMU method), 68
- `_fields` (ducky.reactor.FDCallbacks attribute), 109
- `_fields_` (ducky.cpu.instructions.EncodingA attribute), 47
- `_fields_` (ducky.cpu.instructions.EncodingC attribute), 47

fields (ducky.cpu.instructions.EncodingI attribute), 48
 fields (ducky.cpu.instructions.EncodingR attribute), 48
 fields (ducky.cpu.instructions.EncodingS attribute), 49
 fields (ducky.devices.keyboard.HDTEEntry_Keyboard attribute), 73
 fields (ducky.devices.rtc.HDTEEntry_RTC attribute), 73
 fields (ducky.devices.svga.Char attribute), 80
 fields (ducky.devices.tty.HDTEEntry_TTY attribute), 80
 fields (ducky.hdt.HDTEEntry_Argument attribute), 90
 fields (ducky.hdt.HDTEEntry_CPU attribute), 90
 fields (ducky.hdt.HDTEEntry_Memory attribute), 91
 fields (ducky.hdt.HDTHeader attribute), 91
 fields (ducky.mm.binary.FileFlagsEncoding attribute), 96
 fields (ducky.mm.binary.FileHeader attribute), 97
 fields (ducky.mm.binary.RelocEntry attribute), 97
 fields (ducky.mm.binary.RelocFlagsEncoding attribute), 97
 fields (ducky.mm.binary.SectionFlagsEncoding attribute), 98
 fields (ducky.mm.binary.SectionHeader attribute), 99
 fields (ducky.mm.binary.SymbolEntry attribute), 100
 fields (ducky.mm.binary.SymbolFlagsEncoding attribute), 100
 _flag_busy() (ducky.devices.storage.BlockIO method), 76
 _flag_error() (ducky.devices.storage.BlockIO method), 76
 _flag_finished() (ducky.devices.storage.BlockIO method), 76
 _flags (ducky.cpu.CoreFlags attribute), 66
 _flags (ducky.cpu.coprocessor.control.CoreFlags attribute), 33
 _flags (ducky.mm.PageTableEntry attribute), 106
 _flags (ducky.mm.binary.FileFlags attribute), 96
 _flags (ducky.mm.binary.RelocFlags attribute), 97
 _flags (ducky.mm.binary.SectionFlags attribute), 98
 _flags (ducky.mm.binary.SymbolFlags attribute), 100
 _flags (ducky.util.Flags attribute), 120
 _get_instruction_set() (ducky.cpu.CPUCore method), 64
 _get_mmap_fileno() (ducky.boot.ROMLoader method), 29
 _get_payload() (ducky.mm.binary.Section method), 98
 _get_pg_ops_dict() (ducky.cpu.MMU method), 68
 _get_pg_ops_list() (ducky.cpu.MMU method), 68
 _get_pt_enabled() (ducky.cpu.MMU method), 68
 _get_pte() (ducky.cpu.MMU method), 68
 _get_py2() (ducky.boot.MMapMemoryPage method), 28
 _get_py3() (ducky.boot.MMapMemoryPage method), 28
 _get_vars() (ducky.log.ColorizedLogFormatter method), 92
 _get_vars() (ducky.log.LogFormatter method), 93
 _handle_exception() (ducky.cpu.CPUCore method), 64
 _handle_input_error() (ducky.devices.keyboard.Frontend method), 72
 _handle_python_exception() (ducky.cpu.CPUCore method), 64
 _handle_raw_input() (ducky.devices.keyboard.Frontend method), 72
 _input_read_u8_echo() (ducky.devices.terminal.Terminal method), 79
 _labels (ducky.cpu.CoreFlags attribute), 66
 _labels (ducky.cpu.coprocessor.control.CoreFlags attribute), 33
 _labels (ducky.mm.PageTableEntry attribute), 106
 _labels (ducky.mm.binary.FileFlags attribute), 96
 _labels (ducky.mm.binary.RelocFlags attribute), 97
 _labels (ducky.mm.binary.SectionFlags attribute), 98
 _labels (ducky.mm.binary.SymbolFlags attribute), 100
 _labels (ducky.util.Flags attribute), 120
 _machines (ducky.tools.vm.DuckyProtocol attribute), 117
 _make() (ducky.reactor.FDCallbacks class method), 109
 _match_operand_type() (ducky.cpu.instructions.Descriptor static method), 44
 _member_map_ (ducky.cpu.coprocessor.control.ControlRegisters attribute), 32
 _member_map_ (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOps attribute), 37
 _member_map_ (ducky.cpu.instructions.DuckyOpcodes attribute), 47
 _member_map_ (ducky.cpu.registers.Registers attribute), 62
 _member_map_ (ducky.devices.keyboard.ControlMessages attribute), 72
 _member_map_ (ducky.devices.keyboard.KeyboardPorts attribute), 73
 _member_map_ (ducky.devices.rtc.RTCPorts attribute), 74
 _member_map_ (ducky.devices.storage.BlockIOPorts attribute), 77
 _member_map_ (ducky.devices.svga.SimpleVGACommands attribute), 82
 _member_map_ (ducky.devices.svga.SimpleVGAPorts attribute), 83
 _member_map_ (ducky.devices.tty.TTYPorts attribute), 80
 _member_map_ (ducky.errors.ExceptionList attribute), 86
 _member_map_ (ducky.hdt.HDTEEntryTypes attribute), 89
 _member_map_ (ducky.mm.MMOperationList attribute), 102
 _member_map_ (ducky.mm.binary.SectionTypes attribute), 99
 _member_map_ (ducky.mm.binary.SymbolDataTypes attribute), 100
 _member_names_ (ducky.cpu.coprocessor.control.ControlRegisters attribute), 33
 _member_names_ (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOps attribute), 37

attribute), 37
 _member_names_ (ducky.cpu.instructions.DuckyOpcodes attribute), 47
 _member_names_ (ducky.cpu.registers.Registers attribute), 62
 _member_names_ (ducky.devices.keyboard.ControlMessages attribute), 72
 _member_names_ (ducky.devices.keyboard.KeyboardPorts attribute), 73
 _member_names_ (ducky.devices rtc.RTCPorts attribute), 74
 _member_names_ (ducky.devices.storage.BlockIOPorts attribute), 77
 _member_names_ (ducky.devices.svga.SimpleVGACommands attribute), 82
 _member_names_ (ducky.devices.svga.SimpleVGAPorts attribute), 83
 _member_names_ (ducky.devices.tty.TTYPorts attribute), 80
 _member_names_ (ducky.errors.ExceptionList attribute), 86
 _member_names_ (ducky.hdt.HDTEntryTypes attribute), 90
 _member_names_ (ducky.mm.MMOperationList attribute), 102
 _member_names_ (ducky.mm.binary.SectionTypes attribute), 99
 _member_names_ (ducky.mm.binary.SymbolDataTypes attribute), 100
 _member_type_ (ducky.cpu.coprocessor.control.ControlRegisters attribute), 33
 _member_type_ (ducky.cpu.coprocessor.math_copro.MathCoprocessor attribute), 37
 _member_type_ (ducky.cpu.instructions.DuckyOpcodes attribute), 47
 _member_type_ (ducky.cpu.registers.Registers attribute), 62
 _member_type_ (ducky.devices.keyboard.ControlMessages attribute), 72
 _member_type_ (ducky.devices.keyboard.KeyboardPorts attribute), 73
 _member_type_ (ducky.devices rtc.RTCPorts attribute), 74
 _member_type_ (ducky.devices.storage.BlockIOPorts attribute), 77
 _member_type_ (ducky.devices.svga.SimpleVGACommands attribute), 82
 _member_type_ (ducky.devices.svga.SimpleVGAPorts attribute), 83
 _member_type_ (ducky.devices.tty.TTYPorts attribute), 80
 _member_type_ (ducky.errors.ExceptionList attribute), 86
 _member_type_ (ducky.hdt.HDTEntryTypes attribute), 90
 _member_type_ (ducky.mm.MMOperationList attribute), 102
 _member_type_ (ducky.mm.binary.SectionTypes attribute), 99
 _member_type_ (ducky.mm.binary.SymbolDataTypes attribute), 100
 _nopt_read_u16() (ducky.cpu.MMU method), 68
 _nopt_read_u32() (ducky.cpu.MMU method), 68
 _nopt_read_u8() (ducky.cpu.MMU method), 68
 _nopt_write_u16() (ducky.cpu.MMU method), 68
 _nopt_write_u32() (ducky.cpu.MMU method), 68
 _nopt_write_u8() (ducky.cpu.MMU method), 68
 _open_input() (ducky.devices.keyboard.Frontend method), 72
 pack (ducky.cpu.instructions.EncodingA attribute), 47
 pack (ducky.cpu.instructions.EncodingC attribute), 47
 pack (ducky.cpu.instructions.EncodingI attribute), 48
 pack (ducky.cpu.instructions.EncodingR attribute), 48
 pack (ducky.cpu.instructions.EncodingS attribute), 49
 pack (ducky.devices.svga.Char attribute), 80
 pack (ducky.hdt.HDTStructure attribute), 91
 pack (ducky.mm.binary.FileHeader attribute), 97
 pack (ducky.mm.binary.RelocEntry attribute), 97
 pack (ducky.mm.binary.SectionHeader attribute), 99
 pack (ducky.mm.binary.SymbolEntry attribute), 100
 _patch_echo() (ducky.devices.terminal.Terminal method), 79
 _process_input_events() (ducky.devices.keyboard.Backend method), 72
 _pt_read_u16() (ducky.cpu.MMU method), 68
 _pt_read_u32() (ducky.cpu.MMU method), 68
 _pt_read_u8() (ducky.cpu.MMU method), 68
 _pt_write_u16() (ducky.cpu.MMU method), 68
 _pt_write_u32() (ducky.cpu.MMU method), 68
 _pt_write_u8() (ducky.cpu.MMU method), 68
 _put_mmap_fileno() (ducky.boot.ROMLoader method), 29
 _put_py2() (ducky.boot.MMapMemoryPage method), 28
 _put_py3() (ducky.boot.MMapMemoryPage method), 28
 _raw_pop() (ducky.cpu.CPUCore method), 64
 _raw_push() (ducky.cpu.CPUCore method), 64
 _raw_read_fd() (ducky.streams.Stream method), 114
 _raw_read_stream() (ducky.streams.Stream method), 114
 _raw_write_fd() (ducky.streams.Stream method), 114
 _raw_write_stream() (ducky.streams.Stream method), 114
 _read() (ducky.devices.storage.FileBackedStorage method), 77
 _read() (ducky.mm.binary.File method), 96
 _read_char() (ducky.devices.keyboard.Backend method), 72
 _read_header() (ducky.mm.binary.File method), 96
 _read_header() (ducky.mm.binary.Section method), 98

- `_read_payload()` (ducky.mm.binary.Section method), 98
- `_replace()` (ducky.reactor.FDCallbacks method), 109
- `_set_access_methods()` (ducky.cpu.MMU method), 68
- `_set_instruction_set()` (ducky.cpu.CPUCore method), 64
- `_set_payload()` (ducky.mm.binary.Section method), 98
- `_set_pt_enabled()` (ducky.cpu.MMU method), 69
- `_u32_to_encoding_pypy()` (ducky.cpu.instructions.EncodingContext method), 48
- `_u32_to_encoding_python()` (ducky.cpu.instructions.EncodingContext method), 48
- `_value2member_map_` (ducky.cpu.coprocessor.control.ControlRegisters attribute), 33
- `_value2member_map_` (ducky.cpu.coprocessor.math_copro.MathCoproAttributes attribute), 37
- `_value2member_map_` (ducky.cpu.instructions.DuckyOpcodes attribute), 47
- `_value2member_map_` (ducky.cpu.registers.Registers attribute), 63
- `_value2member_map_` (ducky.devices.keyboard.ControlMessage attribute), 72
- `_value2member_map_` (ducky.devices.keyboard.KeyboardPorts attribute), 73
- `_value2member_map_` (ducky.devices.rtc.RTCPorts attribute), 74
- `_value2member_map_` (ducky.devices.storage.BlockIOPorts attribute), 77
- `_value2member_map_` (ducky.devices.svga.SimpleVGACommands attribute), 82
- `_value2member_map_` (ducky.devices.svga.SimpleVGAPorts attribute), 83
- `_value2member_map_` (ducky.devices.tty.TTYPorts attribute), 80
- `_value2member_map_` (ducky.errors.ExceptionList attribute), 86
- `_value2member_map_` (ducky.hdt.HDTEntryTypes attribute), 90
- `_value2member_map_` (ducky.mm.MMOperationList attribute), 102
- `_value2member_map_` (ducky.mm.binary.SectionTypes attribute), 99
- `_value2member_map_` (ducky.mm.binary.SymbolDataTypes attribute), 100
- `_write()` (ducky.devices.storage.FileBackedStorage method), 77
- `_write()` (ducky.mm.binary.File method), 96
- `_write_header()` (ducky.mm.binary.File method), 96
- `_write_header()` (ducky.mm.binary.Section method), 98
- `_write_payload()` (ducky.mm.binary.Section method), 98
- `act()` (ducky.debugging.LogValueAction method), 70
- `act()` (ducky.debugging.SuspendCoreAction method), 71
- Action (class in ducky.debugging), 69
- ADD (class in ducky.cpu.instructions), 40
- ADD (ducky.cpu.instructions.DuckyOpcodes attribute), 45
- `add_breakpoint()` (ducky.config.MachineConfig method), 30
- `add_call()` (ducky.reactor.Reactor method), 110
- `add_child()` (ducky.snapshot.SnapshotNode method), 111
- `add_common_options()` (in module ducky.tools), 119
- `add_device()` (ducky.config.MachineConfig method), 30
- `add_listener()` (ducky.reactor.Reactor method), 110
- `add_fd()` (ducky.reactor.Reactor method), 110
- `add_mmap()` (ducky.config.MachineConfig method), 30
- `add_point()` (ducky.debugging.DebuggingSet method), 70
- `add_storage()` (ducky.config.MachineConfig method), 30
- `add_task()` (ducky.reactor.Reactor method), 110
- ADDL (class in ducky.cpu.coprocessor.math_copro.MathCoproAttributes), 33
- ADDL (ducky.cpu.coprocessor.math_copro.MathCoproAttributes attribute), 36
- ADDR (ducky.devices.storage.BlockIOPorts attribute), 76
- `addr_to_offset()` (in module ducky.mm), 106
- `addr_to_page()` (in module ducky.mm), 106
- address (ducky.mm.binary.SymbolEntry attribute), 100
- align (in module ducky.util), 121
- `align_file()` (in module ducky.tools.img), 116
- ALLOC (ducky.mm.MMOperationList attribute), 101
- `alloc_page()` (ducky.mm.MemoryController method), 102
- `alloc_pages()` (ducky.mm.MemoryController method), 103
- `alloc_specific_page()` (ducky.mm.MemoryController method), 103
- AND (class in ducky.cpu.instructions), 41
- AND (ducky.cpu.instructions.DuckyOpcodes attribute), 45
- AnonymousMemoryPage (class in ducky.mm), 101
- `archive_files()` (in module ducky.tools.ld), 117
- `area_to_pages()` (in module ducky.mm), 107
- ARGUMENT (ducky.hdt.HDTEntryTypes attribute), 89
- ASCII (ducky.mm.binary.SymbolDataTypes attribute), 99
- `assemble_operands()` (ducky.cpu.coprocessor.math_copro.Descriptor_MAT static method), 34
- `assemble_operands()` (ducky.cpu.coprocessor.math_copro.LOAD static method), 35
- `assemble_operands()` (ducky.cpu.coprocessor.math_copro.LOADUW static method), 35
- `assemble_operands()` (ducky.cpu.coprocessor.math_copro.LOADW static method), 35
- `assemble_operands()` (ducky.cpu.coprocessor.math_copro.SAVE static method), 39

A

AccessViolationError, 84
act() (ducky.debugging.Action method), 69

- ul style="list-style-type: none; padding-left: 0;">
- assemble_operands() (ducky.cpu.coprocessor.math_copro.SAVE static method), 39
- assemble_operands() (ducky.cpu.instructions._BRANCH class method), 59
- assemble_operands() (ducky.cpu.instructions._JUMP static method), 60
- assemble_operands() (ducky.cpu.instructions._LOAD static method), 60
- assemble_operands() (ducky.cpu.instructions._SELECT class method), 60
- assemble_operands() (ducky.cpu.instructions._SET class method), 61
- assemble_operands() (ducky.cpu.instructions._STORE static method), 61
- assemble_operands() (ducky.cpu.instructions.CAS static method), 42
- assemble_operands() (ducky.cpu.instructions.Descriptor static method), 44
- assemble_operands() (ducky.cpu.instructions.Descriptor_R static method), 44
- assemble_operands() (ducky.cpu.instructions.Descriptor_R_1 static method), 45
- assemble_operands() (ducky.cpu.instructions.Descriptor_R_16 static method), 45
- assemble_operands() (ducky.cpu.instructions.Descriptor_R_32 static method), 45
- assemble_operands() (ducky.cpu.instructions.Descriptor_RI static method), 44
- AssemblerError, 84
- AssemblyIllegalCharError, 85
- AssemblyParseError, 85
- ## B
- Backend (class in ducky.devices.keyboard), 72
 - Backend (class in ducky.devices.tty), 79
 - BadLinkerScriptError, 85
 - base (ducky.mm.binary.SectionHeader attribute), 99
 - BE (class in ducky.cpu.instructions), 41
 - BG (class in ducky.cpu.instructions), 41
 - bg (ducky.devices.svga.Char attribute), 80
 - BGE (class in ducky.cpu.instructions), 41
 - binary, 26
 - BinaryFile (class in ducky.util), 119
 - BIO_BUSY (in module ducky.devices.storage), 75
 - BIO_DMA (in module ducky.devices.storage), 75
 - BIO_ERR (in module ducky.devices.storage), 75
 - BIO_RDY (in module ducky.devices.storage), 75
 - BIO_READ (in module ducky.devices.storage), 75
 - BIO_SRST (in module ducky.devices.storage), 75
 - BIO_USER (in module ducky.devices.storage), 75
 - BIO_WRITE (in module ducky.devices.storage), 75
 - BL (class in ducky.cpu.instructions), 41
 - BLE (class in ducky.cpu.instructions), 41
 - blink (ducky.devices.svga.Char attribute), 80
 - block (ducky.devices.storage.BlockIO attribute), 76
 - BLOCK_SIZE (in module ducky.devices.storage), 75
 - BlockIO (class in ducky.devices.storage), 76
 - BlockIOMemoryPage (class in ducky.devices.storage), 76
 - BlockIOPorts (class in ducky.devices.storage), 76
 - blue() (ducky.log.ColorizedLogFormatter method), 92
 - blue() (ducky.log.LogFormatter method), 93
 - BNE (class in ducky.cpu.instructions), 41
 - BNO (class in ducky.cpu.instructions), 41
 - BNS (class in ducky.cpu.instructions), 41
 - BNZ (class in ducky.cpu.instructions), 41
 - BO (class in ducky.cpu.instructions), 42
 - bool2option() (in module ducky.config), 31
 - boot() (ducky.boot.ROMLoader method), 29
 - boot() (ducky.console.ConsoleConnection method), 31
 - boot() (ducky.console.ConsoleMaster method), 31
 - boot() (ducky.cpu.CPU method), 63
 - boot() (ducky.cpu.CPUCore method), 64
 - boot() (ducky.devices.Device method), 83
 - boot() (ducky.devices.keyboard.Backend method), 72
 - boot() (ducky.devices.keyboard.Frontend method), 72
 - boot() (ducky.devices rtc.RTC method), 74
 - boot() (ducky.devices.snapshot.FileSnapshotStorage method), 75
 - boot() (ducky.devices.storage.BlockIO method), 76
 - boot() (ducky.devices.storage.FileBackedStorage method), 77
 - boot() (ducky.devices.svga.Display method), 81
 - boot() (ducky.devices.svga.SimpleVGA method), 82
 - boot() (ducky.devices.terminal.StandalonePTYTerminal method), 78
 - boot() (ducky.devices.terminal.StreamIOTerminal method), 78
 - boot() (ducky.devices.terminal.Terminal method), 79
 - boot() (ducky.devices.tty.Backend method), 79
 - boot() (ducky.devices.tty.Frontend method), 79
 - boot() (ducky.interfaces.IMachineWorker method), 92
 - boot() (ducky.machine.Machine method), 94
 - boot() (ducky.mm.MemoryController method), 103
 - bootloader, 26
 - BRANCH (ducky.cpu.instructions.DuckyOpcodes attribute), 45
 - BreakPoint (class in ducky.debugging), 69
 - BS (class in ducky.cpu.instructions), 42
 - bss (ducky.mm.binary.SectionFlagsEncoding attribute), 98
 - buff (ducky.util.StringTable attribute), 120
 - buff_to_memory() (ducky.devices.storage.BlockIO method), 76
 - buildProtocol() (ducky.tools.vm.DuckySocketServerFactory method), 118
 - BYTE (ducky.mm.binary.SymbolDataTypes attribute), 99

BZ (class in ducky.cpu.instructions), 42

C

CALL (class in ducky.cpu.instructions), 42

CALL (ducky.cpu.instructions.DuckyOpcodes attribute), 45

CallInReactorTask (class in ducky.reactor), 109

capture_state() (ducky.machine.Machine method), 94

capture_vm_state() (ducky.snapshot.VMState static method), 112

CAS (class in ducky.cpu.instructions), 42

CAS (ducky.cpu.instructions.DuckyOpcodes attribute), 45

change_runnable_state() (ducky.cpu.CPUCore method), 65

Char (class in ducky.devices.svga), 80

CHAR (ducky.mm.binary.SymbolDataTypes attribute), 99

check_protected_ins() (ducky.cpu.CPUCore method), 65

clear() (ducky.cpu.InstructionCache_Full method), 67

clear() (ducky.mm.AnonymousMemoryPage method), 101

clear() (ducky.mm.ExternalMemoryPage method), 101

clear() (ducky.mm.MemoryPage method), 105

CLI (class in ducky.cpu.instructions), 42

CLI (ducky.cpu.instructions.DuckyOpcodes attribute), 46

close() (ducky.devices.tty.Frontend method), 79

close() (ducky.streams.StdinStream method), 114

close() (ducky.streams.Stream method), 114

cmd_boot() (in module ducky.machine), 95

cmd_bp_active() (in module ducky.debugging), 71

cmd_bp_add_breakpoint() (in module ducky.debugging), 71

cmd_bp_add_memory_watchpoint() (in module ducky.debugging), 71

cmd_bp_list() (in module ducky.debugging), 72

cmd_bp_remove() (in module ducky.debugging), 72

cmd_halt() (in module ducky.machine), 95

cmd_help() (in module ducky.console), 32

cmd_run() (in module ducky.machine), 95

cmd_snapshot() (in module ducky.machine), 95

CMP (class in ducky.cpu.instructions), 43

CMP (ducky.cpu.instructions.DuckyOpcodes attribute), 46

CMPU (class in ducky.cpu.instructions), 43

CMPU (ducky.cpu.instructions.DuckyOpcodes attribute), 46

CNT (ducky.cpu.registers.Registers attribute), 61

codepoint (ducky.devices.svga.Char attribute), 81

colorize() (ducky.log.ColorizedLogFormatter method), 93

colorize() (ducky.log.LogFormatter method), 93

ColorizedLogFormatter (class in ducky.log), 92

COLS (ducky.devices.svga.SimpleVGACommands attribute), 82

CommChannel (class in ducky.machine), 93

CommQueue (class in ducky.machine), 93

ConflictingNamesError, 85

connect() (ducky.console.ConsoleMaster method), 31

console_id (ducky.console.ConsoleMaster attribute), 32

ConsoleConnection (class in ducky.console), 31

ConsoleMaster (class in ducky.console), 31

CONTROL (ducky.devices.svga.SimpleVGAPorts attribute), 83

CONTROL_MESSAGE_FIRST (ducky.devices.keyboard.ControlMessages attribute), 72

ControlCoproprocessor (class in ducky.cpu.coprocessor.control), 32

ControlMessages (class in ducky.devices.keyboard), 72

ControlRegisters (class in ducky.cpu.coprocessor.control), 32

Coprocessor (class in ducky.cpu.coprocessor), 40

CoprocessorError, 85

CoprocessorError (ducky.errors.ExceptionList attribute), 85

core() (ducky.machine.Machine method), 94

CoreDumpFile (class in ducky.snapshot), 111

CoreFlags (class in ducky.cpu), 66

CoreFlags (class in ducky.cpu.coprocessor.control), 33

cores (ducky.machine.Machine attribute), 94

COUNT (ducky.devices.storage.BlockIOPorts attribute), 76

COUNT (ducky.errors.ExceptionList attribute), 85

CPU (class in ducky.cpu), 63

CPU (ducky.hdt.HDTEntryTypes attribute), 89

CPUCore (class in ducky.cpu), 63

CPUCoreState (class in ducky.cpu), 66

CPUState (class in ducky.cpu), 66

CR0 (ducky.cpu.coprocessor.control.ControlRegisters attribute), 32

CR1 (ducky.cpu.coprocessor.control.ControlRegisters attribute), 32

CR2 (ducky.cpu.coprocessor.control.ControlRegisters attribute), 32

CR3 (ducky.cpu.coprocessor.control.ControlRegisters attribute), 32

create() (ducky.hdt.HDT method), 89

create() (ducky.hdt.HDTEntry class method), 89

create() (ducky.hdt.HDTEntry_Argument class method), 90

create() (ducky.streams.InputStream static method), 113

create() (ducky.streams.OutputStream static method), 113

create() (ducky.util.Flags class method), 120

create_binary_image() (in module ducky.tools.img), 116

create_frame() (ducky.cpu.CPUCore method), 65

[create_from_config\(\)](#) (ducky.debugging.BreakPoint static method), 69
[create_from_config\(\)](#) (ducky.debugging.LogMemoryContentAction static method), 70
[create_from_config\(\)](#) (ducky.debugging.LogRegisterContentAction static method), 70
[create_from_config\(\)](#) (ducky.debugging.MemoryWatchPoint static method), 71
[create_from_config\(\)](#) (ducky.debugging.SuspendCoreAction static method), 71
[create_from_config\(\)](#) (ducky.devices.Device static method), 83
[create_from_config\(\)](#) (ducky.devices.keyboard.Backend static method), 72
[create_from_config\(\)](#) (ducky.devices.keyboard.Frontend static method), 72
[create_from_config\(\)](#) (ducky.devices.rtc.RTC static method), 74
[create_from_config\(\)](#) (ducky.devices.snapshot.DefaultFileSnapshotStorage static method), 75
[create_from_config\(\)](#) (ducky.devices.snapshot.FileSnapshotStorage static method), 75
[create_from_config\(\)](#) (ducky.devices.snapshot.SnapshotStorage static method), 75
[create_from_config\(\)](#) (ducky.devices.storage.BlockIO static method), 76
[create_from_config\(\)](#) (ducky.devices.storage.FileBackedStorage static method), 77
[create_from_config\(\)](#) (ducky.devices.svga.Display static method), 81
[create_from_config\(\)](#) (ducky.devices.svga.SimpleVGA static method), 82
[create_from_config\(\)](#) (ducky.devices.terminal.StandalonePTYTerminal static method), 78
[create_from_config\(\)](#) (ducky.devices.terminal.StandardIOTerminal static method), 78
[create_from_config\(\)](#) (ducky.devices.terminal.StreamIOTerminal static method), 78
[create_from_config\(\)](#) (ducky.devices.tty.Backend static method), 79
[create_from_config\(\)](#) (ducky.devices.tty.Frontend static method), 79
[create_getters\(\)](#) (ducky.config.MachineConfig method), 30
[create_hdt_entries\(\)](#) (ducky.devices.Device static method), 83
[create_hdt_entries\(\)](#) (ducky.devices.keyboard.Backend static method), 72
[create_hdt_entries\(\)](#) (ducky.devices.rtc.RTC static method), 74
[create_hdt_entries\(\)](#) (ducky.devices.tty.Backend static method), 79
[create_hdt_image\(\)](#) (in module ducky.tools.img), 116
[create_logger\(\)](#) (in module ducky.log), 93
[create_queue\(\)](#) (ducky.machine.CommChannel method), 93
[create_section\(\)](#) (ducky.mm.binary.File method), 96
[create_stats\(\)](#) (ducky.profiler.DummyCPUCoreProfiler method), 107
[create_stats\(\)](#) (ducky.profiler.DummyMachineProfiler method), 108
[create_text\(\)](#) (ducky.errors.AssemblerError method), 85
[CTR](#) (class in ducky.cpu.instructions), 43
[CTR](#) (ducky.cpu.instructions.DuckyOpcodes attribute), 46
[CTW](#) (class in ducky.cpu.instructions), 43
[CTW](#) (ducky.cpu.instructions.DuckyOpcodes attribute), 46

D

[DATA](#) (ducky.devices.keyboard.KeyboardPorts attribute), 73
[DATA](#) (ducky.devices.storage.BlockIOPorts attribute), 76
[DATA](#) (ducky.devices.svga.SimpleVGAPorts attribute), 83
[DATA](#) (ducky.devices.tty.TTYPorts attribute), 80
[data_size](#) (ducky.mm.binary.SectionHeader attribute), 99
[DAY](#) (ducky.devices.rtc.RTCPorts attribute), 74
[debug\(\)](#) (in module ducky.patch), 107
[DebuggingSet](#) (class in ducky.debugging), 70
[DEC](#) (class in ducky.cpu.instructions), 43
[DEC](#) (ducky.cpu.instructions.DuckyOpcodes attribute), 46
[DECL](#) (class in ducky.cpu.coprocessor.math_copro), 33
[DECL](#) (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 36
[decoding](#) (ducky.cpu.instructions.EncodingContext method), 48
[DEFAULT_BOOT_MODE](#) (in module ducky.devices.svga), 81
[DEFAULT_BOOTLOADER_ADDRESS](#) (in module ducky.boot), 27
[DEFAULT_CORE_INST_CACHE_SIZE](#) (in module ducky.cpu), 66
[DEFAULT_EVT_ADDRESS](#) (in module ducky.cpu), 66
[DEFAULT_HDT_ADDRESS](#) (in module ducky.boot), 27
[DEFAULT_MEMORY_BANKS](#) (in module ducky.devices.svga), 81
[DEFAULT_MEMORY_SIZE](#) (in module ducky.devices.svga), 81
[DEFAULT_MMIO_ADDRESS](#) (in module ducky.devices.svga), 81
[DEFAULT_MODES](#) (in module ducky.devices.svga), 81
[DEFAULT_PT_ADDRESS](#) (in module ducky.cpu), 66
[DefaultFileSnapshotStorage](#) (class in ducky.devices.snapshot), 75
[DEPTH](#) (ducky.devices.svga.SimpleVGACommands attribute), 82

Descriptor (class in ducky.cpu.instructions), 44	static method), 60
Descriptor_MATH (class in ducky.cpu.coprocessor.math_copro), 34	disassemble_operands() (ducky.cpu.instructions._SELECT static method), 60
Descriptor_R (class in ducky.cpu.instructions), 44	disassemble_operands() (ducky.cpu.instructions._SET static method), 61
Descriptor_R_I (class in ducky.cpu.instructions), 45	disassemble_operands() (ducky.cpu.instructions._STORE static method), 61
Descriptor_R_R (class in ducky.cpu.instructions), 45	disassemble_operands() (ducky.cpu.instructions.CAS static method), 42
Descriptor_RI (class in ducky.cpu.instructions), 44	disassemble_operands() (ducky.cpu.instructions.Descriptor static method), 44
destroy_frame() (ducky.cpu.CPUCore method), 65	disassemble_operands() (ducky.cpu.instructions.Descriptor_R static method), 44
Device (class in ducky.devices), 83	disassemble_operands() (ducky.cpu.instructions.Descriptor_R_I static method), 45
DEVICE (ducky.hdt.HDTEEntryTypes attribute), 89	disassemble_operands() (ducky.cpu.instructions.Descriptor_R_R static method), 45
DeviceBackend (class in ducky.devices), 84	disassemble_operands() (ducky.cpu.instructions.Descriptor_R_RI static method), 45
DeviceFrontend (class in ducky.devices), 84	disassemble_operands() (ducky.cpu.instructions.Descriptor_RI static method), 44
die() (ducky.console.ConsoleConnection method), 31	DisassembleMismatchError, 85
die() (ducky.cpu.CPU method), 63	Display (class in ducky.devices.svga), 81
die() (ducky.cpu.CPUCore method), 65	DisplayRefreshTask (class in ducky.devices.svga), 81
die() (ducky.interfaces.IMachineWorker method), 92	DIV (class in ducky.cpu.instructions), 44
die() (ducky.machine.Machine method), 94	DIV (ducky.cpu.instructions.DuckyOpcodes attribute), 46
DIRTY (ducky.mm.PageTableEntry attribute), 106	DivideByZero (ducky.errors.ExceptionList attribute), 86
disable() (ducky.profiler.DummyCPUCoreProfiler method), 108	DivideByZeroError, 85
disable() (ducky.profiler.DummyMachineProfiler method), 108	DIVL (class in ducky.cpu.coprocessor.math_copro), 33
disassemble_instruction() (ducky.cpu.instructions.InstructionSet class method), 50	DIVL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 36
disassemble_mnemonic() (ducky.cpu.instructions._BRANCH static method), 59	do_log_cpu_core_state() (in module ducky.cpu), 69
disassemble_mnemonic() (ducky.cpu.instructions._SELECT static method), 60	do_open() (ducky.util.BinaryFile static method), 119
disassemble_mnemonic() (ducky.cpu.instructions._SET static method), 61	do_read_blocks() (ducky.devices.storage.FileBackedStorage method), 77
disassemble_mnemonic() (ducky.cpu.instructions.Descriptor class method), 44	do_read_blocks() (ducky.devices.storage.Storage method), 77
disassemble_operands() (ducky.cpu.coprocessor.math_copro static method), 34	do_read_blocks() (ducky.cpu.CPUCore method), 65
disassemble_operands() (ducky.cpu.coprocessor.math_copro static method), 35	do_write_blocks() (ducky.devices.storage.FileBackedStorage method), 77
disassemble_operands() (ducky.cpu.coprocessor.math_copro static method), 35	do_write_blocks() (ducky.devices.storage.Storage method), 77
disassemble_operands() (ducky.cpu.coprocessor.math_copro static method), 35	double_fault() (ducky.log.LogFormatter method), 93
disassemble_operands() (ducky.cpu.coprocessor.math_copro static method), 35	DOUBLEFAULT (ducky.errors.ExceptionList attribute), 86
disassemble_operands() (ducky.cpu.coprocessor.math_copro static method), 39	DROP (class in ducky.cpu.coprocessor.math_copro), 34
disassemble_operands() (ducky.cpu.coprocessor.math_copro static method), 39	DROP (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 36
disassemble_operands() (ducky.cpu.coprocessor.math_copro static method), 39	ducky (module), 27
disassemble_operands() (ducky.cpu.instructions._BRANCH static method), 59	ducky.config (module), 30
disassemble_operands() (ducky.cpu.instructions._JUMP static method), 60	ducky.console (module), 31
disassemble_operands() (ducky.cpu.instructions._LOAD static method), 60	ducky.cpu (module), 63
	ducky.cpu.coprocessor (module), 40
	ducky.cpu.coprocessor.control (module), 32
	ducky.cpu.coprocessor.math_copro (module), 33

- ducky.cpu.instructions (module), 40
 - ducky.cpu.registers (module), 61
 - ducky.debugging (module), 69
 - ducky.devices (module), 83
 - ducky.devices.keyboard (module), 72
 - ducky.devices.rtc (module), 73
 - ducky.devices.snapshot (module), 75
 - ducky.devices.storage (module), 75
 - ducky.devices.svga (module), 80
 - ducky.devices.terminal (module), 78
 - ducky.devices.tty (module), 79
 - ducky.errors (module), 84
 - ducky.hdt (module), 89
 - ducky.interfaces (module), 91
 - ducky.log (module), 92
 - ducky.machine (module), 93
 - ducky.mm (module), 101
 - ducky.mm.binary (module), 96
 - ducky.patch (module), 107
 - ducky.profiler (module), 107
 - ducky.reactor (module), 109
 - ducky.snapshot (module), 111
 - ducky.streams (module), 112
 - ducky.tools (module), 119
 - ducky.tools.as (module), 115
 - ducky.tools.coredump (module), 116
 - ducky.tools.defs (module), 116
 - ducky.tools.img (module), 116
 - ducky.tools.ld (module), 116
 - ducky.tools.objdump (module), 117
 - ducky.tools.profile (module), 117
 - ducky.tools.vm (module), 117
 - ducky.util (module), 119
 - DuckyInstructionSet (class in ducky.cpu.instructions), 45
 - DuckyOpcodes (class in ducky.cpu.instructions), 45
 - DuckyProtocol (class in ducky.tools.vm), 117
 - DuckySocketServerFactory (class in ducky.tools.vm), 118
 - DummyCPUCoreProfiler (class in ducky.profiler), 107
 - DummyMachineProfiler (class in ducky.profiler), 108
 - dump_stack() (ducky.cpu.coprocessor.math_copro.MathCoproprocessor method), 36
 - dump_stats() (ducky.profiler.DummyCPUCoreProfiler method), 108
 - dump_stats() (ducky.profiler.DummyMachineProfiler method), 108
 - dump_stats() (ducky.profiler.RealCPUCoreProfiler method), 109
 - dumps() (ducky.config.MachineConfig method), 30
 - DUP (class in ducky.cpu.coprocessor.math_copro), 34
 - DUP (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 36
 - DUP2 (class in ducky.cpu.coprocessor.math_copro), 34
 - DUP2 (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 36
- ## E
- emit_instruction() (ducky.cpu.instructions.Descriptor static method), 44
 - EmptyMathStackError, 34
 - enable() (ducky.profiler.DummyCPUCoreProfiler method), 108
 - enable() (ducky.profiler.DummyMachineProfiler method), 108
 - enable_cpu() (ducky.profiler.ProfilerStore method), 108
 - enable_machine() (ducky.profiler.ProfilerStore method), 108
 - encode() (ducky.cpu.instructions.EncodingContext method), 48
 - encode_blob() (in module ducky.tools.as), 115
 - encode_string() (in module ducky.hdt), 91
 - Encoding (class in ducky.cpu.instructions), 47
 - encoding (ducky.cpu.instructions._BINOP attribute), 59
 - encoding (ducky.cpu.instructions._BITOP attribute), 59
 - encoding (ducky.cpu.instructions._BRANCH attribute), 59
 - encoding (ducky.cpu.instructions._CMP attribute), 59
 - encoding (ducky.cpu.instructions._JUMP attribute), 60
 - encoding (ducky.cpu.instructions._LOAD attribute), 60
 - encoding (ducky.cpu.instructions._SELECT attribute), 60
 - encoding (ducky.cpu.instructions._SET attribute), 61
 - encoding (ducky.cpu.instructions._STORE attribute), 61
 - encoding (ducky.cpu.instructions.CAS attribute), 42
 - encoding (ducky.cpu.instructions.CLI attribute), 42
 - encoding (ducky.cpu.instructions.CTR attribute), 43
 - encoding (ducky.cpu.instructions.CTW attribute), 43
 - encoding (ducky.cpu.instructions.Descriptor attribute), 44
 - encoding (ducky.cpu.instructions.Descriptor_R attribute), 44
 - encoding (ducky.cpu.instructions.Descriptor_R_I attribute), 45
 - encoding (ducky.cpu.instructions.Descriptor_R_R attribute), 45
 - encoding (ducky.cpu.instructions.Descriptor_R_RI attribute), 45
 - encoding (ducky.cpu.instructions.Descriptor_RI attribute), 45
 - encoding (ducky.cpu.instructions.FPTC attribute), 49
 - encoding (ducky.cpu.instructions.IDLE attribute), 50
 - encoding (ducky.cpu.instructions.LPM attribute), 52
 - encoding (ducky.cpu.instructions.MOV attribute), 52
 - encoding (ducky.cpu.instructions.NOP attribute), 53
 - encoding (ducky.cpu.instructions.NOT attribute), 53
 - encoding (ducky.cpu.instructions.RET attribute), 53
 - encoding (ducky.cpu.instructions.RETINT attribute), 54
 - encoding (ducky.cpu.instructions.RST attribute), 54
 - encoding (ducky.cpu.instructions.STI attribute), 57

- ul style="list-style-type: none; padding-left: 0;">
- encoding (ducky.cpu.instructions.SWP attribute), 58
- encoding() (ducky.util.Flags class method), 120
- encoding_to_u32() (in module ducky.cpu.instructions), 61
- EncodingA (class in ducky.cpu.instructions), 47
- EncodingC (class in ducky.cpu.instructions), 47
- EncodingContext (class in ducky.cpu.instructions), 48
- EncodingI (class in ducky.cpu.instructions), 48
- EncodingLargeValueError, 85
- EncodingR (class in ducky.cpu.instructions), 48
- EncodingS (class in ducky.cpu.instructions), 49
- enqueue() (ducky.tools.vm.WSInputStream method), 118
- enqueue_input_stream() (ducky.devices.terminal.StreamIOTerminal method), 78
- enqueue_stream() (ducky.devices.keyboard.Frontend method), 72
- enqueue_streams() (ducky.devices.terminal.StreamIOTerminal method), 79
- entries (ducky.hdt.HDTHeader attribute), 91
- ENTRY_HEADER (ducky.hdt.HDTEntry attribute), 89
- ENTRY_HEADER (ducky.hdt.HDTEntry_Device attribute), 90
- Error, 85
- evaluate() (ducky.cpu.instructions._CMP static method), 59
- evaluate() (ducky.cpu.instructions._COND static method), 60
- EventBus (class in ducky.machine), 93
- EVT, 26
- EXCEPTION_INDEX (ducky.errors.CoprocessorError attribute), 85
- EXCEPTION_INDEX (ducky.errors.DivideByZeroError attribute), 85
- EXCEPTION_INDEX (ducky.errors.InvalidExceptionError attribute), 87
- EXCEPTION_INDEX (ducky.errors.InvalidInstructionSetError attribute), 87
- EXCEPTION_INDEX (ducky.errors.InvalidOpcodeError attribute), 87
- EXCEPTION_INDEX (ducky.errors.MemoryAccessError attribute), 88
- EXCEPTION_INDEX (ducky.errors.PrivilegedInstructionError attribute), 88
- EXCEPTION_INDEX (ducky.errors.RegisterAccessError attribute), 88
- EXCEPTION_INDEX (ducky.errors.UnalignedAccessError attribute), 88
- ExceptionList (class in ducky.errors), 85
- exec_f() (in module ducky.patch), 107
- executable (ducky.mm.binary.SectionFlagsEncoding attribute), 98
- EXECUTE (ducky.mm.PageTableEntry attribute), 106
- execute() (ducky.console.ConsoleConnection method), 31
- execute() (ducky.cpu.coprocessor.math_copro.ADDL static method), 33
- execute() (ducky.cpu.coprocessor.math_copro.DECL static method), 33
- execute() (ducky.cpu.coprocessor.math_copro.DIVL static method), 33
- execute() (ducky.cpu.coprocessor.math_copro.DROP static method), 34
- execute() (ducky.cpu.coprocessor.math_copro.DUP static method), 34
- execute() (ducky.cpu.coprocessor.math_copro.DUP2 static method), 34
- execute() (ducky.cpu.coprocessor.math_copro.INCL static method), 34
- execute() (ducky.cpu.coprocessor.math_copro.LOAD static method), 35
- execute() (ducky.cpu.coprocessor.math_copro.LOADUW static method), 35
- execute() (ducky.cpu.coprocessor.math_copro.LOADW static method), 35
- execute() (ducky.cpu.coprocessor.math_copro.MODL static method), 35
- execute() (ducky.cpu.coprocessor.math_copro.MULL static method), 35
- execute() (ducky.cpu.coprocessor.math_copro.POPL static method), 37
- execute() (ducky.cpu.coprocessor.math_copro.POPUW static method), 37
- execute() (ducky.cpu.coprocessor.math_copro.POPW static method), 38
- execute() (ducky.cpu.coprocessor.math_copro.PUSHL static method), 38
- execute() (ducky.cpu.coprocessor.math_copro.PUSHW static method), 38
- execute() (ducky.cpu.coprocessor.math_copro.SAVE static method), 39
- execute() (ducky.cpu.coprocessor.math_copro.SAVEW static method), 39
- execute() (ducky.cpu.coprocessor.math_copro.SWP static method), 39
- execute() (ducky.cpu.coprocessor.math_copro.SYMDIVL static method), 39
- execute() (ducky.cpu.coprocessor.math_copro.SYMMODL static method), 40
- execute() (ducky.cpu.coprocessor.math_copro.UDIVL static method), 40
- execute() (ducky.cpu.coprocessor.math_copro.UMODL static method), 40
- execute() (ducky.cpu.instructions._BINOP static method), 59
- execute() (ducky.cpu.instructions._BITOP static method), 59
- execute() (ducky.cpu.instructions._BRANCH static method), 59

execute() (ducky.cpu.instructions._LOAD static method), 60

execute() (ducky.cpu.instructions._LOAD_IMM class method), 60

execute() (ducky.cpu.instructions._SELECT static method), 60

execute() (ducky.cpu.instructions._SET static method), 61

execute() (ducky.cpu.instructions._STORE static method), 61

execute() (ducky.cpu.instructions.CALL static method), 42

execute() (ducky.cpu.instructions.CAS static method), 42

execute() (ducky.cpu.instructions.CLI static method), 42

execute() (ducky.cpu.instructions.CMP static method), 43

execute() (ducky.cpu.instructions.CMPU static method), 43

execute() (ducky.cpu.instructions.CTR static method), 43

execute() (ducky.cpu.instructions.CTW static method), 43

execute() (ducky.cpu.instructions.DEC static method), 43

execute() (ducky.cpu.instructions.Descriptor static method), 44

execute() (ducky.cpu.instructions.FPTC static method), 49

execute() (ducky.cpu.instructions.HLT static method), 49

execute() (ducky.cpu.instructions.IDLE static method), 50

execute() (ducky.cpu.instructions.INC static method), 50

execute() (ducky.cpu.instructions.INT static method), 50

execute() (ducky.cpu.instructions.IPI static method), 50

execute() (ducky.cpu.instructions.J static method), 51

execute() (ducky.cpu.instructions.LPM static method), 52

execute() (ducky.cpu.instructions.MOV static method), 52

execute() (ducky.cpu.instructions.NOP static method), 53

execute() (ducky.cpu.instructions.NOT static method), 53

execute() (ducky.cpu.instructions.POP static method), 53

execute() (ducky.cpu.instructions.PUSH static method), 53

execute() (ducky.cpu.instructions.RET static method), 54

execute() (ducky.cpu.instructions.RETINT static method), 54

execute() (ducky.cpu.instructions.RST static method), 54

execute() (ducky.cpu.instructions.SIS static method), 57

execute() (ducky.cpu.instructions.STI static method), 58

execute() (ducky.cpu.instructions.SWP static method), 58

ExecutionException, 86

ExecutionException__SimpleESR (class in ducky.errors), 86

exit_code (ducky.machine.Machine attribute), 94

extend_with_push() (ducky.cpu.coprocessor.math_copro.MathCopro-
cessor method), 36

ExternalMemoryPage (class in ducky.mm), 101

F

fd_blocking() (in module ducky.streams), 115

FDCallbacks (class in ducky.reactor), 109

FDInputStream (class in ducky.streams), 112

FDOutputStream (class in ducky.streams), 112

fg (ducky.devices.svga.Char attribute), 81

field (ducky.mm.binary.FileFlags attribute), 96

field (ducky.mm.binary.RelocFlags attribute), 97

field (ducky.mm.binary.SectionFlags attribute), 98

field (ducky.mm.binary.SymbolFlags attribute), 100

File (class in ducky.mm.binary), 96

file_size (ducky.mm.binary.SectionHeader attribute), 99

FileBackedStorage (class in ducky.devices.storage), 77

FileFlags (class in ducky.mm.binary), 96

FileFlagsEncoding (class in ducky.mm.binary), 96

FileHeader (class in ducky.mm.binary), 96

FileInputStream (class in ducky.streams), 112

filename (ducky.mm.binary.SymbolEntry attribute), 100

FileOutputStream (class in ducky.streams), 112

FileSnapshotStorage (class in ducky.devices.snapshot), 75

fill_reloc_slot() (ducky.cpu.instructions._BRANCH static method), 59

fill_reloc_slot() (ducky.cpu.instructions.Descriptor static method), 44

fill_reloc_slot() (ducky.cpu.instructions.EncodingC static method), 47

fill_reloc_slot() (ducky.cpu.instructions.EncodingI static method), 48

fill_reloc_slot() (ducky.cpu.instructions.EncodingR static method), 48

fill_reloc_slot() (ducky.cpu.instructions.EncodingS static method), 49

find_module() (ducky.patch.Importer method), 107

FIRST_HW (ducky.errors.ExceptionList attribute), 86

FIRST_SW (ducky.errors.ExceptionList attribute), 86

fix_offsets() (ducky.mm.binary.File method), 96

fix_section_bases() (in module ducky.tools.ld), 117

flag (ducky.cpu.instructions.EncodingC attribute), 47

flag (ducky.cpu.instructions.EncodingS attribute), 49

Flags (class in ducky.util), 120

flags (ducky.cpu.CPUCore attribute), 65

FLAGS (ducky.cpu.instructions._COND attribute), 60

flags (ducky.devices.keyboard.HDTEEntry_Keyboard attribute), 73

flags (ducky.devices rtc.HDTEEntry_RTC attribute), 73

flags (ducky.devices.tty.HDTEEntry_TTY attribute), 80

flags (ducky.mm.binary.FileHeader attribute), 97

flags (ducky.mm.binary.RelocEntry attribute), 97

flags (ducky.mm.binary.SectionHeader attribute), 99

flags (ducky.mm.binary.SymbolEntry attribute), 100

flush() (ducky.devices.tty.Frontend method), 79

flush() (ducky.streams.OutputStream method), 113

format() (ducky.log.LogFormatter method), 93

format_field() (ducky.util.Formatter method), 120
 format_int() (ducky.util.Formatter method), 120
 Formatter (class in ducky.util), 120
 FP (ducky.cpu.registers.Registers attribute), 61
 FP() (ducky.cpu.CPUCore method), 63
 FPTC (class in ducky.cpu.instructions), 49
 FPTC (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 FREE (ducky.mm.MMOperationList attribute), 101
 free_page() (ducky.mm.MemoryController method), 103
 free_pages() (ducky.mm.MemoryController method), 103
 frequency (ducky.devices.rtc.RTC attribute), 74
 FREQUENCY (ducky.devices.rtc.RTCPorts attribute), 74
 from_encoding() (ducky.util.Flags class method), 120
 from_int() (ducky.util.Flags class method), 120
 from_string() (ducky.devices.svga.Mode class method), 81
 from_string() (ducky.util.Flags class method), 120
 from_u16() (ducky.devices.svga.Char static method), 81
 from_u8() (ducky.devices.svga.Char static method), 81
 Frontend (class in ducky.devices.keyboard), 72
 Frontend (class in ducky.devices.tty), 79
 FrontendFlushTask (class in ducky.devices.tty), 79
 FullMathStackError, 34
 FUNCTION (ducky.mm.binary.SymbolDataTypes attribute), 99

G

get() (ducky.boot.MMapMemoryPage method), 28
 get() (ducky.config.MachineConfig method), 30
 get() (ducky.devices.svga.SimpleVGAMemoryPage method), 83
 get() (ducky.mm.ExternalMemoryPage method), 101
 get_assembler_process() (in module ducky.tools.as), 115
 get_child() (ducky.snapshot.SnapshotNode method), 112
 get_children() (ducky.snapshot.SnapshotNode method), 112
 get_code() (ducky.patch.ModuleLoader method), 107
 get_core_profiler() (ducky.profiler.ProfilerStore method), 108
 get_core_state_by_id() (ducky.cpu.CPUState method), 66
 get_core_states() (ducky.cpu.CPUState method), 66
 get_cpu_state_by_id() (ducky.machine.MachineState method), 95
 get_cpu_states() (ducky.machine.MachineState method), 95
 get_device_by_name() (ducky.machine.Machine method), 94
 get_driver() (in module ducky.devices), 84
 get_instruction_set() (in module ducky.cpu.instructions), 61
 get_machine_profiler() (ducky.profiler.ProfilerStore method), 108
 get_master() (ducky.devices.Device method), 84

get_message() (ducky.debugging.LogMemoryContentAction method), 70
 get_message() (ducky.debugging.LogRegisterContentAction method), 70
 get_message() (ducky.debugging.LogValueAction method), 70
 get_page() (ducky.mm.MemoryController method), 103
 get_page_states() (ducky.mm.MemoryState method), 106
 get_pages() (ducky.mm.MemoryController method), 103
 get_queue() (ducky.machine.CommChannel method), 93
 get_section_by_index() (ducky.mm.binary.File method), 96
 get_section_by_name() (ducky.mm.binary.File method), 96
 get_section_by_type() (ducky.mm.binary.File method), 96
 get_selectee() (ducky.streams.StdinStream method), 114
 get_slave_devices() (in module ducky.devices.terminal), 79
 get_slave_gpu() (ducky.devices.svga.Display static method), 81
 get_source() (ducky.patch.ModuleLoader method), 107
 get_storage_by_id() (ducky.machine.Machine method), 95
 get_string() (ducky.util.StringTable method), 120
 get_strings_section() (ducky.mm.binary.File method), 96
 get_symbol() (ducky.util.SymbolTable method), 121
 get_values() (ducky.debugging.LogMemoryContentAction method), 70
 get_values() (ducky.debugging.LogRegisterContentAction method), 70
 get_values() (ducky.debugging.LogValueAction method), 71
 getbool() (ducky.config.MachineConfig method), 31
 getfloat() (ducky.config.MachineConfig method), 31
 getint() (ducky.config.MachineConfig method), 31
 GFLAGS (ducky.cpu.instructions._COND attribute), 60
 globally_visible (ducky.mm.binary.SectionFlagsEncoding attribute), 98
 globally_visible (ducky.mm.binary.SymbolFlagsEncoding attribute), 100
 GRAPHIC (ducky.devices.svga.SimpleVGACommands attribute), 82
 green() (ducky.log.ColorizedLogFormatter method), 93
 green() (ducky.log.LogFormatter method), 93

H

HALT (ducky.devices.keyboard.ControlMessages attribute), 72
 halt() (ducky.boot.ROMLoader method), 29
 halt() (ducky.console.ConsoleConnection method), 31
 halt() (ducky.console.ConsoleMaster method), 32
 halt() (ducky.console.TerminalConsoleConnection method), 32

[halt\(\)](#) (ducky.cpu.CPU method), 63
[halt\(\)](#) (ducky.cpu.CPUCore method), 65
[halt\(\)](#) (ducky.cpu.MMU method), 69
[halt\(\)](#) (ducky.devices.Device method), 84
[halt\(\)](#) (ducky.devices.keyboard.Backend method), 72
[halt\(\)](#) (ducky.devices.keyboard.Frontend method), 72
[halt\(\)](#) (ducky.devices.rtc.RTC method), 74
[halt\(\)](#) (ducky.devices.snapshot.FileSnapshotStorage method), 75
[halt\(\)](#) (ducky.devices.snapshot.SnapshotStorage method), 75
[halt\(\)](#) (ducky.devices.storage.BlockIO method), 76
[halt\(\)](#) (ducky.devices.storage.FileBackedStorage method), 77
[halt\(\)](#) (ducky.devices.svga.Display method), 81
[halt\(\)](#) (ducky.devices.svga.SimpleVGA method), 82
[halt\(\)](#) (ducky.devices.terminal.StandalonePTYTerminal method), 78
[halt\(\)](#) (ducky.devices.terminal.StreamIOTerminal method), 79
[halt\(\)](#) (ducky.devices.terminal.Terminal method), 79
[halt\(\)](#) (ducky.devices.tty.Backend method), 79
[halt\(\)](#) (ducky.devices.tty.Frontend method), 79
[halt\(\)](#) (ducky.interfaces.IMachineWorker method), 92
[halt\(\)](#) (ducky.machine.Machine method), 95
[halt\(\)](#) (ducky.mm.MemoryController method), 104
[HaltMachineTask](#) (class in ducky.machine), 94
[has_coprocessor\(\)](#) (ducky.cpu.CPUCore method), 65
[has_fd\(\)](#) (ducky.streams.Stream method), 115
[has_poll_support\(\)](#) (ducky.streams.Stream method), 115
[has_poll_support\(\)](#) (ducky.tools.vm.WSInputStream method), 118
[HDT](#), 26
[HDT](#) (class in ducky.hdt), 89
[HDT_MAGIC](#) (in module ducky.hdt), 91
[HDTEntry](#) (class in ducky.hdt), 89
[HDTEntry_Argument](#) (class in ducky.hdt), 90
[HDTEntry_CPU](#) (class in ducky.hdt), 90
[HDTEntry_Device](#) (class in ducky.hdt), 90
[HDTEntry_Keyboard](#) (class in ducky.devices.keyboard), 73
[HDTEntry_Memory](#) (class in ducky.hdt), 91
[HDTEntry_RTC](#) (class in ducky.devices.rtc), 73
[HDTEntry_TTY](#) (class in ducky.devices.tty), 80
[HDTEntryTypes](#) (class in ducky.hdt), 89
[HDTHeader](#) (class in ducky.hdt), 91
[HDTStructure](#) (class in ducky.hdt), 91
[header](#) (ducky.mm.binary.File attribute), 96
[header](#) (ducky.mm.binary.Section attribute), 98
[HLT](#) (class in ducky.cpu.instructions), 49
[HLT](#) (ducky.cpu.instructions.DuckyOpcodes attribute), 46
[HOUR](#) (ducky.devices.rtc.RTCPorts attribute), 74
[hw_setup\(\)](#) (ducky.machine.Machine method), 95

I

[ident](#) (ducky.devices.keyboard.HDTEntry_Keyboard attribute), 73
[ident](#) (ducky.devices.rtc.HDTEntry_RTC attribute), 73
[ident](#) (ducky.devices.tty.HDTEntry_TTY attribute), 80
[IDLE](#) (class in ducky.cpu.instructions), 50
[IDLE](#) (ducky.cpu.instructions.DuckyOpcodes attribute), 46
[IE_FLAG\(\)](#) (in module ducky.cpu.instructions), 50
[IE_IMM\(\)](#) (in module ducky.cpu.instructions), 50
[IE_OPCODE\(\)](#) (in module ducky.cpu.instructions), 50
[IE_REG\(\)](#) (in module ducky.cpu.instructions), 50
[IMachineWorker](#) (class in ducky.interfaces), 91
[immediate](#) (ducky.cpu.instructions.EncodingC attribute), 47
[immediate](#) (ducky.cpu.instructions.EncodingI attribute), 48
[immediate](#) (ducky.cpu.instructions.EncodingR attribute), 48
[immediate](#) (ducky.cpu.instructions.EncodingS attribute), 49
[immediate_flag](#) (ducky.cpu.instructions.EncodingC attribute), 48
[immediate_flag](#) (ducky.cpu.instructions.EncodingI attribute), 48
[immediate_flag](#) (ducky.cpu.instructions.EncodingR attribute), 48
[immediate_flag](#) (ducky.cpu.instructions.EncodingS attribute), 49
[Importer](#) (class in ducky.patch), 107
[INC](#) (class in ducky.cpu.instructions), 50
[INC](#) (ducky.cpu.instructions.DuckyOpcodes attribute), 46
[INCL](#) (class in ducky.cpu.coprocessor.math_copro), 34
[INCL](#) (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 36
[IncompatibleSectionFlagsError](#), 87
[IncompleteDirectiveError](#), 87
[index](#) (ducky.mm.binary.SectionHeader attribute), 99
[init\(\)](#) (ducky.cpu.instructions.InstructionSet class method), 50
[init_debug_set\(\)](#) (ducky.cpu.CPUCore method), 66
[InputStream](#) (class in ducky.streams), 113
[inst_aligned](#) (ducky.cpu.instructions._BRANCH attribute), 59
[inst_aligned](#) (ducky.cpu.instructions._JUMP attribute), 60
[inst_aligned](#) (ducky.cpu.instructions.Descriptor attribute), 44
[inst_aligned](#) (ducky.mm.binary.RelocFlagsEncoding attribute), 97
[instruction_set](#) (ducky.cpu.CPUCore attribute), 66
[instruction_set_id](#) (ducky.cpu.coprocessor.math_copro.MathCoproprocessorInstructions attribute), 36
[instruction_set_id](#) (ducky.cpu.instructions.DuckyInstructionSet attribute), 45

- ul style="list-style-type: none; padding-left: 0;">
- instruction_set_id (ducky.cpu.instructions.InstructionSet attribute), 50
- InstructionCache_Base (class in ducky.cpu), 66
- InstructionCache_Full (class in ducky.cpu), 67
- instructions (ducky.cpu.coprocessor.math_copro.MathCopro attribute), 36
- instructions (ducky.cpu.instructions.DuckyInstructionSet attribute), 45
- instructions (ducky.cpu.instructions.InstructionSet attribute), 50
- InstructionSet (class in ducky.cpu.instructions), 50
- InstructionSetMetaclass (class in ducky.cpu.instructions), 51
- INT (class in ducky.cpu.instructions), 50
- INT (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- INT (ducky.mm.binary.SymbolDataTypes attribute), 99
- InterruptVector (class in ducky.cpu), 67
- InvalidException (ducky.errors.ExceptionList attribute), 86
- InvalidExceptionError, 87
- InvalidFrameError, 87
- InvalidInstructionSetError, 87
- InvalidInstSet (ducky.errors.ExceptionList attribute), 86
- InvalidOpcode (ducky.errors.ExceptionList attribute), 86
- InvalidOpcodeError, 87
- InvalidResourceError, 87
- IP (ducky.cpu.registers.Registers attribute), 61
- IP() (ducky.cpu.CPUCore method), 63
- IPI (class in ducky.cpu.instructions), 50
- IPI (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- IReactorTask (class in ducky.interfaces), 92
- irq() (ducky.cpu.CPUCore method), 66
- IRQRouterTask (class in ducky.machine), 94
- is_cpu_enabled() (ducky.profiler.ProfilerStore method), 108
- is_empty_in() (ducky.machine.CommQueue method), 93
- is_empty_out() (ducky.machine.CommQueue method), 93
- is_machine_enabled() (ducky.profiler.ProfilerStore method), 109
- is_registered_command() (ducky.console.ConsoleMaster method), 32
- is_slave() (ducky.devices.Device method), 84
- is_triggered() (ducky.debugging.BreakPoint method), 69
- is_triggered() (ducky.debugging.MemoryWatchPoint method), 71
- is_triggered() (ducky.debugging.Point method), 71
- isfile() (in module ducky.util), 121
- ISnapshotable (class in ducky.interfaces), 92
- iter_breakpoints() (ducky.config.MachineConfig method), 31
- iter_devices() (ducky.config.MachineConfig method), 31
- iter_mmaps() (ducky.config.MachineConfig method), 31
- iter_storages() (ducky.config.MachineConfig method), 31
- IVirtualInterrupt (class in ducky.interfaces), 92
- ## J
- J (class in ducky.cpu.instructions), 51
 - jit() (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 - jit() (ducky.cpu.coprocessor.math_copro.LOADUW static method), 35
 - jit() (ducky.cpu.coprocessor.math_copro.LOADW static method), 35
 - jit() (ducky.cpu.coprocessor.math_copro.MULL static method), 36
 - jit() (ducky.cpu.coprocessor.math_copro.POPW static method), 38
 - jit() (ducky.cpu.coprocessor.math_copro.SAVE static method), 39
 - jit() (ducky.cpu.coprocessor.math_copro.SAVEW static method), 39
 - jit() (ducky.cpu.instructions._BRANCH static method), 59
 - jit() (ducky.cpu.instructions._LOAD static method), 60
 - jit() (ducky.cpu.instructions._SELECT static method), 61
 - jit() (ducky.cpu.instructions._STORE static method), 61
 - jit() (ducky.cpu.instructions.ADD static method), 40
 - jit() (ducky.cpu.instructions.AND static method), 41
 - jit() (ducky.cpu.instructions.CALL static method), 42
 - jit() (ducky.cpu.instructions.CLI static method), 42
 - jit() (ducky.cpu.instructions.CMP static method), 43
 - jit() (ducky.cpu.instructions.CMPU static method), 43
 - jit() (ducky.cpu.instructions.CTR static method), 43
 - jit() (ducky.cpu.instructions.CTW static method), 43
 - jit() (ducky.cpu.instructions.DEC static method), 43
 - jit() (ducky.cpu.instructions.Descriptor static method), 44
 - jit() (ducky.cpu.instructions.DIV static method), 44
 - jit() (ducky.cpu.instructions.FPTC static method), 49
 - jit() (ducky.cpu.instructions.INC static method), 50
 - jit() (ducky.cpu.instructions.J static method), 51
 - jit() (ducky.cpu.instructions.LA static method), 51
 - jit() (ducky.cpu.instructions.LI static method), 51
 - jit() (ducky.cpu.instructions.LIU static method), 51
 - jit() (ducky.cpu.instructions.MOD static method), 52
 - jit() (ducky.cpu.instructions.MOV static method), 52
 - jit() (ducky.cpu.instructions.MUL static method), 52
 - jit() (ducky.cpu.instructions.OR static method), 53
 - jit() (ducky.cpu.instructions.POP static method), 53
 - jit() (ducky.cpu.instructions.PUSH static method), 53
 - jit() (ducky.cpu.instructions.RET static method), 54
 - jit() (ducky.cpu.instructions.RETINT static method), 54
 - jit() (ducky.cpu.instructions.SHL static method), 57
 - jit() (ducky.cpu.instructions.SHR static method), 57
 - jit() (ducky.cpu.instructions.SHRS static method), 57
 - jit() (ducky.cpu.instructions.SIS static method), 57
 - jit() (ducky.cpu.instructions.STI static method), 58
 - jit() (ducky.cpu.instructions.SUB static method), 58
 - jit() (ducky.cpu.instructions.SWP static method), 58

jit() (ducky.cpu.instructions.UDIV static method), 58
 jit() (ducky.cpu.instructions.XOR static method), 59
 JUMP() (in module ducky.cpu.instructions), 51

K

KeyboardMMIOMemoryPage (class in ducky.devices.keyboard), 73
 KeyboardPorts (class in ducky.devices.keyboard), 73
 classes (ducky.hdt.HDT attribute), 89

L

LA (class in ducky.cpu.instructions), 51
 LA (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 LAST (ducky.devices.keyboard.KeyboardPorts attribute), 73
 LAST_HW (ducky.errors.ExceptionList attribute), 86
 LAST_SW (ducky.errors.ExceptionList attribute), 86
 LB (class in ducky.cpu.instructions), 51
 LB (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 length (ducky.devices.keyboard.HDTEEntry_Keyboard attribute), 73
 length (ducky.devices rtc.HDTEEntry_RTC attribute), 73
 length (ducky.devices.tty.HDTEEntry_TTY attribute), 80
 length (ducky.hdt.HDTEEntry_Argument attribute), 90
 length (ducky.hdt.HDTEEntry_CPU attribute), 90
 length (ducky.hdt.HDTEEntry_Memory attribute), 91
 length (ducky.hdt.HDTHeader attribute), 91
 LI (class in ducky.cpu.instructions), 51
 LI (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 lineno (ducky.mm.binary.SymbolEntry attribute), 100
 link_files() (in module ducky.tools.ld), 117
 linker, 26
 LinkerError, 87
 LinkerInfo (class in ducky.tools.ld), 116
 LinkerScript (class in ducky.tools.ld), 116
 LIU (class in ducky.cpu.instructions), 51
 LIU (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 LOAD (class in ducky.cpu.coprocessor.math_copro), 34
 LOAD (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 36
 load() (ducky.cpu.instructions._LOAD_IMM class method), 60
 load() (ducky.cpu.instructions.LA class method), 51
 load() (ducky.cpu.instructions.LI class method), 51
 load() (ducky.cpu.instructions.LIU class method), 51
 load() (ducky.cpu.InterruptVector static method), 67
 load() (ducky.snapshot.CoreDumpFile method), 111
 load_encoding() (ducky.util.Flags method), 120
 load_int() (ducky.util.Flags method), 120
 load_module() (ducky.patch.ModuleLoader method), 107
 load_state() (ducky.boot.MMapArea method), 27
 load_state() (ducky.cpu.coprocessor.math_copro.MathCoproprocessor method), 36

load_state() (ducky.cpu.coprocessor.math_copro.RegisterSet method), 38
 load_state() (ducky.cpu.CPU method), 63
 load_state() (ducky.cpu.CPUCore method), 66
 load_state() (ducky.interfaces.ISnapshotable method), 92
 load_state() (ducky.machine.Machine method), 95
 load_state() (ducky.mm.MemoryController method), 104
 load_state() (ducky.mm.MemoryPage method), 105
 load_state() (ducky.mm.MemoryRegion method), 106
 load_string() (ducky.util.Flags method), 120
 load_vm_state() (ducky.snapshot.VMState static method), 112
 loadable (ducky.mm.binary.SectionFlagsEncoding attribute), 98
 loader_for_path() (ducky.patch.Importer method), 107
 LOADUW (class in ducky.cpu.coprocessor.math_copro), 35
 LOADUW (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 36
 LOADW (class in ducky.cpu.coprocessor.math_copro), 35
 LOADW (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 36
 log() (ducky.console.ConsoleConnection method), 31
 log() (ducky.errors.AssemblerError method), 85
 log() (ducky.errors.Error method), 85
 log_cpu_core_state() (in module ducky.cpu), 69
 LogFormatter (class in ducky.log), 93
 LoggingCapable (class in ducky.util), 120
 LogMemoryContentAction (class in ducky.debugging), 70
 LogRegisterContentAction (class in ducky.debugging), 70
 LogValueAction (class in ducky.debugging), 70
 LPM (class in ducky.cpu.instructions), 51
 LPM (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 LS (class in ducky.cpu.instructions), 52
 LS (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 LW (class in ducky.cpu.instructions), 52
 LW (ducky.cpu.instructions.DuckyOpcodes attribute), 46

M

machine, 26
 Machine (class in ducky.machine), 94
 MachineConfig (class in ducky.config), 30
 MachineState (class in ducky.machine), 95
 magic (ducky.hdt.HDTHeader attribute), 91
 MAGIC (ducky.mm.binary.File attribute), 96
 magic (ducky.mm.binary.FileHeader attribute), 97
 main() (in module ducky.tools.as), 116
 main() (in module ducky.tools.coredump), 116
 main() (in module ducky.tools.defs), 116
 main() (in module ducky.tools.img), 116

- main() (in module ducky.tools.ld), 117
- main() (in module ducky.tools.objdump), 117
- main() (in module ducky.tools.profile), 117
- main() (in module ducky.tools.vm), 119
- MalformedBinaryError, 102
- MathCoproprocessor (class in ducky.cpu.coproprocessor.math_copro), 36
- MathCoproprocessorInstructionSet (class in ducky.cpu.coproprocessor.math_copro), 36
- MathCoproprocessorOpcodes (class in ducky.cpu.coproprocessor.math_copro), 36
- MathCoproprocessorState (class in ducky.cpu.coproprocessor.math_copro), 37
- MAX_IDENT_LENGTH (ducky.hdt.HDTEntry_Device attribute), 90
- MAX_NAME_LENGTH (ducky.hdt.HDTEntry_Argument attribute), 90
- MAX_NAME_LENGTH (ducky.hdt.HDTEntry_Device attribute), 90
- MEMORY (ducky.hdt.HDTEntryTypes attribute), 89
- MEMORY_BANK_ID (ducky.devices.svga.SimpleVGACommand attribute), 82
- memory_to_buff() (ducky.devices.storage.BlockIO method), 76
- MemoryAccess (ducky.errors.ExceptionList attribute), 86
- MemoryAccessError, 88
- MemoryController (class in ducky.mm), 102
- MemoryPage (class in ducky.mm), 104
- MemoryPageState (class in ducky.mm), 106
- MemoryRegion (class in ducky.mm), 106
- MemoryRegionState (class in ducky.mm), 106
- MemoryState (class in ducky.mm), 106
- MemoryWatchPoint (class in ducky.debugging), 71
- merge() (ducky.profiler.ProfileRecord method), 108
- merge_object_into() (in module ducky.tools.ld), 117
- MethodInputStream (class in ducky.streams), 113
- MethodOutputStream (class in ducky.streams), 113
- MINUTE (ducky.devices.rtc.RTCPorts attribute), 74
- MMAP (ducky.mm.MMOperationList attribute), 102
- mmap_area() (ducky.boot.ROMLoader method), 29
- MMapArea (class in ducky.boot), 27
- MMapAreaState (class in ducky.boot), 27
- MMapMemoryPage (class in ducky.boot), 28
- mmio_address (ducky.devices.keyboard.HDTEntry_Keyboard attribute), 73
- mmio_address (ducky.devices.rtc.HDTEntry_RTC attribute), 74
- mmio_address (ducky.devices.tty.HDTEntry_TTY attribute), 80
- MMIOMemoryPage (class in ducky.devices), 84
- MMOperationList (class in ducky.mm), 101
- MMU (class in ducky.cpu), 67
- mnemonic (ducky.cpu.coproprocessor.math_copro.ADDL attribute), 33
- mnemonic (ducky.cpu.coproprocessor.math_copro.DECL attribute), 33
- mnemonic (ducky.cpu.coproprocessor.math_copro.DIVL attribute), 33
- mnemonic (ducky.cpu.coproprocessor.math_copro.DROP attribute), 34
- mnemonic (ducky.cpu.coproprocessor.math_copro.DUP attribute), 34
- mnemonic (ducky.cpu.coproprocessor.math_copro.DUP2 attribute), 34
- mnemonic (ducky.cpu.coproprocessor.math_copro.INCL attribute), 34
- mnemonic (ducky.cpu.coproprocessor.math_copro.LOAD attribute), 35
- mnemonic (ducky.cpu.coproprocessor.math_copro.LOADUW attribute), 35
- mnemonic (ducky.cpu.coproprocessor.math_copro.LOADW attribute), 35
- mnemonic (ducky.cpu.coproprocessor.math_copro.MODL attribute), 35
- mnemonic (ducky.cpu.coproprocessor.math_copro.MULL attribute), 36
- mnemonic (ducky.cpu.coproprocessor.math_copro.POPL attribute), 37
- mnemonic (ducky.cpu.coproprocessor.math_copro.POPUW attribute), 37
- mnemonic (ducky.cpu.coproprocessor.math_copro.POPW attribute), 38
- mnemonic (ducky.cpu.coproprocessor.math_copro.PUSHL attribute), 38
- mnemonic (ducky.cpu.coproprocessor.math_copro.PUSHW attribute), 38
- mnemonic (ducky.cpu.coproprocessor.math_copro.SAVE attribute), 39
- mnemonic (ducky.cpu.coproprocessor.math_copro.SAVEW attribute), 39
- mnemonic (ducky.cpu.coproprocessor.math_copro.SWP attribute), 39
- mnemonic (ducky.cpu.coproprocessor.math_copro.SYMDIVL attribute), 39
- mnemonic (ducky.cpu.coproprocessor.math_copro.SYMMODL attribute), 40
- mnemonic (ducky.cpu.coproprocessor.math_copro.UDIVL attribute), 40
- mnemonic (ducky.cpu.coproprocessor.math_copro.UMODL attribute), 40
- mnemonic (ducky.cpu.instructions.ADD attribute), 40
- mnemonic (ducky.cpu.instructions.AND attribute), 41
- mnemonic (ducky.cpu.instructions.BE attribute), 41
- mnemonic (ducky.cpu.instructions.BG attribute), 41
- mnemonic (ducky.cpu.instructions.BGE attribute), 41
- mnemonic (ducky.cpu.instructions.BL attribute), 41
- mnemonic (ducky.cpu.instructions.BLE attribute), 41
- mnemonic (ducky.cpu.instructions.BNE attribute), 41

mnemonic (ducky.cpu.instructions.BNO attribute), 41
 mnemonic (ducky.cpu.instructions.BNS attribute), 41
 mnemonic (ducky.cpu.instructions.BNZ attribute), 42
 mnemonic (ducky.cpu.instructions.BO attribute), 42
 mnemonic (ducky.cpu.instructions.BS attribute), 42
 mnemonic (ducky.cpu.instructions.BZ attribute), 42
 mnemonic (ducky.cpu.instructions.CALL attribute), 42
 mnemonic (ducky.cpu.instructions.CAS attribute), 42
 mnemonic (ducky.cpu.instructions.CLI attribute), 42
 mnemonic (ducky.cpu.instructions.CMP attribute), 43
 mnemonic (ducky.cpu.instructions.CMPU attribute), 43
 mnemonic (ducky.cpu.instructions.CTR attribute), 43
 mnemonic (ducky.cpu.instructions.CTW attribute), 43
 mnemonic (ducky.cpu.instructions.DEC attribute), 44
 mnemonic (ducky.cpu.instructions.Descriptor attribute), 44
 mnemonic (ducky.cpu.instructions.DIV attribute), 44
 mnemonic (ducky.cpu.instructions.FPTC attribute), 49
 mnemonic (ducky.cpu.instructions.HLT attribute), 49
 mnemonic (ducky.cpu.instructions.IDLE attribute), 50
 mnemonic (ducky.cpu.instructions.INC attribute), 50
 mnemonic (ducky.cpu.instructions.INT attribute), 50
 mnemonic (ducky.cpu.instructions.IPI attribute), 50
 mnemonic (ducky.cpu.instructions.J attribute), 51
 mnemonic (ducky.cpu.instructions.LA attribute), 51
 mnemonic (ducky.cpu.instructions.LB attribute), 51
 mnemonic (ducky.cpu.instructions.LI attribute), 51
 mnemonic (ducky.cpu.instructions.LIU attribute), 51
 mnemonic (ducky.cpu.instructions.LPM attribute), 52
 mnemonic (ducky.cpu.instructions.LS attribute), 52
 mnemonic (ducky.cpu.instructions.LW attribute), 52
 mnemonic (ducky.cpu.instructions.MOD attribute), 52
 mnemonic (ducky.cpu.instructions.MOV attribute), 52
 mnemonic (ducky.cpu.instructions.MUL attribute), 52
 mnemonic (ducky.cpu.instructions.NOP attribute), 53
 mnemonic (ducky.cpu.instructions.NOT attribute), 53
 mnemonic (ducky.cpu.instructions.OR attribute), 53
 mnemonic (ducky.cpu.instructions.POP attribute), 53
 mnemonic (ducky.cpu.instructions.PUSH attribute), 53
 mnemonic (ducky.cpu.instructions.RET attribute), 54
 mnemonic (ducky.cpu.instructions.RETINT attribute), 54
 mnemonic (ducky.cpu.instructions.RST attribute), 54
 mnemonic (ducky.cpu.instructions.SELE attribute), 54
 mnemonic (ducky.cpu.instructions.SELG attribute), 54
 mnemonic (ducky.cpu.instructions.SELGE attribute), 54
 mnemonic (ducky.cpu.instructions.SELL attribute), 55
 mnemonic (ducky.cpu.instructions.SELLE attribute), 55
 mnemonic (ducky.cpu.instructions.SELNE attribute), 55
 mnemonic (ducky.cpu.instructions.SELNO attribute), 55
 mnemonic (ducky.cpu.instructions.SELNS attribute), 55
 mnemonic (ducky.cpu.instructions.SELNZ attribute), 55
 mnemonic (ducky.cpu.instructions.SELO attribute), 55
 mnemonic (ducky.cpu.instructions.SELS attribute), 55
 mnemonic (ducky.cpu.instructions.SELZ attribute), 55

mnemonic (ducky.cpu.instructions.SETE attribute), 55
 mnemonic (ducky.cpu.instructions.SETG attribute), 56
 mnemonic (ducky.cpu.instructions.SETGE attribute), 56
 mnemonic (ducky.cpu.instructions.SETL attribute), 56
 mnemonic (ducky.cpu.instructions.SETLE attribute), 56
 mnemonic (ducky.cpu.instructions.SETNE attribute), 56
 mnemonic (ducky.cpu.instructions.SETNO attribute), 56
 mnemonic (ducky.cpu.instructions.SETNS attribute), 56
 mnemonic (ducky.cpu.instructions.SETNZ attribute), 56
 mnemonic (ducky.cpu.instructions.SETO attribute), 56
 mnemonic (ducky.cpu.instructions.SETS attribute), 56
 mnemonic (ducky.cpu.instructions.SETZ attribute), 57
 mnemonic (ducky.cpu.instructions.SHL attribute), 57
 mnemonic (ducky.cpu.instructions.SHR attribute), 57
 mnemonic (ducky.cpu.instructions.SHRS attribute), 57
 mnemonic (ducky.cpu.instructions.SIS attribute), 57
 mnemonic (ducky.cpu.instructions.STB attribute), 57
 mnemonic (ducky.cpu.instructions.STI attribute), 58
 mnemonic (ducky.cpu.instructions.STS attribute), 58
 mnemonic (ducky.cpu.instructions.STW attribute), 58
 mnemonic (ducky.cpu.instructions.SUB attribute), 58
 mnemonic (ducky.cpu.instructions.SWP attribute), 58
 mnemonic (ducky.cpu.instructions.UDIV attribute), 58
 mnemonic (ducky.cpu.instructions.XOR attribute), 59
 MNEMONICS (ducky.cpu.instructions._COND attribute), 60
 MOD (class in ducky.cpu.instructions), 52
 MOD (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 Mode (class in ducky.devices.svga), 81
 MODL (class in ducky.cpu.coprocessor.math_copro), 35
 MODL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 36
 ModuleLoader (class in ducky.patch), 107
 MONTH (ducky.devices.rtc.RTCPorts attribute), 74
 MOV (class in ducky.cpu.instructions), 52
 MOV (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 MUL (class in ducky.cpu.instructions), 52
 MUL (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 MULL (class in ducky.cpu.coprocessor.math_copro), 35
 MULL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 36

N

name (ducky.devices.keyboard.HDTEEntry_Keyboard attribute), 73
 name (ducky.devices.rtc.HDTEEntry_RTC attribute), 74
 name (ducky.devices.tty.HDTEEntry_TTY attribute), 80
 name (ducky.hdt.HDTEEntry_Argument attribute), 90
 name (ducky.mm.binary.RelocEntry attribute), 97
 name (ducky.mm.binary.Section attribute), 98
 name (ducky.mm.binary.SectionHeader attribute), 99

- name (ducky.mm.binary.SymbolEntry attribute), 100
- name_length (ducky.devices.keyboard.HDTEEntry_Keyboard attribute), 73
- name_length (ducky.devices rtc.HDTEEntry_RTC attribute), 74
- name_length (ducky.devices.tty.HDTEEntry_TTY attribute), 80
- name_length (ducky.hdt.HDTEEntry_Argument attribute), 90
- NOP (class in ducky.cpu.instructions), 52
- NOP (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- NOT (class in ducky.cpu.instructions), 53
- NOT (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- nr_cores (ducky.hdt.HDTEEntry_CPU attribute), 90
- nr_cpus (ducky.hdt.HDTEEntry_CPU attribute), 90
- ## O
- object file, 26
- offset (ducky.mm.binary.SectionHeader attribute), 99
- OFFSET_FMT() (in module ducky.mm), 106
- on_core_alive() (ducky.cpu.CPU method), 63
- on_core_alive() (ducky.machine.Machine method), 95
- on_core_halted() (ducky.cpu.CPU method), 63
- on_core_halted() (ducky.machine.Machine method), 95
- on_core_running() (ducky.cpu.CPU method), 63
- on_core_suspended() (ducky.cpu.CPU method), 63
- on_error (ducky.reactor.FDCallbacks attribute), 110
- on_read (ducky.reactor.FDCallbacks attribute), 110
- on_tick() (ducky.devices rtc.RTCTask method), 74
- on_tick() (ducky.devices.svga.DisplayRefreshTask method), 81
- on_write (ducky.reactor.FDCallbacks attribute), 110
- onClose() (ducky.tools.vm.DuckyProtocol method), 117
- onMessage() (ducky.tools.vm.DuckyProtocol method), 117
- onOpen() (ducky.tools.vm.DuckyProtocol method), 118
- opcode (ducky.cpu.coprocessor.math_copro.ADDL attribute), 33
- opcode (ducky.cpu.coprocessor.math_copro.DECL attribute), 33
- opcode (ducky.cpu.coprocessor.math_copro.DIVL attribute), 33
- opcode (ducky.cpu.coprocessor.math_copro.DROP attribute), 34
- opcode (ducky.cpu.coprocessor.math_copro.DUP attribute), 34
- opcode (ducky.cpu.coprocessor.math_copro.DUP2 attribute), 34
- opcode (ducky.cpu.coprocessor.math_copro.INCL attribute), 34
- opcode (ducky.cpu.coprocessor.math_copro.LOAD attribute), 35
- opcode (ducky.cpu.coprocessor.math_copro.LOADUW attribute), 35
- opcode (ducky.cpu.coprocessor.math_copro.LOADW attribute), 35
- opcode (ducky.cpu.coprocessor.math_copro.MODL attribute), 35
- opcode (ducky.cpu.coprocessor.math_copro.MULL attribute), 36
- opcode (ducky.cpu.coprocessor.math_copro.POPL attribute), 37
- opcode (ducky.cpu.coprocessor.math_copro.POPUW attribute), 37
- opcode (ducky.cpu.coprocessor.math_copro.POPW attribute), 38
- opcode (ducky.cpu.coprocessor.math_copro.PUSHL attribute), 38
- opcode (ducky.cpu.coprocessor.math_copro.PUSHW attribute), 38
- opcode (ducky.cpu.coprocessor.math_copro.SAVE attribute), 39
- opcode (ducky.cpu.coprocessor.math_copro.SAVEW attribute), 39
- opcode (ducky.cpu.coprocessor.math_copro.SWP attribute), 39
- opcode (ducky.cpu.coprocessor.math_copro.SYMDIVL attribute), 39
- opcode (ducky.cpu.coprocessor.math_copro.SYMMODL attribute), 40
- opcode (ducky.cpu.coprocessor.math_copro.UDIVL attribute), 40
- opcode (ducky.cpu.coprocessor.math_copro.UMODL attribute), 40
- opcode (ducky.cpu.instructions._BRANCH attribute), 59
- opcode (ducky.cpu.instructions._SELECT attribute), 61
- opcode (ducky.cpu.instructions._SET attribute), 61
- opcode (ducky.cpu.instructions.ADD attribute), 40
- opcode (ducky.cpu.instructions.AND attribute), 41
- opcode (ducky.cpu.instructions.CALL attribute), 42
- opcode (ducky.cpu.instructions.CAS attribute), 42
- opcode (ducky.cpu.instructions.CLI attribute), 43
- opcode (ducky.cpu.instructions.CMP attribute), 43
- opcode (ducky.cpu.instructions.CMPU attribute), 43
- opcode (ducky.cpu.instructions.CTR attribute), 43
- opcode (ducky.cpu.instructions.CTW attribute), 43
- opcode (ducky.cpu.instructions.DEC attribute), 44
- opcode (ducky.cpu.instructions.Descriptor attribute), 44
- opcode (ducky.cpu.instructions.DIV attribute), 44
- opcode (ducky.cpu.instructions.EncodingA attribute), 47
- opcode (ducky.cpu.instructions.EncodingC attribute), 48
- opcode (ducky.cpu.instructions.EncodingI attribute), 48
- opcode (ducky.cpu.instructions.EncodingR attribute), 48
- opcode (ducky.cpu.instructions.EncodingS attribute), 49
- opcode (ducky.cpu.instructions.FPTC attribute), 49
- opcode (ducky.cpu.instructions.HLT attribute), 49

- opcode (ducky.cpu.instructions.IDLE attribute), 50
- opcode (ducky.cpu.instructions.INC attribute), 50
- opcode (ducky.cpu.instructions.INT attribute), 50
- opcode (ducky.cpu.instructions.IPI attribute), 50
- opcode (ducky.cpu.instructions.J attribute), 51
- opcode (ducky.cpu.instructions.LA attribute), 51
- opcode (ducky.cpu.instructions.LB attribute), 51
- opcode (ducky.cpu.instructions.LI attribute), 51
- opcode (ducky.cpu.instructions.LIU attribute), 51
- opcode (ducky.cpu.instructions.LPM attribute), 52
- opcode (ducky.cpu.instructions.LS attribute), 52
- opcode (ducky.cpu.instructions.LW attribute), 52
- opcode (ducky.cpu.instructions.MOD attribute), 52
- opcode (ducky.cpu.instructions.MOV attribute), 52
- opcode (ducky.cpu.instructions.MUL attribute), 52
- opcode (ducky.cpu.instructions.NOP attribute), 53
- opcode (ducky.cpu.instructions.NOT attribute), 53
- opcode (ducky.cpu.instructions.OR attribute), 53
- opcode (ducky.cpu.instructions.POP attribute), 53
- opcode (ducky.cpu.instructions.PUSH attribute), 53
- opcode (ducky.cpu.instructions.RET attribute), 54
- opcode (ducky.cpu.instructions.RETINT attribute), 54
- opcode (ducky.cpu.instructions.RST attribute), 54
- opcode (ducky.cpu.instructions.SHL attribute), 57
- opcode (ducky.cpu.instructions.SHR attribute), 57
- opcode (ducky.cpu.instructions.SHRS attribute), 57
- opcode (ducky.cpu.instructions.SIS attribute), 57
- opcode (ducky.cpu.instructions.STB attribute), 57
- opcode (ducky.cpu.instructions.STI attribute), 58
- opcode (ducky.cpu.instructions.STS attribute), 58
- opcode (ducky.cpu.instructions.STW attribute), 58
- opcode (ducky.cpu.instructions.SUB attribute), 58
- opcode (ducky.cpu.instructions.SWP attribute), 58
- opcode (ducky.cpu.instructions.UDIV attribute), 58
- opcode (ducky.cpu.instructions.XOR attribute), 59
- opcode_desc_map (ducky.cpu.coprocessor.math_copro.MathCoproInstructionSet attribute), 36
- opcode_desc_map (ducky.cpu.instructions.DuckyInstructionSet attribute), 45
- opcode_encoding_map (ducky.cpu.coprocessor.math_copro.MathCoproInstructionSet attribute), 36
- opcode_encoding_map (ducky.cpu.instructions.DuckyInstructionSet attribute), 45
- opcodes (ducky.cpu.coprocessor.math_copro.MathCoproInstructionSet attribute), 36
- opcodes (ducky.cpu.instructions.DuckyInstructionSet attribute), 45
- opcodes (ducky.cpu.instructions.InstructionSet attribute), 50
- open() (ducky.mm.binary.File static method), 96
- open() (ducky.snapshot.CoreDumpFile static method), 111
- open() (ducky.util.BinaryFile static method), 119
- OperandMismatchError, 88
- operands (ducky.cpu.coprocessor.math_copro.ADDL attribute), 33
- operands (ducky.cpu.coprocessor.math_copro.DECL attribute), 33
- operands (ducky.cpu.coprocessor.math_copro.Descriptor_MATH attribute), 34
- operands (ducky.cpu.coprocessor.math_copro.DIVL attribute), 33
- operands (ducky.cpu.coprocessor.math_copro.DROP attribute), 34
- operands (ducky.cpu.coprocessor.math_copro.DUP attribute), 34
- operands (ducky.cpu.coprocessor.math_copro.DUP2 attribute), 34
- operands (ducky.cpu.coprocessor.math_copro.INCL attribute), 34
- operands (ducky.cpu.coprocessor.math_copro.LOAD attribute), 35
- operands (ducky.cpu.coprocessor.math_copro.LOADUW attribute), 35
- operands (ducky.cpu.coprocessor.math_copro.LOADW attribute), 35
- operands (ducky.cpu.coprocessor.math_copro.MODL attribute), 35
- operands (ducky.cpu.coprocessor.math_copro.MULL attribute), 36
- operands (ducky.cpu.coprocessor.math_copro.POPL attribute), 37
- operands (ducky.cpu.coprocessor.math_copro.POPUW attribute), 37
- operands (ducky.cpu.coprocessor.math_copro.POPW attribute), 38
- operands (ducky.cpu.coprocessor.math_copro.PUSHL attribute), 38
- operands (ducky.cpu.coprocessor.math_copro.PUSHW attribute), 38
- operands (ducky.cpu.coprocessor.math_copro.SAVE attribute), 39
- operands (ducky.cpu.coprocessor.math_copro.SAVEW attribute), 39
- operands (ducky.cpu.coprocessor.math_copro.SWP attribute), 39
- operands (ducky.cpu.coprocessor.math_copro.SYMDIVL attribute), 40
- operands (ducky.cpu.coprocessor.math_copro.SYMMODL attribute), 40
- operands (ducky.cpu.coprocessor.math_copro.UDIVL attribute), 40
- operands (ducky.cpu.coprocessor.math_copro.UMODL attribute), 40
- operands (ducky.cpu.instructions._BRANCH attribute), 59
- operands (ducky.cpu.instructions._JUMP attribute), 60
- operands (ducky.cpu.instructions._LOAD attribute), 60

- operands (ducky.cpu.instructions._SELECT attribute), 61
 - operands (ducky.cpu.instructions._SET attribute), 61
 - operands (ducky.cpu.instructions._STORE attribute), 61
 - operands (ducky.cpu.instructions.ADD attribute), 40
 - operands (ducky.cpu.instructions.AND attribute), 41
 - operands (ducky.cpu.instructions.BE attribute), 41
 - operands (ducky.cpu.instructions.BG attribute), 41
 - operands (ducky.cpu.instructions.BGE attribute), 41
 - operands (ducky.cpu.instructions.BL attribute), 41
 - operands (ducky.cpu.instructions.BLE attribute), 41
 - operands (ducky.cpu.instructions.BNE attribute), 41
 - operands (ducky.cpu.instructions.BNO attribute), 41
 - operands (ducky.cpu.instructions.BNS attribute), 41
 - operands (ducky.cpu.instructions.BNZ attribute), 42
 - operands (ducky.cpu.instructions.BO attribute), 42
 - operands (ducky.cpu.instructions.BS attribute), 42
 - operands (ducky.cpu.instructions.BZ attribute), 42
 - operands (ducky.cpu.instructions.CALL attribute), 42
 - operands (ducky.cpu.instructions.CAS attribute), 42
 - operands (ducky.cpu.instructions.CLI attribute), 43
 - operands (ducky.cpu.instructions.CMP attribute), 43
 - operands (ducky.cpu.instructions.CMPU attribute), 43
 - operands (ducky.cpu.instructions.CTR attribute), 43
 - operands (ducky.cpu.instructions.CTW attribute), 43
 - operands (ducky.cpu.instructions.DEC attribute), 44
 - operands (ducky.cpu.instructions.Descriptor attribute), 44
 - operands (ducky.cpu.instructions.Descriptor_R attribute), 44
 - operands (ducky.cpu.instructions.Descriptor_R_I attribute), 45
 - operands (ducky.cpu.instructions.Descriptor_R_R attribute), 45
 - operands (ducky.cpu.instructions.Descriptor_R_RI attribute), 45
 - operands (ducky.cpu.instructions.Descriptor_RI attribute), 45
 - operands (ducky.cpu.instructions.DIV attribute), 44
 - operands (ducky.cpu.instructions.FPTC attribute), 49
 - operands (ducky.cpu.instructions.HLT attribute), 49
 - operands (ducky.cpu.instructions.IDLE attribute), 50
 - operands (ducky.cpu.instructions.INC attribute), 50
 - operands (ducky.cpu.instructions.INT attribute), 50
 - operands (ducky.cpu.instructions.IPI attribute), 50
 - operands (ducky.cpu.instructions.J attribute), 51
 - operands (ducky.cpu.instructions.LA attribute), 51
 - operands (ducky.cpu.instructions.LB attribute), 51
 - operands (ducky.cpu.instructions.LI attribute), 51
 - operands (ducky.cpu.instructions.LIU attribute), 51
 - operands (ducky.cpu.instructions.LPM attribute), 52
 - operands (ducky.cpu.instructions.LS attribute), 52
 - operands (ducky.cpu.instructions.LW attribute), 52
 - operands (ducky.cpu.instructions.MOD attribute), 52
 - operands (ducky.cpu.instructions.MOV attribute), 52
 - operands (ducky.cpu.instructions.MUL attribute), 52
 - operands (ducky.cpu.instructions.NOP attribute), 53
 - operands (ducky.cpu.instructions.NOT attribute), 53
 - operands (ducky.cpu.instructions.OR attribute), 53
 - operands (ducky.cpu.instructions.POP attribute), 53
 - operands (ducky.cpu.instructions.PUSH attribute), 53
 - operands (ducky.cpu.instructions.RET attribute), 54
 - operands (ducky.cpu.instructions.RETINT attribute), 54
 - operands (ducky.cpu.instructions.RST attribute), 54
 - operands (ducky.cpu.instructions.SELE attribute), 54
 - operands (ducky.cpu.instructions.SELG attribute), 54
 - operands (ducky.cpu.instructions.SELGE attribute), 54
 - operands (ducky.cpu.instructions.SELL attribute), 55
 - operands (ducky.cpu.instructions.SELLE attribute), 55
 - operands (ducky.cpu.instructions.SELNE attribute), 55
 - operands (ducky.cpu.instructions.SELNO attribute), 55
 - operands (ducky.cpu.instructions.SELNS attribute), 55
 - operands (ducky.cpu.instructions.SELNZ attribute), 55
 - operands (ducky.cpu.instructions.SELO attribute), 55
 - operands (ducky.cpu.instructions.SELS attribute), 55
 - operands (ducky.cpu.instructions.SELZ attribute), 55
 - operands (ducky.cpu.instructions.SETE attribute), 55
 - operands (ducky.cpu.instructions.SETG attribute), 56
 - operands (ducky.cpu.instructions.SETGE attribute), 56
 - operands (ducky.cpu.instructions.SETL attribute), 56
 - operands (ducky.cpu.instructions.SETLE attribute), 56
 - operands (ducky.cpu.instructions.SETNE attribute), 56
 - operands (ducky.cpu.instructions.SETNO attribute), 56
 - operands (ducky.cpu.instructions.SETNS attribute), 56
 - operands (ducky.cpu.instructions.SETNZ attribute), 56
 - operands (ducky.cpu.instructions.SETO attribute), 56
 - operands (ducky.cpu.instructions.SETS attribute), 57
 - operands (ducky.cpu.instructions.SETZ attribute), 57
 - operands (ducky.cpu.instructions.SHL attribute), 57
 - operands (ducky.cpu.instructions.SHR attribute), 57
 - operands (ducky.cpu.instructions.SHRS attribute), 57
 - operands (ducky.cpu.instructions.SIS attribute), 57
 - operands (ducky.cpu.instructions.STB attribute), 57
 - operands (ducky.cpu.instructions.STI attribute), 58
 - operands (ducky.cpu.instructions.STS attribute), 58
 - operands (ducky.cpu.instructions.STW attribute), 58
 - operands (ducky.cpu.instructions.SUB attribute), 58
 - operands (ducky.cpu.instructions.SWP attribute), 58
 - operands (ducky.cpu.instructions.UDIV attribute), 58
 - operands (ducky.cpu.instructions.XOR attribute), 59
 - OR (class in ducky.cpu.instructions), 53
 - OR (ducky.cpu.instructions.DuckyOpcodes attribute), 46
 - OutputStream (class in ducky.streams), 113
- ## P
- padding (ducky.mm.binary.SectionHeader attribute), 99
 - PAGE_SIZE (in module ducky.mm), 106
 - pages_in_area() (ducky.mm.MemoryController method), 104
 - PageTableEntry (class in ducky.mm), 106

- ul style="list-style-type: none; padding-left: 0;">
- parse_io_streams() (in module ducky.devices.terminal), 79
- parse_options() (in module ducky.tools), 119
- parse_template() (in module ducky.tools.defs), 116
- patch() (ducky.tools.Id.RelocationPatcher method), 117
- patch_add (ducky.mm.binary.RelocEntry attribute), 97
- patch_address (ducky.mm.binary.RelocEntry attribute), 97
- patch_offset (ducky.mm.binary.RelocEntry attribute), 97
- patch_section (ducky.mm.binary.RelocEntry attribute), 97
- patch_size (ducky.mm.binary.RelocEntry attribute), 97
- PatchTooLargeError, 88
- payload (ducky.mm.binary.Section attribute), 98
- Point (class in ducky.debugging), 71
- poke() (ducky.boot.ROMLoader method), 29
- POP (class in ducky.cpu.instructions), 53
- POP (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- pop() (ducky.cpu.coprocessor.math_copro.RegisterSet method), 38
- pop() (ducky.cpu.CPUCore method), 66
- pop_flags() (ducky.cpu.CPUCore method), 66
- pop_frame() (ducky.cpu.CPUCore method), 66
- POPL (class in ducky.cpu.coprocessor.math_copro), 37
- POPL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
- POPUW (class in ducky.cpu.coprocessor.math_copro), 37
- POPUW (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
- POPW (class in ducky.cpu.coprocessor.math_copro), 37
- POPW (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
- post_memory() (ducky.debugging.DebuggingSet method), 70
- post_step() (ducky.debugging.DebuggingSet method), 70
- pre_memory() (ducky.debugging.DebuggingSet method), 70
- pre_step() (ducky.debugging.DebuggingSet method), 70
- prepare_write() (ducky.mm.binary.Section method), 98
- print_machine_stats() (in module ducky.tools.vm), 119
- print_node() (ducky.snapshot.SnapshotNode method), 112
- PrivilegedInstr (ducky.errors.ExceptionList attribute), 86
- PrivilegedInstructionError, 88
- process_config_options() (in module ducky.tools.vm), 119
- ProfileRecord (class in ducky.profiler), 108
- ProfilerStore (class in ducky.profiler), 108
- PROGBITS (ducky.mm.binary.SectionTypes attribute), 99
- prompt() (ducky.console.ConsoleConnection method), 31
- pt_enabled (ducky.cpu.MMU attribute), 69
- PUSH (class in ducky.cpu.instructions), 53
- PUSH (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- push() (ducky.cpu.coprocessor.math_copro.RegisterSet method), 38
- push() (ducky.cpu.CPUCore method), 66
- push_flags() (ducky.cpu.CPUCore method), 66
- PUSHL (class in ducky.cpu.coprocessor.math_copro), 38
- PUSHL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
- PUSHW (class in ducky.cpu.coprocessor.math_copro), 38
- PUSHW (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
- put() (ducky.boot.MMapMemoryPage method), 28
- put() (ducky.devices.svga.SimpleVGAMemoryPage method), 83
- put() (ducky.mm.ExternalMemoryPage method), 101
- put_string() (ducky.util.StringTable method), 120
- ## R
- R00 (ducky.cpu.registers.Registers attribute), 61
 - R01 (ducky.cpu.registers.Registers attribute), 62
 - R02 (ducky.cpu.registers.Registers attribute), 62
 - R03 (ducky.cpu.registers.Registers attribute), 62
 - R04 (ducky.cpu.registers.Registers attribute), 62
 - R05 (ducky.cpu.registers.Registers attribute), 62
 - R06 (ducky.cpu.registers.Registers attribute), 62
 - R07 (ducky.cpu.registers.Registers attribute), 62
 - R08 (ducky.cpu.registers.Registers attribute), 62
 - R09 (ducky.cpu.registers.Registers attribute), 62
 - R10 (ducky.cpu.registers.Registers attribute), 62
 - R11 (ducky.cpu.registers.Registers attribute), 62
 - R12 (ducky.cpu.registers.Registers attribute), 62
 - R13 (ducky.cpu.registers.Registers attribute), 62
 - R14 (ducky.cpu.registers.Registers attribute), 62
 - R15 (ducky.cpu.registers.Registers attribute), 62
 - R16 (ducky.cpu.registers.Registers attribute), 62
 - R17 (ducky.cpu.registers.Registers attribute), 62
 - R18 (ducky.cpu.registers.Registers attribute), 62
 - R19 (ducky.cpu.registers.Registers attribute), 62
 - R20 (ducky.cpu.registers.Registers attribute), 62
 - R21 (ducky.cpu.registers.Registers attribute), 62
 - R22 (ducky.cpu.registers.Registers attribute), 62
 - R23 (ducky.cpu.registers.Registers attribute), 62
 - R24 (ducky.cpu.registers.Registers attribute), 62
 - R25 (ducky.cpu.registers.Registers attribute), 62
 - R26 (ducky.cpu.registers.Registers attribute), 62
 - R27 (ducky.cpu.registers.Registers attribute), 62
 - R28 (ducky.cpu.registers.Registers attribute), 62
 - R29 (ducky.cpu.registers.Registers attribute), 62
 - Reactor (class in ducky.reactor), 110
 - READ (ducky.mm.PageTableEntry attribute), 106
 - read() (ducky.config.MachineConfig method), 31
 - read() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 32

[read\(\)](#) (ducky.streams.InputStream method), 113
[read\(\)](#) (ducky.streams.OutputStream method), 113
[read\(\)](#) (ducky.streams.Stream method), 115
[read\(\)](#) (ducky.tools.vm.WSInputStream method), 118
[read_blocks\(\)](#) (ducky.devices.storage.Storage method), 77
[read_cr0\(\)](#) (ducky.cpu.coprocessor.control.ControlCoproces method), 32
[read_cr1\(\)](#) (ducky.cpu.coprocessor.control.ControlCoproces method), 32
[read_cr2\(\)](#) (ducky.cpu.coprocessor.control.ControlCoproces method), 32
[read_cr3\(\)](#) (ducky.cpu.coprocessor.control.ControlCoproces method), 32
[read_data\(\)](#) (ducky.devices.storage.BlockIO method), 76
[read_in\(\)](#) (ducky.machine.CommQueue method), 93
[read_input\(\)](#) (ducky.console.ConsoleConnection method), 31
[read_out\(\)](#) (ducky.machine.CommQueue method), 93
[read_profiling_data\(\)](#) (in module ducky.tools.profile), 117
[read_struct\(\)](#) (ducky.util.BinaryFile method), 119
[read_u16\(\)](#) (ducky.devices.svga.SimpleVGAMMIOMemoryPage method), 82
[read_u16\(\)](#) (ducky.mm.AnonymousMemoryPage method), 101
[read_u16\(\)](#) (ducky.mm.ExternalMemoryPage method), 101
[read_u16\(\)](#) (ducky.mm.MemoryController method), 104
[read_u16\(\)](#) (ducky.mm.MemoryPage method), 105
[read_u32\(\)](#) (ducky.devices.storage.BlockIOMMIOMemoryPage method), 76
[read_u32\(\)](#) (ducky.mm.AnonymousMemoryPage method), 101
[read_u32\(\)](#) (ducky.mm.ExternalMemoryPage method), 101
[read_u32\(\)](#) (ducky.mm.MemoryController method), 104
[read_u32\(\)](#) (ducky.mm.MemoryPage method), 105
[read_u8\(\)](#) (ducky.devices.keyboard.KeyboardMMIOMemoryPage method), 73
[read_u8\(\)](#) (ducky.devices rtc.RTCMMIOMemoryPage method), 74
[read_u8\(\)](#) (ducky.mm.AnonymousMemoryPage method), 101
[read_u8\(\)](#) (ducky.mm.ExternalMemoryPage method), 101
[read_u8\(\)](#) (ducky.mm.MemoryController method), 104
[read_u8\(\)](#) (ducky.mm.MemoryPage method), 105
[readable](#) (ducky.mm.binary.SectionFlagsEncoding attribute), 98
[RealCPUCoreProfiler](#) (class in ducky.profiler), 109
[red\(\)](#) (ducky.log.ColorizedLogFormatter method), 93
[red\(\)](#) (ducky.log.LogFormatter method), 93
[REFRESH](#) (ducky.devices.svga.SimpleVGACommands attribute), 82
[reg](#) (ducky.cpu.instructions.EncodingC attribute), 48
[reg](#) (ducky.cpu.instructions.EncodingI attribute), 48
[REG\(\)](#) (ducky.cpu.CPUCore method), 63
[reg1](#) (ducky.cpu.instructions.EncodingA attribute), 47
[reg1](#) (ducky.cpu.instructions.EncodingR attribute), 49
[reg1](#) (ducky.cpu.instructions.EncodingS attribute), 49
[reg2](#) (ducky.cpu.instructions.EncodingA attribute), 47
[reg2](#) (ducky.cpu.instructions.EncodingR attribute), 49
[reg2](#) (ducky.cpu.instructions.EncodingS attribute), 49
[reg3](#) (ducky.cpu.instructions.EncodingA attribute), 47
[region_id](#) (ducky.mm.MemoryRegion attribute), 106
[register_command\(\)](#) (ducky.console.ConsoleMaster method), 32
[register_commands\(\)](#) (ducky.console.ConsoleMaster method), 32
[REGISTER_COUNT](#) (ducky.cpu.registers.Registers attribute), 62
[register_page\(\)](#) (ducky.mm.MemoryController method), 104
[REGISTER_SPECIAL](#) (ducky.cpu.registers.Registers attribute), 62
[register_with_reactor\(\)](#) (ducky.streams.Stream method), 115
[register_with_reactor\(\)](#) (ducky.tools.vm.WSInputStream method), 118
[RegisterAccess](#) (ducky.errors.ExceptionList attribute), 86
[RegisterAccessError](#), 88
[Registers](#) (class in ducky.cpu.registers), 61
[RegisterSet](#) (class in ducky.cpu.coprocessor.math_copro), 38
[RegisterSet](#) (class in ducky.cpu.registers), 61
[relative](#) (ducky.mm.binary.RelocFlagsEncoding attribute), 97
[relative_address](#) (ducky.cpu.instructions._BRANCH attribute), 59
[relative_address](#) (ducky.cpu.instructions._JUMP attribute), 60
[relative_address](#) (ducky.cpu.instructions.Descriptor attribute), 44
[relative_address](#) (ducky.cpu.instructions.LA attribute), 51
[release_ptes\(\)](#) (ducky.cpu.MMU method), 69
[RELOC](#) (ducky.mm.binary.SectionTypes attribute), 99
[RelocationPatcher](#) (class in ducky.tools.ld), 116
[RelocEntry](#) (class in ducky.mm.binary), 97
[RelocFlags](#) (class in ducky.mm.binary), 97
[RelocFlagsEncoding](#) (class in ducky.mm.binary), 97
[remove_fd\(\)](#) (ducky.reactor.Reactor method), 110
[remove_fd\(\)](#) (ducky.reactor.SelectTask method), 111
[remove_listener\(\)](#) (ducky.machine.EventBus method), 94
[remove_point\(\)](#) (ducky.debugging.DebuggingSet method), 70
[remove_task\(\)](#) (ducky.reactor.Reactor method), 110
[RemoveLoggingVisitor](#) (class in ducky.patch), 107

- ul style="list-style-type: none; padding-left: 0;">
- repr() (ducky.cpu.instructions.Encoding static method), 47
- reserved (ducky.mm.binary.FileFlagsEncoding attribute), 96
- RESET (ducky.devices.svga.SimpleVGACommands attribute), 82
- reset() (ducky.cpu.CPUCore method), 66
- reset() (ducky.cpu.MMU method), 69
- reset() (ducky.devices.storage.BlockIO method), 76
- reset() (ducky.devices.svga.SimpleVGA method), 82
- resolve_relocations() (in module ducky.tools.ld), 117
- resolve_symbols() (in module ducky.tools.ld), 117
- RET (class in ducky.cpu.instructions), 53
- RET (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- RETINT (class in ducky.cpu.instructions), 54
- RETINT (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- RI_ADDR() (in module ducky.cpu.instructions), 54
- RI_VAL() (in module ducky.cpu.instructions), 54
- ROMLoader (class in ducky.boot), 28
- ROWS (ducky.devices.svga.SimpleVGACommands attribute), 82
- RST (class in ducky.cpu.instructions), 54
- RST (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- RTC (class in ducky.devices.rtc), 74
- RTCMMIOMemoryPage (class in ducky.devices.rtc), 74
- RTCPorts (class in ducky.devices.rtc), 74
- RTCTask (class in ducky.devices.rtc), 74
- run() (ducky.cpu.CPUCore method), 66
- run() (ducky.devices.tty.FrontendFlushTask method), 80
- run() (ducky.interfaces.IMachineWorker method), 92
- run() (ducky.interfaces.IReactorTask method), 92
- run() (ducky.interfaces.IVirtualInterrupt method), 92
- run() (ducky.machine.HaltMachineTask method), 94
- run() (ducky.machine.IRQRouterTask method), 94
- run() (ducky.machine.Machine method), 95
- run() (ducky.reactor.CallInReactorTask method), 109
- run() (ducky.reactor.Reactor method), 110
- run() (ducky.reactor.RunInIntervalTask method), 111
- run() (ducky.reactor.SelectTask method), 111
- run_machine() (ducky.tools.vm.DuckyProtocol method), 118
- RunInIntervalTask (class in ducky.reactor), 110
- runtime_handle() (ducky.errors.ExecutionException method), 86
- runtime_handle() (ducky.errors.ExecutionException__Simple method), 87
- runtime_handle() (ducky.errors.InvalidFrameError method), 87
- S**
- SAVE (class in ducky.cpu.coprocessor.math_copro), 39
- SAVE (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
- save() (ducky.mm.binary.File method), 96
- save() (ducky.profiler.ProfilerStore method), 109
- save() (ducky.snapshot.CoreDumpFile method), 111
- save() (ducky.snapshot.VMState method), 112
- save_encoding() (ducky.util.Flags method), 120
- save_object_file() (in module ducky.tools.as), 116
- save_snapshot() (ducky.devices.snapshot.FileSnapshotStorage method), 75
- save_snapshot() (ducky.devices.snapshot.SnapshotStorage method), 75
- save_state() (ducky.boot.MMapArea method), 27
- save_state() (ducky.cpu.coprocessor.math_copro.MathCoproprocessor method), 36
- save_state() (ducky.cpu.coprocessor.math_copro.RegisterSet method), 38
- save_state() (ducky.cpu.CPU method), 63
- save_state() (ducky.cpu.CPUCore method), 66
- save_state() (ducky.interfaces.ISnapshotable method), 92
- save_state() (ducky.machine.Machine method), 95
- save_state() (ducky.mm.ExternalMemoryPage method), 101
- save_state() (ducky.mm.MemoryController method), 104
- save_state() (ducky.mm.MemoryPage method), 105
- save_state() (ducky.mm.MemoryRegion method), 106
- save_state() (ducky.mm.VirtualMemoryPage method), 106
- SAVEW (class in ducky.cpu.coprocessor.math_copro), 39
- SAVEW (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
- SECOND (ducky.devices.rtc.RTCPorts attribute), 74
- Section (class in ducky.mm.binary), 98
- section (ducky.mm.binary.SymbolEntry attribute), 100
- section_ordering() (ducky.tools.ld.LinkerScript method), 116
- SectionFlags (class in ducky.mm.binary), 98
- SectionFlagsEncoding (class in ducky.mm.binary), 98
- SectionHeader (class in ducky.mm.binary), 98
- sections (ducky.mm.binary.File attribute), 96
- sections (ducky.mm.binary.FileHeader attribute), 97
- SectionTypes (class in ducky.mm.binary), 99
- SELE (class in ducky.cpu.instructions), 54
- SELECT (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- select_storage() (ducky.devices.storage.BlockIO method), 76
- SELECTTask (class in ducky.reactor), 111
- SELG (class in ducky.cpu.instructions), 54
- SELGE (class in ducky.cpu.instructions), 54
- SELL (class in ducky.cpu.instructions), 54
- SELLE (class in ducky.cpu.instructions), 55
- SELNE (class in ducky.cpu.instructions), 55
- SELNO (class in ducky.cpu.instructions), 55

- SELNS (class in ducky.cpu.instructions), 55
- SELNZ (class in ducky.cpu.instructions), 55
- SELO (class in ducky.cpu.instructions), 55
- SELS (class in ducky.cpu.instructions), 55
- SELZ (class in ducky.cpu.instructions), 55
- SET (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- set() (ducky.config.MachineConfig method), 31
- set_backend() (ducky.devices.DeviceFrontend method), 84
- set_condition() (ducky.cpu.instructions._COND static method), 60
- set_frontend() (ducky.devices.DeviceBackend method), 84
- set_mode() (ducky.devices.svga.SimpleVGA method), 82
- set_output() (ducky.devices.tty.Frontend method), 79
- set_output() (ducky.devices.tty.FrontendFlushTask method), 80
- SETE (class in ducky.cpu.instructions), 55
- SETG (class in ducky.cpu.instructions), 56
- SETGE (class in ducky.cpu.instructions), 56
- SETL (class in ducky.cpu.instructions), 56
- SETLE (class in ducky.cpu.instructions), 56
- SETNE (class in ducky.cpu.instructions), 56
- SETNO (class in ducky.cpu.instructions), 56
- SETNS (class in ducky.cpu.instructions), 56
- SETNZ (class in ducky.cpu.instructions), 56
- SETO (class in ducky.cpu.instructions), 56
- SETS (class in ducky.cpu.instructions), 56
- setup() (ducky.mm.binary.File method), 96
- setup() (ducky.util.BinaryFile method), 119
- setup_bootloader() (ducky.boot.ROMLoader method), 29
- setup_debugging() (ducky.boot.ROMLoader method), 29
- setup_devices() (ducky.machine.Machine method), 95
- setup_hdt() (ducky.boot.ROMLoader method), 29
- setup_logger() (in module ducky.tools), 119
- setup_mmmaps() (ducky.boot.ROMLoader method), 30
- SETZ (class in ducky.cpu.instructions), 57
- SHL (class in ducky.cpu.instructions), 57
- SHL (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- SHORT (ducky.mm.binary.SymbolDataTypes attribute), 100
- show_cores() (in module ducky.tools.coredump), 116
- show_disassemble() (in module ducky.tools.objdump), 117
- show_dump() (in module ducky.tools.coredump), 116
- show_file_header() (in module ducky.tools.objdump), 117
- show_forth_dict() (in module ducky.tools.coredump), 116
- show_forth_word() (in module ducky.tools.coredump), 116
- show_header() (in module ducky.tools.coredump), 116
- show_memory() (in module ducky.tools.coredump), 116
- show_pages() (in module ducky.tools.coredump), 116
- show_reloc() (in module ducky.tools.objdump), 117
- show_sections() (in module ducky.tools.objdump), 117
- show_symbols() (in module ducky.tools.objdump), 117
- SHR (class in ducky.cpu.instructions), 57
- SHR (ducky.cpu.instructions.DuckyOpcodes attribute), 46
- SHRS (class in ducky.cpu.instructions), 57
- SHRS (ducky.cpu.instructions.DuckyOpcodes attribute), 47
- SID (ducky.devices.storage.BlockIOPorts attribute), 76
- sign_extend_immediate() (ducky.cpu.instructions.Encoding static method), 47
- sign_extend_immediate() (ducky.cpu.instructions.EncodingC static method), 48
- sign_extend_immediate() (ducky.cpu.instructions.EncodingI static method), 48
- sign_extend_immediate() (ducky.cpu.instructions.EncodingR static method), 49
- sign_extend_immediate() (ducky.cpu.instructions.EncodingS static method), 49
- sign_extend_with_push() (ducky.cpu.coprocessor.math_copro.MathCoproprocessor method), 36
- SimpleVGA (class in ducky.devices.svga), 81
- SimpleVGACommands (class in ducky.devices.svga), 82
- SimpleVGAMemoryPage (class in ducky.devices.svga), 83
- SimpleVGAMMIOMemoryPage (class in ducky.devices.svga), 82
- SimpleVGAPorts (class in ducky.devices.svga), 83
- SIS (class in ducky.cpu.instructions), 57
- SIS (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
- SIS (ducky.cpu.instructions.DuckyOpcodes attribute), 47
- SIZE (ducky.cpu.InterruptVector attribute), 67
- size (ducky.hdt.HDTEntry_Memory attribute), 91
- size (ducky.mm.binary.SymbolEntry attribute), 100
- SIZE_FMT() (in module ducky.mm), 106
- sizeof_fmt() (in module ducky.util), 121
- sleep_flush() (ducky.devices.tty.Frontend method), 79
- SnapshotNode (class in ducky.snapshot), 111
- SnapshotStorage (class in ducky.devices.snapshot), 75
- SP (ducky.cpu.registers.Registers attribute), 62
- SP() (ducky.cpu.CPUCore method), 63
- STACK_DEPTH (in module ducky.cpu.coprocessor.math_copro), 39
- StackFrame (class in ducky.cpu), 69
- StandalonePTYTerminal (class in ducky.devices.terminal), 78

- StandardIOTerminal (class in ducky.devices.terminal), 78
- STATUS (ducky.devices.keyboard.KeyboardPorts attribute), 73
- STATUS (ducky.devices.storage.BlockIOPorts attribute), 76
- status_write() (ducky.devices.storage.BlockIO method), 76
- STB (class in ducky.cpu.instructions), 57
- STB (ducky.cpu.instructions.DuckyOpcodes attribute), 47
- StderrStream (class in ducky.streams), 113
- StdinStream (class in ducky.streams), 114
- StdoutStream (class in ducky.streams), 114
- step() (ducky.cpu.CPUCore method), 66
- STI (class in ducky.cpu.instructions), 57
- STI (ducky.cpu.instructions.DuckyOpcodes attribute), 47
- Storage (class in ducky.devices.storage), 77
- StorageAccessError, 78
- STORE (in module ducky.profiler), 109
- str2int() (in module ducky.util), 121
- Stream (class in ducky.streams), 114
- StreamHandler (class in ducky.log), 93
- StreamIOTerminal (class in ducky.devices.terminal), 78
- STRING (ducky.mm.binary.SymbolDataTypes attribute), 100
- string_table (ducky.mm.binary.File attribute), 96
- STRINGS (ducky.mm.binary.SectionTypes attribute), 99
- StringTable (class in ducky.util), 120
- STS (class in ducky.cpu.instructions), 58
- STS (ducky.cpu.instructions.DuckyOpcodes attribute), 47
- STW (class in ducky.cpu.instructions), 58
- STW (ducky.cpu.instructions.DuckyOpcodes attribute), 47
- SUB (class in ducky.cpu.instructions), 58
- SUB (ducky.cpu.instructions.DuckyOpcodes attribute), 47
- suspend() (ducky.cpu.CPU method), 63
- suspend() (ducky.cpu.CPUCore method), 66
- suspend() (ducky.interfaces.IMachineWorker method), 92
- suspend() (ducky.machine.Machine method), 95
- SuspendCoreAction (class in ducky.debugging), 71
- SWP (class in ducky.cpu.coprocessor.math_copro), 39
- SWP (class in ducky.cpu.instructions), 58
- SWP (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
- SWP (ducky.cpu.instructions.DuckyOpcodes attribute), 47
- SymbolDataTypes (class in ducky.mm.binary), 99
- SymbolEntry (class in ducky.mm.binary), 100
- SymbolFlags (class in ducky.mm.binary), 100
- SymbolFlagsEncoding (class in ducky.mm.binary), 100
- SYMBOLS (ducky.mm.binary.SectionTypes attribute), 99
- SymbolTable (class in ducky.util), 121
- SYMDIVL (class in ducky.cpu.coprocessor.math_copro), 39
- SYMDIVL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
- SYMMODL (class in ducky.cpu.coprocessor.math_copro), 40
- SYMMODL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
- ## T
- table() (ducky.console.ConsoleConnection method), 31
- take_sample() (ducky.profiler.DummyCPUCoreProfiler method), 108
- take_sample() (ducky.profiler.RealCPUCoreProfiler method), 109
- task_runnable() (ducky.reactor.Reactor method), 110
- task_suspended() (ducky.reactor.Reactor method), 110
- tenh() (ducky.devices.tty.Backend method), 79
- tenh() (ducky.machine.Machine method), 95
- tenh_close_stream() (ducky.devices.tty.Backend method), 79
- tenh_enable() (ducky.devices.tty.Backend method), 79
- tenh_enable() (ducky.devices.tty.Frontend method), 79
- tenh_flush_stream() (ducky.devices.tty.Backend method), 79
- Terminal (class in ducky.devices.terminal), 79
- TerminalConsoleConnection (class in ducky.console), 32
- to_encoding() (ducky.util.Flags method), 120
- to_int() (ducky.util.Flags method), 120
- to_pretty_string() (ducky.devices.svga.Mode method), 81
- to_string() (ducky.devices.svga.Mode method), 81
- to_string() (ducky.util.Flags method), 120
- to_u8() (ducky.devices.svga.Char method), 81
- TooManyLabelsError, 88
- tos() (ducky.cpu.coprocessor.math_copro.RegisterSet method), 38
- tos1() (ducky.cpu.coprocessor.math_copro.RegisterSet method), 39
- trigger() (ducky.machine.EventBus method), 94
- trigger_irq() (ducky.machine.Machine method), 95
- TTYMMIOMemoryPage (class in ducky.devices.tty), 80
- TTYPorts (class in ducky.devices.tty), 80
- type (ducky.devices.keyboard.HDTEEntry_Keyboard attribute), 73
- type (ducky.devices.rtc.HDTEEntry_RTC attribute), 74
- type (ducky.devices.tty.HDTEEntry_TTY attribute), 80
- type (ducky.hdt.HDTEEntry_Argument attribute), 90
- type (ducky.hdt.HDTEEntry_CPU attribute), 90
- type (ducky.hdt.HDTEEntry_Memory attribute), 91
- type (ducky.mm.binary.SectionHeader attribute), 99
- type (ducky.mm.binary.SymbolEntry attribute), 100
- ## U
- UDIV (class in ducky.cpu.instructions), 58

- UDIV (ducky.cpu.instructions.DuckyOpcodes attribute), 47
 - UDIVL (class in ducky.cpu.coprocessor.math_copro), 40
 - UDIVL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
 - UINT20_FMT() (in module ducky.cpu.instructions), 58
 - UMODL (class in ducky.cpu.coprocessor.math_copro), 40
 - UMODL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 37
 - UnalignedAccess (ducky.errors.ExceptionList attribute), 86
 - UnalignedAccessError, 88
 - UnalignedJumpTargetError, 88
 - UNDEFINED (ducky.hdt.HDTEntryTypes attribute), 89
 - UNKNOWN (ducky.mm.binary.SectionTypes attribute), 99
 - UNKNOWN (ducky.mm.binary.SymbolDataTypes attribute), 100
 - UnknownDestinationSectionError, 88
 - UnknownFileError, 88
 - UnknownInstructionError, 89
 - UnknownPatternError, 89
 - UnknownSymbolError, 89
 - UNMMAP (ducky.mm.MMOperationList attribute), 102
 - unmmap_area() (ducky.boot.ROMLoader method), 30
 - unregister_command() (ducky.console.ConsoleMaster method), 32
 - unregister_page() (ducky.mm.MemoryController method), 104
 - unregister_queue() (ducky.machine.CommChannel method), 93
 - unregister_with_reactor() (ducky.streams.Stream method), 115
 - unregister_with_reactor() (ducky.tools.vm.WSInputStream method), 118
 - unused (ducky.devices.svga.Char attribute), 81
 - UNUSED (ducky.mm.MMOperationList attribute), 102
 - update_arith_flags() (in module ducky.cpu.instructions), 61
 - update_tick() (ducky.devices rtc.RTCTask method), 74
- ## V
- value (ducky.cpu.instructions.EncodingC attribute), 48
 - value (ducky.cpu.instructions.EncodingS attribute), 49
 - value (ducky.hdt.HDTEntry_Argument attribute), 90
 - value_length (ducky.hdt.HDTEntry_Argument attribute), 90
 - VERSION (ducky.mm.binary.File attribute), 96
 - version (ducky.mm.binary.FileHeader attribute), 97
 - VirtualMemoryPage (class in ducky.mm), 106
 - visit_Expr() (ducky.patch.RemoveLoggingVisitor method), 107
 - visit_For() (ducky.patch.RemoveLoggingVisitor method), 107
 - visit_If() (ducky.patch.RemoveLoggingVisitor method), 107
 - VM, 26
 - VMState (class in ducky.snapshot), 112
- ## W
- wake_up() (ducky.cpu.CPU method), 63
 - wake_up() (ducky.cpu.CPUCore method), 66
 - wake_up() (ducky.interfaces.IMachineWorker method), 92
 - wake_up() (ducky.machine.Machine method), 95
 - wakeup_flush() (ducky.devices.tty.Frontend method), 79
 - where_to_base() (ducky.tools.ld.LinkerScript method), 116
 - where_to_merge() (ducky.tools.ld.LinkerScript method), 116
 - white() (ducky.log.ColorizedLogFormatter method), 93
 - white() (ducky.log.LogFormatter method), 93
 - writable (ducky.mm.binary.SectionFlagsEncoding attribute), 98
 - WRITE (ducky.mm.PageTableEntry attribute), 106
 - write() (ducky.console.ConsoleConnection method), 31
 - write() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 32
 - write() (ducky.mm.binary.Section method), 98
 - write() (ducky.streams.InputStream method), 113
 - write() (ducky.streams.OutputStream method), 113
 - write() (ducky.streams.Stream method), 115
 - write() (ducky.tools.vm.WSOutputStream method), 119
 - write_blocks() (ducky.devices.storage.Storage method), 78
 - write_cr1() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 32
 - write_cr2() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 32
 - write_cr3() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 32
 - write_data() (ducky.devices.storage.BlockIO method), 76
 - write_in() (ducky.machine.CommQueue method), 93
 - write_out() (ducky.machine.CommQueue method), 93
 - write_struct() (ducky.util.BinaryFile method), 119
 - write_u16() (ducky.devices.svga.SimpleVGAMMIOMemoryPage method), 82
 - write_u16() (ducky.mm.AnonymousMemoryPage method), 101
 - write_u16() (ducky.mm.ExternalMemoryPage method), 101
 - write_u16() (ducky.mm.MemoryController method), 104
 - write_u16() (ducky.mm.MemoryPage method), 105
 - write_u32() (ducky.devices.storage.BlockIOMMIOMemoryPage method), 76

`write_u32()` (ducky.mm.AnonymousMemoryPage method), [101](#)
`write_u32()` (ducky.mm.ExternalMemoryPage method), [101](#)
`write_u32()` (ducky.mm.MemoryController method), [104](#)
`write_u32()` (ducky.mm.MemoryPage method), [105](#)
`write_u8()` (ducky.devices.rtc.RTCMMIOMemoryPage method), [74](#)
`write_u8()` (ducky.devices.tty.TTYMMIOMemoryPage method), [80](#)
`write_u8()` (ducky.mm.AnonymousMemoryPage method), [101](#)
`write_u8()` (ducky.mm.ExternalMemoryPage method), [101](#)
`write_u8()` (ducky.mm.MemoryController method), [104](#)
`write_u8()` (ducky.mm.MemoryPage method), [106](#)
`writeln()` (ducky.console.ConsoleConnection method), [31](#)
`WSInputStream` (class in ducky.tools.vm), [118](#)
`WStream` (class in ducky.tools.vm), [118](#)

X

`XOR` (class in ducky.cpu.instructions), [59](#)
`XOR` (ducky.cpu.instructions.DuckyOpcodes attribute), [47](#)

Y

`YEAR` (ducky.devices.rtc.RTCPorts attribute), [74](#)