
Ducky Documentation

Release 1.0

Milos Prchlik

April 24, 2016

1	About	3
1.1	ducky	3
1.2	Getting started	4
1.3	Virtual hardware	9
1.4	Software	15
1.5	Tools	16
1.6	Examples	22
2	Code Documentation	23
2.1	ducky.boot module	23
2.2	ducky.config module	25
2.3	ducky.console module	26
2.4	ducky.cc package	27
2.5	ducky.cpu package	35
2.6	ducky.debugging module	66
2.7	ducky.devices package	69
2.8	ducky.errors module	77
2.9	ducky.hdt module	77
2.10	ducky.interfaces module	79
2.11	ducky.log module	80
2.12	ducky.machine module	81
2.13	ducky.mm package	83
2.14	ducky.patch module	92
2.15	ducky.profiler module	93
2.16	ducky.reactor module	94
2.17	ducky.snapshot module	97
2.18	ducky.streams module	97
2.19	ducky.tools package	99
2.20	ducky.util module	103
3	Indices and tables	105
	Python Module Index	107

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 approach, 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 assembler 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
- and `cc`, an experimental C compiler

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__
'2.0'
```

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.

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` directory. If you check it out, it’s a plain and simple assembler:

```
.include "tty.asm"

.data

.type stack, space
.space 64

.type message, string
.string "Hello, world!"

.text

main:
    la sp, &stack
    add sp, 64

    la r0, &message
    call &writesn
    hlt 0x00

outb:
    ; > r0: port
    ; > r1: byte
    outb r0, r1
    ret

writesn:
    ; > r0: string address
```

```
; ...
; r0: port
; r1: current byte
; r2: string ptr
push r1
push r2
mov r2, r0
li r0, $TTY_PORT_DATA
.__writesn_loop:
lb r1, r2
bz &.__writesn_write_nl
call &outb
inc r2
j &.__writesn_loop
.__writesn_write_nl:
; \n
li r1, 0x0000000A
call &outb
; \r
li r1, 0x0000000D
call &outb
li r0, 0x00000000
pop r2
pop r1
ret
```

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

Binary

Virtual machine needs binary (or bytecode, as you wish...) code, and there's a tool for it:

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

This command will translate source code to object file, containing instructions for VM and other resources. 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.bin
[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] message                .data      0x000000    string (2)          14   examples/hello-world.asm
[INFO] main                    .text      0x000100    function (3)         0   examples/hello-world.asm
[INFO] outb                    .text      0x000110    function (3)         0   examples/hello-world.asm
[INFO] writesn                 .text      0x000118    function (3)         0   examples/hello-world.asm
[INFO] __fn_writesn_loop       .text      0x00012C    function (3)         0   examples/hello-world.asm
[INFO] __fn_writesn_write_nl   .text      0x000140    function (3)         0   examples/hello-world.asm
[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]

```

You can see internal sections in the object file, list of symbols, and disassembled instructions, 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 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 sections in object files, and updates addresses used by instructions to retrieve data and to perform jumps. You can inspect the binary file using `objdump` tool, too:

```
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 files, 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 -g
```

There are two other command line options that deserve some explanation:

- `-g` - by default, VM prepares itself, and waits for user to press `Enter` to actually start running the loaded binaries. This option tells it to skip “press any key” phase and go ahead.

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 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 15, 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 - as I said, terminal is dedicated to VM itself.

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 equipped 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 `r28` are general purpose registers - `r30` is reserved, and used as a stack pointer register, `SP` - contains address of the last push'ed value on stack - `r29` is reserved, and used as a frame pointer register, `FP` - contains content of `SP` in time of the last `call` or `int` instruction - `r31` is reserved, and used as a instruction pointer register, `IP` - contains address of **next** instruction to be executed
- flags register

`IP` and flags registers are protected, and cannot be modified by standard means (`push flags;` `<modify flags>;` `pop flags`) when CPU is in user mode

Flags register

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

Instruction set

CPU supports multiple instruction set. 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.

Design principles

- load and store operations are performed by dedicated instructions
- all memory-register transfers work with 32-bit operands, 16-bit and 8-bit operands are handled by special instructions when necessary
- all memory-register transfers work with addresses that are aligned to the size of their operands (1 byte alignment - so no alignment at all - for 8-bit operands)
- in most cases, destination operand is the first one. Exceptions are instructions that work with IO ports.
- when content of a register is changed by an 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 `r28` - 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

Ducky instruction set

Design principles

- basic data unit is *a word*, 4 bytes, 32 bits. Other units are *short* and *byte*. 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)
- in most cases, destination operand is the first one. Exceptions are instructions that work with IO ports.
- 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 `r28` - 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

Stack frames Several instructions transfer control to other routines, with possibility of returning back to 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>)
shifl rA, (rB|<value>)
shiftr 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 bne bs bns bz bnz bo bno bg bge bl ble	$e = 1$ $e = 0$ $s = 1$ $s = 0$ $z = 1$ $z = 0$ $o = 1$ $o = 0$ $e = 0$ and $s = 1$ $e = 1$ 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.
cmpu rA, (rB|<value>) - treat operands as unsigned values, immediate value is zero-extended to 32 bits.
```

Port IO All IO instructions take two operands: port number, specified by register or immediate value, and register.

`inw (rA|<port>), rB` - read word from port and store it in `rB`

`ins (rA|<port>), rB` - read short from port and store it in `rB`

`inb (rA|<port>), rB` - read byte from port and store it in `rB`

`outw (rA|<port>), rB` - write value from `rB` to port

`outs (rA|<port>), rB` - write lower short of `rB` to port

`outb (rA|<port>), rB` - write lowest byte of `rB` to port

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 `IVT` 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

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`

`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`

`ltol`

`ltol`

`iitol`

dupl

Arithmetic operations incl

decl

addl

subl

mull

divl

modl

syndivl

symmodl

1.3.2 Memory

Memory model

- the full addressable memory is 4 GB, 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

Interrupt Vector table

Interrupt vector table (*IVT*), located in main memory, is by default located at address 0x00000000. *IVT* address can be set per CPU core. *IVT* is 256 bytes long, providing enough space for 64 entries. Typically, lower 32 entries are reserved for hardware interrupts, provided by devices, and upper 32 entries leads to software routines that provide additional functionality for binaries. Each entry has the same layout:

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

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

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 *IVT* 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 IVT, 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

IVT 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 transferred from storage into memory
- 1 - *write mode*, blocks are transferred 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

IVT entry	34	
		QUIET mode
Parameters	r0	0
	r1	0 for <i>quiet</i> mode, anything else for <i>full</i> mode
Returns	r0	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 beginning of the code, and turn it on again at the beginning 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` is 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, symbols 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.

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

ivt-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.StandalonePTY
```

--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.

--stdio-console

Enable console terminal with stdin and stdout as its IO streams. User can then enter commands in via the keyboard, while VM and its binaries use e.g. stand-alone pty terminals.

`-g, --go-on`

By default, `ducky-vm` creates a VM and boots it, but before handing the control to it, `ducky-vm` will ask user to press any key. `-g` option tells `ducky-vm` to skip this part, and immediately start execution of binaries.

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

Code Documentation

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 can 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 `mm.MMapArea` instances, and if such area was opened as *shared*, every change in this page content will affect the content of

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 (`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`

boot ()

halt ()

load_data (`base`, `content`)

load_text (`base`, `content`)

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*)

setup_debugging ()

setup_hdt ()

Initialize memory area that contains HDT.

If VM config file specifies HDT image file, it is loaded, otherwise HDT is constructed for the actual configuration. It is then 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** – 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.

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_mmaps** ()**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.cc package

2.4.1 Subpackages

ducky.cc.passes package

Submodules

ducky.cc.passes.ast_codegen module

```

class ducky.cc.passes.ast_codegen.CodegenVisitor (*args, **kwargs)
    Bases: ducky.cc.passes.ASTVisitor
    block (stage=None, *args, **kwargs)
    emit_epilog ()
    emit_prolog ()
    emit_string_literals ()
    emit_trampoline ()
    generic_visit (node, **kwargs)
    get_new_label (name=None)
    get_new_literal_label ()
    get_new_local_storage (size)
    make_current (block)
    materialize ()
    pop_scope ()
    priority = 1000
    process_cond (node, iftrue_label=None, iffalse_label=None)

```

```
push_scope ()
reset_scope ()
visit (node, **kwargs)
visit_ArrayRef (node, preferred=None, keep=None)
visit_Assignment (node, preferred=None, keep=None)
visit_BinaryOp (node, preferred=None, keep=None)
visit_Cast (node, **kwargs)
visit_Compound (node, create_scope=True)
visit_Constant (node, preferred=None, keep=None)
visit_Decl (node)
visit_ExprList (node)
visit_FileAST (node)
visit_For (node)
visit_FuncCall (node, preferred=None, keep=None)
visit_FuncDef (node)
visit_ID (node, preferred=None, keep=None)
visit_If (node)
visit_Return (node)
visit_StructRef (node, **kwargs)
visit_TypeDecl (node, **kwargs)
visit_Typedef (node)
visit_Typename (node, **kwargs)
visit_UnaryOp (node, preferred=None, keep=None)
visit_While (node)
visit_constant_value (node)
visit_expr (node, preferred=None, keep=None)
```

ducky.cc.passes.ast_constprop module

```
class ducky.cc.passes.ast_constprop.ConstantFoldingVisitor (logger, *args, **kwargs)
    Bases: ducky.cc.passes.ASTOptVisitor

    priority = 100
    visit_BinaryOp (node)
```

ducky.cc.passes.ast_dce module

```
class ducky.cc.passes.ast_dce.DSEVisitor (logger, *args, **kwargs)
    Bases: ducky.cc.passes.ASTOptVisitor

    priority = 500
    visit_Compound (node)
```


ducky.cc.passes.ast_visualise module

```

class ducky.cc.passes.ast_visualise.ASTVisualiseVisitor(*args, **kwargs)
    Bases: ducky.cc.passes.ASTVisitor

    generic_visit(node, shape='box')

    get_index(node)

    visit_BinaryOp(node, shape='box')

    visit_Constant(node, shape='box')

    visit_FileAST(node)

    visit_ID(node, shape='box')

    visit_If(node)

    visit_UnaryOp(node, shape='box')

    visit_While(node)

```

ducky.cc.passes.bt_peephole module

```

class ducky.cc.passes.bt_peephole.BTPeeppholeVisitor(logger, *args, **kwargs)
    Bases: ducky.cc.passes.BlockVisitor

    do_visit_block(block)

    priority = 100

```

ducky.cc.passes.bt_simplify module

```

class ducky.cc.passes.bt_simplify.BlockTreeSimplifyVisitor(logger, *args, **kwargs)
    Bases: ducky.cc.passes.BlockVisitor

    do_visit_fn(fn)

    priority = 500

```

ducky.cc.passes.bt_visualise module

```

class ducky.cc.passes.bt_visualise.BlockTreeVisualiseVisitor(*args, **kwargs)
    Bases: ducky.cc.passes.BlockVisitor

    do_visit_block(block)

    priority = 1000

    visit(cv)

```

Module contents

```

class ducky.cc.passes.ASTOptVisitor(logger, *args, **kwargs)
    Bases: ducky.cc.passes.ASTVisitor

    replace_child(current_node, new_node)

class ducky.cc.passes.ASTVisitor(logger, *args, **kwargs)
    Bases: pycparser.c_ast.NodeVisitor

    DOWN()

    UP()

```

```

    generic_visit (node)

    priority = 99

    visit (node, **kwargs)

class ducky.cc.passes.BlockVisitor (logger, *args, **kwargs)
    Bases: object

    DOWN ()

    UP ()

    do_visit (cv)

    do_visit_block (block)

    do_visit_fn (fn)

    priority = 99

    visit (cv)

    visit_block (block)

    visit_fn (fn)

ducky.cc.passes.load (logger)

```

2.4.2 Submodules

ducky.cc.types module

```

class ducky.cc.types.ArrayType (item_type, size=None, *args, **kwargs)
    Bases: ducky.cc.types.CType

class ducky.cc.types.CType (visitor, decl=None)
    Bases: object

    static create_from_decl (visitor, decl)

    static get_from_decl (visitor, decl)

    static get_from_desc (visitor, desc)

    types = {}

class ducky.cc.types.CharType (visitor, decl=None)
    Bases: ducky.cc.types.CType

class ducky.cc.types.FunctionType (*args, **kwargs)
    Bases: ducky.cc.types.CType

class ducky.cc.types.IntType (visitor, decl=None)
    Bases: ducky.cc.types.CType

class ducky.cc.types.PointerType (ptr_to_type, *args, **kwargs)
    Bases: ducky.cc.types.CType

class ducky.cc.types.StructType (name, *args, **kwargs)
    Bases: ducky.cc.types.CType

    field_offset (name)

    field_type (name)

```

```
class ducky.cc.types.UnsignedCharType (visitor, decl=None)
    Bases: ducky.cc.types.CharType
```

```
class ducky.cc.types.UnsignedIntType (visitor, decl=None)
    Bases: ducky.cc.types.IntType
```

```
class ducky.cc.types.VoidType (visitor, decl=None)
    Bases: ducky.cc.types.CType
```

2.4.3 Module contents

```
class ducky.cc.ADD (*operands)
    Bases: ducky.cc.Instruction
```

```
class ducky.cc.AND (*operands)
    Bases: ducky.cc.Instruction
```

```
class ducky.cc.BE (label)
    Bases: ducky.cc.Instruction
```

```
class ducky.cc.BG (label)
    Bases: ducky.cc.Instruction
```

```
class ducky.cc.BGE (label)
    Bases: ducky.cc.Instruction
```

```
class ducky.cc.BL (label)
    Bases: ducky.cc.Instruction
```

```
class ducky.cc.BLE (label)
    Bases: ducky.cc.Instruction
```

```
class ducky.cc.BNE (label)
    Bases: ducky.cc.Instruction
```

```
class ducky.cc.Block (name=None, comment=None)
    Bases: object
```

```
    add_incoming (block)
```

```
    add_name (name)
```

```
    add_outgoing (block)
```

```
    connect (next)
```

```
    emit (inst)
```

```
    id = 0
```

```
    instructions ()
```

```
    materialize (code)
```

```
class ducky.cc.CALL (label)
    Bases: ducky.cc.Instruction
```

```
class ducky.cc.CMP (left, right)
    Bases: ducky.cc.Instruction
```

```
class ducky.cc.Comment (comment)
    Bases: object
```

```
    materialize ()
```

```
exception ducky.cc.CompilerError (location, msg)
    Bases: exceptions.Exception

class ducky.cc.ConstantValue (value)
    Bases: ducky.cc.NamedValue

class ducky.cc.Directive (directive)
    Bases: ducky.cc.Instruction

    materialize()

class ducky.cc.Expression (value=None, type=None, klass=<ExpressionClass.RVALUE: 2>)
    Bases: object

    is_lvalue()

    is_mlvalue()

    is_rvalue()

    to_rvalue (visitor, preferred=None, keep=None)

class ducky.cc.ExpressionClass
    Bases: enum.Enum

    LVALUE = <ExpressionClass.LVALUE: 0>

    MLVALUE = <ExpressionClass.MLVALUE: 1>

    RVALUE = <ExpressionClass.RVALUE: 2>

class ducky.cc.Function (visitor, decl, ftype, args_types=None)
    Bases: object

    args_block()

    block (*args, **kwargs)

    body_block()

    epilog_block()

    finish()

    header_block()

    materialize()

    prolog_block()

class ducky.cc.HLT (isr)
    Bases: ducky.cc.Instruction

class ducky.cc.INC (reg)
    Bases: ducky.cc.Instruction

class ducky.cc.INT (isr)
    Bases: ducky.cc.Instruction

exception ducky.cc.IncompatibleTypesError (loc, t1, t2)
    Bases: ducky.cc.CompilerError

class ducky.cc.InlineAsm (code)
    Bases: ducky.cc.Instruction

    materialize()
```

```

class ducky.cc.Instruction(opcode, *operands)
    Bases: object

    materialize()

exception ducky.cc.IsAPointerError(loc, t)
    Bases: ducky.cc.CompilerError

class ducky.cc.J(label)
    Bases: ducky.cc.Instruction

class ducky.cc.LA(reg, value)
    Bases: ducky.cc.Instruction

class ducky.cc.LB(reg, addr)
    Bases: ducky.cc.Instruction

class ducky.cc.LI(reg, value)
    Bases: ducky.cc.Instruction

class ducky.cc.LS(reg, addr)
    Bases: ducky.cc.Instruction

class ducky.cc.LValueExpression(*args, **kwargs)
    Bases: ducky.cc.Expression

class ducky.cc.LW(reg, addr)
    Bases: ducky.cc.Instruction

class ducky.cc.MLValueExpression(*args, **kwargs)
    Bases: ducky.cc.Expression

class ducky.cc.MOV(*operands)
    Bases: ducky.cc.Instruction

class ducky.cc.MUL(*operands)
    Bases: ducky.cc.Instruction

class ducky.cc.MemorySlotStorage(symbol, label)
    Bases: ducky.cc.SymbolStorage

    addrof(register, emit)

    name()

class ducky.cc.MemorySlotValue(storage)
    Bases: ducky.cc.NamedValue

class ducky.cc.NOT(*operands)
    Bases: ducky.cc.Instruction

class ducky.cc.NamedValue(name)
    Bases: object

    backing_register()

    can_register_backed()

    is_register_backed()

exception ducky.cc.NotAPointerError(loc, t)
    Bases: ducky.cc.CompilerError

class ducky.cc.OR(*operands)
    Bases: ducky.cc.Instruction

```

```
class ducky.cc.POP (reg)
    Bases: ducky.cc.Instruction

class ducky.cc.PUSH (reg)
    Bases: ducky.cc.Instruction

class ducky.cc.RET
    Bases: ducky.cc.Instruction

class ducky.cc.RValueExpression (*args, **kwargs)
    Bases: ducky.cc.Expression

class ducky.cc.Register (rset, index)
    Bases: object

    free ()

    put ()

class ducky.cc.RegisterMemorySlotValue (register)
    Bases: ducky.cc.NamedValue

class ducky.cc.RegisterSet (fn)
    Bases: object

    get (preferred=None, keep=None)

    restore_callee_saves (block)

    save_callee_saves (block)

class ducky.cc.RegisterValue (register)
    Bases: ducky.cc.NamedValue

class ducky.cc.SHL (reg, ri)
    Bases: ducky.cc.Instruction

class ducky.cc.SHR (reg, ri)
    Bases: ducky.cc.Instruction

class ducky.cc.STB (addr, reg)
    Bases: ducky.cc.Instruction

class ducky.cc.STS (addr, reg)
    Bases: ducky.cc.Instruction

class ducky.cc.STW (addr, reg)
    Bases: ducky.cc.Instruction

class ducky.cc.SUB (*operands)
    Bases: ducky.cc.Instruction

class ducky.cc.Scope (visitor, parent=None)
    Bases: object

    add (loc, symbol)

    get (name)

    scope_id = 0

class ducky.cc.StackSlotStorage (symbol, offset)
    Bases: ducky.cc.SymbolStorage

    addrof (register, emit)
```

```

    name ()

class ducky.cc.StackSlotValue (storage)
    Bases: ducky.cc.NamedValue

class ducky.cc.StringConstantValue (value)
    Bases: ducky.cc.ConstantValue

class ducky.cc.Symbol (visitor, name, decl_type, extern=False, defined=False, const=False)
    Bases: object

exception ducky.cc.SymbolAlreadyDefinedError (loc, symbol)
    Bases: ducky.cc.CompilerError

exception ducky.cc.SymbolConflictError (location, msg)
    Bases: ducky.cc.CompilerError

class ducky.cc.SymbolStorage (symbol, register=None)
    Bases: object

    acquire_register (register)

    addrof (reg, emit)

    has_register ()

    name ()

    release_register ()

    spill_register (visitor)

    unspill_register (visitor, register)

exception ducky.cc.SymbolUndefined (loc, symbol)
    Bases: ducky.cc.CompilerError

exception ducky.cc.UnableToImplicitCastError (loc, t1, t2)
    Bases: ducky.cc.CompilerError

exception ducky.cc.UndefinedStructMemberError (loc, s, m)
    Bases: ducky.cc.CompilerError

ducky.cc.dump_node (node)

ducky.cc.show_node (node)

```

2.5 ducky.cpu package

2.5.1 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.Coprocessor

    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 = <ControlRegisters.CR0: 0>
    CR1 = <ControlRegisters.CR1: 1>
    CR2 = <ControlRegisters.CR2: 2>
    CR3 = <ControlRegisters.CR3: 3>
class ducky.cpu.coprocessor.control.CoreFlags
    Bases: ducky.util.Flags
exception ducky.cpu.coprocessor.control.ReadOnlyRegisterError (r, *args, **kwargs)
    Bases: ducky.cpu.CPUException
exception ducky.cpu.coprocessor.control.WriteOnlyRegisterError (r, *args, **kwargs)
    Bases: ducky.cpu.CPUException

```

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 = <MathCoproprocessorOpcodes.ADDL: 32>
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 = <MathCoproprocessorOpcodes.DECL: 31>
class ducky.cpu.coproprocessor.math_copro.DIVL(instruction_set)
    Bases: ducky.cpu.coproprocessor.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 = <MathCoproprocessorOpcodes.DIVL: 11>
class ducky.cpu.coproprocessor.math_copro.DROP(instruction_set)
    Bases: ducky.cpu.coproprocessor.math_copro.Descriptor_MATH
    static execute(core, inst)
    mnemonic = 'drop'
    opcode = <MathCoproprocessorOpcodes.DROP: 23>
class ducky.cpu.coproprocessor.math_copro.DUP(instruction_set)
    Bases: ducky.cpu.coproprocessor.math_copro.Descriptor_MATH
    static execute(core, inst)
    mnemonic = 'dup'
    opcode = <MathCoproprocessorOpcodes.DUP: 20>
class ducky.cpu.coproprocessor.math_copro.DUP2(instruction_set)
    Bases: ducky.cpu.coproprocessor.math_copro.Descriptor_MATH
    static execute(core, inst)
    mnemonic = 'dup2'
    opcode = <MathCoproprocessorOpcodes.DUP2: 21>
class ducky.cpu.coproprocessor.math_copro.Descriptor_MATH(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_R
    static assemble_operands(logger, buffer, inst, operands)
    static disassemble_operands(logger, inst)
    operands = ''
exception ducky.cpu.coproprocessor.math_copro.EmptyMathStackError(*args, **kwargs)
    Bases: ducky.cpu.CPUException
    Raised when operation expects at least one value on math stack but stack is empty.
exception ducky.cpu.coproprocessor.math_copro.FullMathStackError(*args, **kwargs)
    Bases: ducky.cpu.CPUException
    Raised when operation tries to put value on math stack but there is no empty spot available.
class ducky.cpu.coproprocessor.math_copro.INCL(instruction_set)
    Bases: ducky.cpu.coproprocessor.math_copro.Descriptor_MATH
    Increment top of the stack by one.
    static execute(core, inst)
    mnemonic = 'incl'
    opcode = <MathCoproprocessorOpcodes.INCL: 30>

```

```
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 (logger, buffer, inst, operands)

    static disassemble_operands (logger, inst)

    static execute (core, inst)

    mnemonic = 'load'

    opcode = <MathCoproprocessorOpcodes.LOAD: 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 (logger, buffer, inst, operands)

    static disassemble_operands (logger, inst)

    static execute (core, inst)

    mnemonic = 'loaduw'

    opcode = <MathCoproprocessorOpcodes.LOADUW: 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 (logger, buffer, inst, operands)

    static disassemble_operands (logger, inst)

    static execute (core, inst)

    mnemonic = 'loadw'

    opcode = <MathCoproprocessorOpcodes.LOADW: 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 = <MathCoproprocessorOpcodes.MODL: 12>

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)

    mnemonic = 'mull'

    opcode = <MathCoproprocessorOpcodes.MULL: 10>
```

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 at 0x7f05443da650>, <ducky.cpu.coprocessor.math_copro.POPW object at 0x7f05443da650>]

opcode_desc_map = {<MathCoproprocessorOpcodes.POPW: 0>: <ducky.cpu.coprocessor.math_copro.POPW object at 0x7f05443da650>, <MathCoproprocessorOpcodes.POPW: 1>: <ducky.cpu.coprocessor.math_copro.POPW object at 0x7f05443da650>}

opcode_encoding_map = {<MathCoproprocessorOpcodes.POPW: 0>: <class ‘ducky.cpu.instructions.EncodingR’>, <MathCoproprocessorOpcodes.POPW: 1>: <class ‘ducky.cpu.instructions.EncodingR’>}

opcodes

alias of `MathCoproprocessorOpcodes`

class `ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes`

Bases: `enum.IntEnum`

Math coprocessor’s instruction opcodes.

ADDL = <MathCoproprocessorOpcodes.ADDL: 32>

DECL = <MathCoproprocessorOpcodes.DECL: 31>

DIVL = <MathCoproprocessorOpcodes.DIVL: 11>

DROP = <MathCoproprocessorOpcodes.DROP: 23>

DUP = <MathCoproprocessorOpcodes.DUP: 20>

DUP2 = <MathCoproprocessorOpcodes.DUP2: 21>

INCL = <MathCoproprocessorOpcodes.INCL: 30>

LOAD = <MathCoproprocessorOpcodes.LOAD: 9>

LOADUW = <MathCoproprocessorOpcodes.LOADUW: 5>

LOADW = <MathCoproprocessorOpcodes.LOADW: 4>

MODL = <MathCoproprocessorOpcodes.MODL: 12>

MULL = <MathCoproprocessorOpcodes.MULL: 10>

POP = <MathCoproprocessorOpcodes.POP: 6>

POPUW = <MathCoproprocessorOpcodes.POPUW: 1>

POPW = <MathCoproprocessorOpcodes.POPW: 0>

PUSH = <MathCoproprocessorOpcodes.PUSH: 8>

PUSHW = <MathCoproprocessorOpcodes.PUSHW: 2>
SAVE = <MathCoproprocessorOpcodes.SAVE: 7>
SAVEW = <MathCoproprocessorOpcodes.SAVEW: 3>
SIS = <MathCoproprocessorOpcodes.SIS: 63>
SWP = <MathCoproprocessorOpcodes.SWP: 22>
SYMDIVL = <MathCoproprocessorOpcodes.SYMDIVL: 13>
SYMMODL = <MathCoproprocessorOpcodes.SYMMODL: 14>
UDIVL = <MathCoproprocessorOpcodes.UDIVL: 15>
UMODL = <MathCoproprocessorOpcodes.UMODL: 16>

class `ducky.cpu.coproprocessor.math_copro.MathCoproprocessorState`
Bases: `ducky.snapshot.SnapshotNode`

Snapshot node holding the state of math coprocessor.

class `ducky.cpu.coproprocessor.math_copro.POP(instruction_set)`
Bases: `ducky.cpu.coproprocessor.math_copro.Descriptor_MATH`

Pop the long from data stack, and make it new TOS.

static execute (*core, inst*)

mnemonic = 'pop'

opcode = <MathCoproprocessorOpcodes.POP: 6>

class `ducky.cpu.coproprocessor.math_copro.POPUW(instruction_set)`
Bases: `ducky.cpu.coproprocessor.math_copro.Descriptor_MATH`

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

static execute (*core, inst*)

mnemonic = 'popuw'

opcode = <MathCoproprocessorOpcodes.POPUW: 1>

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

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

static execute (*core, inst*)

mnemonic = 'popw'

opcode = <MathCoproprocessorOpcodes.POPW: 0>

class `ducky.cpu.coproprocessor.math_copro.PUSH(instruction_set)`
Bases: `ducky.cpu.coproprocessor.math_copro.Descriptor_MATH`

Push the TOS on the data stack.

static execute (*core, inst*)

mnemonic = 'push'

opcode = <MathCoproprocessorOpcodes.PUSH: 8>

```

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 = <MathCoproprocessorOpcodes.PUSHW: 2>

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 (logger, buffer, inst, operands)

    static disassemble_operands (logger, inst)

    static execute (core, inst)

    mnemonic = 'save'

    opcode = <MathCoproprocessorOpcodes.SAVE: 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 (logger, buffer, inst, operands)
static disassemble_operands (logger, inst)
static execute (core, inst)
mnemonic = 'savew'
opcode = <MathCoproprocessorOpcodes.SAVEW: 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 = 'swp'
    opcode = <MathCoproprocessorOpcodes.SWP: 22>

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 = <MathCoproprocessorOpcodes.SYMDIVL: 13>

class ducky.cpu.coprocessor.math_copro.SYMMODL (instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH
    static execute (core, inst)
    mnemonic = 'symmodl'
    opcode = <MathCoproprocessorOpcodes.SYMMODL: 14>

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 = <MathCoproprocessorOpcodes.UDIVL: 15>

class ducky.cpu.coprocessor.math_copro.UMODL (instruction_set)
    Bases: ducky.cpu.coprocessor.math_copro.Descriptor_MATH
    static execute (core, inst)
    mnemonic = 'umodl'
    opcode = <MathCoproprocessorOpcodes.UMODL: 16>
```

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.
```

2.5.2 Submodules

ducky.cpu.assemble module

```
class ducky.cpu.assemble.AlignSlot (boundary)
    Bases: ducky.cpu.assemble.DataSlot

class ducky.cpu.assemble.AsciiSlot
    Bases: ducky.cpu.assemble.DataSlot

    close ()

    symbol_type = <SymbolDataTypes.ASCII: 5>

class ducky.cpu.assemble.BssSection (s_name, flags=None, **kwargs)
    Bases: ducky.cpu.assemble.Section

class ducky.cpu.assemble.Buffer (logger, filename, buff)
    Bases: object

    get_error (cls, info, column=None, length=None, **kwargs)

    get_line ()

    has_lines ()

    put_buffer (buff, filename=None)

    put_line (line)

class ducky.cpu.assemble.ByteSlot
    Bases: ducky.cpu.assemble.DataSlot

    close ()

    symbol_type = <SymbolDataTypes.CHAR: 2>

class ducky.cpu.assemble.BytesSlot
    Bases: ducky.cpu.assemble.DataSlot

    close ()

    symbol_type = <SymbolDataTypes.ASCII: 5>

class ducky.cpu.assemble.CharSlot
    Bases: ducky.cpu.assemble.DataSlot

    close ()

    symbol_type = <SymbolDataTypes.CHAR: 2>

class ducky.cpu.assemble.DataSection (s_name, flags=None, **kwargs)
    Bases: ducky.cpu.assemble.Section
```

```

class ducky.cpu.assemble.DataSlot
    Bases: object

    close()

class ducky.cpu.assemble.FunctionSlot
    Bases: ducky.cpu.assemble.DataSlot

    close()

    symbol_type = <SymbolDataTypes.FUNCTION: 6>

class ducky.cpu.assemble.IntSlot
    Bases: ducky.cpu.assemble.DataSlot

    close()

    symbol_type = <SymbolDataTypes.INT: 0>

class ducky.cpu.assemble.Label (name, section, location)
    Bases: object

ducky.cpu.assemble.PATTERN (pattern)

class ducky.cpu.assemble.RODataSection (s_name, flags=None, **kwargs)
    Bases: ducky.cpu.assemble.Section

class ducky.cpu.assemble.Reference (add=None, label=None)
    Bases: object

class ducky.cpu.assemble.RelocSection (s_name, flags=None, **kwargs)
    Bases: ducky.cpu.assemble.Section

class ducky.cpu.assemble.RelocSlot (name, flags=None, patch_section=None,
    patch_address=None, patch_offset=None, patch_size=None,
    patch_add=None)

    Bases: object

class ducky.cpu.assemble.Section (s_name, s_type, s_flags)
    Bases: object

class ducky.cpu.assemble.ShortSlot
    Bases: ducky.cpu.assemble.DataSlot

    close()

    symbol_type = <SymbolDataTypes.SHORT: 1>

class ducky.cpu.assemble.SourceLocation (filename=None, lineno=None, column=None,
    length=None)

    Bases: object

    copy()

class ducky.cpu.assemble.SpaceSlot
    Bases: ducky.cpu.assemble.DataSlot

    close()

    symbol_type = <SymbolDataTypes.ASCII: 5>

class ducky.cpu.assemble.StringSlot
    Bases: ducky.cpu.assemble.DataSlot

    close()

    symbol_type = <SymbolDataTypes.STRING: 4>

```



```

class ducky.cpu.assemble.SymbolsSection (s_name, flags=None, **kwargs)
    Bases: ducky.cpu.assemble.Section

class ducky.cpu.assemble.TextSection (s_name, flags=None, **kwargs)
    Bases: ducky.cpu.assemble.Section

ducky.cpu.assemble.decode_string (s)

ducky.cpu.assemble.sizeof (o)

ducky.cpu.assemble.translate_buffer (logger, buff, base_address=None, mma-
                                     pable_sections=False, writable_sections=False,
                                     filename=None, defines=None, includes=None, ver-
                                     ify_disassemble=False)

```

ducky.cpu.instructions module

```

class ducky.cpu.instructions.ADD (instruction_set)
    Bases: ducky.cpu.instructions._BINOP
    static jit (core, inst)
    mnemonic = 'add'
    opcode = <DuckyOpcodes.ADD: 28>

class ducky.cpu.instructions.AND (instruction_set)
    Bases: ducky.cpu.instructions._BITOP
    static jit (core, inst)
    mnemonic = 'and'
    opcode = <DuckyOpcodes.AND: 34>

class ducky.cpu.instructions.BE (instruction_set)
    Bases: ducky.cpu.instructions._BRANCH
    mnemonic = 'be'

class ducky.cpu.instructions.BG (instruction_set)
    Bases: ducky.cpu.instructions._BRANCH
    mnemonic = 'bg'

class ducky.cpu.instructions.BGE (instruction_set)
    Bases: ducky.cpu.instructions._BRANCH
    mnemonic = 'bge'

class ducky.cpu.instructions.BL (instruction_set)
    Bases: ducky.cpu.instructions._BRANCH
    mnemonic = 'bl'

class ducky.cpu.instructions.BLE (instruction_set)
    Bases: ducky.cpu.instructions._BRANCH
    mnemonic = 'ble'

class ducky.cpu.instructions.BNE (instruction_set)
    Bases: ducky.cpu.instructions._BRANCH
    mnemonic = 'bne'

```

```

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

    mnemonic = 'bno'

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

    mnemonic = 'bns'

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

    mnemonic = 'bnz'

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

    mnemonic = 'bo'

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

    mnemonic = 'bs'

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

    mnemonic = 'bz'

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

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'call'

    opcode = <DuckyOpcodes.CALL: 15>

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

    static assemble_operands (logger, buffer, inst, operands)

    static disassemble_operands (logger, inst)

    encoding
        alias of EncodingA

    static execute (core, inst)

    mnemonic = 'cas'

    opcode = <DuckyOpcodes.CAS: 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)

    mnemonic = 'cli'

```

```

        opcode = <DuckyOpcodes.CLI: 17>
class ducky.cpu.instructions.CMP (instruction_set)
    Bases: ducky.cpu.instructions._CMP
    static execute (core, inst)
    static jit (core, inst)
    mnemonic = 'cmp'
    opcode = <DuckyOpcodes.CMP: 47>
class ducky.cpu.instructions.CMPU (instruction_set)
    Bases: ducky.cpu.instructions._CMP
    static execute (core, inst)
    mnemonic = 'cmпу'
    opcode = <DuckyOpcodes.CMPU: 48>
class ducky.cpu.instructions.CTR (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_R
    encoding
        alias of EncodingR
    static execute (core, inst)
    mnemonic = 'ctr'
    opcode = <DuckyOpcodes.CTR: 60>
class ducky.cpu.instructions.CTW (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R_R
    encoding
        alias of EncodingR
    static execute (core, inst)
    mnemonic = 'ctw'
    opcode = <DuckyOpcodes.CTW: 61>
class ducky.cpu.instructions.DEC (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_R
    static execute (core, inst)
    static jit (core, inst)
    mnemonic = 'dec'
    opcode = <DuckyOpcodes.DEC: 27>
class ducky.cpu.instructions.DIV (instruction_set)
    Bases: ducky.cpu.instructions._BINOP
    mnemonic = 'div'
    opcode = <DuckyOpcodes.DIV: 31>
class ducky.cpu.instructions.Descriptor (instruction_set)
    Bases: object
    static assemble_operands (logger, buffer, inst, operands)

```

```
classmethod disassemble_mnemonic (inst)
static disassemble_operands (logger, inst)
emit_instruction (logger, buffer, line)
encoding
    alias of EncodingR
static execute (core, inst)
static fill_reloc_slot (logger, inst, slot)
inst_aligned = False
static jit (core, inst)
mnemonic = None
opcode = None
operands = None
pattern = None
relative_address = False

class ducky.cpu.instructions.Descriptor_I (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor
    static assemble_operands (logger, buffer, inst, operands)
    static disassemble_operands (logger, inst)
    operands = 'i'

class ducky.cpu.instructions.Descriptor_R (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor
    static assemble_operands (logger, buffer, 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 (logger, buffer, inst, operands)
    static disassemble_operands (logger, inst)
    encoding
        alias of EncodingI
    operands = 'ri'

class ducky.cpu.instructions.Descriptor_RI_R (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor
    static assemble_operands (logger, buffer, inst, operands)
    static disassemble_operands (logger, inst)
    encoding
        alias of EncodingR
```


CMPU = <DuckyOpcodes.CMPU: 48>
CTR = <DuckyOpcodes.CTR: 60>
CTW = <DuckyOpcodes.CTW: 61>
DEC = <DuckyOpcodes.DEC: 27>
DIV = <DuckyOpcodes.DIV: 31>
FPTC = <DuckyOpcodes.FPTC: 62>
HLT = <DuckyOpcodes.HLT: 20>
IDLE = <DuckyOpcodes.IDLE: 21>
INB = <DuckyOpcodes.INB: 45>
INC = <DuckyOpcodes.INC: 26>
INS = <DuckyOpcodes.INS: 44>
INT = <DuckyOpcodes.INT: 13>
INW = <DuckyOpcodes.INW: 43>
IPI = <DuckyOpcodes.IPI: 23>
J = <DuckyOpcodes.J: 46>
LA = <DuckyOpcodes.LA: 8>
LB = <DuckyOpcodes.LB: 3>
LI = <DuckyOpcodes.LI: 9>
LIU = <DuckyOpcodes.LIU: 10>
LPM = <DuckyOpcodes.LPM: 22>
LS = <DuckyOpcodes.LS: 2>
LW = <DuckyOpcodes.LW: 1>
MOD = <DuckyOpcodes.MOD: 33>
MOV = <DuckyOpcodes.MOV: 11>
MUL = <DuckyOpcodes.MUL: 30>
NOP = <DuckyOpcodes.NOP: 0>
NOT = <DuckyOpcodes.NOT: 37>
OR = <DuckyOpcodes.OR: 35>
OUTB = <DuckyOpcodes.OUTB: 42>
OUTS = <DuckyOpcodes.OUTS: 41>
OUTW = <DuckyOpcodes.OUTW: 40>
POP = <DuckyOpcodes.POP: 25>
PUSH = <DuckyOpcodes.PUSH: 24>
RET = <DuckyOpcodes.RET: 16>
RETINT = <DuckyOpcodes.RETINT: 14>
RST = <DuckyOpcodes.RST: 19>

```

SET = <DuckyOpcodes.SET: 49>
SHIFTL = <DuckyOpcodes.SHIFTL: 38>
SHIFTR = <DuckyOpcodes.SHIFTR: 39>
SIS = <DuckyOpcodes.SIS: 63>
STB = <DuckyOpcodes.STB: 6>
STI = <DuckyOpcodes.STI: 18>
STS = <DuckyOpcodes.STS: 5>
STW = <DuckyOpcodes.STW: 4>
SUB = <DuckyOpcodes.SUB: 29>
SWP = <DuckyOpcodes.SWP: 12>
UDIV = <DuckyOpcodes.UDIV: 32>
XOR = <DuckyOpcodes.XOR: 36>

ducky.cpu.instructions.ENCORE(logger, buffer, inst, field, size, value,
                               raise_on_large_value=False)

class ducky.cpu.instructions.Encoding
    Bases: _ctypes.Structure

    static sign_extend_immediate(logger, inst, sign_mask, ext_mask)

class ducky.cpu.instructions.EncodingA
    Bases: _ctypes.Structure

    opcode
        Structure/Union member

    reg1
        Structure/Union member

    reg2
        Structure/Union member

    reg3
        Structure/Union member

class ducky.cpu.instructions.EncodingC
    Bases: _ctypes.Structure

    static fill_reloc_slot(logger, inst, slot)

    flag
        Structure/Union member

    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.EncodingI
    Bases: _ctypes.Structure
    static fill_reloc_slot (logger, 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
    static fill_reloc_slot (logger, 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.FPTC (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor
    encoding
        alias of EncodingI
    static execute (core, inst)
    mnemonic = 'fptc'
    opcode = <DuckyOpcodes.FPTC: 62>
class ducky.cpu.instructions.HLT (instruction_set)
    Bases: ducky.cpu.instructions.Descriptor_RI
    static execute (core, inst)
    mnemonic = 'hlt'
    opcode = <DuckyOpcodes.HLT: 20>
```



```

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

    encoding
        alias of EncodingI

    static execute(core, inst)

    mnemonic = 'idle'

    opcode = <DuckyOpcodes.IDLE: 21>

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.INB(instruction_set)
    Bases: ducky.cpu.instructions._IN

    mnemonic = 'inb'

    opcode = <DuckyOpcodes.INB: 45>

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

    static execute(core, inst)

    static jit(core, inst)

    mnemonic = 'inc'

    opcode = <DuckyOpcodes.INC: 26>

class ducky.cpu.instructions.INS(instruction_set)
    Bases: ducky.cpu.instructions._IN

    mnemonic = 'ins'

    opcode = <DuckyOpcodes.INS: 44>

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

    static execute(core, inst)

    mnemonic = 'int'

    opcode = <DuckyOpcodes.INT: 13>

class ducky.cpu.instructions.INW(instruction_set)
    Bases: ducky.cpu.instructions._IN

    mnemonic = 'inw'

    opcode = <DuckyOpcodes.INW: 43>

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

    static execute(core, inst)

    mnemonic = 'ipi'

    opcode = <DuckyOpcodes.IPI: 23>

```

```
class ducky.cpu.instructions.InstructionSet
    Bases: object

    classmethod decode_instruction (logger, inst, core=None)

    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 = <DuckyOpcodes.J: 46>

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

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

    static jit (core, inst)

    classmethod load (core, inst)

    mnemonic = 'la'

    opcode = <DuckyOpcodes.LA: 8>

    relative_address = True

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

    mnemonic = 'lb'

    opcode = <DuckyOpcodes.LB: 3>

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

    static jit (core, inst)

    classmethod load (core, inst)

    mnemonic = 'li'

    opcode = <DuckyOpcodes.LI: 9>

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

    classmethod load (core, inst)

    mnemonic = 'liu'
```

```

        opcode = <DuckyOpcodes.LIU: 10>

class ducky.cpu.instructions.LPM(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor
    encoding
        alias of EncodingI
    static execute (core, inst)
    mnemonic = 'lpm'
    opcode = <DuckyOpcodes.LPM: 22>

class ducky.cpu.instructions.LS(instruction_set)
    Bases: ducky.cpu.instructions._LOAD
    mnemonic = 'ls'
    opcode = <DuckyOpcodes.LS: 2>

class ducky.cpu.instructions.LW(instruction_set)
    Bases: ducky.cpu.instructions._LOAD
    mnemonic = 'lw'
    opcode = <DuckyOpcodes.LW: 1>

class ducky.cpu.instructions.MOD(instruction_set)
    Bases: ducky.cpu.instructions._BINOP
    mnemonic = 'mod'
    opcode = <DuckyOpcodes.MOD: 33>

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 = <DuckyOpcodes.MOV: 11>

class ducky.cpu.instructions.MUL(instruction_set)
    Bases: ducky.cpu.instructions._BINOP
    static jit (core, inst)
    mnemonic = 'mul'
    opcode = <DuckyOpcodes.MUL: 30>

class ducky.cpu.instructions.NOP(instruction_set)
    Bases: ducky.cpu.instructions.Descriptor
    encoding
        alias of EncodingI
    static execute (core, inst)
    mnemonic = 'nop'

```

```
        opcode = <DuckyOpcodes.NOP: 0>

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

    encoding
        alias of EncodingR

    static execute (core, inst)

    mnemonic = 'not'

    opcode = <DuckyOpcodes.NOT: 37>

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

    static jit (core, inst)

    mnemonic = 'or'

    opcode = <DuckyOpcodes.OR: 35>

class ducky.cpu.instructions.OUTB (instruction_set)
    Bases: ducky.cpu.instructions._OUT

    mnemonic = 'outb'

    opcode = <DuckyOpcodes.OUTB: 42>

class ducky.cpu.instructions.OUTS (instruction_set)
    Bases: ducky.cpu.instructions._OUT

    mnemonic = 'outs'

    opcode = <DuckyOpcodes.OUTS: 41>

class ducky.cpu.instructions.OUTW (instruction_set)
    Bases: ducky.cpu.instructions._OUT

    mnemonic = 'outw'

    opcode = <DuckyOpcodes.OUTW: 40>

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

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'pop'

    opcode = <DuckyOpcodes.POP: 25>

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

    static execute (core, inst)

    static jit (core, inst)

    mnemonic = 'push'

    opcode = <DuckyOpcodes.PUSH: 24>

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 = <DuckyOpcodes.RET: 16>

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

    encoding
        alias of EncodingI

    static execute (core, inst)

    mnemonic = 'retint'

    opcode = <DuckyOpcodes.RETINT: 14>

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 = <DuckyOpcodes.RST: 19>

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

    mnemonic = 'sete'

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

    mnemonic = 'setg'

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

    mnemonic = 'setge'

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

    mnemonic = 'setl'

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

    mnemonic = 'setle'

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

    mnemonic = 'setne'

```

```

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

    mnemonic = 'setno'

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

    mnemonic = 'setns'

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

    mnemonic = 'setnz'

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

    mnemonic = 'seto'

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

    mnemonic = 'sets'

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

    mnemonic = 'setz'

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

    static jit (core, inst)

    mnemonic = 'shiftl'

    opcode = <DuckyOpcodes.SHIFTL: 38>

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

    static jit (core, inst)

    mnemonic = 'shiftr'

    opcode = <DuckyOpcodes.SHIFTR: 39>

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

    static execute (core, inst)

    mnemonic = 'sis'

    opcode = <DuckyOpcodes.SIS: 63>

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

    mnemonic = 'stb'

    opcode = <DuckyOpcodes.STB: 6>

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

```

```

    encoding
        alias of EncodingI

    static execute (core, inst)

    mnemonic = 'sti'

    opcode = <DuckyOpcodes.STI: 18>

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

    mnemonic = 'sts'

    opcode = <DuckyOpcodes.STS: 5>

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

    mnemonic = 'stw'

    opcode = <DuckyOpcodes.STW: 4>

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

    static jit (core, inst)

    mnemonic = 'sub'

    opcode = <DuckyOpcodes.SUB: 29>

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 = <DuckyOpcodes.SWP: 12>

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

    mnemonic = 'udiv'

    opcode = <DuckyOpcodes.UDIV: 32>

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 = <DuckyOpcodes.XOR: 36>

ducky.cpu.instructions.encoding_to_u32 (inst)

ducky.cpu.instructions.get_instruction_set (i, exc=None)

ducky.cpu.instructions.u32_to_encoding (u, encoding)

```

`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: `object`

class `ducky.cpu.registers.Registers`

Bases: `enum.IntEnum`

CNT = `<Registers.CNT: 33>`

FP = `<Registers.FP: 30>`

IP = `<Registers.IP: 32>`

R00 = `<Registers.R00: 0>`

R01 = `<Registers.R01: 1>`

R02 = `<Registers.R02: 2>`

R03 = `<Registers.R03: 3>`

R04 = `<Registers.R04: 4>`

R05 = `<Registers.R05: 5>`

R06 = `<Registers.R06: 6>`

R07 = `<Registers.R07: 7>`

R08 = `<Registers.R08: 8>`

R09 = `<Registers.R09: 9>`

R10 = `<Registers.R10: 10>`

R11 = `<Registers.R11: 11>`

R12 = `<Registers.R12: 12>`

R13 = `<Registers.R13: 13>`

R14 = `<Registers.R14: 14>`

R15 = `<Registers.R15: 15>`

R16 = `<Registers.R16: 16>`

R17 = `<Registers.R17: 17>`

R18 = `<Registers.R18: 18>`

R19 = `<Registers.R19: 19>`

R20 = `<Registers.R20: 20>`

R21 = `<Registers.R21: 21>`

R22 = `<Registers.R22: 22>`


```

R23 = <Registers.R23: 23>
R24 = <Registers.R24: 24>
R25 = <Registers.R25: 25>
R26 = <Registers.R26: 26>
R27 = <Registers.R27: 27>
R28 = <Registers.R28: 28>
R29 = <Registers.R29: 29>
REGISTER_COUNT = <Registers.REGISTER_COUNT: 34>
REGISTER_SPECIAL = <Registers.FP: 30>
SP = <Registers.SP: 31>

```

2.5.3 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 ()

backtrace ()

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 `AccessViolationError` – if the core is not in privileged mode

check_protected_port (*port*)

create_frame ()

Create new call stack frame. Push `IP` and `FP` registers and set `FP` value to `SP`.

destroy_frame ()

Destroy current call frame. Pop `FP` and `IP` from stack, by popping `FP` restores previous frame.

Raises `CPUException` – if current frame does not match last created frame.

die (*exc*)

do_int (*index*)

Handle software interrupt. Real software interrupts cause CPU state to be saved and new stack and register values are prepared by `__enter_interrupt` method, virtual interrupts are simply triggered without any prior changes of CPU state.

Parameters `index` (*int*) – interrupt number

exit_interrupt ()

Restore CPU state after running a interrupt routine. Call frame is destroyed, registers are restored, stack is returned back to memory pool.

flags

halt ()

has_coprocessor (*name*)

init_debug_set ()

irq (*index*)

load_state (*state*)

pop (**regs*)

push (**regs*)

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

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`

exception `ducky.cpu.CPUException` (`msg`, `core=None`, `ip=None`)

Bases: `exceptions.Exception`

Base class for CPU-related exceptions.

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.

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`

`ducky.cpu.DEFAULT_CORE_INST_CACHE_SIZE = 256`

Default size of core instruction cache, in instructions.

`ducky.cpu.DEFAULT_IVT_ADDRESS = 0`

Default IVT address

`ducky.cpu.DEFAULT_PT_ADDRESS = 65536`

Default PT address

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

Bases: `ducky.util.LRUCache`

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

Parameters

- **core** (`ducky.cpu.CPUCore`) – CPU core that owns this cache.
- **size** (`int`) – maximal number of entries this cache can store.

get_object (*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_object_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`

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

Bases: `object`

Interrupt vector table entry.

SIZE = 8

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

Bases: `ducky.cpu.CPUException`

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

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

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

Bases: `ducky.cpu.CPUException`

Raised when unknown or invalid opcode is found in instruction.

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

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.

check_access (*access, addr, align=None*)

Check attempted access against PTE. Be aware that each check can be turned off by configuration file.

Parameters

- **access** – read, write or execute.
- **addr** (*u24*) – memory address.
- **align** (*int*) – if set, operation is expected to be aligned to this boundary.

Raises `ducky.errors.AccessViolationError` – when access is denied.

full_read_u16 (*addr*)

full_read_u32 (*addr*, *not_execute=True*)

full_read_u8 (*addr*)

full_write_u16 (*addr*, *value*)

full_write_u32 (*addr*, *value*)

full_write_u8 (*addr*, *value*)

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** (u24) – memory address.

halt ()

release_ptes ()

reset ()

set_access_methods ()

Set parent core's memory-access methods to proper shortcuts.

class `ducky.cpu.StackFrame` (*fp*)

Bases: `object`

`ducky.cpu.cmd_bt` (*console*, *cmd*)

Print current backtrace

`ducky.cpu.cmd_cont` (*console*, *cmd*)

Continue execution until next breakpoint is reached: `cont`

`ducky.cpu.cmd_core_state` (*console*, *cmd*)

Print core state

`ducky.cpu.cmd_next` (*console*, *cmd*)

Proceed to the next instruction in the same stack frame.

`ducky.cpu.cmd_set_core` (*console*, *cmd*)

Set core address of default core used by control commands: `sc <coreid>`

`ducky.cpu.cmd_step` (*console*, *cmd*)

Step one instruction forward

`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”.

2.6 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`

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`)

class `ducky.debugging.VMDebugInterrupt` (`machine`)

Bases: `ducky.interfaces.IVirtualInterrupt`

run (`core`)

class `ducky.debugging.VMDebugOperationList`

Bases: `enum.Enum`

LOGGER_VERBOSITY = `<VMDebugOperationList.LOGGER_VERBOSITY: 0>`

class `ducky.debugging.VMVerbosityLevels`

Bases: `enum.Enum`

DEBUG = `<VMVerbosityLevels.DEBUG: 0>`

ERROR = `<VMVerbosityLevels.ERROR: 3>`

INFO = `<VMVerbosityLevels.INFO: 1>`

WARNING = `<VMVerbosityLevels.WARNING: 2>`

`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>

2.7 ducky.devices package

2.7.1 Submodules

ducky.devices.keyboard module

Keyboard controller - provides events for pressed and released keys.

```
class ducky.devices.keyboard.Backend(machine, name, port=None, irq=None)
    Bases: ducky.devices.IRQProvider, ducky.devices.IOProvider,
           ducky.devices.DeviceBackend

    boot()

    static create_from_config(machine, config, section)

    halt()

    read_u8(port)

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

    CONTROL_MESSAGE_FIRST = <ControlMessages.CONTROL_MESSAGE_FIRST: 1024>
    HALT = <ControlMessages.HALT: 1025>

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

    boot()

    static create_from_config(machine, config, section)

    enqueue_stream(stream)

    halt()
```

ducky.devices.rtc module

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

    boot()

    static create_from_config(machine, config, section)

    halt()

    read_u8(port)

    write_u8(port, value)

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. By default, 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.

Each block has its own id. Block IO operations read or write one or more blocks to or from a device. IO is requested by invoking the virtual interrupt, with properly set values in registers.

```
ducky.devices.storage.BLOCK_SIZE = 1024  
    Size of block, in bytes.
```

```
class ducky.devices.storage.BlockIO (machine, name, port=None, irq=None, *args, **kwargs)  
    Bases: ducky.devices.IRQProvider, ducky.devices.IOProvider,  
         ducky.devices.Device  
    boot ()  
    buff_to_memory (addr, buff)  
    static create_from_config (machine, config, section)  
    halt ()  
    memory_to_buff (addr, length)  
    read_u32 (port)  
    reset ()  
    write_u32 (port, value)
```

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.

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`

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, port=None, *args, **kwargs*)

Bases: `ducky.devices.IOProvider`, `ducky.devices.DeviceBackend`

boot ()

static create_from_config (*machine, config, section*)

halt ()

tenh (*s, *args*)

tenh_close_stream ()

tenh_enable ()

tenh_flush_stream ()

write_u8 (*port, value*)

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*)

tenh_enable ()

class `ducky.devices.tty.FrontendFlushTask` (*frontend, queue, stream*)

Bases: `ducky.interfaces.IReactorTask`

run ()

set_output (*stream*)

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`

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_MODES = [(g, 320, 200, 1), (t, 80, 25, 2), (t, 80, 25, 1)]
    Default list of available modes

ducky.devices.svga.DEFAULT_PORT_RANGE = 1008
    Default address of command port

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, port=None, memory_size=None, memory_address=None, memory_banks=None, modes=None, boot_mode=None, *args, **kwargs)
    Bases: ducky.devices.IOProvider, 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.
- **port** (*u16*) – address of the command port.
- **memory_size** (*int*) – size of graphic memory.
- **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 ()

read_u16 (*port*)

reset ()

set_mode (*mode*)

write_u16 (*port, value*)

class `ducky.devices.svga.SimpleVGACommands`

Bases: `enum.IntEnum`

COLS = <SimpleVGACommands.COLS: 33>

DEPTH = <SimpleVGACommands.DEPTH: 35>

GRAPHIC = <SimpleVGACommands.GRAPHIC: 32>

MEMORY_BANK_ID = <SimpleVGACommands.MEMORY_BANK_ID: 48>

REFRESH = <SimpleVGACommands.REFRESH: 2>

RESET = <SimpleVGACommands.RESET: 32769>

ROWS = <SimpleVGACommands.ROWS: 34>

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*)

2.7.2 Module contents

```
class ducky.devices.Device (machine, klass, name)
    Bases: ducky.interfaces.IMachineWorker

    boot ()

    static create_from_config (machine, config, section)

    get_master ()

    halt ()

    is_slave ()

class ducky.devices.DeviceBackend (machine, klass, name)
    Bases: ducky.devices.Device

    set_frontend (device)

class ducky.devices.DeviceFrontend (machine, klass, name)
    Bases: ducky.devices.Device

    set_backend (device)

class ducky.devices.IOPorts
    Bases: enum.IntEnum

    PORT_COUNT = <IOPorts.PORT_COUNT: 65536>

class ducky.devices.IOPProvider
    Bases: object

    is_port_protected (port, read=True)

    read_u16 (port)

    read_u32 (port)

    read_u8 (port)

    write_u16 (port, value)

    write_u32 (port, value)

    write_u8 (port, value)

class ducky.devices.IRQList
    Bases: enum.IntEnum

    List of known IRQ sources.

    BIO = <IRQList.BIO: 2>

    BLOCKIO = <IRQList.BLOCKIO: 17>

    HALT = <IRQList.HALT: 16>

    IRQ_COUNT = <IRQList.IRQ_COUNT: 32>

    KEYBOARD = <IRQList.KEYBOARD: 1>

    TIMER = <IRQList.TIMER: 0>

    VMDEBUG = <IRQList.VMDEBUG: 18>

class ducky.devices.IRQProvider
    Bases: object
```


`ducky.devices.get_driver_creator(driver_class)`

2.8 ducky.errors module

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

Bases: `ducky.errors.Error`

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

Bases: `ducky.errors.Error`

create_text ()

log (*logger*)

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

Bases: `ducky.errors.AssemblerError`

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

Bases: `ducky.errors.AssemblerError`

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

Bases: `exceptions.Exception`

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

Bases: `ducky.errors.Error`

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

Bases: `ducky.errors.AssemblerError`

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

Bases: `ducky.errors.Error`

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

Bases: `ducky.errors.AssemblerError`

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

Bases: `ducky.errors.AssemblerError`

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

Bases: `ducky.errors.AssemblerError`

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

Bases: `ducky.errors.AssemblerError`

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

Bases: `ducky.errors.Error`

2.9 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.

klasses = [<class 'ducky.hdt.HDTEntry_Memory'>, <class 'ducky.hdt.HDTEntry_CPU'>, <class 'ducky.hdt.HDTEntry

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.

classmethod **create** (*args, **kwargs)

class `ducky.hdt.HDTEntryTypes`

Bases: `enum.IntEnum`

Types of different HDT entries.

ARGUMENT = <HDTEntryTypes.ARGUMENT: 3>

CPU = <HDTEntryTypes.CPU: 1>

MEMORY = <HDTEntryTypes.MEMORY: 2>

UNDEFINED = <HDTEntryTypes.UNDEFINED: 0>

class `ducky.hdt.HDTEntry_Argument` (*arg_name, arg_type, arg_value*)

Bases: `ducky.hdt.HDTEntry`

MAX_NAME_LENGTH = 13

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.

length
Structure/Union member

nr_cores
Structure/Union member

nr_cpus
Structure/Union member

type
Structure/Union member

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

Bases: `ducky.hdt.HDTEEntry`

HDT entry describing memory configuration.

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

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.

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.

`ducky.hdt.HDT_MAGIC = 1298034544`

Magic number present in HDT header

2.10 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*)

2.11 ducky.log module

`ducky.log.BLUE` (*s*)

`ducky.log.GREEN` (*s*)

class `ducky.log.LogFormatter` (*fmt=None, datefmt=None*)

Bases: `logging.Formatter`

format (*record*)

`ducky.log.RED` (*s*)

```

class ducky.log.StreamHandler(*args, **kwargs)
    Bases: logging.StreamHandler

ducky.log.WHITE(s)

ducky.log.create_logger(name=None, handler=None)

```

2.12 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.

    boot ()

    capture_state (suspend=False)
        Capture current state of the VM, and store it in it's 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.

        • klass (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.

register_port (port, handler)

run ()

save_state (parent)

setup_devices ()

suspend ()

tenh (s, *args)

trigger_irq (handler)

unregister_port (port)

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

```

2.13 ducky.mm package

2.13.1 Submodules

ducky.mm.binary module

```

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

    MAGIC = 57005

    VERSION = 1

    create_header ()

    create_section ()

    get_header ()

    get_section (i)

    get_section_by_name (name)

    load ()

```

```

    load_symbols ()
    static open (*args, **kwargs)
    save ()
    sections ()
    set_content (header, content)
    setup ()
class ducky.mm.binary.FileFlags
    Bases: ducky.util.Flags
    field = ('mmapable', <class 'ctypes.c_ushort'>, 1)
class ducky.mm.binary.FileFlagsEncoding
    Bases: _ctypes.Structure
    mmapable
        Structure/Union member
class ducky.mm.binary.FileHeader
    Bases: _ctypes.Structure
    flags
        Structure/Union member
    magic
        Structure/Union member
    sections
        Structure/Union member
    version
        Structure/Union member
class ducky.mm.binary.RelocEntry
    Bases: _ctypes.Structure
    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
    field = ('inst_aligned', <class 'ctypes.c_ushort'>, 1)

```



```

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

    inst_aligned
        Structure/Union member

    relative
        Structure/Union member

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

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

class ducky.mm.binary.SectionFlagsEncoding
    Bases: ctypes.Structure

    bss
        Structure/Union member

    executable
        Structure/Union member

    globally_visible
        Structure/Union member

    loadable
        Structure/Union member

    mmapable
        Structure/Union member

    readable
        Structure/Union member

    writable
        Structure/Union member

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

    base
        Structure/Union member

    data_size
        Structure/Union member

    file_size
        Structure/Union member

    flags
        Structure/Union member

    index
        Structure/Union member

    items
        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`

DATA = `<SectionTypes.DATA: 2>`

RELOC = `<SectionTypes.RELOC: 5>`

STRINGS = `<SectionTypes.STRINGS: 4>`

SYMBOLS = `<SectionTypes.SYMBOLS: 3>`

TEXT = `<SectionTypes.TEXT: 1>`

UNKNOWN = `<SectionTypes.UNKNOWN: 0>`

class `ducky.mm.binary.SymbolDataTypes`

Bases: `enum.IntEnum`

ASCII = `<SymbolDataTypes.ASCII: 5>`

BYTE = `<SymbolDataTypes.BYTE: 3>`

CHAR = `<SymbolDataTypes.CHAR: 2>`

FUNCTION = `<SymbolDataTypes.FUNCTION: 6>`

INT = `<SymbolDataTypes.INT: 0>`

SHORT = `<SymbolDataTypes.SHORT: 1>`

STRING = `<SymbolDataTypes.STRING: 4>`

UNKNOWN = `<SymbolDataTypes.UNKNOWN: 7>`

class `ducky.mm.binary.SymbolEntry`

Bases: `_ctypes.Structure`

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

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

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

    globally_visible
        Structure/Union member
```

2.13.2 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 = `<MMOperationList.ALLOC: 3>`

FREE = `<MMOperationList.FREE: 4>`

MMAP = `<MMOperationList.MMAP: 6>`

UNMMAP = `<MMOperationList.UNMMAP: 7>`

UNUSED = `<MMOperationList.UNUSED: 5>`

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`.

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
<code>read</code>	page is readable by executed instructions	<code>False</code>
<code>write</code>	page is writable by executed instructions	<code>False</code>
<code>execute</code>	content of the page can be used as executable instructions	<code>False</code>
<code>dirty</code>	there have been write access to this page, its content has changed	<code>False</code>

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

ducky.mm.SIZE_FMT (size)

ducky.mm.addr_to_offset (addr)

ducky.mm.addr_to_page (addr)

ducky.mm.area_to_pages (addr, size)
```

2.14 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)

```

2.15 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

2.16 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`

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 oportunities, 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()

2.17 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*)

2.18 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 differences.

class `ducky.streams.FDInputStream`(*machine, fd, **kwargs*)

Bases: `ducky.streams.InputStream`

class `ducky.streams.FDOutputStream`(*machine, fd, **kwargs*)

Bases: `ducky.streams.OutputStream`

class `ducky.streams.FileInputStream`(*machine, f, **kwargs*)

Bases: `ducky.streams.InputStream`

class `ducky.streams.FileOutputStream`(*machine, f, **kwargs*)

Bases: `ducky.streams.OutputStream`

class `ducky.streams.InputStream`(*machine, desc, stream=None, fd=None, close=True, allow_close=True*)

Bases: `ducky.streams.Stream`

```

    static create (machine, desc)

    read (size=None)

    write (b)

class ducky.streams.MethodInputStream (machine, desc, **kwargs)
    Bases: ducky.streams.InputStream

class ducky.streams.MethodOutputStream (machine, desc, **kwargs)
    Bases: ducky.streams.OutputStream

class ducky.streams.OutputStream (machine, desc, stream=None, fd=None, close=True, allow_close=True)
    Bases: ducky.streams.Stream

    static create (machine, desc)

    flush ()

    read (size=None)

    write (buff)

class ducky.streams.StderrStream (machine)
    Bases: ducky.streams.OutputStream

class ducky.streams.StdinStream (machine, **kwargs)
    Bases: ducky.streams.InputStream

    close ()

    get_selectee ()

class ducky.streams.StdoutStream (machine)
    Bases: ducky.streams.OutputStream

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.

`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.

2.19 ducky.tools package

2.19.1 Submodules

ducky.tools.as module

`ducky.tools.as.encode_blob` (*logger*, *file_in*, *options*)

`ducky.tools.as.main` ()

`ducky.tools.as.save_object_file` (*logger*, *sections*, *file_out*, *options*)

`ducky.tools.as.translate_buffer` (*logger*, *buffer*, *file_in*, *options*)

ducky.tools.cc module

```
ducky.tools.cc.compile_file(logger, options, file_in, file_out)
ducky.tools.cc.main()
```

ducky.tools.coredump module

```
ducky.tools.coredump.main()
ducky.tools.coredump.show_cores(logger, state)
ducky.tools.coredump.show_forth_dict(logger, state, last)
ducky.tools.coredump.show_forth_trace(logger, state)
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(logger, f_out, boundary)
ducky.tools.img.create_binary_image(logger, f_in, f_out, bio=False)
ducky.tools.img.create_hdt_image(logger, file_in, f_out, options)
ducky.tools.img.main()
```

ducky.tools.ld module

```
class ducky.tools.ld.LinkerInfo
    Bases: object
ducky.tools.ld.align_nop(n)
ducky.tools.ld.fix_section_bases(logger, info, f_out, required_bases)
ducky.tools.ld.main()
ducky.tools.ld.merge_object_into(logger, info, f_dst, f_src)
ducky.tools.ld.process_files(logger, info, files_in, file_out, bases=None)
ducky.tools.ld.resolve_relocations(logger, info, f_out, f_ins)
ducky.tools.ld.resolve_symbols(logger, info, f_out, f_ins)
```


ducky.tools.objdump module

```
ducky.tools.objdump.main()
ducky.tools.objdump.show_disassemble(logger,f)
ducky.tools.objdump.show_file_header(logger,f)
ducky.tools.objdump.show_reloc(logger,f)
ducky.tools.objdump.show_sections(logger,f)
ducky.tools.objdump.show_symbols(logger,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.

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.

enqueue (`buff`)

Called by protocol instance, to add newly received data to stream’s buffer.

Parameters `buff` (`bytearray`) – received bytes.

has_poll_support ()

See :py:meth:‘`ducky.streams.Stream.has_poll_support`’.

read (`size=None`)

See :py:meth:‘`ducky.streams.Stream.read`’.

register_with_reactor (`reactor, on_read=None, on_error=None`)

See :py:meth:‘`ducky.streams.Stream.register_with_reactor`’.

unregister_with_reactor (`reactor`)

See :py:meth:‘`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.

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, disable_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.

2.19.2 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)
```

2.20 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

```
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.LRUCache` (*logger, size, *args, **kwargs*)

Bases: `collections.OrderedDict`

Simple LRU cache, based on `OrderedDict`, with limited size. When limit is reached, the least recently inserted item is removed.

Parameters

- **logger** (*logging.Logger*) – logger object instance should use for logging.
- **size** (*int*) – maximal number of entries allowed.

get_object (*key*)

The real workhorse - responsible for getting requested item from outside when it's not present in cache. Called by `__missing__` method. This method itself makes no changes to cache at all.

make_space ()

This method is called when there is no free space in cache. It's responsible for freeing at least one slot, upper limit of removed entries is not enforced.

class `ducky.util.StringTable`

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 (`\0`).

This is a helper class - it makes working with string, e.g. section and symbol names, much easier.

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, and terminating zero byte (`\0`) is appended to mark end of string.

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*)

Indices and tables

- `genindex`
- `modindex`
- `search`

d

- [ducky.boot](#), 23
- [ducky.cc](#), 31
- [ducky.cc.passes](#), 29
- [ducky.cc.passes.ast_codegen](#), 27
- [ducky.cc.passes.ast_constprop](#), 28
- [ducky.cc.passes.ast_dce](#), 28
- [ducky.cc.passes.ast_visualise](#), 29
- [ducky.cc.passes.bt_peephole](#), 29
- [ducky.cc.passes.bt_simplify](#), 29
- [ducky.cc.passes.bt_visualise](#), 29
- [ducky.cc.types](#), 30
- [ducky.config](#), 25
- [ducky.console](#), 26
- [ducky.cpu](#), 61
- [ducky.cpu.assemble](#), 43
- [ducky.cpu.coprocessor](#), 43
- [ducky.cpu.coprocessor.control](#), 35
- [ducky.cpu.coprocessor.math_copro](#), 36
- [ducky.cpu.instructions](#), 45
- [ducky.cpu.registers](#), 60
- [ducky.debugging](#), 66
- [ducky.devices](#), 76
- [ducky.devices.keyboard](#), 69
- [ducky.devices.rtc](#), 69
- [ducky.devices.snapshot](#), 70
- [ducky.devices.storage](#), 70
- [ducky.devices.svga](#), 73
- [ducky.devices.terminal](#), 72
- [ducky.devices.tty](#), 73
- [ducky.errors](#), 77
- [ducky.hdt](#), 77
- [ducky.interfaces](#), 79
- [ducky.log](#), 80
- [ducky.machine](#), 81
- [ducky.mm](#), 87
- [ducky.mm.binary](#), 83
- [ducky.patch](#), 92
- [ducky.profiler](#), 93
- [ducky.reactor](#), 94
- [ducky.snapshot](#), 97
- [ducky.streams](#), 97
- [ducky.tools](#), 102
- [ducky.tools.as](#), 99
- [ducky.tools.cc](#), 100
- [ducky.tools.coredump](#), 100
- [ducky.tools.defs](#), 100
- [ducky.tools.img](#), 100
- [ducky.tools.ld](#), 100
- [ducky.tools.objdump](#), 101
- [ducky.tools.profile](#), 101
- [ducky.tools.vm](#), 101
- [ducky.util](#), 103

A

AccessViolationError, 77

acquire_register() (ducky.cc.SymbolStorage method), 35

act() (ducky.debugging.Action method), 66

act() (ducky.debugging.LogValueAction method), 67

act() (ducky.debugging.SuspendCoreAction method), 68

Action (class in ducky.debugging), 66

ADD (class in ducky.cc), 31

ADD (class in ducky.cpu.instructions), 45

ADD (ducky.cpu.instructions.DuckyOpcodes attribute), 49

add() (ducky.cc.Scope method), 34

add_breakpoint() (ducky.config.MachineConfig method), 25

add_call() (ducky.reactor.Reactor method), 95

add_child() (ducky.snapshot.SnapshotNode method), 97

add_common_options() (in module ducky.tools), 102

add_device() (ducky.config.MachineConfig method), 25

add_event() (ducky.reactor.Reactor method), 95

add_fd() (ducky.reactor.Reactor method), 95

add_fd() (ducky.reactor.SelectTask method), 96

add_incoming() (ducky.cc.Block method), 31

add_listener() (ducky.machine.EventBus method), 81

add_mmap() (ducky.config.MachineConfig method), 25

add_name() (ducky.cc.Block method), 31

add_outgoing() (ducky.cc.Block method), 31

add_point() (ducky.debugging.DebuggingSet method), 66

add_storage() (ducky.config.MachineConfig method), 25

add_task() (ducky.reactor.Reactor method), 95

ADDL (class in ducky.cpu.coprocessor.math_copro), 36

ADDL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 39

addr_to_offset() (in module ducky.mm), 92

addr_to_page() (in module ducky.mm), 92

address (ducky.mm.binary.SymbolEntry attribute), 86

addrrof() (ducky.cc.MemorySlotStorage method), 33

addrrof() (ducky.cc.StackSlotStorage method), 34

addrrof() (ducky.cc.SymbolStorage method), 35

align() (in module ducky.util), 104

align_file() (in module ducky.tools.img), 100

align_nop() (in module ducky.tools.ld), 100

AlignSlot (class in ducky.cpu.assemble), 43

ALLOC (ducky.mm.MMOperationList attribute), 88

alloc_page() (ducky.mm.MemoryController method), 88

alloc_pages() (ducky.mm.MemoryController method), 88

alloc_specific_page() (ducky.mm.MemoryController method), 88

AND (class in ducky.cc), 31

AND (class in ducky.cpu.instructions), 45

AND (ducky.cpu.instructions.DuckyOpcodes attribute), 49

AnonymousMemoryPage (class in ducky.mm), 87

area_to_pages() (in module ducky.mm), 92

args_block() (ducky.cc.Function method), 32

ARGUMENT (ducky.hdt.HDTEntryTypes attribute), 78

ArrayType (class in ducky.cc.types), 30

ASCII (ducky.mm.binary.SymbolDataTypes attribute), 86

AsciiSlot (class in ducky.cpu.assemble), 43

assemble_operands() (ducky.cpu.coprocessor.math_copro.Descriptor_MAT static method), 37

assemble_operands() (ducky.cpu.coprocessor.math_copro.LOAD static method), 38

assemble_operands() (ducky.cpu.coprocessor.math_copro.LOADUW static method), 38

assemble_operands() (ducky.cpu.coprocessor.math_copro.LOADW static method), 38

assemble_operands() (ducky.cpu.coprocessor.math_copro.SAVE static method), 41

assemble_operands() (ducky.cpu.coprocessor.math_copro.SAVEW static method), 41

assemble_operands() (ducky.cpu.instructions.CAS static method), 46

assemble_operands() (ducky.cpu.instructions.Descriptor static method), 47

assemble_operands() (ducky.cpu.instructions.Descriptor_I static method), 48

assemble_operands() (ducky.cpu.instructions.Descriptor_R static method), 48

assemble_operands() (ducky.cpu.instructions.Descriptor_R_I static method), 49

assemble_operands() (ducky.cpu.instructions.Descriptor_R_R static method), 49

static method), 49
assemble_operands() (ducky.cpu.instructions.Descriptor_RI method), 49
assemble_operands() (ducky.cpu.instructions.Descriptor_RI method), 48
assemble_operands() (ducky.cpu.instructions.Descriptor_RI method), 48
AssemblerError, 77
ASTOptVisitor (class in ducky.cc.passes), 29
ASTVisitor (class in ducky.cc.passes), 29
ASTVisualiseVisitor (class in ducky.cc.passes.ast_visualise), 29

B

Backend (class in ducky.devices.keyboard), 69
Backend (class in ducky.devices.tty), 73
backing_register() (ducky.cc.NamedValue method), 33
backtrace() (ducky.cpu.CPUCore method), 62
base (ducky.mm.binary.SectionHeader attribute), 85
BE (class in ducky.cc), 31
BE (class in ducky.cpu.instructions), 45
BG (class in ducky.cc), 31
BG (class in ducky.cpu.instructions), 45
bg (ducky.devices.svga.Char attribute), 73
BGE (class in ducky.cc), 31
BGE (class in ducky.cpu.instructions), 45
BinaryFile (class in ducky.util), 103
BIO (ducky.devices.IRQList attribute), 76
BL (class in ducky.cc), 31
BL (class in ducky.cpu.instructions), 45
BLE (class in ducky.cc), 31
BLE (class in ducky.cpu.instructions), 45
blink (ducky.devices.svga.Char attribute), 73
Block (class in ducky.cc), 31
block() (ducky.cc.Function method), 32
block() (ducky.cc.passes.ast_codegen.CodegenVisitor method), 27
BLOCK_SIZE (in module ducky.devices.storage), 70
BlockIO (class in ducky.devices.storage), 70
BLOCKIO (ducky.devices.IRQList attribute), 76
BlockTreeSimplifyVisitor (class in ducky.cc.passes.bt_simplify), 29
BlockTreeVisualiseVisitor (class in ducky.cc.passes.bt_visualise), 29
BlockVisitor (class in ducky.cc.passes), 30
BLUE() (in module ducky.log), 80
BNE (class in ducky.cc), 31
BNE (class in ducky.cpu.instructions), 45
BNO (class in ducky.cpu.instructions), 45
BNS (class in ducky.cpu.instructions), 46
BNZ (class in ducky.cpu.instructions), 46
BO (class in ducky.cpu.instructions), 46
body_block() (ducky.cc.Function method), 32
bool2option() (in module ducky.config), 26

boot() (ducky.boot.ROMLoader method), 24
boot() (ducky.console.ConsoleConnection method), 26
boot() (ducky.console.ConsoleMaster method), 26
boot() (ducky.cpu.CPU method), 61
boot() (ducky.cpu.CPUCore method), 62
boot() (ducky.devices.Device method), 76
boot() (ducky.devices.keyboard.Backend method), 69
boot() (ducky.devices.keyboard.Frontend method), 69
boot() (ducky.devices.rtc.RTC method), 69
boot() (ducky.devices.snapshot.FileSnapshotStorage method), 70
boot() (ducky.devices.storage.BlockIO method), 70
boot() (ducky.devices.storage.FileBackedStorage method), 71
boot() (ducky.devices.svga.Display method), 74
boot() (ducky.devices.svga.SimpleVGA method), 75
boot() (ducky.devices.terminal.StandalonePTYTerminal method), 72
boot() (ducky.devices.terminal.StreamIOTerminal method), 72
boot() (ducky.devices.terminal.Terminal method), 72
boot() (ducky.devices.tty.Backend method), 73
boot() (ducky.devices.tty.Frontend method), 73
boot() (ducky.interfaces.IMachineWorker method), 79
boot() (ducky.machine.Machine method), 82
boot() (ducky.mm.MemoryController method), 89
BRANCH (ducky.cpu.instructions.DuckyOpcodes attribute), 49
BreakPoint (class in ducky.debugging), 66
BS (class in ducky.cpu.instructions), 46
bss (ducky.mm.binary.SectionFlagsEncoding attribute), 85
BssSection (class in ducky.cpu.assemble), 43
BTPeepholeVisitor (class in ducky.cc.passes.bt_peephole), 29
buff_to_memory() (ducky.devices.storage.BlockIO method), 70
Buffer (class in ducky.cpu.assemble), 43
buildProtocol() (ducky.tools.vm.DuckySocketServerFactory method), 101
in BYTE (ducky.mm.binary.SymbolDataTypes attribute), 86
ByteSlot (class in ducky.cpu.assemble), 43
in BytesSlot (class in ducky.cpu.assemble), 43
BZ (class in ducky.cpu.instructions), 46

C

CALL (class in ducky.cc), 31
CALL (class in ducky.cpu.instructions), 46
CALL (ducky.cpu.instructions.DuckyOpcodes attribute), 49
CallInReactorTask (class in ducky.reactor), 95
can_register_backed() (ducky.cc.NamedValue method), 33
capture_state() (ducky.machine.Machine method), 82

capture_vm_state() (ducky.snapshot.VMState static method), 97
CAS (class in ducky.cpu.instructions), 46
CAS (ducky.cpu.instructions.DuckyOpcodes attribute), 49
change_runnable_state() (ducky.cpu.CPUCore method), 62
Char (class in ducky.devices.svga), 73
CHAR (ducky.mm.binary.SymbolDataTypes attribute), 86
CharSlot (class in ducky.cpu.assemble), 43
CharType (class in ducky.cc.types), 30
check_access() (ducky.cpu.MMU method), 64
check_protected_ins() (ducky.cpu.CPUCore method), 62
check_protected_port() (ducky.cpu.CPUCore method), 62
clear() (ducky.mm.AnonymousMemoryPage method), 87
clear() (ducky.mm.ExternalMemoryPage method), 87
clear() (ducky.mm.MemoryPage method), 90
CLI (class in ducky.cpu.instructions), 46
CLI (ducky.cpu.instructions.DuckyOpcodes attribute), 49
close() (ducky.cpu.assemble.AsciiSlot method), 43
close() (ducky.cpu.assemble.ByteSlot method), 43
close() (ducky.cpu.assemble.BytesSlot method), 43
close() (ducky.cpu.assemble.CharSlot method), 43
close() (ducky.cpu.assemble.DataSlot method), 44
close() (ducky.cpu.assemble.FunctionSlot method), 44
close() (ducky.cpu.assemble.IntSlot method), 44
close() (ducky.cpu.assemble.ShortSlot method), 44
close() (ducky.cpu.assemble.SpaceSlot method), 44
close() (ducky.cpu.assemble.StringSlot method), 44
close() (ducky.devices.tty.Frontend method), 73
close() (ducky.streams.StdinStream method), 98
close() (ducky.streams.Stream method), 98
cmd_boot() (in module ducky.machine), 83
cmd_bp_active() (in module ducky.debugging), 68
cmd_bp_add_breakpoint() (in module ducky.debugging), 68
cmd_bp_add_memory_watchpoint() (in module ducky.debugging), 68
cmd_bp_list() (in module ducky.debugging), 68
cmd_bp_remove() (in module ducky.debugging), 68
cmd_bt() (in module ducky.cpu), 65
cmd_cont() (in module ducky.cpu), 65
cmd_core_state() (in module ducky.cpu), 65
cmd_halt() (in module ducky.machine), 83
cmd_help() (in module ducky.console), 27
cmd_next() (in module ducky.cpu), 65
cmd_run() (in module ducky.machine), 83
cmd_set_core() (in module ducky.cpu), 65
cmd_snapshot() (in module ducky.machine), 83
cmd_step() (in module ducky.cpu), 65
CMP (class in ducky.cc), 31
CMP (class in ducky.cpu.instructions), 47
CMP (ducky.cpu.instructions.DuckyOpcodes attribute), 49
CMPU (class in ducky.cpu.instructions), 47
CMPU (ducky.cpu.instructions.DuckyOpcodes attribute), 49
CNT (ducky.cpu.registers.Registers attribute), 60
CodegenVisitor (class in ducky.cc.passes.ast_codegen), 27
codepoint (ducky.devices.svga.Char attribute), 73
COLS (ducky.devices.svga.SimpleVGACommands attribute), 75
CommChannel (class in ducky.machine), 81
Comment (class in ducky.cc), 31
CommQueue (class in ducky.machine), 81
compile_file() (in module ducky.tools.cc), 100
CompilerError, 31
connect() (ducky.cc.Block method), 31
connect() (ducky.console.ConsoleMaster method), 27
console_id (ducky.console.ConsoleMaster attribute), 27
ConsoleConnection (class in ducky.console), 26
ConsoleMaster (class in ducky.console), 26
ConstantFoldingVisitor (class in ducky.cc.passes.ast_constprop), 28
ConstantValue (class in ducky.cc), 32
CONTROL_MESSAGE_FIRST (ducky.devices.keyboard.ControlMessages attribute), 69
ControlCoproprocessor (class in ducky.cpu.coprocessor.control), 35
ControlMessages (class in ducky.devices.keyboard), 69
ControlRegisters (class in ducky.cpu.coprocessor.control), 36
Coproprocessor (class in ducky.cpu.coprocessor), 43
copy() (ducky.cpu.assemble.SourceLocation method), 44
core() (ducky.machine.Machine method), 82
CoreDumpFile (class in ducky.snapshot), 97
CoreFlags (class in ducky.cpu), 63
CoreFlags (class in ducky.cpu.coprocessor.control), 36
cores (ducky.machine.Machine attribute), 82
CPU (class in ducky.cpu), 61
CPU (ducky.hdt.HDTEntryTypes attribute), 78
CPUCore (class in ducky.cpu), 61
CPUCoreState (class in ducky.cpu), 63
CPUException, 63
CPUState (class in ducky.cpu), 63
CR0 (ducky.cpu.coprocessor.control.ControlRegisters attribute), 36
CR1 (ducky.cpu.coprocessor.control.ControlRegisters attribute), 36
CR2 (ducky.cpu.coprocessor.control.ControlRegisters attribute), 36
CR3 (ducky.cpu.coprocessor.control.ControlRegisters attribute), 36
create() (ducky.hdt.HDT method), 78

create() (ducky.hdt.HDTEntry class method), 78
 create() (ducky.hdt.HDTEntry_Argument class method), 78
 create() (ducky.streams.InputStream static method), 97
 create() (ducky.streams.OutputStream static method), 98
 create() (ducky.util.Flags class method), 103
 create_binary_image() (in module ducky.tools.img), 100
 create_frame() (ducky.cpu.CPUCore method), 62
 create_from_config() (ducky.debugging.BreakPoint static method), 66
 create_from_config() (ducky.debugging.LogMemoryContentAction static method), 66
 create_from_config() (ducky.debugging.LogRegisterContentAction static method), 67
 create_from_config() (ducky.debugging.MemoryWatchPoint static method), 67
 create_from_config() (ducky.debugging.SuspendCoreAction static method), 68
 create_from_config() (ducky.devices.Device static method), 76
 create_from_config() (ducky.devices.keyboard.Backend static method), 69
 create_from_config() (ducky.devices.keyboard.Frontend static method), 69
 create_from_config() (ducky.devices.rtc.RTC static method), 69
 create_from_config() (ducky.devices.snapshot.DefaultFileSnapshotStorage static method), 70
 create_from_config() (ducky.devices.snapshot.FileSnapshotStorage static method), 70
 create_from_config() (ducky.devices.snapshot.SnapshotStorage static method), 70
 create_from_config() (ducky.devices.storage.BlockIO static method), 70
 create_from_config() (ducky.devices.storage.FileBackedStorage static method), 71
 create_from_config() (ducky.devices.svga.Display static method), 74
 create_from_config() (ducky.devices.svga.SimpleVGA static method), 75
 create_from_config() (ducky.devices.terminal.StandalonePTYTerminal static method), 72
 create_from_config() (ducky.devices.terminal.StandardIOTerminal static method), 72
 create_from_config() (ducky.devices.terminal.StreamIOTerminal static method), 72
 create_from_config() (ducky.devices.tty.Backend static method), 73
 create_from_config() (ducky.devices.tty.Frontend static method), 73
 create_from_decl() (ducky.cc.types.CType static method), 30
 create_getters() (ducky.config.MachineConfig method), 25
 create_hdt_image() (in module ducky.tools.img), 100
 create_header() (ducky.mm.binary.File method), 83
 create_logger() (in module ducky.log), 81
 create_queue() (ducky.machine.CommChannel method), 81
 create_section() (ducky.mm.binary.File method), 83
 create_stats() (ducky.profiler.DummyCPUCoreProfiler method), 93
 create_stats() (ducky.profiler.DummyMachineProfiler method), 93
 create_text() (ducky.errors.AssemblerError method), 77
 CTR (class in ducky.cpu.instructions), 47
 CTR (in ducky.cpu.instructions.DuckyOpcodes attribute), 50
 CTW (class in ducky.cpu.instructions), 47
 CTW (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 CType (class in ducky.cc.types), 30
D
 DATA (ducky.mm.binary.SectionTypes attribute), 86
 data_size (ducky.mm.binary.SectionHeader attribute), 85
 DataSection (class in ducky.cpu.assemble), 43
 DataSlot (class in ducky.cpu.assemble), 44
 DEBUG (ducky.debugging.VMVerbosityLevels attribute), 68
 debug() (in module ducky.patch), 93
 DebuggingSet (class in ducky.debugging), 66
 DEC (class in ducky.cpu.instructions), 47
 DEC (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 DECL (class in ducky.cpu.coprocessor.math_copro), 36
 DECL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 39
 decode_instruction() (ducky.cpu.instructions.InstructionSet class method), 54
 decode_string() (in module ducky.cpu.assemble), 45
 DEFAULT_BOOT_MODE (in module ducky.devices.svga), 74
 DEFAULT_BOOTLOADER_ADDRESS (in module ducky.boot), 23
 DEFAULT_CORE_INST_CACHE_SIZE (in module ducky.cpu), 63
 DEFAULT_HDT_ADDRESS (in module ducky.boot), 23
 DEFAULT_IVT_ADDRESS (in module ducky.cpu), 63
 DEFAULT_MEMORY_BANKS (in module ducky.devices.svga), 74
 DEFAULT_MEMORY_SIZE (in module ducky.devices.svga), 74
 DEFAULT_MODES (in module ducky.devices.svga), 74
 DEFAULT_PORT_RANGE (in module ducky.devices.svga), 74
 DEFAULT_PT_ADDRESS (in module ducky.cpu), 63

DefaultFileSnapshotStorage (class in ducky.devices.snapshot), 70

DEPTH (ducky.devices.svga.SimpleVGACommands attribute), 75

Descriptor (class in ducky.cpu.instructions), 47

Descriptor_I (class in ducky.cpu.instructions), 48

Descriptor_MATH (class in ducky.cpu.coprocessor.math_copro), 37

Descriptor_R (class in ducky.cpu.instructions), 48

Descriptor_R_I (class in ducky.cpu.instructions), 49

Descriptor_R_R (class in ducky.cpu.instructions), 49

Descriptor_R_RI (class in ducky.cpu.instructions), 49

Descriptor_RI (class in ducky.cpu.instructions), 48

Descriptor_RI_R (class in ducky.cpu.instructions), 48

destroy_frame() (ducky.cpu.CPUCore method), 62

Device (class in ducky.devices), 76

DeviceBackend (class in ducky.devices), 76

DeviceFrontend (class in ducky.devices), 76

die() (ducky.console.ConsoleConnection method), 26

die() (ducky.cpu.CPU method), 61

die() (ducky.cpu.CPUCore method), 62

die() (ducky.interfaces.IMachineWorker method), 79

die() (ducky.machine.Machine method), 82

Directive (class in ducky.cc), 32

DIRTY (ducky.mm.PageTableEntry attribute), 92

disable() (ducky.profiler.DummyCPUCoreProfiler method), 93

disable() (ducky.profiler.DummyMachineProfiler method), 93

disassemble_instruction() (ducky.cpu.instructions.InstructionSet class method), 54

disassemble_mnemonic() (ducky.cpu.instructions.Descriptor class method), 47

disassemble_operands() (ducky.cpu.coprocessor.math_copro.Descriptor_MATH static method), 37

disassemble_operands() (ducky.cpu.coprocessor.math_copro.LOAD static method), 38

disassemble_operands() (ducky.cpu.coprocessor.math_copro.LOADUW static method), 38

disassemble_operands() (ducky.cpu.coprocessor.math_copro.LOADW static method), 38

disassemble_operands() (ducky.cpu.coprocessor.math_copro.DROP static method), 41

disassemble_operands() (ducky.cpu.coprocessor.math_copro.DSEW static method), 42

disassemble_operands() (ducky.cpu.instructions.CAS static method), 46

disassemble_operands() (ducky.cpu.instructions.Descriptor static method), 48

disassemble_operands() (ducky.cpu.instructions.Descriptor static method), 48

disassemble_operands() (ducky.cpu.instructions.Descriptor static method), 48

static method), 48

disassemble_operands() (ducky.cpu.instructions.Descriptor_R_I static method), 49

disassemble_operands() (ducky.cpu.instructions.Descriptor_R_R static method), 49

disassemble_operands() (ducky.cpu.instructions.Descriptor_R_RI static method), 49

disassemble_operands() (ducky.cpu.instructions.Descriptor_RI static method), 48

disassemble_operands() (ducky.cpu.instructions.Descriptor_RI_R static method), 48

DisassembleMismatchError, 77

Display (class in ducky.devices.svga), 74

DisplayRefreshTask (class in ducky.devices.svga), 74

DIV (class in ducky.cpu.instructions), 47

DIV (ducky.cpu.instructions.DuckyOpcodes attribute), 50

DIVL (class in ducky.cpu.coprocessor.math_copro), 37

DIVL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 39

do_int() (ducky.cpu.CPUCore method), 62

do_log_cpu_core_state() (in module ducky.cpu), 65

do_open() (ducky.util.BinaryFile static method), 103

do_read_blocks() (ducky.devices.storage.FileBackedStorage method), 71

do_read_blocks() (ducky.devices.storage.Storage method), 71

do_visit() (ducky.cc.passes.BlockVisitor method), 30

do_visit_block() (ducky.cc.passes.BlockVisitor method), 30

do_visit_block() (ducky.cc.passes.bt_peephole.BTPeepholeVisitor method), 29

do_visit_block() (ducky.cc.passes.bt_visualise.BlockTreeVisualiseVisitor method), 29

do_visit_fn() (ducky.cc.passes.BlockVisitor method), 30

do_visit_fn() (ducky.cc.passes.bt_simplify.BlockTreeSimplifyVisitor method), 29

do_write_blocks() (ducky.devices.storage.FileBackedStorage method), 71

do_write_blocks() (ducky.devices.storage.Storage method), 71

DOWN() (ducky.cc.passes.ASTVisitor method), 29

DOWN() (ducky.cc.passes.BlockVisitor method), 30

DROP (class in ducky.cpu.coprocessor.math_copro), 37

DROP (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 39

DSEW (class in ducky.cc.passes.ast_dce), 28

ducky.boot (module), 23

ducky.cc (module), 31

ducky.cc.passes (module), 29

ducky.cc.passes.ast_codegen (module), 27

ducky.cc.passes.ast_constprop (module), 28

ducky.cc.passes.ast_dce (module), 28

ducky.cc.passes.ast_visualise (module), 29

ducky.cc.passes.bt_peephole (module), 29

[ducky.cc.passes.bt_simplify \(module\)](#), 29
[ducky.cc.passes.bt_visualise \(module\)](#), 29
[ducky.cc.types \(module\)](#), 30
[ducky.config \(module\)](#), 25
[ducky.console \(module\)](#), 26
[ducky.cpu \(module\)](#), 61
[ducky.cpu.assemble \(module\)](#), 43
[ducky.cpu.coprocessor \(module\)](#), 43
[ducky.cpu.coprocessor.control \(module\)](#), 35
[ducky.cpu.coprocessor.math_copro \(module\)](#), 36
[ducky.cpu.instructions \(module\)](#), 45
[ducky.cpu.registers \(module\)](#), 60
[ducky.debugging \(module\)](#), 66
[ducky.devices \(module\)](#), 76
[ducky.devices.keyboard \(module\)](#), 69
[ducky.devices rtc \(module\)](#), 69
[ducky.devices.snapshot \(module\)](#), 70
[ducky.devices.storage \(module\)](#), 70
[ducky.devices.svga \(module\)](#), 73
[ducky.devices.terminal \(module\)](#), 72
[ducky.devices.tty \(module\)](#), 73
[ducky.errors \(module\)](#), 77
[ducky.hdt \(module\)](#), 77
[ducky.interfaces \(module\)](#), 79
[ducky.log \(module\)](#), 80
[ducky.machine \(module\)](#), 81
[ducky.mm \(module\)](#), 87
[ducky.mm.binary \(module\)](#), 83
[ducky.patch \(module\)](#), 92
[ducky.profiler \(module\)](#), 93
[ducky.reactor \(module\)](#), 94
[ducky.snapshot \(module\)](#), 97
[ducky.streams \(module\)](#), 97
[ducky.tools \(module\)](#), 102
[ducky.tools.as \(module\)](#), 99
[ducky.tools.cc \(module\)](#), 100
[ducky.tools.coredump \(module\)](#), 100
[ducky.tools.defs \(module\)](#), 100
[ducky.tools.img \(module\)](#), 100
[ducky.tools.ld \(module\)](#), 100
[ducky.tools.objdump \(module\)](#), 101
[ducky.tools.profile \(module\)](#), 101
[ducky.tools.vm \(module\)](#), 101
[ducky.util \(module\)](#), 103
[DuckyInstructionSet \(class in ducky.cpu.instructions\)](#), 49
[DuckyOpcodes \(class in ducky.cpu.instructions\)](#), 49
[DuckyProtocol \(class in ducky.tools.vm\)](#), 101
[DuckySocketServerFactory \(class in ducky.tools.vm\)](#), 101
[DummyCPUCoreProfiler \(class in ducky.profiler\)](#), 93
[DummyMachineProfiler \(class in ducky.profiler\)](#), 93
[dump_node\(\) \(in module ducky.cc\)](#), 35
[dump_stack\(\) \(ducky.cpu.coprocessor.math_copro.MathCoprocesor method\)](#), 39

[dump_stats\(\) \(ducky.profiler.DummyCPUCoreProfiler method\)](#), 93
[dump_stats\(\) \(ducky.profiler.DummyMachineProfiler method\)](#), 93
[dump_stats\(\) \(ducky.profiler.RealCPUCoreProfiler method\)](#), 94
[dumps\(\) \(ducky.config.MachineConfig method\)](#), 25
[DUP \(class in ducky.cpu.coprocessor.math_copro\)](#), 37
[DUP \(ducky.cpu.coprocessor.math_copro.MathCoprocesorOpcodes attribute\)](#), 39
[DUP2 \(class in ducky.cpu.coprocessor.math_copro\)](#), 37
[DUP2 \(ducky.cpu.coprocessor.math_copro.MathCoprocesorOpcodes attribute\)](#), 39

E

[emit\(\) \(ducky.cc.Block method\)](#), 31
[emit_epilog\(\) \(ducky.cc.passes.ast_codegen.CodegenVisitor method\)](#), 27
[emit_instruction\(\) \(ducky.cpu.instructions.Descriptor method\)](#), 48
[emit_prolog\(\) \(ducky.cc.passes.ast_codegen.CodegenVisitor method\)](#), 27
[emit_string_literals\(\) \(ducky.cc.passes.ast_codegen.CodegenVisitor method\)](#), 27
[emit_trampoline\(\) \(ducky.cc.passes.ast_codegen.CodegenVisitor method\)](#), 27
[EmptyMathStackError](#), 37
[enable\(\) \(ducky.profiler.DummyCPUCoreProfiler method\)](#), 93
[enable\(\) \(ducky.profiler.DummyMachineProfiler method\)](#), 94
[enable_cpu\(\) \(ducky.profiler.ProfilerStore method\)](#), 94
[enable_machine\(\) \(ducky.profiler.ProfilerStore method\)](#), 94
[ENCODE\(\) \(in module ducky.cpu.instructions\)](#), 51
[encode_blob\(\) \(in module ducky.tools.as\)](#), 99
[Encoding \(class in ducky.cpu.instructions\)](#), 51
[encoding \(ducky.cpu.instructions.CAS attribute\)](#), 46
[encoding \(ducky.cpu.instructions.CLI attribute\)](#), 46
[encoding \(ducky.cpu.instructions.CTR attribute\)](#), 47
[encoding \(ducky.cpu.instructions.CTW attribute\)](#), 47
[encoding \(ducky.cpu.instructions.Descriptor attribute\)](#), 48
[encoding \(ducky.cpu.instructions.Descriptor_R attribute\)](#), 48
[encoding \(ducky.cpu.instructions.Descriptor_R_I attribute\)](#), 49
[encoding \(ducky.cpu.instructions.Descriptor_R_R attribute\)](#), 49
[encoding \(ducky.cpu.instructions.Descriptor_R_RI attribute\)](#), 49
[encoding \(ducky.cpu.instructions.Descriptor_RI attribute\)](#), 48
[encoding \(ducky.cpu.instructions.Descriptor_RI_R attribute\)](#), 48

- encoding (ducky.cpu.instructions.FPTC attribute), 52
- encoding (ducky.cpu.instructions.IDLE attribute), 53
- encoding (ducky.cpu.instructions.LPM attribute), 55
- encoding (ducky.cpu.instructions.MOV attribute), 55
- encoding (ducky.cpu.instructions.NOP attribute), 55
- encoding (ducky.cpu.instructions.NOT attribute), 56
- encoding (ducky.cpu.instructions.RET attribute), 56
- encoding (ducky.cpu.instructions.RETINT attribute), 57
- encoding (ducky.cpu.instructions.RST attribute), 57
- encoding (ducky.cpu.instructions.STI attribute), 58
- encoding (ducky.cpu.instructions.SWP attribute), 59
- encoding() (ducky.util.Flags class method), 103
- encoding_to_u32() (in module ducky.cpu.instructions), 59
- EncodingA (class in ducky.cpu.instructions), 51
- EncodingC (class in ducky.cpu.instructions), 51
- EncodingI (class in ducky.cpu.instructions), 52
- EncodingLargeValueError, 77
- EncodingR (class in ducky.cpu.instructions), 52
- enqueue() (ducky.tools.vm.WSInputStream method), 102
- enqueue_input_stream() (ducky.devices.terminal.StreamIOTerminal method), 72
- enqueue_stream() (ducky.devices.keyboard.Frontend method), 69
- enqueue_streams() (ducky.devices.terminal.StreamIOTerminal method), 72
- entries (ducky.hdt.HDTHeader attribute), 79
- epilog_block() (ducky.cc.Function method), 32
- Error, 77
- ERROR (ducky.debugging.VMVerbosityLevels attribute), 68
- EventBus (class in ducky.machine), 81
- exec_f() (in module ducky.patch), 93
- executable (ducky.mm.binary.SectionFlagsEncoding attribute), 85
- EXECUTE (ducky.mm.PageTableEntry attribute), 92
- execute() (ducky.console.ConsoleConnection method), 26
- execute() (ducky.cpu.coprocessor.math_copro.ADDL static method), 36
- execute() (ducky.cpu.coprocessor.math_copro.DECL static method), 36
- execute() (ducky.cpu.coprocessor.math_copro.DIVL static method), 37
- execute() (ducky.cpu.coprocessor.math_copro.DROP static method), 37
- execute() (ducky.cpu.coprocessor.math_copro.DUP static method), 37
- execute() (ducky.cpu.coprocessor.math_copro.DUP2 static method), 37
- execute() (ducky.cpu.coprocessor.math_copro.INCL static method), 37
- execute() (ducky.cpu.coprocessor.math_copro.LOAD static method), 38
- execute() (ducky.cpu.coprocessor.math_copro.LOADUW static method), 38
- execute() (ducky.cpu.coprocessor.math_copro.LOADW static method), 38
- execute() (ducky.cpu.coprocessor.math_copro.MODL static method), 38
- execute() (ducky.cpu.coprocessor.math_copro.MULL static method), 38
- execute() (ducky.cpu.coprocessor.math_copro.POP static method), 40
- execute() (ducky.cpu.coprocessor.math_copro.POPUW static method), 40
- execute() (ducky.cpu.coprocessor.math_copro.POPW static method), 40
- execute() (ducky.cpu.coprocessor.math_copro.PUSH static method), 40
- execute() (ducky.cpu.coprocessor.math_copro.PUSHW static method), 41
- execute() (ducky.cpu.coprocessor.math_copro.SAVE static method), 41
- execute() (ducky.cpu.coprocessor.math_copro.SAVEW static method), 42
- execute() (ducky.cpu.coprocessor.math_copro.SWP static method), 42
- execute() (ducky.cpu.coprocessor.math_copro.SYMDIVL static method), 42
- execute() (ducky.cpu.coprocessor.math_copro.SYMMODL static method), 42
- execute() (ducky.cpu.coprocessor.math_copro.UDIVL static method), 42
- execute() (ducky.cpu.coprocessor.math_copro.UMODL static method), 42
- execute() (ducky.cpu.instructions.CALL static method), 46
- execute() (ducky.cpu.instructions.CAS static method), 46
- execute() (ducky.cpu.instructions.CLI static method), 46
- execute() (ducky.cpu.instructions.CMP static method), 47
- execute() (ducky.cpu.instructions.CMPU static method), 47
- execute() (ducky.cpu.instructions.CTR static method), 47
- execute() (ducky.cpu.instructions.CTW static method), 47
- execute() (ducky.cpu.instructions.DEC static method), 47
- execute() (ducky.cpu.instructions.Descriptor static method), 48
- execute() (ducky.cpu.instructions.FPTC static method), 52
- execute() (ducky.cpu.instructions.HLT static method), 52
- execute() (ducky.cpu.instructions.IDLE static method), 53
- execute() (ducky.cpu.instructions.INC static method), 53
- execute() (ducky.cpu.instructions.INT static method), 53
- execute() (ducky.cpu.instructions.IPI static method), 53
- execute() (ducky.cpu.instructions.J static method), 54
- execute() (ducky.cpu.instructions.LPM static method), 55

- execute() (ducky.cpu.instructions.MOV static method), 55
- execute() (ducky.cpu.instructions.NOP static method), 55
- execute() (ducky.cpu.instructions.NOT static method), 56
- execute() (ducky.cpu.instructions.POP static method), 56
- execute() (ducky.cpu.instructions.PUSH static method), 56
- execute() (ducky.cpu.instructions.RET static method), 57
- execute() (ducky.cpu.instructions.RETINT static method), 57
- execute() (ducky.cpu.instructions.RST static method), 57
- execute() (ducky.cpu.instructions.SIS static method), 58
- execute() (ducky.cpu.instructions.STI static method), 59
- execute() (ducky.cpu.instructions.SWP static method), 59
- exit_code (ducky.machine.Machine attribute), 82
- exit_interrupt() (ducky.cpu.CPUCore method), 62
- Expression (class in ducky.cc), 32
- ExpressionClass (class in ducky.cc), 32
- extend_with_push() (ducky.cpu.coprocessor.math_copro.MathCopro method), 39
- ExternalMemoryPage (class in ducky.mm), 87
- F**
- fd_blocking() (in module ducky.streams), 99
- FDCallbacks (class in ducky.reactor), 95
- FDInputStream (class in ducky.streams), 97
- FDOutputStream (class in ducky.streams), 97
- fg (ducky.devices.svga.Char attribute), 73
- field (ducky.mm.binary.FileFlags attribute), 84
- field (ducky.mm.binary.RelocFlags attribute), 84
- field (ducky.mm.binary.SectionFlags attribute), 85
- field (ducky.mm.binary.SymbolFlags attribute), 87
- field_offset() (ducky.cc.types.StructType method), 30
- field_type() (ducky.cc.types.StructType method), 30
- File (class in ducky.mm.binary), 83
- file_size (ducky.mm.binary.SectionHeader attribute), 85
- FileBackedStorage (class in ducky.devices.storage), 70
- FileFlags (class in ducky.mm.binary), 84
- FileFlagsEncoding (class in ducky.mm.binary), 84
- FileHeader (class in ducky.mm.binary), 84
- FileInputStream (class in ducky.streams), 97
- filename (ducky.mm.binary.SymbolEntry attribute), 86
- FileOutputStream (class in ducky.streams), 97
- FileSnapshotStorage (class in ducky.devices.snapshot), 70
- fill_reloc_slot() (ducky.cpu.instructions.Descriptor static method), 48
- fill_reloc_slot() (ducky.cpu.instructions.EncodingC static method), 51
- fill_reloc_slot() (ducky.cpu.instructions.EncodingI static method), 52
- fill_reloc_slot() (ducky.cpu.instructions.EncodingR static method), 52
- find_module() (ducky.patch.Importer method), 92
- finish() (ducky.cc.Function method), 32
- fix_section_bases() (in module ducky.tools.ld), 100
- flag (ducky.cpu.instructions.EncodingC attribute), 51
- Flags (class in ducky.util), 103
- flags (ducky.cpu.CPUCore attribute), 62
- flags (ducky.mm.binary.FileHeader attribute), 84
- flags (ducky.mm.binary.RelocEntry attribute), 84
- flags (ducky.mm.binary.SectionHeader attribute), 85
- flags (ducky.mm.binary.SymbolEntry attribute), 86
- flush() (ducky.devices.tty.Frontend method), 73
- flush() (ducky.streams.OutputStream method), 98
- format() (ducky.log.LogFormatter method), 80
- format_field() (ducky.util.Formatter method), 103
- format_int() (ducky.util.Formatter method), 103
- Formatter (class in ducky.util), 103
- FP (ducky.cpu.registers.Registers attribute), 60
- FP() (ducky.cpu.CPUCore method), 61
- FPTC (class in ducky.cpu.instructions), 52
- FPTC (class in ducky.cpu.instructions.DuckyOpcodes attribute), 50
- FREE (ducky.mm.MMOperationList attribute), 88
- free() (ducky.cc.Register method), 34
- free_page() (ducky.mm.MemoryController method), 89
- free_pages() (ducky.mm.MemoryController method), 89
- from_encoding() (ducky.util.Flags class method), 103
- from_int() (ducky.util.Flags class method), 103
- from_string() (ducky.devices.svga.Mode class method), 74
- from_string() (ducky.util.Flags class method), 103
- from_u16() (ducky.devices.svga.Char static method), 74
- from_u8() (ducky.devices.svga.Char static method), 74
- Frontend (class in ducky.devices.keyboard), 69
- Frontend (class in ducky.devices.tty), 73
- FrontendFlushTask (class in ducky.devices.tty), 73
- full_read_u16() (ducky.cpu.MMU method), 64
- full_read_u32() (ducky.cpu.MMU method), 65
- full_read_u8() (ducky.cpu.MMU method), 65
- full_write_u16() (ducky.cpu.MMU method), 65
- full_write_u32() (ducky.cpu.MMU method), 65
- full_write_u8() (ducky.cpu.MMU method), 65
- FullMathStackError, 37
- Function (class in ducky.cc), 32
- FUNCTION (ducky.mm.binary.SymbolDataTypes attribute), 86
- FunctionSlot (class in ducky.cpu.assemble), 44
- FunctionType (class in ducky.cc.types), 30
- G**
- generic_visit() (ducky.cc.passes.ast_codegen.CodegenVisitor method), 27
- generic_visit() (ducky.cc.passes.ast_visualise.ASTVisualiseVisitor method), 29
- generic_visit() (ducky.cc.passes.ASTVisitor method), 29
- get() (ducky.boot.MMapMemoryPage method), 24

- get() (ducky.cc.RegisterSet method), 34
 - get() (ducky.cc.Scope method), 34
 - get() (ducky.config.MachineConfig method), 25
 - get() (ducky.devices.svga.SimpleVGAMemoryPage method), 75
 - get() (ducky.mm.ExternalMemoryPage method), 87
 - get_child() (ducky.snapshot.SnapshotNode method), 97
 - get_children() (ducky.snapshot.SnapshotNode method), 97
 - get_code() (ducky.patch.ModuleLoader method), 92
 - get_core_profiler() (ducky.profiler.ProfilerStore method), 94
 - get_core_state_by_id() (ducky.cpu.CPUState method), 63
 - get_core_states() (ducky.cpu.CPUState method), 63
 - get_cpu_state_by_id() (ducky.machine.MachineState method), 83
 - get_cpu_states() (ducky.machine.MachineState method), 83
 - get_device_by_name() (ducky.machine.Machine method), 82
 - get_driver_creator() (in module ducky.devices), 76
 - get_error() (ducky.cpu.assemble.Buffer method), 43
 - get_from_decl() (ducky.cc.types.CType static method), 30
 - get_from_desc() (ducky.cc.types.CType static method), 30
 - get_header() (ducky.mm.binary.File method), 83
 - get_index() (ducky.cc.passes.ast_visualise.ASTVisualiseVisitor method), 29
 - get_instruction_set() (in module ducky.cpu.instructions), 59
 - get_line() (ducky.cpu.assemble.Buffer method), 43
 - get_machine_profiler() (ducky.profiler.ProfilerStore method), 94
 - get_master() (ducky.devices.Device method), 76
 - get_message() (ducky.debugging.LogMemoryContentAction method), 66
 - get_message() (ducky.debugging.LogRegisterContentAction method), 67
 - get_message() (ducky.debugging.LogValueAction method), 67
 - get_new_label() (ducky.cc.passes.ast_codegen.CodegenVisitor method), 27
 - get_new_literal_label() (ducky.cc.passes.ast_codegen.CodegenVisitor method), 27
 - get_new_local_storage() (ducky.cc.passes.ast_codegen.CodegenVisitor method), 27
 - get_object() (ducky.cpu.InstructionCache method), 63
 - get_object() (ducky.util.LRUCache method), 104
 - get_object_jit() (ducky.cpu.InstructionCache method), 64
 - get_page() (ducky.mm.MemoryController method), 89
 - get_page_states() (ducky.mm.MemoryState method), 92
 - get_pages() (ducky.mm.MemoryController method), 89
 - get_pte() (ducky.cpu.MMU method), 65
 - get_queue() (ducky.machine.CommChannel method), 81
 - get_section() (ducky.mm.binary.File method), 83
 - get_section_by_name() (ducky.mm.binary.File method), 83
 - get_selectee() (ducky.streams.StdinStream method), 98
 - get_slave_devices() (in module ducky.devices.terminal), 72
 - get_slave_gpu() (ducky.devices.svga.Display static method), 74
 - get_source() (ducky.patch.ModuleLoader method), 92
 - get_storage_by_id() (ducky.machine.Machine method), 82
 - get_string() (ducky.util.StringTable method), 104
 - get_symbol() (ducky.util.SymbolTable method), 104
 - get_values() (ducky.debugging.LogMemoryContentAction method), 66
 - get_values() (ducky.debugging.LogRegisterContentAction method), 67
 - get_values() (ducky.debugging.LogValueAction method), 67
 - getbool() (ducky.config.MachineConfig method), 26
 - getfloat() (ducky.config.MachineConfig method), 26
 - getint() (ducky.config.MachineConfig method), 26
 - globally_visible (ducky.mm.binary.SectionFlagsEncoding attribute), 85
 - globally_visible (ducky.mm.binary.SymbolFlagsEncoding attribute), 87
 - GRAPHIC (ducky.devices.svga.SimpleVGACommands attribute), 75
 - GREEN() (in module ducky.log), 80
- ## H
- HALT (ducky.devices.IRQList attribute), 76
 - HALT (ducky.devices.keyboard.ControlMessages attribute), 69
 - halt() (ducky.boot.ROMLoader method), 24
 - halt() (ducky.console.ConsoleConnection method), 26
 - halt() (ducky.console.ConsoleMaster method), 27
 - halt() (ducky.console.TerminalConsoleConnection method), 27
 - halt() (ducky.cpu.CPU method), 61
 - halt() (ducky.cpu.CPUCore method), 62
 - halt() (ducky.cpu.MMU method), 65
 - halt() (ducky.devices.Device method), 76
 - halt() (ducky.devices.keyboard.Backend method), 69
 - halt() (ducky.devices.keyboard.Frontend method), 69
 - halt() (ducky.devices.rtc.RTC method), 69
 - halt() (ducky.devices.snapshot.FileSnapshotStorage method), 70
 - halt() (ducky.devices.snapshot.SnapshotStorage method), 70
 - halt() (ducky.devices.storage.BlockIO method), 70
 - halt() (ducky.devices.storage.FileBackedStorage method), 71

- halt() (ducky.devices.svga.Display method), 74
- halt() (ducky.devices.svga.SimpleVGA method), 75
- halt() (ducky.devices.terminal.StandalonePTYTerminal method), 72
- halt() (ducky.devices.terminal.StreamIOTerminal method), 72
- halt() (ducky.devices.terminal.Terminal method), 72
- halt() (ducky.devices.tty.Backend method), 73
- halt() (ducky.devices.tty.Frontend method), 73
- halt() (ducky.interfaces.IMachineWorker method), 80
- halt() (ducky.machine.Machine method), 82
- halt() (ducky.mm.MemoryController method), 89
- HaltMachineTask (class in ducky.machine), 81
- has_coprocessor() (ducky.cpu.CPUCore method), 62
- has_fd() (ducky.streams.Stream method), 98
- has_lines() (ducky.cpu.assemble.Buffer method), 43
- has_poll_support() (ducky.streams.Stream method), 99
- has_poll_support() (ducky.tools.vm.WSInputStream method), 102
- has_register() (ducky.cc.SymbolStorage method), 35
- HDT (class in ducky.hdt), 77
- HDT_MAGIC (in module ducky.hdt), 79
- HDTEntry (class in ducky.hdt), 78
- HDTEntry_Argument (class in ducky.hdt), 78
- HDTEntry_CPU (class in ducky.hdt), 78
- HDTEntry_Memory (class in ducky.hdt), 79
- HDTEntryTypes (class in ducky.hdt), 78
- HDTHeader (class in ducky.hdt), 79
- HDTStructure (class in ducky.hdt), 79
- header_block() (ducky.cc.Function method), 32
- HLT (class in ducky.cc), 32
- HLT (class in ducky.cpu.instructions), 52
- HLT (ducky.cpu.instructions.DuckyOpcodes attribute), 50
- hw_setup() (ducky.machine.Machine method), 82
- I
- id (ducky.cc.Block attribute), 31
- IDLE (class in ducky.cpu.instructions), 52
- IDLE (ducky.cpu.instructions.DuckyOpcodes attribute), 50
- IE_FLAG() (in module ducky.cpu.instructions), 53
- IE_IMM() (in module ducky.cpu.instructions), 53
- IE_OPCODE() (in module ducky.cpu.instructions), 53
- IE_REG() (in module ducky.cpu.instructions), 53
- IMachineWorker (class in ducky.interfaces), 79
- immediate (ducky.cpu.instructions.EncodingC attribute), 51
- immediate (ducky.cpu.instructions.EncodingI attribute), 52
- immediate (ducky.cpu.instructions.EncodingR attribute), 52
- immediate_flag (ducky.cpu.instructions.EncodingC attribute), 51
- immediate_flag (ducky.cpu.instructions.EncodingI attribute), 52
- immediate_flag (ducky.cpu.instructions.EncodingR attribute), 52
- Importer (class in ducky.patch), 92
- INB (class in ducky.cpu.instructions), 53
- INB (ducky.cpu.instructions.DuckyOpcodes attribute), 50
- INC (class in ducky.cc), 32
- INC (class in ducky.cpu.instructions), 53
- INC (ducky.cpu.instructions.DuckyOpcodes attribute), 50
- INCL (class in ducky.cpu.coprocessor.math_copro), 37
- INCL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 39
- IncompatibleLinkerFlagsError, 77
- IncompatibleTypeError, 32
- IncompleteDirectiveError, 77
- index (ducky.mm.binary.SectionHeader attribute), 85
- INFO (ducky.debugging.VMVerbosityLevels attribute), 68
- init() (ducky.cpu.instructions.InstructionSet class method), 54
- init_debug_set() (ducky.cpu.CPUCore method), 62
- InlineAsm (class in ducky.cc), 32
- InputStream (class in ducky.streams), 97
- INS (class in ducky.cpu.instructions), 53
- INS (ducky.cpu.instructions.DuckyOpcodes attribute), 50
- inst_aligned (ducky.cpu.instructions.Descriptor attribute), 48
- inst_aligned (ducky.mm.binary.RelocFlagsEncoding attribute), 85
- Instruction (class in ducky.cc), 32
- instruction_set_id (ducky.cpu.coprocessor.math_copro.MathCoproprocessorInstructions attribute), 39
- instruction_set_id (ducky.cpu.instructions.DuckyInstructionSet attribute), 49
- instruction_set_id (ducky.cpu.instructions.InstructionSet attribute), 54
- InstructionCache (class in ducky.cpu), 63
- instructions (ducky.cpu.coprocessor.math_copro.MathCoproprocessorInstructions attribute), 39
- instructions (ducky.cpu.instructions.DuckyInstructionSet attribute), 49
- instructions (ducky.cpu.instructions.InstructionSet attribute), 54
- instructions() (ducky.cc.Block method), 31
- InstructionSet (class in ducky.cpu.instructions), 53
- InstructionSetMetaclass (class in ducky.cpu.instructions), 54
- INT (class in ducky.cc), 32
- INT (class in ducky.cpu.instructions), 53
- INT (ducky.cpu.instructions.DuckyOpcodes attribute), 50
- INT (ducky.mm.binary.SymbolDataTypes attribute), 86
- InterruptVector (class in ducky.cpu), 64
- IntSlot (class in ducky.cpu.assemble), 44

IntType (class in ducky.cc.types), 30
 InvalidInstructionSetError, 64
 InvalidOpcodeError, 64
 InvalidResourceError, 77
 INW (class in ducky.cpu.instructions), 53
 INW (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 IOPorts (class in ducky.devices), 76
 IOProvider (class in ducky.devices), 76
 IP (ducky.cpu.registers.Registers attribute), 60
 IP() (ducky.cpu.CPUCore method), 62
 IPI (class in ducky.cpu.instructions), 53
 IPI (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 IReactorTask (class in ducky.interfaces), 80
 irq() (ducky.cpu.CPUCore method), 62
 IRQ_COUNT (ducky.devices.IRQList attribute), 76
 IRQList (class in ducky.devices), 76
 IRQProvider (class in ducky.devices), 76
 IRQRouterTask (class in ducky.machine), 81
 is_cpu_enabled() (ducky.profiler.ProfilerStore method), 94
 is_empty_in() (ducky.machine.CommQueue method), 81
 is_empty_out() (ducky.machine.CommQueue method), 81
 is_lvalue() (ducky.cc.Expression method), 32
 is_machine_enabled() (ducky.profiler.ProfilerStore method), 94
 is_mlvalue() (ducky.cc.Expression method), 32
 is_port_protected() (ducky.devices.IOProvider method), 76
 is_register_backed() (ducky.cc.NamedValue method), 33
 is_registered_command() (ducky.console.ConsoleMaster method), 27
 is_rvalue() (ducky.cc.Expression method), 32
 is_slave() (ducky.devices.Device method), 76
 is_triggered() (ducky.debugging.BreakPoint method), 66
 is_triggered() (ducky.debugging.MemoryWatchPoint method), 67
 is_triggered() (ducky.debugging.Point method), 68
 IsAPointerError, 33
 isfile() (in module ducky.util), 104
 ISnapshotable (class in ducky.interfaces), 80
 items (ducky.mm.binary.SectionHeader attribute), 85
 iter_breakpoints() (ducky.config.MachineConfig method), 26
 iter_devices() (ducky.config.MachineConfig method), 26
 iter_mmaps() (ducky.config.MachineConfig method), 26
 iter_storages() (ducky.config.MachineConfig method), 26
 IVirtualInterrupt (class in ducky.interfaces), 80

J

J (class in ducky.cc), 33
 J (class in ducky.cpu.instructions), 54
 J (ducky.cpu.instructions.DuckyOpcodes attribute), 50

jit() (ducky.cpu.instructions.ADD static method), 45
 jit() (ducky.cpu.instructions.AND static method), 45
 jit() (ducky.cpu.instructions.CALL static method), 46
 jit() (ducky.cpu.instructions.CMP static method), 47
 jit() (ducky.cpu.instructions.DEC static method), 47
 jit() (ducky.cpu.instructions.Descriptor static method), 48
 jit() (ducky.cpu.instructions.INC static method), 53
 jit() (ducky.cpu.instructions.J static method), 54
 jit() (ducky.cpu.instructions.LA static method), 54
 jit() (ducky.cpu.instructions.LI static method), 54
 jit() (ducky.cpu.instructions.MOV static method), 55
 jit() (ducky.cpu.instructions.MUL static method), 55
 jit() (ducky.cpu.instructions.OR static method), 56
 jit() (ducky.cpu.instructions.POP static method), 56
 jit() (ducky.cpu.instructions.PUSH static method), 56
 jit() (ducky.cpu.instructions.RET static method), 57
 jit() (ducky.cpu.instructions.SHIFTL static method), 58
 jit() (ducky.cpu.instructions.SHIFTR static method), 58
 jit() (ducky.cpu.instructions.SUB static method), 59
 jit() (ducky.cpu.instructions.SWP static method), 59
 jit() (ducky.cpu.instructions.XOR static method), 59
 JUMP() (in module ducky.cpu.instructions), 54

K

KEYBOARD (ducky.devices.IRQList attribute), 76
 classes (ducky.hdt.HDT attribute), 78

L

LA (class in ducky.cc), 33
 LA (class in ducky.cpu.instructions), 54
 LA (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 Label (class in ducky.cpu.assemble), 44
 LB (class in ducky.cc), 33
 LB (class in ducky.cpu.instructions), 54
 LB (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 length (ducky.hdt.HDTEntry_Argument attribute), 78
 length (ducky.hdt.HDTEntry_CPU attribute), 79
 length (ducky.hdt.HDTEntry_Memory attribute), 79
 length (ducky.hdt.HDTHeader attribute), 79
 LI (class in ducky.cc), 33
 LI (class in ducky.cpu.instructions), 54
 LI (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 lineno (ducky.mm.binary.SymbolEntry attribute), 86
 LinkerInfo (class in ducky.tools.ld), 100
 LIU (class in ducky.cpu.instructions), 54
 LIU (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 LOAD (class in ducky.cpu.coprocessor.math_copro), 37
 LOAD (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 39
 load() (ducky.cpu.instructions.LA class method), 54
 load() (ducky.cpu.instructions.LI class method), 54
 load() (ducky.cpu.instructions.LIU class method), 54
 load() (ducky.mm.binary.File method), 83
 load() (ducky.snapshot.CoreDumpFile method), 97

- load() (in module ducky.cc.passes), 30
 - load_data() (ducky.boot.ROMLoader method), 24
 - load_encoding() (ducky.util.Flags method), 103
 - load_int() (ducky.util.Flags method), 103
 - load_module() (ducky.patch.ModuleLoader method), 92
 - load_state() (ducky.boot.MMapArea method), 23
 - load_state() (ducky.cpu.coprocessor.math_copro.MathCoprocesor method), 39
 - load_state() (ducky.cpu.coprocessor.math_copro.RegisterSet method), 41
 - load_state() (ducky.cpu.CPU method), 61
 - load_state() (ducky.cpu.CPUCore method), 62
 - load_state() (ducky.interfaces.ISnapshotable method), 80
 - load_state() (ducky.machine.Machine method), 82
 - load_state() (ducky.mm.MemoryController method), 89
 - load_state() (ducky.mm.MemoryPage method), 90
 - load_state() (ducky.mm.MemoryRegion method), 92
 - load_string() (ducky.util.Flags method), 103
 - load_symbols() (ducky.mm.binary.File method), 83
 - load_text() (ducky.boot.ROMLoader method), 24
 - load_vm_state() (ducky.snapshot.VMState static method), 97
 - loadable (ducky.mm.binary.SectionFlagsEncoding attribute), 85
 - loader_for_path() (ducky.patch.Importer method), 92
 - LOADUW (class in ducky.cpu.coprocessor.math_copro), 38
 - LOADUW (ducky.cpu.coprocessor.math_copro.MathCoprocesor attribute), 39
 - LOADW (class in ducky.cpu.coprocessor.math_copro), 38
 - LOADW (ducky.cpu.coprocessor.math_copro.MathCoprocesorOpcodes attribute), 39
 - log() (ducky.console.ConsoleConnection method), 26
 - log() (ducky.errors.AssemblerError method), 77
 - log_cpu_core_state() (in module ducky.cpu), 65
 - LogFormatter (class in ducky.log), 80
 - LOGGER_VERBOSITY (ducky.debugging.VMDebugOperationList attribute), 68
 - LogMemoryContentAction (class in ducky.debugging), 66
 - LogRegisterContentAction (class in ducky.debugging), 66
 - LogValueAction (class in ducky.debugging), 67
 - LPM (class in ducky.cpu.instructions), 55
 - LPM (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 - LRUCache (class in ducky.util), 103
 - LS (class in ducky.cc), 33
 - LS (class in ducky.cpu.instructions), 55
 - LS (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 - LVALUE (ducky.cc.ExpressionClass attribute), 32
 - LValueExpression (class in ducky.cc), 33
 - LW (class in ducky.cc), 33
 - LW (class in ducky.cpu.instructions), 55
 - LW (ducky.cpu.instructions.DuckyOpcodes attribute), 50
- ## M
- Machine (class in ducky.machine), 81
 - MachineConfig (class in ducky.config), 25
 - MachineState (class in ducky.machine), 83
 - magic (ducky.hdt.HDTHeader attribute), 79
 - MAGIC (ducky.mm.binary.File attribute), 83
 - magic (ducky.mm.binary.FileHeader attribute), 84
 - main() (in module ducky.tools.as), 99
 - main() (in module ducky.tools.cc), 100
 - main() (in module ducky.tools.coredump), 100
 - main() (in module ducky.tools.defs), 100
 - main() (in module ducky.tools.img), 100
 - main() (in module ducky.tools.ld), 100
 - main() (in module ducky.tools.objdump), 101
 - main() (in module ducky.tools.profile), 101
 - main() (in module ducky.tools.vm), 102
 - make_current() (ducky.cc.passes.ast_codegen.CodegenVisitor method), 27
 - make_space() (ducky.util.LRUCache method), 104
 - MalformedBinaryError, 88
 - materialize() (ducky.cc.Block method), 31
 - materialize() (ducky.cc.Comment method), 31
 - materialize() (ducky.cc.Directive method), 32
 - materialize() (ducky.cc.Function method), 32
 - materialize() (ducky.cc.InlineAsm method), 32
 - materialize() (ducky.cc.Instruction method), 33
 - materialize() (ducky.cc.passes.ast_codegen.CodegenVisitor method), 27
 - MathCoprocesor (class in ducky.cpu.coprocessor.math_copro), 38
 - MathCoprocesorInstructionSet (class in ducky.cpu.coprocessor.math_copro), 39
 - MathCoprocesorOpcodes (class in ducky.cpu.coprocessor.math_copro), 39
 - MathCoprocesorState (class in ducky.cpu.coprocessor.math_copro), 40
 - MAX_NAME_LENGTH (ducky.hdt.HDTEntry_Argument attribute), 78
 - MEMORY (ducky.hdt.HDTEntryTypes attribute), 78
 - MEMORY_BANK_ID (ducky.devices.svga.SimpleVGACommands attribute), 75
 - memory_to_buff() (ducky.devices.storage.BlockIO method), 70
 - MemoryController (class in ducky.mm), 88
 - MemoryPage (class in ducky.mm), 90
 - MemoryPageState (class in ducky.mm), 91
 - MemoryRegion (class in ducky.mm), 91
 - MemoryRegionState (class in ducky.mm), 92
 - MemorySlotStorage (class in ducky.cc), 33
 - MemorySlotValue (class in ducky.cc), 33

- MemoryState (class in ducky.mm), 92
- MemoryWatchPoint (class in ducky.debugging), 67
- merge() (ducky.profiler.ProfileRecord method), 94
- merge_object_into() (in module ducky.tools.ld), 100
- MethodInputStream (class in ducky.streams), 98
- MethodOutputStream (class in ducky.streams), 98
- MLVALUE (ducky.cc.ExpressionClass attribute), 32
- MLValueExpression (class in ducky.cc), 33
- MMAP (ducky.mm.MMOperationList attribute), 88
- mmap_area() (ducky.boot.ROMLoader method), 24
- mmapable (ducky.mm.binary.FileFlagsEncoding attribute), 84
- mmapable (ducky.mm.binary.SectionFlagsEncoding attribute), 85
- MMapArea (class in ducky.boot), 23
- MMapAreaState (class in ducky.boot), 23
- MMapMemoryPage (class in ducky.boot), 23
- MMOperationList (class in ducky.mm), 88
- MMU (class in ducky.cpu), 64
- mnemonic (ducky.cpu.coprocessor.math_copro.ADDL attribute), 36
- mnemonic (ducky.cpu.coprocessor.math_copro.DECL attribute), 36
- mnemonic (ducky.cpu.coprocessor.math_copro.DIVL attribute), 37
- mnemonic (ducky.cpu.coprocessor.math_copro.DROP attribute), 37
- mnemonic (ducky.cpu.coprocessor.math_copro.DUP attribute), 37
- mnemonic (ducky.cpu.coprocessor.math_copro.DUP2 attribute), 37
- mnemonic (ducky.cpu.coprocessor.math_copro.INCL attribute), 37
- mnemonic (ducky.cpu.coprocessor.math_copro.LOAD attribute), 38
- mnemonic (ducky.cpu.coprocessor.math_copro.LOADUW attribute), 38
- mnemonic (ducky.cpu.coprocessor.math_copro.LOADW attribute), 38
- mnemonic (ducky.cpu.coprocessor.math_copro.MODL attribute), 38
- mnemonic (ducky.cpu.coprocessor.math_copro.MULL attribute), 38
- mnemonic (ducky.cpu.coprocessor.math_copro.POP attribute), 40
- mnemonic (ducky.cpu.coprocessor.math_copro.POPUW attribute), 40
- mnemonic (ducky.cpu.coprocessor.math_copro.POPW attribute), 40
- mnemonic (ducky.cpu.coprocessor.math_copro.PUSH attribute), 40
- mnemonic (ducky.cpu.coprocessor.math_copro.PUSHW attribute), 41
- mnemonic (ducky.cpu.coprocessor.math_copro.SAVE attribute), 41
- mnemonic (ducky.cpu.coprocessor.math_copro.SAVEW attribute), 42
- mnemonic (ducky.cpu.coprocessor.math_copro.SWP attribute), 42
- mnemonic (ducky.cpu.coprocessor.math_copro.SYMDIVL attribute), 42
- mnemonic (ducky.cpu.coprocessor.math_copro.SYMMODL attribute), 42
- mnemonic (ducky.cpu.coprocessor.math_copro.UDIVL attribute), 42
- mnemonic (ducky.cpu.coprocessor.math_copro.UMODL attribute), 42
- mnemonic (ducky.cpu.instructions.ADD attribute), 45
- mnemonic (ducky.cpu.instructions.AND attribute), 45
- mnemonic (ducky.cpu.instructions.BE attribute), 45
- mnemonic (ducky.cpu.instructions.BG attribute), 45
- mnemonic (ducky.cpu.instructions.BGE attribute), 45
- mnemonic (ducky.cpu.instructions.BL attribute), 45
- mnemonic (ducky.cpu.instructions.BLE attribute), 45
- mnemonic (ducky.cpu.instructions.BNE attribute), 45
- mnemonic (ducky.cpu.instructions.BNO attribute), 46
- mnemonic (ducky.cpu.instructions.BNS attribute), 46
- mnemonic (ducky.cpu.instructions.BNZ attribute), 46
- mnemonic (ducky.cpu.instructions.BO attribute), 46
- mnemonic (ducky.cpu.instructions.BS attribute), 46
- mnemonic (ducky.cpu.instructions.BZ attribute), 46
- mnemonic (ducky.cpu.instructions.CALL attribute), 46
- mnemonic (ducky.cpu.instructions.CAS attribute), 46
- mnemonic (ducky.cpu.instructions.CLI attribute), 46
- mnemonic (ducky.cpu.instructions.CMP attribute), 47
- mnemonic (ducky.cpu.instructions.CMPU attribute), 47
- mnemonic (ducky.cpu.instructions.CTR attribute), 47
- mnemonic (ducky.cpu.instructions.CTW attribute), 47
- mnemonic (ducky.cpu.instructions.DEC attribute), 47
- mnemonic (ducky.cpu.instructions.Descriptor attribute), 48
- mnemonic (ducky.cpu.instructions.DIV attribute), 47
- mnemonic (ducky.cpu.instructions.FPTC attribute), 52
- mnemonic (ducky.cpu.instructions.HLT attribute), 52
- mnemonic (ducky.cpu.instructions.IDLE attribute), 53
- mnemonic (ducky.cpu.instructions.INB attribute), 53
- mnemonic (ducky.cpu.instructions.INC attribute), 53
- mnemonic (ducky.cpu.instructions.INS attribute), 53
- mnemonic (ducky.cpu.instructions.INT attribute), 53
- mnemonic (ducky.cpu.instructions.INW attribute), 53
- mnemonic (ducky.cpu.instructions.IPI attribute), 53
- mnemonic (ducky.cpu.instructions.J attribute), 54
- mnemonic (ducky.cpu.instructions.LA attribute), 54
- mnemonic (ducky.cpu.instructions.LB attribute), 54
- mnemonic (ducky.cpu.instructions.LI attribute), 54
- mnemonic (ducky.cpu.instructions.LIU attribute), 54
- mnemonic (ducky.cpu.instructions.LPM attribute), 55
- mnemonic (ducky.cpu.instructions.LS attribute), 55

mnemonic (ducky.cpu.instructions.LW attribute), 55
 mnemonic (ducky.cpu.instructions.MOD attribute), 55
 mnemonic (ducky.cpu.instructions.MOV attribute), 55
 mnemonic (ducky.cpu.instructions.MUL attribute), 55
 mnemonic (ducky.cpu.instructions.NOP attribute), 55
 mnemonic (ducky.cpu.instructions.NOT attribute), 56
 mnemonic (ducky.cpu.instructions.OR attribute), 56
 mnemonic (ducky.cpu.instructions.OUTB attribute), 56
 mnemonic (ducky.cpu.instructions.OUTS attribute), 56
 mnemonic (ducky.cpu.instructions.OUTW attribute), 56
 mnemonic (ducky.cpu.instructions.POP attribute), 56
 mnemonic (ducky.cpu.instructions.PUSH attribute), 56
 mnemonic (ducky.cpu.instructions.RET attribute), 57
 mnemonic (ducky.cpu.instructions.RETINT attribute), 57
 mnemonic (ducky.cpu.instructions.RST attribute), 57
 mnemonic (ducky.cpu.instructions.SETE attribute), 57
 mnemonic (ducky.cpu.instructions.SETG attribute), 57
 mnemonic (ducky.cpu.instructions.SETGE attribute), 57
 mnemonic (ducky.cpu.instructions.SETL attribute), 57
 mnemonic (ducky.cpu.instructions.SETLE attribute), 57
 mnemonic (ducky.cpu.instructions.SETNE attribute), 57
 mnemonic (ducky.cpu.instructions.SETNO attribute), 58
 mnemonic (ducky.cpu.instructions.SETNS attribute), 58
 mnemonic (ducky.cpu.instructions.SETNZ attribute), 58
 mnemonic (ducky.cpu.instructions.SETO attribute), 58
 mnemonic (ducky.cpu.instructions.SETS attribute), 58
 mnemonic (ducky.cpu.instructions.SETZ attribute), 58
 mnemonic (ducky.cpu.instructions.SHIFTL attribute), 58
 mnemonic (ducky.cpu.instructions.SHIFTR attribute), 58
 mnemonic (ducky.cpu.instructions.SIS attribute), 58
 mnemonic (ducky.cpu.instructions.STB attribute), 58
 mnemonic (ducky.cpu.instructions.STI attribute), 59
 mnemonic (ducky.cpu.instructions.STS attribute), 59
 mnemonic (ducky.cpu.instructions.STW attribute), 59
 mnemonic (ducky.cpu.instructions.SUB attribute), 59
 mnemonic (ducky.cpu.instructions.SWP attribute), 59
 mnemonic (ducky.cpu.instructions.UDIV attribute), 59
 mnemonic (ducky.cpu.instructions.XOR attribute), 59
 MOD (class in ducky.cpu.instructions), 55
 MOD (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 Mode (class in ducky.devices.svga), 74
 MODL (class in ducky.cpu.coprocessor.math_copro), 38
 MODL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 39
 ModuleLoader (class in ducky.patch), 92
 MOV (class in ducky.cc), 33
 MOV (class in ducky.cpu.instructions), 55
 MOV (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 MUL (class in ducky.cc), 33
 MUL (class in ducky.cpu.instructions), 55
 MUL (ducky.cpu.instructions.DuckyOpcodes attribute), 50

MULL (class in ducky.cpu.coprocessor.math_copro), 38
 MULL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 39

N

name (ducky.hdt.HDTEntry_Argument attribute), 78
 name (ducky.mm.binary.RelocEntry attribute), 84
 name (ducky.mm.binary.SectionHeader attribute), 85
 name (ducky.mm.binary.SymbolEntry attribute), 86
 name() (ducky.cc.MemorySlotStorage method), 33
 name() (ducky.cc.StackSlotStorage method), 34
 name() (ducky.cc.SymbolStorage method), 35
 name_length (ducky.hdt.HDTEntry_Argument attribute), 78
 NamedValue (class in ducky.cc), 33
 NOP (class in ducky.cpu.instructions), 55
 NOP (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 NOT (class in ducky.cc), 33
 NOT (class in ducky.cpu.instructions), 56
 NOT (ducky.cpu.instructions.DuckyOpcodes attribute), 50
 NotAPointerError, 33
 nr_cores (ducky.hdt.HDTEntry_CPU attribute), 79
 nr_cpus (ducky.hdt.HDTEntry_CPU attribute), 79

O

offset (ducky.mm.binary.SectionHeader attribute), 85
 OFFSET_FMT() (in module ducky.mm), 92
 on_core_alive() (ducky.cpu.CPU method), 61
 on_core_alive() (ducky.machine.Machine method), 82
 on_core_halted() (ducky.cpu.CPU method), 61
 on_core_halted() (ducky.machine.Machine method), 82
 on_core_running() (ducky.cpu.CPU method), 61
 on_core_suspended() (ducky.cpu.CPU method), 61
 on_error (ducky.reactor.FDCallbacks attribute), 95
 on_read (ducky.reactor.FDCallbacks attribute), 95
 on_tick() (ducky.devices rtc.RTC task method), 69
 on_tick() (ducky.devices.svga.DisplayRefreshTask method), 74
 on_write (ducky.reactor.FDCallbacks attribute), 95
 onClose() (ducky.tools.vm.DuckyProtocol method), 101
 onMessage() (ducky.tools.vm.DuckyProtocol method), 101
 onOpen() (ducky.tools.vm.DuckyProtocol method), 101
 opcode (ducky.cpu.coprocessor.math_copro.ADDL attribute), 36
 opcode (ducky.cpu.coprocessor.math_copro.DECL attribute), 36
 opcode (ducky.cpu.coprocessor.math_copro.DIVL attribute), 37
 opcode (ducky.cpu.coprocessor.math_copro.DROP attribute), 37

opcode (ducky.cpu.coprocessor.math_copro.DUP attribute), 37

opcode (ducky.cpu.coprocessor.math_copro.DUP2 attribute), 37

opcode (ducky.cpu.coprocessor.math_copro.INCL attribute), 37

opcode (ducky.cpu.coprocessor.math_copro.LOAD attribute), 38

opcode (ducky.cpu.coprocessor.math_copro.LOADUW attribute), 38

opcode (ducky.cpu.coprocessor.math_copro.LOADW attribute), 38

opcode (ducky.cpu.coprocessor.math_copro.MODL attribute), 38

opcode (ducky.cpu.coprocessor.math_copro.MULL attribute), 38

opcode (ducky.cpu.coprocessor.math_copro.POP attribute), 40

opcode (ducky.cpu.coprocessor.math_copro.POPUW attribute), 40

opcode (ducky.cpu.coprocessor.math_copro.POPW attribute), 40

opcode (ducky.cpu.coprocessor.math_copro.PUSH attribute), 40

opcode (ducky.cpu.coprocessor.math_copro.PUSHW attribute), 41

opcode (ducky.cpu.coprocessor.math_copro.SAVE attribute), 41

opcode (ducky.cpu.coprocessor.math_copro.SAVEW attribute), 42

opcode (ducky.cpu.coprocessor.math_copro.SWP attribute), 42

opcode (ducky.cpu.coprocessor.math_copro.SYMDIVL attribute), 42

opcode (ducky.cpu.coprocessor.math_copro.SYMMODL attribute), 42

opcode (ducky.cpu.coprocessor.math_copro.UDIVL attribute), 42

opcode (ducky.cpu.coprocessor.math_copro.UMODL attribute), 42

opcode (ducky.cpu.instructions.ADD attribute), 45

opcode (ducky.cpu.instructions.AND attribute), 45

opcode (ducky.cpu.instructions.CALL attribute), 46

opcode (ducky.cpu.instructions.CAS attribute), 46

opcode (ducky.cpu.instructions.CLI attribute), 46

opcode (ducky.cpu.instructions.CMP attribute), 47

opcode (ducky.cpu.instructions.CMPU attribute), 47

opcode (ducky.cpu.instructions.CTR attribute), 47

opcode (ducky.cpu.instructions.CTW attribute), 47

opcode (ducky.cpu.instructions.DEC attribute), 47

opcode (ducky.cpu.instructions.Descriptor attribute), 48

opcode (ducky.cpu.instructions.DIV attribute), 47

opcode (ducky.cpu.instructions.EncodingA attribute), 51

opcode (ducky.cpu.instructions.EncodingC attribute), 51

opcode (ducky.cpu.instructions.EncodingI attribute), 52

opcode (ducky.cpu.instructions.EncodingR attribute), 52

opcode (ducky.cpu.instructions.FPTC attribute), 52

opcode (ducky.cpu.instructions.HLT attribute), 52

opcode (ducky.cpu.instructions.IDLE attribute), 53

opcode (ducky.cpu.instructions.INB attribute), 53

opcode (ducky.cpu.instructions.INC attribute), 53

opcode (ducky.cpu.instructions.INS attribute), 53

opcode (ducky.cpu.instructions.INT attribute), 53

opcode (ducky.cpu.instructions.INW attribute), 53

opcode (ducky.cpu.instructions.IPI attribute), 53

opcode (ducky.cpu.instructions.J attribute), 54

opcode (ducky.cpu.instructions.LA attribute), 54

opcode (ducky.cpu.instructions.LB attribute), 54

opcode (ducky.cpu.instructions.LI attribute), 54

opcode (ducky.cpu.instructions.LIU attribute), 54

opcode (ducky.cpu.instructions.LPM attribute), 55

opcode (ducky.cpu.instructions.LS attribute), 55

opcode (ducky.cpu.instructions.LW attribute), 55

opcode (ducky.cpu.instructions.MOD attribute), 55

opcode (ducky.cpu.instructions.MOV attribute), 55

opcode (ducky.cpu.instructions.MUL attribute), 55

opcode (ducky.cpu.instructions.NOP attribute), 55

opcode (ducky.cpu.instructions.NOT attribute), 56

opcode (ducky.cpu.instructions.OR attribute), 56

opcode (ducky.cpu.instructions.OUTB attribute), 56

opcode (ducky.cpu.instructions.OUTS attribute), 56

opcode (ducky.cpu.instructions.OUTW attribute), 56

opcode (ducky.cpu.instructions.POP attribute), 56

opcode (ducky.cpu.instructions.PUSH attribute), 56

opcode (ducky.cpu.instructions.RET attribute), 57

opcode (ducky.cpu.instructions.RETINT attribute), 57

opcode (ducky.cpu.instructions.RST attribute), 57

opcode (ducky.cpu.instructions.SHIFTL attribute), 58

opcode (ducky.cpu.instructions.SHIFTR attribute), 58

opcode (ducky.cpu.instructions.SIS attribute), 58

opcode (ducky.cpu.instructions.STB attribute), 58

opcode (ducky.cpu.instructions.STI attribute), 59

opcode (ducky.cpu.instructions.STS attribute), 59

opcode (ducky.cpu.instructions.STW attribute), 59

opcode (ducky.cpu.instructions.SUB attribute), 59

opcode (ducky.cpu.instructions.SWP attribute), 59

opcode (ducky.cpu.instructions.UDIV attribute), 59

opcode (ducky.cpu.instructions.XOR attribute), 59

opcode_desc_map (ducky.cpu.coprocessor.math_copro.MathCoproprocessorInstructionSet attribute), 39

opcode_desc_map (ducky.cpu.instructions.DuckyInstructionSet attribute), 49

opcode_encoding_map (ducky.cpu.coprocessor.math_copro.MathCoproprocessorInstructionSet attribute), 39

opcode_encoding_map (ducky.cpu.instructions.DuckyInstructionSet attribute), 49

opcodes (ducky.cpu.coprocessor.math_copro.MathCoproprocessorInstructionSet attribute), 39

- ul style="list-style-type: none; padding-left: 0;">
- opcodes (ducky.cpu.instructions.DuckyInstructionSet attribute), [49](#)
- opcodes (ducky.cpu.instructions.InstructionSet attribute), [54](#)
- open() (ducky.mm.binary.File static method), [84](#)
- open() (ducky.snapshot.CoreDumpFile static method), [97](#)
- open() (ducky.util.BinaryFile static method), [103](#)
- operands (ducky.cpu.coprocessor.math_copro.Descriptor_MATH attribute), [37](#)
- operands (ducky.cpu.coprocessor.math_copro.LOAD attribute), [38](#)
- operands (ducky.cpu.coprocessor.math_copro.LOADUW attribute), [38](#)
- operands (ducky.cpu.coprocessor.math_copro.LOADW attribute), [38](#)
- operands (ducky.cpu.coprocessor.math_copro.SAVE attribute), [41](#)
- operands (ducky.cpu.coprocessor.math_copro.SAVEW attribute), [42](#)
- operands (ducky.cpu.instructions.CAS attribute), [46](#)
- operands (ducky.cpu.instructions.Descriptor attribute), [48](#)
- operands (ducky.cpu.instructions.Descriptor_I attribute), [48](#)
- operands (ducky.cpu.instructions.Descriptor_R attribute), [48](#)
- operands (ducky.cpu.instructions.Descriptor_R_I attribute), [49](#)
- operands (ducky.cpu.instructions.Descriptor_R_R attribute), [49](#)
- operands (ducky.cpu.instructions.Descriptor_R_RI attribute), [49](#)
- operands (ducky.cpu.instructions.Descriptor_RI attribute), [48](#)
- operands (ducky.cpu.instructions.Descriptor_RI_R attribute), [48](#)
- OR (class in ducky.cc), [33](#)
- OR (class in ducky.cpu.instructions), [56](#)
- OR (ducky.cpu.instructions.DuckyOpcodes attribute), [50](#)
- OUTB (class in ducky.cpu.instructions), [56](#)
- OUTB (ducky.cpu.instructions.DuckyOpcodes attribute), [50](#)
- OutputStream (class in ducky.streams), [98](#)
- OUTS (class in ducky.cpu.instructions), [56](#)
- OUTS (ducky.cpu.instructions.DuckyOpcodes attribute), [50](#)
- OUTW (class in ducky.cpu.instructions), [56](#)
- OUTW (ducky.cpu.instructions.DuckyOpcodes attribute), [50](#)
- ## P
- padding (ducky.mm.binary.SectionHeader attribute), [85](#)
 - PAGE_SIZE (in module ducky.mm), [92](#)
 - pages_in_area() (ducky.mm.MemoryController method), [89](#)
 - PageTableEntry (class in ducky.mm), [92](#)
 - parse_io_streams() (in module ducky.devices.terminal), [72](#)
 - parse_options() (in module ducky.tools), [102](#)
 - parse_template() (in module ducky.tools.defs), [100](#)
 - patch_add (ducky.mm.binary.RelocEntry attribute), [84](#)
 - patch_address (ducky.mm.binary.RelocEntry attribute), [84](#)
 - patch_offset (ducky.mm.binary.RelocEntry attribute), [84](#)
 - patch_section (ducky.mm.binary.RelocEntry attribute), [84](#)
 - patch_size (ducky.mm.binary.RelocEntry attribute), [84](#)
 - pattern (ducky.cpu.instructions.Descriptor attribute), [48](#)
 - PATTERN() (in module ducky.cpu.assemble), [44](#)
 - Point (class in ducky.debugging), [67](#)
 - PointerType (class in ducky.cc.types), [30](#)
 - poke() (ducky.boot.ROMLoader method), [24](#)
 - POP (class in ducky.cc), [33](#)
 - POP (class in ducky.cpu.coprocessor.math_copro), [40](#)
 - POP (class in ducky.cpu.instructions), [56](#)
 - POP (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), [39](#)
 - POP (ducky.cpu.instructions.DuckyOpcodes attribute), [50](#)
 - pop() (ducky.cpu.coprocessor.math_copro.RegisterSet method), [41](#)
 - pop() (ducky.cpu.CPUCore method), [62](#)
 - pop_scope() (ducky.cc.passes.ast_codegen.CodegenVisitor method), [27](#)
 - POPUW (class in ducky.cpu.coprocessor.math_copro), [40](#)
 - POPUW (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), [39](#)
 - POPW (class in ducky.cpu.coprocessor.math_copro), [40](#)
 - POPW (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), [39](#)
 - PORT_COUNT (ducky.devices.IOPorts attribute), [76](#)
 - post_memory() (ducky.debugging.DebuggingSet method), [66](#)
 - post_step() (ducky.debugging.DebuggingSet method), [66](#)
 - pre_memory() (ducky.debugging.DebuggingSet method), [66](#)
 - pre_step() (ducky.debugging.DebuggingSet method), [66](#)
 - print_machine_stats() (in module ducky.tools.vm), [102](#)
 - print_node() (ducky.snapshot.SnapshotNode method), [97](#)
 - priority (ducky.cc.passes.ast_codegen.CodegenVisitor attribute), [27](#)
 - priority (ducky.cc.passes.ast_constprop.ConstantFoldingVisitor attribute), [28](#)
 - priority (ducky.cc.passes.ast_dce.DSEVisitor attribute), [28](#)
 - priority (ducky.cc.passes.ASTVisitor attribute), [30](#)
 - priority (ducky.cc.passes.BlockVisitor attribute), [30](#)
 - priority (ducky.cc.passes.bt_peephole.BTPeepholeVisitor attribute), [29](#)

- ul style="list-style-type: none; padding-left: 0;">
- priority (ducky.cc.passes.bt_simplify.BlockTreeSimplifyVisitor attribute), 29
- priority (ducky.cc.passes.bt_visualise.BlockTreeVisualiseVisitor attribute), 29
- process_cond() (ducky.cc.passes.ast_codegen.CodegenVisitor method), 27
- process_config_options() (in module ducky.tools.vm), 102
- process_files() (in module ducky.tools.ld), 100
- ProfileRecord (class in ducky.profiler), 94
- ProfilerStore (class in ducky.profiler), 94
- prolog_block() (ducky.cc.Function method), 32
- prompt() (ducky.console.ConsoleConnection method), 26
- PUSH (class in ducky.cc), 34
- PUSH (class in ducky.cpu.coprocessor.math_copro), 40
- PUSH (class in ducky.cpu.instructions), 56
- PUSH (ducky.cpu.coprocessor.math_copro.MathCoproprocessorRegisters attribute), 39
- PUSH (ducky.cpu.instructions.DuckyOpcodes attribute), 50
- push() (ducky.cpu.coprocessor.math_copro.RegisterSet method), 41
- push() (ducky.cpu.CPUCore method), 62
- push_scope() (ducky.cc.passes.ast_codegen.CodegenVisitor method), 27
- PUSHW (class in ducky.cpu.coprocessor.math_copro), 40
- PUSHW (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 39
- put() (ducky.boot.MMapMemoryPage method), 24
- put() (ducky.cc.Register method), 34
- put() (ducky.devices.svga.SimpleVGAMemoryPage method), 75
- put() (ducky.mm.ExternalMemoryPage method), 87
- put_buffer() (ducky.cpu.assemble.Buffer method), 43
- put_line() (ducky.cpu.assemble.Buffer method), 43
- put_string() (ducky.util.StringTable method), 104
- ## R
- R00 (ducky.cpu.registers.Registers attribute), 60
 - R01 (ducky.cpu.registers.Registers attribute), 60
 - R02 (ducky.cpu.registers.Registers attribute), 60
 - R03 (ducky.cpu.registers.Registers attribute), 60
 - R04 (ducky.cpu.registers.Registers attribute), 60
 - R05 (ducky.cpu.registers.Registers attribute), 60
 - R06 (ducky.cpu.registers.Registers attribute), 60
 - R07 (ducky.cpu.registers.Registers attribute), 60
 - R08 (ducky.cpu.registers.Registers attribute), 60
 - R09 (ducky.cpu.registers.Registers attribute), 60
 - R10 (ducky.cpu.registers.Registers attribute), 60
 - R11 (ducky.cpu.registers.Registers attribute), 60
 - R12 (ducky.cpu.registers.Registers attribute), 60
 - R13 (ducky.cpu.registers.Registers attribute), 60
 - R14 (ducky.cpu.registers.Registers attribute), 60
 - R15 (ducky.cpu.registers.Registers attribute), 60
 - R16 (ducky.cpu.registers.Registers attribute), 60
 - R17 (ducky.cpu.registers.Registers attribute), 60
 - R18 (ducky.cpu.registers.Registers attribute), 60
 - R19 (ducky.cpu.registers.Registers attribute), 60
 - R20 (ducky.cpu.registers.Registers attribute), 60
 - R21 (ducky.cpu.registers.Registers attribute), 60
 - R22 (ducky.cpu.registers.Registers attribute), 60
 - R23 (ducky.cpu.registers.Registers attribute), 60
 - R24 (ducky.cpu.registers.Registers attribute), 61
 - R25 (ducky.cpu.registers.Registers attribute), 61
 - R26 (ducky.cpu.registers.Registers attribute), 61
 - R27 (ducky.cpu.registers.Registers attribute), 61
 - R28 (ducky.cpu.registers.Registers attribute), 61
 - R29 (ducky.cpu.registers.Registers attribute), 61
 - raw_pop() (ducky.cpu.CPUCore method), 62
 - raw_push() (ducky.cpu.CPUCore method), 62
 - Reactor (class in ducky.reactor), 95
 - READ (ducky.mm.PageTableEntry attribute), 92
 - read() (ducky.config.MachineConfig method), 26
 - read() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 35
 - read() (ducky.streams.InputStream method), 98
 - read() (ducky.streams.OutputStream method), 98
 - read() (ducky.streams.Stream method), 99
 - read() (ducky.tools.vm.WSInputStream method), 102
 - read_blocks() (ducky.devices.storage.Storage method), 25
 - read_cr0() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 35
 - read_cr1() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 36
 - read_cr2() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 36
 - read_cr3() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 36
 - read_in() (ducky.machine.CommQueue method), 81
 - read_input() (ducky.console.ConsoleConnection method), 26
 - read_out() (ducky.machine.CommQueue method), 81
 - read_profiling_data() (in module ducky.tools.profile), 101
 - read_struct() (ducky.util.BinaryFile method), 103
 - read_u16() (ducky.devices.IOProvider method), 76
 - read_u16() (ducky.devices.svga.SimpleVGA method), 75
 - read_u16() (ducky.mm.AnonymousMemoryPage method), 87
 - read_u16() (ducky.mm.ExternalMemoryPage method), 87
 - read_u16() (ducky.mm.MemoryController method), 90
 - read_u16() (ducky.mm.MemoryPage method), 90
 - read_u32() (ducky.devices.IOProvider method), 76
 - read_u32() (ducky.devices.storage.BlockIO method), 70
 - read_u32() (ducky.mm.AnonymousMemoryPage method), 87

[read_u32\(\)](#) (ducky.mm.ExternalMemoryPage method), [87](#)
[read_u32\(\)](#) (ducky.mm.MemoryController method), [90](#)
[read_u32\(\)](#) (ducky.mm.MemoryPage method), [91](#)
[read_u8\(\)](#) (ducky.devices.IOProvider method), [76](#)
[read_u8\(\)](#) (ducky.devices.keyboard.Backend method), [69](#)
[read_u8\(\)](#) (ducky.devices.rtc.RTC method), [69](#)
[read_u8\(\)](#) (ducky.mm.AnonymousMemoryPage method), [87](#)
[read_u8\(\)](#) (ducky.mm.ExternalMemoryPage method), [87](#)
[read_u8\(\)](#) (ducky.mm.MemoryController method), [90](#)
[read_u8\(\)](#) (ducky.mm.MemoryPage method), [91](#)
[readable](#) (ducky.mm.binary.SectionFlagsEncoding attribute), [85](#)
[ReadOnlyRegisterError](#), [36](#)
[RealCPUCoreProfiler](#) (class in ducky.profiler), [94](#)
[RED\(\)](#) (in module ducky.log), [80](#)
[Reference](#) (class in ducky.cpu.assemble), [44](#)
[REFRESH](#) (ducky.devices.svga.SimpleVGACommands attribute), [75](#)
[reg](#) (ducky.cpu.instructions.EncodingC attribute), [51](#)
[reg](#) (ducky.cpu.instructions.EncodingI attribute), [52](#)
[REG\(\)](#) (ducky.cpu.CPUCore method), [62](#)
[reg1](#) (ducky.cpu.instructions.EncodingA attribute), [51](#)
[reg1](#) (ducky.cpu.instructions.EncodingR attribute), [52](#)
[reg2](#) (ducky.cpu.instructions.EncodingA attribute), [51](#)
[reg2](#) (ducky.cpu.instructions.EncodingR attribute), [52](#)
[reg3](#) (ducky.cpu.instructions.EncodingA attribute), [51](#)
[region_id](#) (ducky.mm.MemoryRegion attribute), [92](#)
[Register](#) (class in ducky.cc), [34](#)
[register_command\(\)](#) (ducky.console.ConsoleMaster method), [27](#)
[register_commands\(\)](#) (ducky.console.ConsoleMaster method), [27](#)
[REGISTER_COUNT](#) (ducky.cpu.registers.Registers attribute), [61](#)
[register_page\(\)](#) (ducky.mm.MemoryController method), [90](#)
[register_port\(\)](#) (ducky.machine.Machine method), [83](#)
[REGISTER_SPECIAL](#) (ducky.cpu.registers.Registers attribute), [61](#)
[register_with_reactor\(\)](#) (ducky.streams.Stream method), [99](#)
[register_with_reactor\(\)](#) (ducky.tools.vm.WSInputStream method), [102](#)
[RegisterMemorySlotValue](#) (class in ducky.cc), [34](#)
[Registers](#) (class in ducky.cpu.registers), [60](#)
[RegisterSet](#) (class in ducky.cc), [34](#)
[RegisterSet](#) (class in ducky.cpu.coprocessor.math_copro), [41](#)
[RegisterSet](#) (class in ducky.cpu.registers), [60](#)
[RegisterValue](#) (class in ducky.cc), [34](#)
[relative](#) (ducky.mm.binary.RelocFlagsEncoding attribute), [85](#)
[relative_address](#) (ducky.cpu.instructions.Descriptor attribute), [48](#)
[relative_address](#) (ducky.cpu.instructions.LA attribute), [54](#)
[release_ptes\(\)](#) (ducky.cpu.MMU method), [65](#)
[release_register\(\)](#) (ducky.cc.SymbolStorage method), [35](#)
[RELOC](#) (ducky.mm.binary.SectionTypes attribute), [86](#)
[RelocEntry](#) (class in ducky.mm.binary), [84](#)
[RelocFlags](#) (class in ducky.mm.binary), [84](#)
[RelocFlagsEncoding](#) (class in ducky.mm.binary), [84](#)
[RelocSection](#) (class in ducky.cpu.assemble), [44](#)
[RelocSlot](#) (class in ducky.cpu.assemble), [44](#)
[remove_fd\(\)](#) (ducky.reactor.Reactor method), [95](#)
[remove_fd\(\)](#) (ducky.reactor.SelectTask method), [96](#)
[remove_listener\(\)](#) (ducky.machine.EventBus method), [81](#)
[remove_point\(\)](#) (ducky.debugging.DebuggingSet method), [66](#)
[remove_task\(\)](#) (ducky.reactor.Reactor method), [96](#)
[RemoveLoggingVisitor](#) (class in ducky.patch), [92](#)
[replace_child\(\)](#) (ducky.cc.passes.ASTOptVisitor method), [29](#)
[RESET](#) (ducky.devices.svga.SimpleVGACommands attribute), [75](#)
[reset\(\)](#) (ducky.cpu.CPUCore method), [63](#)
[reset\(\)](#) (ducky.cpu.MMU method), [65](#)
[reset\(\)](#) (ducky.devices.storage.BlockIO method), [70](#)
[reset\(\)](#) (ducky.devices.svga.SimpleVGA method), [75](#)
[reset_scope\(\)](#) (ducky.cc.passes.ast_codegen.CodegenVisitor method), [28](#)
[resolve_relocations\(\)](#) (in module ducky.tools.ld), [100](#)
[resolve_symbols\(\)](#) (in module ducky.tools.ld), [100](#)
[restore_callee_saves\(\)](#) (ducky.cc.RegisterSet method), [34](#)
[RET](#) (class in ducky.cc), [34](#)
[RET](#) (class in ducky.cpu.instructions), [56](#)
[RET](#) (ducky.cpu.instructions.DuckyOpcodes attribute), [50](#)
[RETINT](#) (class in ducky.cpu.instructions), [57](#)
[RETINT](#) (ducky.cpu.instructions.DuckyOpcodes attribute), [50](#)
[RI_ADDR\(\)](#) (in module ducky.cpu.instructions), [57](#)
[RI_VAL\(\)](#) (in module ducky.cpu.instructions), [57](#)
[RODataSection](#) (class in ducky.cpu.assemble), [44](#)
[ROMLoader](#) (class in ducky.boot), [24](#)
[ROWS](#) (ducky.devices.svga.SimpleVGACommands attribute), [75](#)
[RST](#) (class in ducky.cpu.instructions), [57](#)
[RST](#) (ducky.cpu.instructions.DuckyOpcodes attribute), [50](#)
[RTC](#) (class in ducky.devices.rtc), [69](#)
[RTCTask](#) (class in ducky.devices.rtc), [69](#)
[run\(\)](#) (ducky.cpu.CPUCore method), [63](#)
[run\(\)](#) (ducky.debugging.VMDebugInterrupt method), [68](#)
[run\(\)](#) (ducky.devices.tty.FrontendFlushTask method), [73](#)
[run\(\)](#) (ducky.interfaces.IMachineWorker method), [80](#)
[run\(\)](#) (ducky.interfaces.IReactorTask method), [80](#)

run() (ducky.interfaces.IVirtualInterrupt method), 80
run() (ducky.machine.HaltMachineTask method), 81
run() (ducky.machine.IRQRouterTask method), 81
run() (ducky.machine.Machine method), 83
run() (ducky.reactor.CallInReactorTask method), 95
run() (ducky.reactor.Reactor method), 96
run() (ducky.reactor.RunInIntervalTask method), 96
run() (ducky.reactor.SelectTask method), 97
run_machine() (ducky.tools.vm.DuckyProtocol method), 101
RunInIntervalTask (class in ducky.reactor), 96
RVALUE (ducky.cc.ExpressionClass attribute), 32
RValueExpression (class in ducky.cc), 34

S

SAVE (class in ducky.cpu.coprocessor.math_copro), 41
SAVE (ducky.cpu.coprocessor.math_copro.MathCoproprocessor attribute), 40
save() (ducky.mm.binary.File method), 84
save() (ducky.profiler.ProfilerStore method), 94
save() (ducky.snapshot.CoreDumpFile method), 97
save() (ducky.snapshot.VMState method), 97
save_callee_saves() (ducky.cc.RegisterSet method), 34
save_encoding() (ducky.util.Flags method), 103
save_object_file() (in module ducky.tools.as), 99
save_snapshot() (ducky.devices.snapshot.FileSnapshotStorage method), 70
save_snapshot() (ducky.devices.snapshot.SnapshotStorage method), 70
save_state() (ducky.boot.MMapArea method), 23
save_state() (ducky.cpu.coprocessor.math_copro.MathCoproprocessor method), 39
save_state() (ducky.cpu.coprocessor.math_copro.RegisterSet method), 41
save_state() (ducky.cpu.CPU method), 61
save_state() (ducky.cpu.CPUCore method), 63
save_state() (ducky.interfaces.ISnapshotable method), 80
save_state() (ducky.machine.Machine method), 83
save_state() (ducky.mm.ExternalMemoryPage method), 87
save_state() (ducky.mm.MemoryController method), 90
save_state() (ducky.mm.MemoryPage method), 91
save_state() (ducky.mm.MemoryRegion method), 92
SAVEW (class in ducky.cpu.coprocessor.math_copro), 41
SAVEW (ducky.cpu.coprocessor.math_copro.MathCoproprocessor attribute), 40
Scope (class in ducky.cc), 34
scope_id (ducky.cc.Scope attribute), 34
Section (class in ducky.cpu.assemble), 44
section (ducky.mm.binary.SymbolEntry attribute), 86
SectionFlags (class in ducky.mm.binary), 85
SectionFlagsEncoding (class in ducky.mm.binary), 85
SectionHeader (class in ducky.mm.binary), 85
sections (ducky.mm.binary.FileHeader attribute), 84
sections() (ducky.mm.binary.File method), 84
SectionTypes (class in ducky.mm.binary), 86
SelectTask (class in ducky.reactor), 96
SET (ducky.cpu.instructions.DuckyOpcodes attribute), 50
set() (ducky.config.MachineConfig method), 26
set_access_methods() (ducky.cpu.MMU method), 65
set_backend() (ducky.devices.DeviceFrontend method), 76
set_content() (ducky.mm.binary.File method), 84
set_frontend() (ducky.devices.DeviceBackend method), 76
set_mode() (ducky.devices.svga.SimpleVGA method), 75
set_output() (ducky.devices.tty.Frontend method), 73
set_output() (ducky.devices.tty.FrontendFlushTask method), 73
SETC (class in ducky.cpu.instructions), 57
SETG (class in ducky.cpu.instructions), 57
SETGE (class in ducky.cpu.instructions), 57
SETL (class in ducky.cpu.instructions), 57
SETLE (class in ducky.cpu.instructions), 57
SETNE (class in ducky.cpu.instructions), 57
SETNO (class in ducky.cpu.instructions), 57
SETNS (class in ducky.cpu.instructions), 58
SETNZ (class in ducky.cpu.instructions), 58
SETO (class in ducky.cpu.instructions), 58
SETS (class in ducky.cpu.instructions), 58
setup() (ducky.mm.binary.File method), 84
setup() (ducky.util.BinaryFile method), 103
setup_bootloader() (ducky.boot.ROMLoader method), 24
setup_debugging() (ducky.boot.ROMLoader method), 25
setup_devices() (ducky.machine.Machine method), 83
setup_hdt() (ducky.boot.ROMLoader method), 25
setup_logger() (in module ducky.tools), 102
setup_mmmaps() (ducky.boot.ROMLoader method), 25
SETZ (class in ducky.cpu.instructions), 58
SHIFTL (class in ducky.cpu.instructions), 58
SHIFTL (ducky.cpu.instructions.DuckyOpcodes attribute), 51
SHIFTR (class in ducky.cpu.instructions), 58
SHIFTR (ducky.cpu.instructions.DuckyOpcodes attribute), 51
SHL (class in ducky.cc), 34
SHORT (ducky.mm.binary.SymbolDataTypes attribute), 86
ShortSlot (class in ducky.cpu.assemble), 44
show_cores() (in module ducky.tools.coredump), 100
show_disassemble() (in module ducky.tools.objdump), 101
show_file_header() (in module ducky.tools.objdump), 101
show_forth_dict() (in module ducky.tools.coredump), 100
show_forth_trace() (in module ducky.tools.coredump), 100

[show_forth_word\(\)](#) (in module `ducky.tools.coredump`), [100](#)
[show_header\(\)](#) (in module `ducky.tools.coredump`), [100](#)
[show_memory\(\)](#) (in module `ducky.tools.coredump`), [100](#)
[show_node\(\)](#) (in module `ducky.cc`), [35](#)
[show_pages\(\)](#) (in module `ducky.tools.coredump`), [100](#)
[show_reloc\(\)](#) (in module `ducky.tools.objdump`), [101](#)
[show_sections\(\)](#) (in module `ducky.tools.objdump`), [101](#)
[show_symbols\(\)](#) (in module `ducky.tools.objdump`), [101](#)
[SHR](#) (class in `ducky.cc`), [34](#)
[sign_extend_immediate\(\)](#)
 (`ducky.cpu.instructions.Encoding` static method), [51](#)
[sign_extend_immediate\(\)](#)
 (`ducky.cpu.instructions.EncodingC` static method), [51](#)
[sign_extend_immediate\(\)](#)
 (`ducky.cpu.instructions.EncodingI` static method), [52](#)
[sign_extend_immediate\(\)](#)
 (`ducky.cpu.instructions.EncodingR` static method), [52](#)
[sign_extend_with_push\(\)](#)
 (`ducky.cpu.coprocessor.math_copro.MathCoproces` static method), [39](#)
[SimpleVGA](#) (class in `ducky.devices.svga`), [74](#)
[SimpleVGACommands](#) (class in `ducky.devices.svga`), [75](#)
[SimpleVGAMemoryPage](#) (class in `ducky.devices.svga`), [75](#)
[SIS](#) (class in `ducky.cpu.instructions`), [58](#)
[SIS](#) (`ducky.cpu.coprocessor.math_copro.MathCoproces` static attribute), [40](#)
[SIS](#) (`ducky.cpu.instructions.DuckyOpcodes` attribute), [51](#)
[SIZE](#) (`ducky.cpu.InterruptVector` attribute), [64](#)
[size](#) (`ducky.hdt.HDTEntry_Memory` attribute), [79](#)
[size](#) (`ducky.mm.binary.SymbolEntry` attribute), [86](#)
[SIZE_FMT\(\)](#) (in module `ducky.mm`), [92](#)
[sizeof\(\)](#) (in module `ducky.cpu.assemble`), [45](#)
[sizeof_fmt\(\)](#) (in module `ducky.util`), [104](#)
[SnapshotNode](#) (class in `ducky.snapshot`), [97](#)
[SnapshotStorage](#) (class in `ducky.devices.snapshot`), [70](#)
[SourceLocation](#) (class in `ducky.cpu.assemble`), [44](#)
[SP](#) (`ducky.cpu.registers.Registers` attribute), [61](#)
[SP\(\)](#) (`ducky.cpu.CPUCore` method), [62](#)
[SpaceSlot](#) (class in `ducky.cpu.assemble`), [44](#)
[spill_register\(\)](#) (`ducky.cc.SymbolStorage` method), [35](#)
[STACK_DEPTH](#) (in module `ducky.cpu.coprocessor.math_copro`), [42](#)
[StackFrame](#) (class in `ducky.cpu`), [65](#)
[StackSlotStorage](#) (class in `ducky.cc`), [34](#)
[StackSlotValue](#) (class in `ducky.cc`), [35](#)
[StandalonePTYTerminal](#) (class in `ducky.devices.terminal`), [72](#)
[StandardIOTerminal](#) (class in `ducky.devices.terminal`), [72](#)
[STB](#) (class in `ducky.cc`), [34](#)
[STB](#) (class in `ducky.cpu.instructions`), [58](#)
[STB](#) (`ducky.cpu.instructions.DuckyOpcodes` attribute), [51](#)
[StderrStream](#) (class in `ducky.streams`), [98](#)
[StdinStream](#) (class in `ducky.streams`), [98](#)
[StdoutStream](#) (class in `ducky.streams`), [98](#)
[step\(\)](#) (`ducky.cpu.CPUCore` method), [63](#)
[STI](#) (class in `ducky.cpu.instructions`), [58](#)
[STI](#) (`ducky.cpu.instructions.DuckyOpcodes` attribute), [51](#)
[Storage](#) (class in `ducky.devices.storage`), [71](#)
[StorageAccessError](#), [72](#)
[STORE](#) (in module `ducky.profiler`), [94](#)
[str2int\(\)](#) (in module `ducky.util`), [104](#)
[Stream](#) (class in `ducky.streams`), [98](#)
[StreamHandler](#) (class in `ducky.log`), [80](#)
[StreamIOTerminal](#) (class in `ducky.devices.terminal`), [72](#)
[STRING](#) (`ducky.mm.binary.SymbolDataTypes` attribute), [86](#)
[StringConstantValue](#) (class in `ducky.cc`), [35](#)
[STRINGS](#) (`ducky.mm.binary.SectionTypes` attribute), [86](#)
[StringSlot](#) (class in `ducky.cpu.assemble`), [44](#)
[StringTable](#) (class in `ducky.util`), [104](#)
[StructType](#) (class in `ducky.cc.types`), [30](#)
[STS](#) (class in `ducky.cc`), [34](#)
[STS](#) (class in `ducky.cpu.instructions`), [59](#)
[STS](#) (`ducky.cpu.instructions.DuckyOpcodes` attribute), [51](#)
[STW](#) (class in `ducky.cc`), [34](#)
[STW](#) (class in `ducky.cpu.instructions`), [59](#)
[STW](#) (`ducky.cpu.instructions.DuckyOpcodes` attribute), [51](#)
[SUB](#) (class in `ducky.cc`), [34](#)
[SUB](#) (class in `ducky.cpu.instructions`), [59](#)
[SUB](#) (`ducky.cpu.instructions.DuckyOpcodes` attribute), [51](#)
[suspend\(\)](#) (`ducky.cpu.CPU` method), [61](#)
[suspend\(\)](#) (`ducky.cpu.CPUCore` method), [63](#)
[suspend\(\)](#) (`ducky.interfaces.IMachineWorker` method), [80](#)
[suspend\(\)](#) (`ducky.machine.Machine` method), [83](#)
[SuspendCoreAction](#) (class in `ducky.debugging`), [68](#)
[SWP](#) (class in `ducky.cpu.coprocessor.math_copro`), [42](#)
[SWP](#) (class in `ducky.cpu.instructions`), [59](#)
[SWP](#) (`ducky.cpu.coprocessor.math_copro.MathCoproces` static attribute), [40](#)
[SWP](#) (`ducky.cpu.instructions.DuckyOpcodes` attribute), [51](#)
[Symbol](#) (class in `ducky.cc`), [35](#)
[symbol_type](#) (`ducky.cpu.assemble.AsciiSlot` attribute), [43](#)
[symbol_type](#) (`ducky.cpu.assemble.ByteSlot` attribute), [43](#)
[symbol_type](#) (`ducky.cpu.assemble.BytesSlot` attribute), [43](#)
[symbol_type](#) (`ducky.cpu.assemble.CharSlot` attribute), [43](#)
[symbol_type](#) (`ducky.cpu.assemble.FunctionSlot` attribute), [44](#)

- ul style="list-style-type: none; padding-left: 0;">
- symbol_type (ducky.cpu.assemble.IntSlot attribute), 44
- symbol_type (ducky.cpu.assemble.ShortSlot attribute), 44
- symbol_type (ducky.cpu.assemble.SpaceSlot attribute), 44
- symbol_type (ducky.cpu.assemble.StringSlot attribute), 44
- SymbolAlreadyDefinedError, 35
- SymbolConflictError, 35
- SymbolDataTypes (class in ducky.mm.binary), 86
- SymbolEntry (class in ducky.mm.binary), 86
- SymbolFlags (class in ducky.mm.binary), 86
- SymbolFlagsEncoding (class in ducky.mm.binary), 87
- SYMBOLS (ducky.mm.binary.SectionTypes attribute), 86
- SymbolsSection (class in ducky.cpu.assemble), 44
- SymbolStorage (class in ducky.cc), 35
- SymbolTable (class in ducky.util), 104
- SymbolUndefined, 35
- SYMDIVL (class in ducky.cpu.coprocessor.math_copro), 42
- SYMDIVL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 40
- SYMMODL (class in ducky.cpu.coprocessor.math_copro), 42
- SYMMODL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 40
- ## T
- table() (ducky.console.ConsoleConnection method), 26
 - take_sample() (ducky.profiler.DummyCPUCoreProfiler method), 93
 - take_sample() (ducky.profiler.RealCPUCoreProfiler method), 94
 - task_runnable() (ducky.reactor.Reactor method), 96
 - task_suspended() (ducky.reactor.Reactor method), 96
 - tenh() (ducky.devices.tty.Backend method), 73
 - tenh() (ducky.machine.Machine method), 83
 - tenh_close_stream() (ducky.devices.tty.Backend method), 73
 - tenh_enable() (ducky.devices.tty.Backend method), 73
 - tenh_enable() (ducky.devices.tty.Frontend method), 73
 - tenh_flush_stream() (ducky.devices.tty.Backend method), 73
 - Terminal (class in ducky.devices.terminal), 72
 - TerminalConsoleConnection (class in ducky.console), 27
 - TEXT (ducky.mm.binary.SectionTypes attribute), 86
 - TextSection (class in ducky.cpu.assemble), 45
 - TIMER (ducky.devices.IRQList attribute), 76
 - to_encoding() (ducky.util.Flags method), 103
 - to_int() (ducky.util.Flags method), 103
 - to_pretty_string() (ducky.devices.svga.Mode method), 74
 - to_rvalue() (ducky.cc.Expression method), 32
 - to_string() (ducky.devices.svga.Mode method), 74
 - to_string() (ducky.util.Flags method), 103
 - to_u8() (ducky.devices.svga.Char method), 74
 - TooManyLabelsError, 77
 - tos() (ducky.cpu.coprocessor.math_copro.RegisterSet method), 41
 - tos1() (ducky.cpu.coprocessor.math_copro.RegisterSet method), 41
 - translate_buffer() (in module ducky.cpu.assemble), 45
 - translate_buffer() (in module ducky.tools.as), 99
 - trigger() (ducky.machine.EventBus method), 81
 - trigger_irq() (ducky.machine.Machine method), 83
 - type (ducky.hdt.HDTEntry_Argument attribute), 78
 - type (ducky.hdt.HDTEntry_CPU attribute), 79
 - type (ducky.hdt.HDTEntry_Memory attribute), 79
 - type (ducky.mm.binary.SectionHeader attribute), 86
 - type (ducky.mm.binary.SymbolEntry attribute), 86
 - types (ducky.cc.types.CType attribute), 30
- ## U
- u32_to_encoding() (in module ducky.cpu.instructions), 59
 - UDIV (class in ducky.cpu.instructions), 59
 - UDIV (ducky.cpu.instructions.DuckyOpcodes attribute), 51
 - UDIVL (class in ducky.cpu.coprocessor.math_copro), 42
 - UDIVL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 40
 - UINT20_FMT() (in module ducky.cpu.instructions), 59
 - UMODL (class in ducky.cpu.coprocessor.math_copro), 42
 - UMODL (ducky.cpu.coprocessor.math_copro.MathCoproprocessorOpcodes attribute), 40
 - UnableToImplicitCastError, 35
 - UnalignedJumpTargetError, 77
 - UNDEFINED (ducky.hdt.HDTEntryTypes attribute), 78
 - UndefinedStructMemberError, 35
 - UNKNOWN (ducky.mm.binary.SectionTypes attribute), 86
 - UNKNOWN (ducky.mm.binary.SymbolDataTypes attribute), 86
 - UnknownFileError, 77
 - UnknownPatternError, 77
 - UnknownSymbolError, 77
 - UNMMAP (ducky.mm.MMOperationList attribute), 88
 - unmmap_area() (ducky.boot.ROMLoader method), 25
 - unregister_command() (ducky.console.ConsoleMaster method), 27
 - unregister_page() (ducky.mm.MemoryController method), 90
 - unregister_port() (ducky.machine.Machine method), 83
 - unregister_queue() (ducky.machine.CommChannel method), 81
 - unregister_with_reactor() (ducky.streams.Stream method), 99

unregister_with_reactor()
(ducky.tools.vm.WSInputStream method),
102

UnsignedCharType (class in ducky.cc.types), 30

UnsignedIntType (class in ducky.cc.types), 31

unspill_register() (ducky.cc.SymbolStorage method), 35

unused (ducky.devices.svga.Char attribute), 74

UNUSED (ducky.mm.MMOperationList attribute), 88

UP() (ducky.cc.passes.ASTVisitor method), 29

UP() (ducky.cc.passes.BlockVisitor method), 30

update_arith_flags() (in module ducky.cpu.instructions),
59

update_tick() (ducky.devices.rtc.RTCTask method), 69

V

value (ducky.cpu.instructions.EncodingC attribute), 52

value (ducky.hdt.HDTEntry_Argument attribute), 78

value_length (ducky.hdt.HDTEntry_Argument attribute),
78

VERSION (ducky.mm.binary.File attribute), 83

version (ducky.mm.binary.FileHeader attribute), 84

visit() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit() (ducky.cc.passes.ASTVisitor method), 30

visit() (ducky.cc.passes.BlockVisitor method), 30

visit() (ducky.cc.passes.bt_visualise.BlockTreeVisualiseVisitor
method), 29

visit_ArrayRef() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_Assignment() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_BinaryOp() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_BinaryOp() (ducky.cc.passes.ast_constprop.ConstantFoldingVisitor
method), 28

visit_BinaryOp() (ducky.cc.passes.ast_visualise.ASTVisualiseVisitor
method), 29

visit_block() (ducky.cc.passes.BlockVisitor method), 30

visit_Cast() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_Compound() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_Compound() (ducky.cc.passes.ast_dce.DSEVisitor
method), 28

visit_Constant() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_Constant() (ducky.cc.passes.ast_visualise.ASTVisualiseVisitor
method), 29

visit_constant_value() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_Decl() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_expr() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_Expr() (ducky.patch.RemoveLoggingVisitor
method), 92

visit_ExprList() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_FileAST() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_FileAST() (ducky.cc.passes.ast_visualise.ASTVisualiseVisitor
method), 29

visit_fn() (ducky.cc.passes.BlockVisitor method), 30

visit_For() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_For() (ducky.patch.RemoveLoggingVisitor
method), 92

visit_FuncCall() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_FuncDef() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_ID() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_ID() (ducky.cc.passes.ast_visualise.ASTVisualiseVisitor
method), 29

visit_If() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_If() (ducky.cc.passes.ast_visualise.ASTVisualiseVisitor
method), 29

visit_If() (ducky.patch.RemoveLoggingVisitor method),
92

visit_Return() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_StructRef() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_TypeDecl() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_TypeDef() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_TypeName() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_UnaryOp() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_UnaryOp() (ducky.cc.passes.ast_visualise.ASTVisualiseVisitor
method), 29

visit_While() (ducky.cc.passes.ast_codegen.CodegenVisitor
method), 28

visit_While() (ducky.cc.passes.ast_visualise.ASTVisualiseVisitor
method), 29

VMDEBUG (ducky.devices.IRQList attribute), 76

VMDebugInterrupt (class in ducky.debugging), 68

VMDebugOperationList (class in ducky.debugging), 68

VMState (class in ducky.snapshot), 97

VMVerbosityLevels (class in ducky.debugging), 68

VoidType (class in ducky.cc.types), 31

W

wake_up() (ducky.cpu.CPU method), 61

- wake_up() (ducky.cpu.CPUCore method), 63
- wake_up() (ducky.interfaces.IMachineWorker method), 80
- wake_up() (ducky.machine.Machine method), 83
- WARNING (ducky.debugging.VMVerbosityLevels attribute), 68
- WHITE() (in module ducky.log), 81
- writable (ducky.mm.binary.SectionFlagsEncoding attribute), 85
- WRITE (ducky.mm.PageTableEntry attribute), 92
- write() (ducky.console.ConsoleConnection method), 26
- write() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 36
- write() (ducky.streams.InputStream method), 98
- write() (ducky.streams.OutputStream method), 98
- write() (ducky.streams.Stream method), 99
- write() (ducky.tools.vm.WSOutputStream method), 102
- write_blocks() (ducky.devices.storage.Storage method), 71
- write_cr1() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 36
- write_cr2() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 36
- write_cr3() (ducky.cpu.coprocessor.control.ControlCoproprocessor method), 36
- write_in() (ducky.machine.CommQueue method), 81
- write_out() (ducky.machine.CommQueue method), 81
- write_struct() (ducky.util.BinaryFile method), 103
- write_u16() (ducky.devices.IOProvider method), 76
- write_u16() (ducky.devices.svga.SimpleVGA method), 75
- write_u16() (ducky.mm.AnonymousMemoryPage method), 87
- write_u16() (ducky.mm.ExternalMemoryPage method), 88
- write_u16() (ducky.mm.MemoryController method), 90
- write_u16() (ducky.mm.MemoryPage method), 91
- write_u32() (ducky.devices.IOProvider method), 76
- write_u32() (ducky.devices.storage.BlockIO method), 70
- write_u32() (ducky.mm.AnonymousMemoryPage method), 87
- write_u32() (ducky.mm.ExternalMemoryPage method), 88
- write_u32() (ducky.mm.MemoryController method), 90
- write_u32() (ducky.mm.MemoryPage method), 91
- write_u8() (ducky.devices.IOProvider method), 76
- write_u8() (ducky.devices.rtc.RTC method), 69
- write_u8() (ducky.devices.tty.Backend method), 73
- write_u8() (ducky.mm.AnonymousMemoryPage method), 87
- write_u8() (ducky.mm.ExternalMemoryPage method), 88
- write_u8() (ducky.mm.MemoryController method), 90
- write_u8() (ducky.mm.MemoryPage method), 91
- writeln() (ducky.console.ConsoleConnection method), 26
- WriteOnlyRegisterError, 36
- WSInputStream (class in ducky.tools.vm), 101
- WStream (class in ducky.tools.vm), 102
- X
- XOR (class in ducky.cpu.instructions), 59
- XOR (ducky.cpu.instructions.DuckyOpcodes attribute), 51