CPU Opcodes

Release 0.3.14

Apr 24, 2018

Contents

1 opcodes package **3**
1.1 opcodes.x86 module 3
1.2 opcodes.x86_64 module 12
1.3 opcodes.k1om module 22

2 Indices and tables **33**

Python Module Index **35**

Contents:

CHAPTER 1

opcodes package

1.1 opcodes.x86 module

class opcodes.x86.**CodeOffset**
Relative code offset embedded into instruction encoding.

Offset is relative to the end of the instruction.

Variables

- **size** – size of the offset in bytes. Possible values are 1 or 4.
- **value** – value of the offset. Must be a reference to an instruction operand.

  The instruction operand has "rel" type of the matching size.

class opcodes.x86.**DataOffset**
Absolute data offset embedded into instruction encoding.

Only MOV instruction has forms that use direct data offset.

Variables

- **size** – size of the offset in bytes. Always equals 4.
- **value** – value of the offset. Must be a reference to an instruction operand.

  The instruction operand has "moffs" type of the matching size.

class opcodes.x86.**EVEX**
EVEX prefix.

Encoding may have only one EVEX prefix and if present, it immediately precedes the opcode, and no other prefix is allowed.

Variables

- **mm** – the EVEX mm (compressed legacy escape) field. Identical to two low bits of VEX.mmmmm field. Possible values are:

0b01 Implies 0x0F leading opcode byte.

0b10 Implies 0x0F 0x38 leading opcode bytes.

0b11 Implies 0x0F 0x3A leading opcode bytes.

- **pp** – the EVEX pp (compressed legacy prefix) field. Possible values are:

  0b00 No implied prefix.

  0b01 Implied 0x66 prefix.

  0b10 Implied 0xF3 prefix.

  0b11 Implied 0xF2 prefix.

- **W** – the EVEX.W bit. Possible values are 0, 1, and None.

  None indicates that the bit is ignored.

- **LL** – the EVEX.L'L bits. Specify either vector length for the operation, or explicit rounding control (in which case operation is 512 bits wide). Possible values:

  None Indicates that the EVEX.L'L field is ignored.

  0b00 128-bits wide operation.

  0b01 256-bits wide operation.

  0b10 512-bits wide operation.

  Reference to the last instruction operand EVEX.L'L are interpreted as rounding control and set to the value specified by the operand. If the rounding control operand is omitted, EVEX.L'L is set to 0b10 (embedded rounding control is only supported for 512-bit wide operations).

- **RR** – the EVEX.R'R bits. Always equals 0b00 in 32-bit x86 architecture.

- **B** – the EVEX.B bit. Always equals 0 in 32-bit x86 architecture.

- **X** – the EVEX.X bit. Always equals 0 in 32-bit x86 architecture.

- **vvvv** – the EVEX vvvv field. Possible values are 0b0000 or a reference to one of the instruction operands.

  The value 0b0000 indicates that this field is not used. If vvvv is a reference to an instruction operand, the operand is of register type and EVEX.vvvv field encodes the register number.

- **V** – the EVEX V field. Always equals 0 in 32-bit x86 architecture.

- **b** – the EVEX b (broadcast/rounding control/suppress all exceptions context) bit. Possible values are 0 or a reference to one of the instruction operands.

  The value 0 indicates that this field is not used. If b is a reference to an instruction operand, the operand can be a memory operand with optional broadcasting, an optional rounding specification, or an optional Suppress-all-exceptions specification. If b is a reference to a memory operand, EVEX.b encodes whether broadcasting is used to the operand. If b is a reference to a optional rounding control specification, EVEX.b encodes whether explicit rounding control is used. If b is a reference to a suppress-all-exceptions specification, EVEX.b encodes whether suppress-all-exceptions is enabled.

- **aaa** – the EVEX aaa (embedded opmask register specifier) field. Possible values are 0 or a reference to one of the instruction operands.

  The value 0 indicates that this field is not used. If aaa is a reference to an instruction operand, the operand supports register mask, and EVEX.aaa encodes the mask register.

- **z** – the EVEX z bit. Possible values are None, 0 or a reference to one of the instruction operands.

  None indicates that the bit is ignored. The value 0 indicates that the bit is not used. If z is a reference to an instruction operand, the operand supports zero-masking with register mask, and EVEX.z indicates whether zero-masking is used.

- **disp8xN** – the N value used for encoding compressed 8-bit displacement. Possible values are powers of 2 in [1, 64] range or None.

  None indicates that this instruction form does not use displacement (the form has no memory operands).

set_ignored(*w=0*, *ll=0*, *z=0*)
Sets values for ignored bits

Parameters

- **w** (*int*) – the value (0 or 1) to be assigned to EVEX.W bit if it is ignored.
- **ll** (*int*) – the value (0b00, 0b01, 0b10, or 0b11) to be assigned to EVEX.L'L field if it is ignored.
- **z** (*int*) – the value (0 or 1) to be assigned to EVEX.z bit if it is ignored.

class opcodes.x86.**Encoding**
Instruction encoding

Variables **components** – a list of *Prefix*, *VEX*, *Opcode*, *ModRM*, *RegisterByte*, *Immediate*, *DataOffset*, *CodeOffset* objects that specify the components of encoded instruction

class opcodes.x86.**ISAExtension**(*name*)

score
A number that can be used to order a list of ISA extensions

class opcodes.x86.**Immediate**
Immediate constant embedded into instruction encoding.

Variables

- **size** – size of the constant in bytes. Possible values are 1, 2, or 4.
- **value** – value of the constant. Can be an int value or a reference to an instruction operand.

  If value is a reference to an instruction operand, the operand has "imm" type of the matching size.

class opcodes.x86.**Instruction**(*name*)
Instruction is defined by its mnemonic name (in Intel-style assembly).

An instruction may have multiple forms, that mainly differ by operand types.

Variables

- **name** – instruction name in Intel-style assembly (PeachPy, NASM and YASM assemblers).
- **summary** – a summary description of the instruction name.
- **forms** – a list of *InstructionForm* objects representing the instruction forms.

class opcodes.x86.**InstructionForm**(*name*)
Instruction form is a combination of mnemonic name and operand types.

An instruction form may have multiple possible encodings.

Variables

- **name** – instruction name in PeachPy, NASM and YASM assemblers.
- **gas_name** – instruction form name in GNU assembler (gas).
- **go_name** – instruction form name in Go/Plan 9 assembler (8a).

  None means instruction is not supported in Go/Plan 9 assembler.
- **mmx_mode** – MMX technology state required or forced by this instruction. Possible values are:

  "FPU" Instruction requires the MMX technology state to be clear.

  "MMX" Instruction causes transition to MMX technology state.

  None Instruction neither affects nor cares about the MMX technology state.
- **xmm_mode** – XMM registers state accessed by this instruction. Possible values are:

  "SSE" Instruction accesses XMM registers in legacy SSE mode.

  "AVX" Instruction accesses XMM registers in AVX mode.

  None Instruction does not affect XMM registers and does not change XMM registers access mode.
- **cancelling_inputs** – indicates that the instruction form has not dependency on the values of input operands when they refer to the same register. E.g. **VPXOR xmm1, xmm0, xmm0** does not depend on *xmm0*.

  Instruction forms with cancelling inputs have only two input operands, which have the same register type.
- **operands** – a list of *Operand* objects representing the instruction operands.
- **implicit_inputs** – a set of register names that are implicitly read by this instruction.
- **implicit_outputs** – a set of register names that are implicitly written by this instruction.
- **isa_extensions** – a list of *ISAExtension* objects that represent the ISA extensions required to execute the instruction.
- **encodings** – a list of *Encoding* objects representing the possible encodings for this instruction.

class opcodes.x86.**ModRM**

Mod R/M byte that can encode a register operand, a memory operand, or provide an opcode extension.

If memory operand requires SIB byte, the SIB byte immediately follows the Mod R/M byte in instruction encoding.

Variables

- **mode** – addressing mode. Possible values are 0b11 or a reference to an instruction operand.

  If mode value is 0b11, the Mod R/M encodes two register operands or a register operand and an opcode extension.

  If mode is a reference to an instruction operand, the operand has memory type and its addressing mode must be coded instruction the Mod R/M mode field.

- **rm** – a register or memory operand. Must be a reference to an instruction operand.

  If rm is a reference to a operand, rm specifies bits 0-2 of the register number. If the operand is of memory type, rm specifies bits 0-2 of the base register number unless a SIB byte is used.

- **reg** – a register or an opcode extension. Possible values are an int value, or a reference to an instruction operand.

  If reg is an int value, this value extends the opcode and must be directly coded in the reg field.

  If reg is a reference to an instruction operand, the operand is of register type, and the reg field specifies bits 0-2 of the register number.

set_ignored(*mode=3*, *rm=0*)
Sets values for ignored fields

Parameters

- **mode** (*int*) – the value (0b00, 0b01, 0b10, or 0b11) to be assigned to Mod R/M mode field if it is ignored.
- **rm** (*int*) – the value (an integer, 0 <= rm <= 7) to be assigned to Mod R/M rm field if it is ignored.

class opcodes.x86.**Opcode**(*byte*)
Operation code

Encoding may include more than one opcode. Opcodes do not necessarily go in sequence.

Variables

- **byte** – operation code as a byte integer (0 <= *byte* <= 255)
- **addend** – None or a reference to an instruction operand.

  If addend is a reference to an instruction operand, the operand is of register type and the three lowest bits of its number must be ORed with *byte* to produce the final opcode value.

class opcodes.x86.**Operand**(*type*)
An explicit instruction operand.

Variables

- **type** – the type of the instruction operand. Possible values are:

  "1" The constant value *1*.

  "3" The constant value *3*.

  "al" The al register.

  "ax" The ax register.

  "eax" The eax register.

  "cl" The cl register.

  "xmm0" The xmm0 register.

  "rel8" An 8-bit signed offset relative to the address of instruction end.

  "rel32" A 32-bit signed offset relative to the address of instruction end.

  "imm4" A 4-bit immediate value.

  "imm8" An 8-bit immediate value.

"imm16" A 16-bit immediate value.

"imm32" A 32-bit immediate value.

"r8" An 8-bit general-purpose register (al, ah, bl, bh, cl, ch, dl, dh).

"r16" A 16-bit general-purpose register (ax, bx, cx, dx, si, di, bp, sp).

"r32" A 32-bit general-purpose register (eax, ebx, ecx, edx, esi, edi, ebp, esp).

"mm" A 64-bit MMX SIMD register (mm0-mm7).

"xmm" A 128-bit XMM SIMD register (xmm0-xmm31).

"xmm{k}" A 128-bit XMM SIMD register (xmm0-xmm31), optionally merge-masked by an AVX-512 mask register (k1-k7).

"xmm{k}{z}" A 128-bit XMM SIMD register (xmm0-xmm31), optionally masked by an AVX-512 mask register (k1-k7).

"ymm" A 256-bit YMM SIMD register (ymm0-ymm31).

"ymm{k}" A 256-bit YMM SIMD register (ymm0-ymm31), optionally merge-masked by an AVX-512 mask register (k1-k7).

"ymm{k}{z}" A 256-bit YMM SIMD register (ymm0-ymm31), optionally masked by an AVX-512 mask register (k1-k7).

"zmm" A 512-bit ZMM SIMD register (zmm0-zmm31).

"zmm{k}" A 512-bit ZMM SIMD register (zmm0-zmm31), optionally merge-masked by an AVX-512 mask register (k1-k7).

"zmm{k}{z}" A 512-bit ZMM SIMD register (zmm0-zmm31), optionally masked by an AVX-512 mask register (k1-k7).

"k" An AVX-512 mask register (k0-k7).

"k{k}" An AVX-512 mask register (k0-k7), optionally merge-masked by an AVX-512 mask register (k1-k7).

"m" A memory operand of any size.

"m8" An 8-bit memory operand.

"m16" A 16-bit memory operand.

"m16{k}{z}" A 16-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m32" A 32-bit memory operand.

"m32{k}" A 32-bit memory operand, optionally merge-masked by an AVX-512 mask register (k1-k7).

"m32{k}{z}" A 32-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m64" A 64-bit memory operand.

"m64{k}" A 64-bit memory operand, optionally merge-masked by an AVX-512 mask register (k1-k7).

"m64{k}{z}" A 64-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m80" An 80-bit memory operand.

"m128" A 128-bit memory operand.

"m128{k}{z}" A 128-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m256" A 256-bit memory operand.

"m256{k}{z}" A 256-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m512" A 512-bit memory operand.

"m512{k}{z}" A 512-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m64/m32bcst" A 64-bit memory operand or a 32-bit memory operand broadcasted to 64 bits {1to2}.

"m128/m32bcst" A 128-bit memory operand or a 32-bit memory operand broadcasted to 128 bits {1to4}.

"m256/m32bcst" A 256-bit memory operand or a 32-bit memory operand broadcasted to 256 bits {1to8}.

"m512/m32bcst" A 512-bit memory operand or a 32-bit memory operand broadcasted to 512 bits {1to16}.

"m128/m64bcst" A 128-bit memory operand or a 64-bit memory operand broadcasted to 128 bits {1to2}.

"m256/m64bcst" A 256-bit memory operand or a 64-bit memory operand broadcasted to 256 bits {1to4}.

"m512/m64bcst" A 512-bit memory operand or a 64-bit memory operand broadcasted to 512 bits {1to8}.

"vm32x" A vector of memory addresses using VSIB with 32-bit indices in XMM register.

"vm32x{k}" A vector of memory addresses using VSIB with 32-bit indices in XMM register merge-masked by an AVX-512 mask register (k1-k7).

"vm32y" A vector of memory addresses using VSIB with 32-bit indices in YMM register.

"vm32y{k}" A vector of memory addresses using VSIB with 32-bit indices in YMM register merge-masked by an AVX-512 mask register (k1-k7).

"vm32z" A vector of memory addresses using VSIB with 32-bit indices in ZMM register.

"vm32z{k}" A vector of memory addresses using VSIB with 32-bit indices in ZMM register merge-masked by an AVX-512 mask register (k1-k7).

"vm64x" A vector of memory addresses using VSIB with 64-bit indices in XMM register.

"vm64x{k}" A vector of memory addresses using VSIB with 64-bit indices in XMM register merge-masked by an AVX-512 mask register (k1-k7).

"vm64y" A vector of memory addresses using VSIB with 64-bit indices in YMM register.

"vm64y{k}" A vector of memory addresses using VSIB with 64-bit indices in YMM register merge-masked by an AVX-512 mask register (k1-k7).

"vm64z" A vector of memory addresses using VSIB with 64-bit indices in ZMM register.

"vm64z{k}" A vector of memory addresses using VSIB with 64-bit indices in ZMM register merge-masked by an AVX-512 mask register (k1-k7).

"{sae}" Suppress-all-exceptions modifier. This operand is optional and can be omitted.

"{er}" Embedded rounding control. This operand is optional and can be omitted.

- **is_input** – indicates if the instruction reads the variable specified by this operand.
- **is_output** – indicates if the instruction writes the variable specified by this operand.
- **extended_size** – for immediate operands the size of the value in bytes after size-extension.

  The extended size affects which operand values can be encoded. E.g. a signed imm8 operand would normally encode values in the [-128, 127] range. But if it is extended to 4 bytes, it can also encode values in [2**32 - 128, 2**32 - 1] range.

is_immediate
Indicates whether this operand is an immediate constant

is_memory
Indicates whether this operand specifies a memory location

is_register
Indicates whether this operand specifies a register

is_variable
Indicates whether this operand refers to a variable (i.e. specifies either a register or a memory location)

class opcodes.x86.**Prefix**
0x66/0xF2/0xF3 prefix

Variables

- **is_mandatory** – indicates that the prefix is used not for its primary purpose, but for extending instruction opcode. Mandatory prefixes are common in SSE instructions. Non-mandatory prefix is usually 0x66 that modifies the instruction to operate on 16-bit operands.
- **byte** – numerical representation of the prefix byte.

class opcodes.x86.**RegisterByte**
Byte that encodes a register in the low 4 bits and optionally encodes an immediate value in the high 4 bits.

Variables

- **register** – a reference to an instruction operand of register type. The register number is encoded in the low 4 bits of the byte (register number is in 0..15 for all instructions which use this encoding component).
- **payload** – value of the high 4 bits of the byte. Can be None or a reference to an instruction operand of imm4 type.

  None indicates that this high 4 bits are not used. The only instructions that use the payload are VPERMIL2PD and VPERMIL2PS from XOP instruction set.

class opcodes.x86.**VEX**
VEX or XOP prefix.

VEX and XOP prefixes use the same format and differ only by leading byte. The *type* property helps to differentiate between the two prefix types.

Encoding may have only one VEX prefix and if present, it immediately precedes the opcode, and no other prefix is allowed.

Variables

- **type** – the type of the leading byte for VEX encoding. Possible values are:

"VEX" The VEX prefix (0xC4 or 0xC5) is used.

"XOP" The XOP prefix (0x8F) is used.

- **mmmmm** – the VEX m-mmmm (implied leading opcode bytes) field. In AMD documentation this field is called map_select. Possible values are:

  0b00001 Implies 0x0F leading opcode byte.

  0b00010 Implies 0x0F 0x38 leading opcode bytes.

  0b00011 Implies 0x0F 0x3A leading opcode bytes.

  0b01000 This value does not have opcode byte interpretation. Only XOP instructions use this value.

  0b01001 This value does not have opcode byte interpretation. Only XOP and TBM instructions use this value.

  0b01010 This value does not have opcode byte interpretation. Only TBM instructions use this value.

  Only VEX prefix with m-mmmm equal to 0b00001 could be encoded in two bytes.

- **pp** – the VEX pp (implied legacy prefix) field. Possible values are:

  0b00 No implied prefix.

  0b01 Implied 0x66 prefix.

  0b10 Implied 0xF3 prefix.

  0b11 Implied 0xF2 prefix.

- **W** – the VEX.W bit. Possible values are 0, 1, and None.

  None indicates that the bit is ignored.

- **L** – the VEX.L bit. Possible values are 0, 1, and None.

  None indicates that the bit is ignored.

- **R** – the VEX.R bit. Always equals 0 in 32-bit x86 architecture.
- **B** – the VEX.B bit. Always equals 0 in 32-bit x86 architecture.
- **X** – the VEX.X bit. Always equals 0 in 32-bit x86 architecture.
- **vvvv** – the VEX vvvv field. Possible values are 0b0000 or a reference to one of the instruction operands.

  The value 0b0000 indicates that this field is not used. If vvvv is a reference to an instruction operand, the operand is of register type and VEX.vvvv field specifies its number.

set_ignored(*w=0*, *l=0*)
Sets values for ignored bits

Parameters

- **w** (*int*) – the value (0 or 1) to be assigned to VEX.W bit if it is ignored.
- **l** (*int*) – the value (0 or 1) to be assigned to VEX.L bit if it is ignored.

opcodes.x86.**read_instruction_set**(*filename=None*)
Reads instruction set data from an XML file and returns a list of *Instruction* objects

Parameters **filename** – path to an XML file with instruction set data

1.2 opcodes.x86_64 module

class opcodes.x86_64.**CodeOffset**
Relative code offset embedded into instruction encoding.

Offset is relative to the end of the instruction.

Variables

- **size** – size of the offset in bytes. Possible values are 1 or 4.
- **value** – value of the offset. Must be a reference to an instruction operand.

  The instruction operand has "rel" type of the matching size.

class opcodes.x86_64.**DataOffset**
Absolute data offset embedded into instruction encoding.

Only MOV instruction has forms that use direct data offset.

Variables

- **size** – size of the offset in bytes. Possible values are 4 or 8.
- **value** – value of the offset. Must be a reference to an instruction operand.

  The instruction operand has "moffs" type of the matching size.

class opcodes.x86_64.**EVEX**
EVEX prefix.

Encoding may have only one EVEX prefix and if present, it immediately precedes the opcode, and no other prefix is allowed.

Variables

- **mm** – the EVEX mm (compressed legacy escape) field. Identical to two low bits of VEX.mmmmm field. Possible values are:

  0b01 Implies 0x0F leading opcode byte.

  0b10 Implies 0x0F 0x38 leading opcode bytes.

  0b11 Implies 0x0F 0x3A leading opcode bytes.

- **pp** – the EVEX pp (compressed legacy prefix) field. Possible values are:

  0b00 No implied prefix.

  0b01 Implied 0x66 prefix.

  0b10 Implied 0xF3 prefix.

  0b11 Implied 0xF2 prefix.

- **W** – the EVEX.W bit. Possible values are 0, 1, and None.

  None indicates that the bit is ignored.

- **LL** – the EVEX.L'L bits. Specify either vector length for the operation, or explicit rounding control (in which case operation is 512 bits wide). Possible values:

  None Indicates that the EVEX.L'L field is ignored.

  0b00 128-bits wide operation.

  0b01 256-bits wide operation.

0b10 512-bits wide operation.

Reference to the last instruction operand EVEX.L'L are interpreted as rounding control and set to the value specified by the operand. If the rounding control operand is omitted, EVEX.L'L is set to 0b10 (embedded rounding control is only supported for 512-bit wide operations).

- **RR** – the EVEX.R'R bits. Possible values are None, or a reference to an register-type instruction operand.

  None indicates that the field is ignored. The R' bit specifies bit 4 of the register number and the R bit specifies bit 3 of the register number.

- **B** – the EVEX.B bit. Possible values are None, or a reference to one of the instruction operands.

  None indicates that this bit is ignored. If R is a reference to an instruction operand, the operand can be of register or memory type. If the operand is of register type, the EVEX.R bit specifies the high bit (bit 3) of the register number, and the EVEX.X bit is ignored. If the operand is of memory type, the EVEX.R bit specifies the high bit (bit 3) of the base register number, and the X instance variable refers to the same operand.

- **X** – the EVEX.X bit. Possible values are None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If X is a reference to an instruction operand, the operand is of memory type and the EVEX.X bit specifies the high bit (bit 3) of the index register number, and the B instance variable refers to the same operand.

- **vvvv** – the EVEX vvvv field. Possible values are 0b0000 or a reference to one of the instruction operands.

  The value 0b0000 indicates that this field is not used. If vvvv is a reference to an instruction operand, the operand is of register type and EVEX.vvvv field specifies the register number.

- **V** – the EVEX V field. Possible values are 0, or a reference to one of the instruction operands.

  The value 0 indicates that this field is not used (EVEX.vvvv is not used or encodes a general-purpose register).

- **b** – the EVEX b (broadcast/rounding control/suppress all exceptions context) bit. Possible values are 0 or a reference to one of the instruction operands.

  The value 0 indicates that this field is not used. If b is a reference to an instruction operand, the operand can be a memory operand with optional broadcasting, an optional rounding specification, or an optional Suppress-all-exceptions specification. If b is a reference to a memory operand, EVEX.b encodes whether broadcasting is used to the operand. If b is a reference to a optional rounding control specification, EVEX.b encodes whether explicit rounding control is used. If b is a reference to a suppress-all-exceptions specification, EVEX.b encodes whether suppress-all-exceptions is enabled.

- **aaa** – the EVEX aaa (embedded opmask register specifier) field. Possible values are 0 or a reference to one of the instruction operands.

  The value 0 indicates that this field is not used. If aaa is a reference to an instruction operand, the operand supports register mask, and EVEX.aaa encodes the mask register.

- **z** – the EVEX z bit. Possible values are None, 0 or a reference to one of the instruction operands.

None indicates that the bit is ignored. The value 0 indicates that the bit is not used. If z is a reference to an instruction operand, the operand supports zero-masking with register mask, and EVEX.z indicates whether zero-masking is used.

- **disp8xN** – the N value used for encoding compressed 8-bit displacement. Possible values are powers of 2 in [1, 64] range or None.

  None indicates that this instruction form does not use displacement (the form has no memory operands).

set_ignored(*w=0*, *ll=0*, *rr=0*, *x=0*, *z=0*)
Sets values for ignored bits

Parameters

- **w** (*int*) – the value (0 or 1) to be assigned to EVEX.W bit if it is ignored.
- **ll** (*int*) – the value (0b00, 0b01, 0b10, or 0b11) to be assigned to EVEX.L'L field if it is ignored.
- **rr** (*int*) – the value (0b00, 0b01, 0b10, or 0b11) to be assigned to EVEX.R'R field if it is ignored.
- **x** (*int*) – the value (0 or 1) to be assigned to EVEX.X bit if it is ignored.
- **z** (*int*) – the value (0 or 1) to be assigned to EVEX.z bit if it is ignored.

class opcodes.x86_64.**Encoding**
Instruction encoding

Variables **components** – a list of *Prefix*, *REX*, *VEX*, *Opcode*, *ModRM*, *RegisterByte*, *Immediate*, *DataOffset*, *CodeOffset* objects that specify the components of encoded instruction

class opcodes.x86_64.**ISAExtension**(*name*)

score
A number that can be used to order a list of ISA extensions

class opcodes.x86_64.**Immediate**
Immediate constant embedded into instruction encoding.

Variables

- **size** – size of the constant in bytes. Possible values are 1, 2, 4, or 8.
- **value** – value of the constant. Can be an int value or a reference to an instruction operand.

  If value is a reference to an instruction operand, the operand has "imm" type of the matching size.

class opcodes.x86_64.**Instruction**(*name*)
Instruction is defined by its mnemonic name (in Intel-style assembly).

An instruction may have multiple forms, that mainly differ by operand types.

Variables

- **name** – instruction name in Intel-style assembly (PeachPy, NASM and YASM assemblers).
- **summary** – a summary description of the instruction name.
- **forms** – a list of *InstructionForm* objects representing the instruction forms.

class opcodes.x86_64.**InstructionForm**(*name*)

Instruction form is a combination of mnemonic name and operand types.

An instruction form may have multiple possible encodings.

Variables

- **name** – instruction name in PeachPy, NASM and YASM assemblers.
- **gas_name** – instruction form name in GNU assembler (gas).
- **go_name** – instruction form name in Go/Plan 9 assembler (8a).

  None means instruction is not supported in Go/Plan 9 assembler.
- **mmx_mode** – MMX technology state required or forced by this instruction. Possible values are:

  "FPU" Instruction requires the MMX technology state to be clear.

  "MMX" Instruction causes transition to MMX technology state.

  None Instruction neither affects nor cares about the MMX technology state.
- **xmm_mode** – XMM registers state accessed by this instruction. Possible values are:

  "SSE" Instruction accesses XMM registers in legacy SSE mode.

  "AVX" Instruction accesses XMM registers in AVX mode.

  None Instruction does not affect XMM registers and does not change XMM registers access mode.
- **cancelling_inputs** – indicates that the instruction form has not dependency on the values of input operands when they refer to the same register. E.g. **VPXOR xmm1, xmm0, xmm0** does not depend on *xmm0*.

  Instruction forms with cancelling inputs have only two input operands, which have the same register type.
- **nacl_version** – indicates the earliest Pepper API version where validator supports this instruction.

  Possible values are integers >= 33 or None. Pepper 33 is the earliest version for which information on supported instructions is available; if instruction forms supported before Pepper 33 would have nacl_version == 33. None means instruction is either not yet supported by Native Client validator, or is forbidden in Native Client SFI model.
- **nacl_zero_extends_outputs** – indicates that Native Client validator recognizes that the instruction zeroes the upper 32 bits of the output registers.

  In x86-64 Native Client SFI model this means that the subsequent instruction can use registers written by this instruction for memory addressing.
- **operands** – a list of *Operand* objects representing the instruction operands.
- **implicit_inputs** – a set of register names that are implicitly read by this instruction.
- **implicit_outputs** – a set of register names that are implicitly written by this instruction.
- **isa_extensions** – a list of *ISAExtension* objects that represent the ISA extensions required to execute the instruction.
- **encodings** – a list of *Encoding* objects representing the possible encodings for this instruction.

class opcodes.x86_64.**ModRM**
Mod R/M byte that can encode a register operand, a memory operand, or provide an opcode extension.

If memory operand requires SIB byte, the SIB byte immediately follows the Mod R/M byte in instruction encoding.

Variables

- **mode** – addressing mode. Possible values are 0b11 or a reference to an instruction operand.

  If mode value is 0b11, the Mod R/M encodes two register operands or a register operand and an opcode extension.

  If mode is a reference to an instruction operand, the operand has memory type and its addressing mode must be coded instruction the Mod R/M mode field.

- **rm** – a register or memory operand. Must be a reference to an instruction operand.

  If rm is a reference to a operand, rm specifies bits 0-2 of the register number. If the operand is of memory type, rm specifies bits 0-2 of the base register number unless a SIB byte is used.

- **reg** – a register or an opcode extension. Possible values are an int value, or a reference to an instruction operand.

  If reg is an int value, this value extends the opcode and must be directly coded in the reg field.

  If reg is a reference to an instruction operand, the operand is of register type, and the reg field specifies bits 0-2 of the register number.

set_ignored(*mode=3*, *rm=0*)
Sets values for ignored fields

Parameters

- **mode** (*int*) – the value (0b00, 0b01, 0b10, or 0b11) to be assigned to Mod R/M mode field if it is ignored.
- **rm** (*int*) – the value (an integer, 0 <= rm <= 7) to be assigned to Mod R/M rm field if it is ignored.

class opcodes.x86_64.**Opcode**(*byte*)
Operation code

Encoding may include more than one opcode. Opcodes do not necessarily go in sequence.

Variables

- **byte** – operation code as a byte integer (0 <= *byte* <= 255)
- **addend** – None or a reference to an instruction operand.

  If addend is a reference to an instruction operand, the operand is of register type and the three lowest bits of its number must be ORed with *byte* to produce the final opcode value.

class opcodes.x86_64.**Operand**(*type*)
An explicit instruction operand.

Variables

- **type** – the type of the instruction operand. Possible values are:

  "1" The constant value *1*.

  "3" The constant value *3*.

"al" The al register.

"ax" The ax register.

"eax" The eax register.

"rax" The rax register.

"cl" The cl register.

"xmm0" The xmm0 register.

"rel8" An 8-bit signed offset relative to the address of instruction end.

"rel32" A 32-bit signed offset relative to the address of instruction end.

"imm4" A 4-bit immediate value.

"imm8" An 8-bit immediate value.

"imm16" A 16-bit immediate value.

"imm32" A 32-bit immediate value.

"imm64" A 64-bit immediate value.

"r8" An 8-bit general-purpose register (al, bl, cl, dl, sil, dil, bpl, spl, r8b-r15b).

"r16" A 16-bit general-purpose register (ax, bx, cx, dx, si, di, bp, sp, r8w-r15w).

"r32" A 32-bit general-purpose register (eax, ebx, ecx, edx, esi, edi, ebp, esp, r8d-r15d).

"r64" A 64-bit general-purpose register (rax, rbx, rcx, rdx, rsi, rdi, rbp, rsp, r8-r15).

"mm" A 64-bit MMX SIMD register (mm0-mm7).

"xmm" A 128-bit XMM SIMD register (xmm0-xmm31).

"xmm{k}" A 128-bit XMM SIMD register (xmm0-xmm31), optionally merge-masked by an AVX-512 mask register (k1-k7).

"xmm{k}{z}" A 128-bit XMM SIMD register (xmm0-xmm31), optionally masked by an AVX-512 mask register (k1-k7).

"ymm" A 256-bit YMM SIMD register (ymm0-ymm31).

"ymm{k}" A 256-bit YMM SIMD register (ymm0-ymm31), optionally merge-masked by an AVX-512 mask register (k1-k7).

"ymm{k}{z}" A 256-bit YMM SIMD register (ymm0-ymm31), optionally masked by an AVX-512 mask register (k1-k7).

"zmm" A 512-bit ZMM SIMD register (zmm0-zmm31).

"zmm{k}" A 512-bit ZMM SIMD register (zmm0-zmm31), optionally merge-masked by an AVX-512 mask register (k1-k7).

"zmm{k}{z}" A 512-bit ZMM SIMD register (zmm0-zmm31), optionally masked by an AVX-512 mask register (k1-k7).

"k" An AVX-512 mask register (k0-k7).

"k{k}" An AVX-512 mask register (k0-k7), optionally merge-masked by an AVX-512 mask register (k1-k7).

"m" A memory operand of any size.

"m8" An 8-bit memory operand.

"m16" A 16-bit memory operand.

"m16{k}{z}" A 16-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m32" A 32-bit memory operand.

"m32{k}" A 32-bit memory operand, optionally merge-masked by an AVX-512 mask register (k1-k7).

"m32{k}{z}" A 32-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m64" A 64-bit memory operand.

"m64{k}" A 64-bit memory operand, optionally merge-masked by an AVX-512 mask register (k1-k7).

"m64{k}{z}" A 64-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m80" An 80-bit memory operand.

"m128" A 128-bit memory operand.

"m128{k}{z}" A 128-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m256" A 256-bit memory operand.

"m256{k}{z}" A 256-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m512" A 512-bit memory operand.

"m512{k}{z}" A 512-bit memory operand, optionally masked by an AVX-512 mask register (k1-k7).

"m64/m32bcst" A 64-bit memory operand or a 32-bit memory operand broadcasted to 64 bits {1to2}.

"m128/m32bcst" A 128-bit memory operand or a 32-bit memory operand broadcasted to 128 bits {1to4}.

"m256/m32bcst" A 256-bit memory operand or a 32-bit memory operand broadcasted to 256 bits {1to8}.

"m512/m32bcst" A 512-bit memory operand or a 32-bit memory operand broadcasted to 512 bits {1to16}.

"m128/m64bcst" A 128-bit memory operand or a 64-bit memory operand broadcasted to 128 bits {1to2}.

"m256/m64bcst" A 256-bit memory operand or a 64-bit memory operand broadcasted to 256 bits {1to4}.

"m512/m64bcst" A 512-bit memory operand or a 64-bit memory operand broadcasted to 512 bits {1to8}.

"vm32x" A vector of memory addresses using VSIB with 32-bit indices in XMM register.

"vm32x{k}" A vector of memory addresses using VSIB with 32-bit indices in XMM register merge-masked by an AVX-512 mask register (k1-k7).

"vm32y" A vector of memory addresses using VSIB with 32-bit indices in YMM register.

"vm32y{k}" A vector of memory addresses using VSIB with 32-bit indices in YMM register merge-masked by an AVX-512 mask register (k1-k7).

"vm32z" A vector of memory addresses using VSIB with 32-bit indices in ZMM register.

"vm32z{k}" A vector of memory addresses using VSIB with 32-bit indices in ZMM register merge-masked by an AVX-512 mask register (k1-k7).

"vm64x" A vector of memory addresses using VSIB with 64-bit indices in XMM register.

"vm64x{k}" A vector of memory addresses using VSIB with 64-bit indices in XMM register merge-masked by an AVX-512 mask register (k1-k7).

"vm64y" A vector of memory addresses using VSIB with 64-bit indices in YMM register.

"vm64y{k}" A vector of memory addresses using VSIB with 64-bit indices in YMM register merge-masked by an AVX-512 mask register (k1-k7).

"vm64z" A vector of memory addresses using VSIB with 64-bit indices in ZMM register.

"vm64z{k}" A vector of memory addresses using VSIB with 64-bit indices in ZMM register merge-masked by an AVX-512 mask register (k1-k7).

"{sae}" Suppress-all-exceptions modifier. This operand is optional and can be omitted.

"{er}" Embedded rounding control. This operand is optional and can be omitted.

- **is_input** – indicates if the instruction reads the variable specified by this operand.
- **is_output** – indicates if the instruction writes the variable specified by this operand.
- **extended_size** – for immediate operands the size of the value in bytes after size-extension.

  The extended size affects which operand values can be encoded. E.g. a signed imm8 operand would normally encode values in the [-128, 127] range. But if it is extended to 4 bytes, it can also encode values in [2**32 - 128, 2**32 - 1] range.

is_immediate
Indicates whether this operand is an immediate constant

is_memory
Indicates whether this operand specifies a memory location

is_register
Indicates whether this operand specifies a register

is_variable
Indicates whether this operand refers to a variable (i.e. specifies either a register or a memory location)

class opcodes.x86_64.**Prefix**
0x66/0xF2/0xF3 prefix

Variables

- **is_mandatory** – indicates that the prefix is used not for its primary purpose, but for extending instruction opcode. Mandatory prefixes are common in SSE instructions. Non-mandatory prefix is usually 0x66 that modifies the instruction to operate on 16-bit operands.
- **byte** – numerical representation of the prefix byte.

class opcodes.x86_64.**REX**
REX prefix.

Encoding may have only one REX prefix and if present, it immediately precedes the opcode.

Variables

- **is_mandatory** – indicates whether the REX prefix must be encoded even if no extended registers are used.

  REX is mandatory for most 64-bit instructions (encoded with REX.W = 1) and instructions that operate on the extended set of 8-bit registers (to indicate access to dil/sil/bpl/spl as opposed to ah/bh/ch/dh which use the same ModR/M).

- **W** – the REX.W bit. Possible values are 0, 1, and None.

  None indicates that the bit is ignored.

- **R** – the REX.R bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If R is a reference to an instruction operand, the operand is of register type and REX.R bit specifies the high bit (bit 3) of the register number.

- **B** – the REX.B bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If R is a reference to an instruction operand, the operand can be of register or memory type. If the operand is of register type, the REX.R bit specifies the high bit (bit 3) of the register number, and the REX.X bit is ignored. If the operand is of memory type, the REX.R bit specifies the high bit (bit 3) of the base register number, and the X instance variable refers to the same operand.

- **X** – the REX.X bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If X is a reference to an instruction operand, the operand is of memory type and the REX.X bit specifies the high bit (bit 3) of the index register number, and the B instance variable refers to the same operand.

set_ignored(*w=0*, *r=0*, *x=0*, *b=0*)

Sets values for ignored bits

Parameters

- **w** (*int*) – the value (0 or 1) to be assigned to REX.W bit if it is ignored.
- **r** (*int*) – the value (0 or 1) to be assigned to REX.R bit if it is ignored.
- **x** (*int*) – the value (0 or 1) to be assigned to REX.X bit if it is ignored.
- **b** (*int*) – the value (0 or 1) to be assigned to REX.B bit if it is ignored.

class opcodes.x86_64.**RegisterByte**

Byte that encodes a register in the low 4 bits and optionally encodes an immediate value in the high 4 bits.

Variables

- **register** – a reference to an instruction operand of register type. The register number is encoded in the low 4 bits of the byte (register number is in 0..15 for all instructions which use this encoding component).
- **payload** – value of the high 4 bits of the byte. Can be None or a reference to an instruction operand of imm4 type.

  None indicates that this high 4 bits are not used. The only instructions that use the payload are VPERMIL2PD and VPERMIL2PS from XOP instruction set.

class opcodes.x86_64.**VEX**

VEX or XOP prefix.

VEX and XOP prefixes use the same format and differ only by leading byte. The *type* property helps to differentiate between the two prefix types.

Encoding may have only one VEX prefix and if present, it immediately precedes the opcode, and no other prefix is allowed.

Variables

- **type** – the type of the leading byte for VEX encoding. Possible values are:

  "VEX" The VEX prefix (0xC4 or 0xC5) is used.

  "XOP" The XOP prefix (0x8F) is used.

- **mmmmm** – the VEX m-mmmm (implied leading opcode bytes) field. In AMD documentation this field is called map_select. Possible values are:

  0b00001 Implies 0x0F leading opcode byte.

  0b00010 Implies 0x0F 0x38 leading opcode bytes.

  0b00011 Implies 0x0F 0x3A leading opcode bytes.

  0b01000 This value does not have opcode byte interpretation. Only XOP instructions use this value.

  0b01001 This value does not have opcode byte interpretation. Only XOP and TBM instructions use this value.

  0b01010 This value does not have opcode byte interpretation. Only TBM instructions use this value.

  Only VEX prefix with m-mmmm equal to 0b00001 could be encoded in two bytes.

- **pp** – the VEX pp (implied legacy prefix) field. Possible values are:

  0b00 No implied prefix.

  0b01 Implied 0x66 prefix.

  0b10 Implied 0xF3 prefix.

  0b11 Implied 0xF2 prefix.

- **W** – the VEX.W bit. Possible values are 0, 1, and None.

  None indicates that the bit is ignored.

- **L** – the VEX.L bit. Possible values are 0, 1, and None.

  None indicates that the bit is ignored.

- **R** – the VEX.R bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If R is a reference to an instruction operand, the operand is of register type and VEX.R bit specifies the high bit (bit 3) of the register number.

- **B** – the VEX.B bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If R is a reference to an instruction operand, the operand can be of register or memory type. If the operand is of register type, the VEX.R

bit specifies the high bit (bit 3) of the register number, and the VEX.X bit is ignored. If the operand is of memory type, the VEX.R bit specifies the high bit (bit 3) of the base register number, and the X instance variable refers to the same operand.

- **X** – the VEX.X bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If X is a reference to an instruction operand, the operand is of memory type and the VEX.X bit specifies the high bit (bit 3) of the index register number, and the B instance variable refers to the same operand.

- **vvvv** – the VEX vvvv field. Possible values are 0b0000 or a reference to one of the instruction operands.

  The value 0b0000 indicates that this field is not used. If vvvv is a reference to an instruction operand, the operand is of register type and VEX.vvvv field specifies its number.

set_ignored(*w=0*, *l=0*, *r=0*, *x=0*, *b=0*)

Sets values for ignored bits

Parameters

- **w** (*int*) – the value (0 or 1) to be assigned to VEX.W bit if it is ignored.
- **l** (*int*) – the value (0 or 1) to be assigned to VEX.L bit if it is ignored.
- **r** (*int*) – the value (0 or 1) to be assigned to VEX.R bit if it is ignored.
- **x** (*int*) – the value (0 or 1) to be assigned to VEX.X bit if it is ignored.
- **b** (*int*) – the value (0 or 1) to be assigned to VEX.B bit if it is ignored.

opcodes.x86_64.**read_instruction_set**(*filename=None*)

Reads instruction set data from an XML file and returns a list of *Instruction* objects

Parameters **filename** – path to an XML file with instruction set data

1.3 opcodes.k1om module

class opcodes.k1om.**CodeOffset**

Relative code offset embedded into instruction encoding.

Offset is relative to the end of the instruction.

Variables

- **size** – size of the offset in bytes. Possible values are 1 or 4.
- **value** – value of the offset. Must be a reference to an instruction operand.

  The instruction operand has "rel" type of the matching size.

class opcodes.k1om.**DataOffset**

Absolute data offset embedded into instruction encoding.

Only MOV instruction has forms that use direct data offset.

Variables

- **size** – size of the offset in bytes. Possible values are 4 or 8.
- **value** – value of the offset. Must be a reference to an instruction operand.

  The instruction operand has "moffs" type of the matching size.

class opcodes.k1om.**Encoding**
Instruction encoding

Variables **components** – a list of *Prefix*, *REX*, *VEX*, *Opcode*, *ModRM*, *RegisterByte*, *Immediate*, *DataOffset*, *CodeOffset* objects that specify the components of encoded instruction

class opcodes.k1om.**Immediate**
Immediate constant embedded into instruction encoding.

Variables

- **size** – size of the constant in bytes. Possible values are 1, 2, 4, or 8.
- **value** – value of the constant. Can be an int value or a reference to an instruction operand.

  If value is a reference to an instruction operand, the operand has "imm" type of the matching size.

class opcodes.k1om.**Instruction**(*name*)
Instruction is defined by its mnemonic name (in Intel-style assembly).

An instruction may have multiple forms, that mainly differ by operand types.

Variables

- **name** – instruction name in Intel-style assembly (PeachPy, NASM and YASM assemblers).
- **summary** – a summary description of the instruction name.
- **forms** – a list of *InstructionForm* objects representing the instruction forms.

class opcodes.k1om.**InstructionForm**(*name*)
Instruction form is a combination of mnemonic name and operand types.

An instruction form may have multiple possible encodings.

Variables

- **name** – instruction name in PeachPy assembler.
- **gas_name** – instruction form name in GNU assembler (gas).
- **cancelling_inputs** – indicates that the instruction form has not dependency on the values of input operands when they refer to the same register. E.g. **VPXORD zmm1, zmm0, zmm0** does not depend on *zmm0*.

  Instruction forms with cancelling inputs have only two input operands, which have the same register type.
- **operands** – a list of *Operand* objects representing the instruction operands.
- **implicit_inputs** – a set of register names that are implicitly read by this instruction.
- **implicit_outputs** – a set of register names that are implicitly written by this instruction.
- **encodings** – a list of *Encoding* objects representing the possible encodings for this instruction.

class opcodes.k1om.**MVEX**
MVEX prefix.

Encoding may have only one MVEX prefix and if present, it immediately precedes the opcode, and no other prefix is allowed.

Variables

- **mmmm** – the MVEX mmmm (compressed legacy escape) field. Identical to two low bits of VEX.m-mmmm field. Possible values are:

  0b0001 Implies 0x0F leading opcode byte.

  0b0010 Implies 0x0F 0x38 leading opcode bytes.

  0b0011 Implies 0x0F 0x3A leading opcode bytes.

- **pp** – the MVEX pp (compressed legacy prefix) field. Possible values are:

  0b00 No implied prefix.

  0b01 Implied 0x66 prefix.

  0b10 Implied 0xF3 prefix.

  0b11 Implied 0xF2 prefix.

- **W** – the MVEX.W bit. Possible values are 0, 1, and None.

  None indicates that the bit is ignored.

- **RR** – the MVEX.R'R bits. Possible values are None, or a reference to an register-type instruction operand.

  None indicates that the field is ignored. The R' bit specifies bit 4 of the register number and the R bit specifies bit 3 of the register number.

- **B** – the MVEX.B bit. Possible values are None, or a reference to one of the instruction operands.

  None indicates that this bit is ignored. If R is a reference to an instruction operand, the operand can be of register or memory type. If the operand is of register type, the MVEX.R bit specifies the high bit (bit 3) of the register number, and the MVEX.X bit is ignored. If the operand is of memory type, the MVEX.R bit specifies the high bit (bit 3) of the base register number, and the X instance variable refers to the same operand.

- **X** – the MVEX.X bit. Possible values are None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If X is a reference to an instruction operand, the operand is of memory type and the MVEX.X bit specifies the high bit (bit 3) of the index register number, and the B instance variable refers to the same operand.

- **vvvv** – the MVEX vvvv field. Possible values are 0b0000 or a reference to one of the instruction operands.

  The value 0b0000 indicates that this field is not used. If vvvv is a reference to an instruction operand, the operand is of register type and MVEX.vvvv field specifies the register number.

- **V** – the MVEX V field. Possible values are 0, or a reference to one of the instruction operands.

  The value 0 indicates that this field is not used (MVEX.vvvv is not used or encodes a general-purpose register).

- **SSS** – the MVEX SSS (swizzle/broadcast/up-convert/down-convert) field. Possible values are 0, or a reference to one of the instruction operands.

  The value 0 indicates that this field is not used. If SSS is a reference to an instruction operand, the operand type either includes a swizzle, broadcast, or conversion primitive, or the operand type is {er} (static rounding control), or the operand type is {sae}. If SSS is a reference to a memory/register operand, it encodes the primitive applied to the operand. If SSS is a reference to a static rounding control operand, it the high bit of MVEX.SSS

encodes suppress-all-exceptions mode (1 = enabled, 0 = disabled) and the two low bits encode rounding mode (round to nearest even = 0b00, round down = 0b01, round up = 0b10, round toward zero = 0b11) If SSS is a reference to a suppress-all-exceptions operand, the high bit of MVEX.SSS encodes the suppress-all-exceptions mode (1 = enabled, 0 = disabled) and the two low bits are ignored.

- **aaa** – the MVEX aaa (embedded opmask register specifier) field. Possible values are 0 or a reference to one of the instruction operands.

  The value 0 indicates that this field is not used. If aaa is a reference to an instruction operand, the operand supports register mask, and MVEX.aaa encodes the mask register.

- **E** – the MVEX E (eviction hint/MVEX.SSS override) bit. Possible values are 0, 1, or a reference to an instruction operand.

  The value 0 indicates that MVEX.SSS field specifies swizzle primitive for a register operand. The value 1 indicates that MVEX.SSS field specifies static rounding mode and/or suppress-all-exceptions mode for the instruction. If E is a reference to an instruction operand, the operand is of memory type, and MVEX.E encodes whether eviction hint applies to the operand (1 = eviction hint set, 0 = eviction hint not set).

- **disp8xN** – the N value used for encoding compressed 8-bit displacement of memory operands when no broadcast or conversion is specified. Possible values are powers of 2 in [4, 64] range or None.

  None indicates that this instruction form does not use displacement (the form has no memory operands).

  When broadcast or conversion is specified, N is decreased by the following factors:

  {1to16} N is decreased by 16

  {4to16} N is decreased by 4

  {1to8} N is decreased by 8

  {4to8} N is decreased by 2

  {float16} N is decreased by 2

  {uint16} N is decreased by 2

  {sint16} N is decreased by 2

  {uint8} N is decreased by 4

  {sint8} N is decreased by 4

set_ignored(*w=0*, *rr=0*, *x=0*, *z=0*)
Sets values for ignored bits

Parameters

- **w** (*int*) – the value (0 or 1) to be assigned to MVEX.W bit if it is ignored.
- **rr** (*int*) – the value (0b00, 0b01, 0b10, or 0b11) to be assigned to MVEX.R'R field if it is ignored.
- **x** (*int*) – the value (0 or 1) to be assigned to MVEX.X bit if it is ignored.
- **z** (*int*) – the value (0 or 1) to be assigned to MVEX.z bit if it is ignored.

class opcodes.k1om.**ModRM**
Mod R/M byte that can encode a register operand, a memory operand, or provide an opcode extension.

If memory operand requires SIB byte, the SIB byte immediately follows the Mod R/M byte in instruction encoding.

Variables

- **mode** – addressing mode. Possible values are 0b11 or a reference to an instruction operand.

  If mode value is 0b11, the Mod R/M encodes two register operands or a register operand and an opcode extension.

  If mode is a reference to an instruction operand, the operand has memory type and its addressing mode must be coded instruction the Mod R/M mode field.

- **rm** – a register or memory operand. Must be a reference to an instruction operand.

  If rm is a reference to a operand, rm specifies bits 0-2 of the register number. If the operand is of memory type, rm specifies bits 0-2 of the base register number unless a SIB byte is used.

- **reg** – a register or an opcode extension. Possible values are an int value, or a reference to an instruction operand.

  If reg is an int value, this value extends the opcode and must be directly coded in the reg field.

  If reg is a reference to an instruction operand, the operand is of register type, and the reg field specifies bits 0-2 of the register number.

set_ignored(*mode=3*, *rm=0*)
Sets values for ignored fields

Parameters

- **mode** (*int*) – the value (0b00, 0b01, 0b10, or 0b11) to be assigned to Mod R/M mode field if it is ignored.
- **rm** (*int*) – the value (an integer, 0 <= rm <= 7) to be assigned to Mod R/M rm field if it is ignored.

class opcodes.k1om.**Opcode**(*byte*)
Operation code

Encoding may include more than one opcode. Opcodes do not necessarily go in sequence.

Variables

- **byte** – operation code as a byte integer (0 <= *byte* <= 255)
- **addend** – None or a reference to an instruction operand.

  If addend is a reference to an instruction operand, the operand is of register type and the three lowest bits of its number must be ORed with *byte* to produce the final opcode value.

class opcodes.k1om.**Operand**(*type*)
An explicit instruction operand.

Variables

- **type** – the type of the instruction operand. Possible values are:

  "1" The constant value *1*.

  "3" The constant value *3*.

  "al" The al register.

  "ax" The ax register.

"eax" The eax register.

"rax" The rax register.

"cl" The cl register.

"rel8" An 8-bit signed offset relative to the address of instruction end.

"rel32" A 32-bit signed offset relative to the address of instruction end.

"imm8" An 8-bit immediate value.

"imm16" A 16-bit immediate value.

"imm32" A 32-bit immediate value.

"imm64" A 64-bit immediate value.

"r8" An 8-bit general-purpose register (al, bl, cl, dl, sil, dil, bpl, spl, r8b-r15b).

"r16" A 16-bit general-purpose register (ax, bx, cx, dx, si, di, bp, sp, r8w-r15w).

"r32" A 32-bit general-purpose register (eax, ebx, ecx, edx, esi, edi, ebp, esp, r8d-r15d).

"r64" A 64-bit general-purpose register (rax, rbx, rcx, rdx, rsi, rdi, rbp, rsp, r8-r15).

"zmm" A 512-bit ZMM SIMD register (zmm0-zmm31).

"zmm{k}" A 512-bit ZMM SIMD register (zmm0-zmm31), optionally masked by a mask register (k1-k7).

"S(zmm)" A 512-bit ZMM SIMD register (zmm0-zmm31), optionally with an elements swizzle ({cdab}, {badc}, {dacb}, {aaaa}, {bbbb}, {cccc}, {dddd}).

"Cf32(zmm)" A 512-bit ZMM SIMD register (zmm0-zmm31), optionally with a single-precision store down-conversion ({float16}, {uint8}, {sint8}, {uint16}, {sint16}).

"Ci32(zmm)" A 512-bit ZMM SIMD register (zmm0-zmm31), optionally with a 32-bit integer store down-conversion ({uint8}, {sint8}, {uint16}, {sint16}).

"k" A mask register (k0-k7).

"k{k}" A mask register (k0-k7), optionally masked by another mask register (k1-k7).

"m" A memory operand of any size.

"m8" An 8-bit memory operand.

"m16" A 16-bit memory operand.

"m32" A 32-bit memory operand.

"m64" A 64-bit memory operand.

"m80" An 80-bit memory operand.

"m128" A 128-bit memory operand.

"m512" A 512-bit memory operand.

"m512{k}" A 512-bit memory operand, optionally masked by a mask register (k1-k7).

"BCf32(m512)" A 512-bit memory operand, optionally with a single-precision memory broadcast/conversion ({1to16}, {4to16}, {float16}, {uint8}, {uint16}, {sint16})

"BCi32(m512)" A 512-bit memory operand, optionally with a 32-bit integer memory broadcast/conversion ({1to16}, {4to16}, {uint8}, {sint8}, {uint16}, {sint16})

"B64(m512)" A 512-bit memory operand, optionally with a 64-bit elements broadcast ({1to8}, {4to8})

"Cf32(m512)" A 512-bit memory operand, optionally with a single-precision memory up-conversion ({float16}, {uint8}, {sint8}, {uint16}, {sint16})

"Ci32(m512)" A 512-bit memory operand, optionally with a 32-bit integer memory up-conversion ({uint8}, {sint8}, {uint16}, {sint16})

"vm32z" A vector of memory addresses using VSIB with 32-bit indices in ZMM register.

"vm32z{k}" A vector of memory addresses using VSIB with 32-bit indices in ZMM register masked by a mask register (k1-k7).

"Cf32(vm32z)" A vector of memory addresses using VSIB with 32-bit indices in ZMM register, optionally with a single-precision memory up-conversion ({float16}, {uint8}, {sint8}, {uint16}, {sint16}).

"Ci32(vm32z)" A vector of memory addresses using VSIB with 32-bit indices in ZMM register, optionally with a 32-bit integer memory up-conversion ({uint8}, {sint8}, {uint16}, {sint16}).

"Cf32(vm32z){k}" A vector of memory addresses using VSIB with 32-bit indices in ZMM register masked by a mask register, optionally with a single-precision memory up-conversion ({float16}, {uint8}, {sint8}, {uint16}, {sint16}).

"{sae}" Suppress-all-exceptions modifier. This operand is optional and can be omitted.

"{er}" Embedded rounding control. This operand is optional and can be omitted.

- **allow_conversion** – for a memory operand with BCf32/BCi32 primitive indicates if memory conversion primitive can be used for the operand. For all other types of operands this variable is meaningless, and its value is None.
- **allow_1to16** – for a memory operand with BCf32/BCi32 primitive indicates if {1to16} primitive can be used for the operand. For all other types of operands this variable is meaningless, and its value is None.
- **is_input** – indicates if the instruction reads the variable specified by this operand.
- **is_output** – indicates if the instruction writes the variable specified by this operand.
- **extended_size** – for immediate operands the size of the value in bytes after size-extension.

  The extended size affects which operand values can be encoded. E.g. a signed imm8 operand would normally encode values in the [-128, 127] range. But if it is extended to 4 bytes, it can also encode values in [2**32 - 128, 2**32 - 1] range.

is_immediate
Indicates whether this operand is an immediate constant

is_memory
Indicates whether this operand specifies a memory location

is_register
Indicates whether this operand specifies a register

is_variable
Indicates whether this operand refers to a variable (i.e. specifies either a register or a memory location)

class opcodes.k1om.**Prefix**
0x66/0xF2/0xF3 prefix

Variables

- **is_mandatory** – indicates that the prefix is used not for its primary purpose, but for extending instruction opcode. Mandatory prefixes are common in SSE instructions. Non-mandatory prefix is usually 0x66 that modifies the instruction to operate on 16-bit operands.
- **byte** – numerical representation of the prefix byte.

class opcodes.k1om.**REX**

REX prefix.

Encoding may have only one REX prefix and if present, it immediately precedes the opcode.

Variables

- **is_mandatory** – indicates whether the REX prefix must be encoded even if no extended registers are used.

  REX is mandatory for most 64-bit instructions (encoded with REX.W = 1) and instructions that operate on the extended set of 8-bit registers (to indicate access to dil/sil/bpl/spl as opposed to ah/bh/ch/dh which use the same ModR/M).
- **W** – the REX.W bit. Possible values are 0, 1, and None.

  None indicates that the bit is ignored.
- **R** – the REX.R bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If R is a reference to an instruction operand, the operand is of register type and REX.R bit specifies the high bit (bit 3) of the register number.
- **B** – the REX.B bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If R is a reference to an instruction operand, the operand can be of register or memory type. If the operand is of register type, the REX.R bit specifies the high bit (bit 3) of the register number, and the REX.X bit is ignored. If the operand is of memory type, the REX.R bit specifies the high bit (bit 3) of the base register number, and the X instance variable refers to the same operand.
- **X** – the REX.X bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If X is a reference to an instruction operand, the operand is of memory type and the REX.X bit specifies the high bit (bit 3) of the index register number, and the B instance variable refers to the same operand.

set_ignored(*w=0*, *r=0*, *x=0*, *b=0*)

Sets values for ignored bits

Parameters

- **w** (*int*) – the value (0 or 1) to be assigned to REX.W bit if it is ignored.
- **r** (*int*) – the value (0 or 1) to be assigned to REX.R bit if it is ignored.
- **x** (*int*) – the value (0 or 1) to be assigned to REX.X bit if it is ignored.
- **b** (*int*) – the value (0 or 1) to be assigned to REX.B bit if it is ignored.

class opcodes.k1om.**RegisterByte**

class opcodes.k1om.**VEX**

VEX prefix.

Encoding may have only one VEX prefix and if present, it immediately precedes the opcode, and no other prefix is allowed.

Variables

- **mmmmm** – the VEX m-mmmm (implied leading opcode bytes) field. Possible values are:

  0b00001 Implies 0x0F leading opcode byte.

  0b00011 Implies 0x0F 0x3A leading opcode bytes.

  Only VEX prefix with m-mmmm equal to 0b00001 could be encoded in two bytes.

- **pp** – the VEX pp (implied legacy prefix) field. Possible values are:

  0b00 No implied prefix.

  0b01 Implied 0x66 prefix.

  0b10 Implied 0xF3 prefix.

  0b11 Implied 0xF2 prefix.

- **W** – the VEX.W bit. Possible values are 0, 1, and None.

  None indicates that the bit is ignored.

- **L** – the VEX.L bit. Possible values are 0 or 1.

- **R** – the VEX.R bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If R is a reference to an instruction operand, the operand is of register type and VEX.R bit specifies the high bit (bit 3) of the register number.

- **B** – the VEX.B bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If R is a reference to an instruction operand, the operand can be of register or memory type. If the operand is of register type, the VEX.R bit specifies the high bit (bit 3) of the register number, and the VEX.X bit is ignored. If the operand is of memory type, the VEX.R bit specifies the high bit (bit 3) of the base register number, and the X instance variable refers to the same operand.

- **X** – the VEX.X bit. Possible values are 0, 1, None, or a reference to one of the instruction operands.

  The value None indicates that this bit is ignored. If X is a reference to an instruction operand, the operand is of memory type and the VEX.X bit specifies the high bit (bit 3) of the index register number, and the B instance variable refers to the same operand.

- **vvvv** – the VEX vvvv field. Possible values are 0b0000 or a reference to one of the instruction operands.

  The value 0b0000 indicates that this field is not used. If vvvv is a reference to an instruction operand, the operand is of register type and VEX.vvvv field specifies its number.

set_ignored(*w=0*, *r=0*, *x=0*, *b=0*)

Sets values for ignored bits

Parameters

- **w** (*int*) – the value (0 or 1) to be assigned to VEX.W bit if it is ignored.
- **r** (*int*) – the value (0 or 1) to be assigned to VEX.R bit if it is ignored.
- **x** (*int*) – the value (0 or 1) to be assigned to VEX.X bit if it is ignored.
- **b** (*int*) – the value (0 or 1) to be assigned to VEX.B bit if it is ignored.

opcodes.k1om.**read_instruction_set**(*filename=None*)

Reads instruction set data from an XML file and returns a list of *Instruction* objects

Parameters **filename** – path to an XML file with instruction set data

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

opcodes.k1om, 22
opcodes.x86, 3
opcodes.x86_64, 12

Index

C

CodeOffset (class in opcodes.k1om), 22
CodeOffset (class in opcodes.x86), 3
CodeOffset (class in opcodes.x86_64), 12

D

DataOffset (class in opcodes.k1om), 22
DataOffset (class in opcodes.x86), 3
DataOffset (class in opcodes.x86_64), 12

E

Encoding (class in opcodes.k1om), 22
Encoding (class in opcodes.x86), 5
Encoding (class in opcodes.x86_64), 14
EVEX (class in opcodes.x86), 3
EVEX (class in opcodes.x86_64), 12

I

Immediate (class in opcodes.k1om), 23
Immediate (class in opcodes.x86), 5
Immediate (class in opcodes.x86_64), 14
Instruction (class in opcodes.k1om), 23
Instruction (class in opcodes.x86), 5
Instruction (class in opcodes.x86_64), 14
InstructionForm (class in opcodes.k1om), 23
InstructionForm (class in opcodes.x86), 5
InstructionForm (class in opcodes.x86_64), 14
is_immediate (opcodes.k1om.Operand attribute), 28
is_immediate (opcodes.x86.Operand attribute), 10
is_immediate (opcodes.x86_64.Operand attribute), 19
is_memory (opcodes.k1om.Operand attribute), 28
is_memory (opcodes.x86.Operand attribute), 10
is_memory (opcodes.x86_64.Operand attribute), 19
is_register (opcodes.k1om.Operand attribute), 28
is_register (opcodes.x86.Operand attribute), 10
is_register (opcodes.x86_64.Operand attribute), 19
is_variable (opcodes.k1om.Operand attribute), 28
is_variable (opcodes.x86.Operand attribute), 10
is_variable (opcodes.x86_64.Operand attribute), 19
ISAExtension (class in opcodes.x86), 5
ISAExtension (class in opcodes.x86_64), 14

M

ModRM (class in opcodes.k1om), 25
ModRM (class in opcodes.x86), 6
ModRM (class in opcodes.x86_64), 15
MVEX (class in opcodes.k1om), 23

O

Opcode (class in opcodes.k1om), 26
Opcode (class in opcodes.x86), 7
Opcode (class in opcodes.x86_64), 16
opcodes.k1om (module), 22
opcodes.x86 (module), 3
opcodes.x86_64 (module), 12
Operand (class in opcodes.k1om), 26
Operand (class in opcodes.x86), 7
Operand (class in opcodes.x86_64), 16

P

Prefix (class in opcodes.k1om), 28
Prefix (class in opcodes.x86), 10
Prefix (class in opcodes.x86_64), 19

R

read_instruction_set() (in module opcodes.k1om), 31
read_instruction_set() (in module opcodes.x86), 11
read_instruction_set() (in module opcodes.x86_64), 22
RegisterByte (class in opcodes.k1om), 29
RegisterByte (class in opcodes.x86), 10
RegisterByte (class in opcodes.x86_64), 20
REX (class in opcodes.k1om), 29
REX (class in opcodes.x86_64), 19

S

score (opcodes.x86.ISAExtension attribute), 5
score (opcodes.x86_64.ISAExtension attribute), 14
set_ignored() (opcodes.k1om.ModRM method), 26

set_ignored() (opcodes.k1om.MVEX method), 25
set_ignored() (opcodes.k1om.REX method), 29
set_ignored() (opcodes.k1om.VEX method), 30
set_ignored() (opcodes.x86.EVEX method), 5
set_ignored() (opcodes.x86.ModRM method), 7
set_ignored() (opcodes.x86.VEX method), 11
set_ignored() (opcodes.x86_64.EVEX method), 14
set_ignored() (opcodes.x86_64.ModRM method), 16
set_ignored() (opcodes.x86_64.REX method), 20
set_ignored() (opcodes.x86_64.VEX method), 22

V

VEX (class in opcodes.k1om), 29
VEX (class in opcodes.x86), 10
VEX (class in opcodes.x86_64), 20