https://ForgeVM.org

VOLUME 5

x86-64

Reference Tables and Encoding Atlas

Volume 5 - Registers, Prefixes, Opcodes, ModR/M, SIB, Flags, Features, and Relocations

ENCODING ATLAS

REG	SIZE	PREFIX	MAP	MODR/M	SIB	DISP	IMM

REGISTER FILE

RAX-R15 / XMM-ZMM / K / TMM

PREFIX LEDGER

LEGACY / REX / VEX / EVEX

ADDRESS ATLAS

MODR/M + SIB + DISP

BYTE MATRICES

MODR/M

MOD	REG	R/M

SIB

SS	INDEX	BASE

CONDITION NIBBLE

0	1	2	3
4	5	6	7
8	9	A	B
C	D	E	F

01001000 10001011 01000100 10001011 00010000
11100001 00000000 00000000 00000000 00000000
01001000 10001011 01000100 10001011 00010000
11100001 00000000 00000000 00000000 00000000
01001000 10001011 01000100 10001011 00010000
11100001 00000000 00000000 00000000 00000000
01001000 10001011 01000100 10001011 00010000

Prepared by Ayman Alheraki

TRS

TECHNICAL REFERENCE SERIES / VOLUME 5

x86-64 Reference Tables and Encoding Atlas

A professional desk-side reference for registers, operand and address sizes, prefix families, opcode maps, ModR/M, SIB, effective addresses, immediates, flags, condition codes, CPU features, instruction families, and object relocations.

PREPARED BY Ayman Alheraki

PROJECT ForgeVM.org

EDITION 2026 Technical Edition

SYNTAX NASM / Intel

Designed as the printable lookup companion to Volumes 1-4 of the ForgeVM x86-64 Instruction Encoding Technical Reference Series.

Publication and Technical Scope

Title: x86-64 Reference Tables and Encoding Atlas - Premium Volume 5

Prepared by: Ayman Alheraki

Technical project: https://ForgeVM.org

Reference snapshot

- Intel 64 and IA-32 Architectures Software Developer's Manual portal edition published June 22, 2026.
- AMD64 Architecture Programmer's Manual, combined Volumes 1-5, Revision 4.09, March 9, 2026.
- NASM 3.02 documentation and instruction-list conventions.
- Intel XED encoder/decoder reference for instruction-form and byte-level verification.
- Intel APX architecture specification and Intel AVX10.2 architecture specification for forward-looking encoding families.
- System V ELF/x86-64 psABI, Microsoft PE/COFF, and Apple Mach-O references for relocation tables.

This volume is a secondary engineering atlas. Exact instruction availability, reserved encodings, exceptions, privilege rules, and future-extension behavior must be checked against current primary specifications before production release.

Contents

How to Use This Atlas

This volume is optimized for lookup rather than linear reading. Begin with the form-selection workflow in Chapter 1, then jump to the table that owns the next encoding decision: registers, size, prefix, map, ModR/M, SIB, displacement, immediate, flags, features, or relocation.

- **Assembler and disassembler implementers**
- **Compiler, JIT, and code-generator engineers**
- **Binary-analysis and reverse-engineering developers**
- **Systems programmers hand-encoding or validating x86-64**

Hex matrices are arranged exactly by field bits so they can be used to build generated lookup tables or to audit emitted bytes manually. Quick sheets in the appendices are designed for printing beside a development workstation.

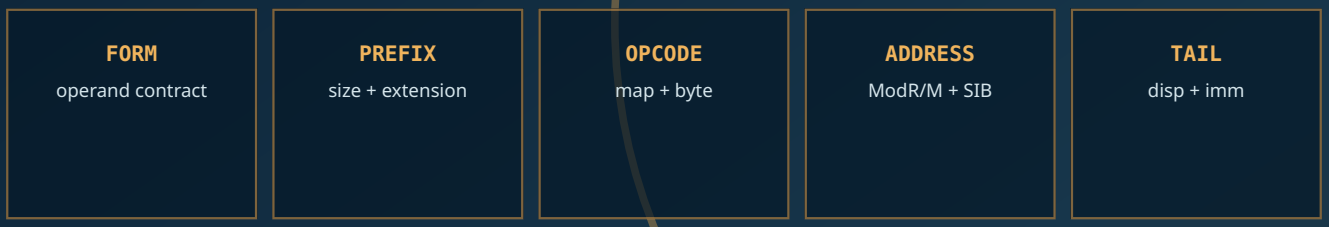
Use the atlas with an authoritative form database

Tables explain encoding mechanics; they do not replace the instruction-specific legality, exception, feature, and operand constraints in the Intel/AMD manuals or a verified instruction database.

CHAPTER 1

How to Read the Encoding Atlas

A desk-side map from assembly operands to prefixes, opcode maps, ModR/M, SIB, displacements, immediates, features, and relocations.



1.1 The Seven-Question Encoding Workflow

1 MODE	2 FORM	3 SIZE	4 PREFIX	5 OPCODE	6 ADDRESS	7 TAIL
64-bit, compatibility, or legacy?	Which operand pattern?	8, 16, 32, 64, vector?	Legacy, REX, VEX, EVEX?	Map, opcode, group?	ModR/M, SIB, disp?	Immediate or relative field?

Every table in this volume answers one of these questions. Start from the instruction form, not merely the mnemonic. A mnemonic such as `MOV` expands into many distinct forms with different operand roles, opcode maps, immediate widths, and address rules.

> **Form-first design**
> An assembler should select an instruction form before it emits bytes. The atlas is organized to support that same decision order.

1.2 Canonical Instruction Layout

Not every instruction contains every field. Legacy instructions may use one-byte, two-byte, or three-byte opcode maps. VEX and EVEX encode the map in the prefix payload. The architectural maximum instruction length remains 15 bytes.

Field	Present when	Primary decision
Legacy prefix	lock/repeat, segment, operand-size, address-size, or mandatory opcode prefix	Prefix groups and form metadata
REX / REX2	64-bit operand or extended GPR/address fields	W/R/X/B extension bits and APX capability
VEX / EVEX	Vector or EVEX-encoded legacy/APX form	Map, W, vector length, pp, vvvv, opmask/ decorators
ModR/M	Form contains `/r` or `/digit`	Register operand and register/memory operand
SIB	Scaled index, RSP/R12 base, or no-base indexed form	Scale, index, base
Displacement	Address form or relative branch needs an offset	Width and relocation policy
Immediate	Form declares immediate or embedded control byte	Width, sign extension, and range

1.3 Notation Used in Tables

Notation	Meaning
`r8/r16/r32/r64`	General-purpose register operand of the stated width
`r/m64`	Register or memory operand encoded through ModR/M.r/m
`/r`	ModR/M.reg encodes an operand
`/0 ... /7`	ModR/M.reg is an opcode extension constant
`+rd`	Low register bits are embedded in the opcode byte
`ib/iw/id/io`	Immediate byte, word, dword, or qword
`cb/cw/cd`	Relative displacement byte, word, or dword
`NP`	No legacy prefix
`66/F2/F3`	Mandatory prefix or size override according to the form

The same byte can have different meanings in different encoding spaces. For example, 66 may be an operand-size override or a mandatory opcode-selection prefix. Interpret it from the selected form.

CHAPTER 2

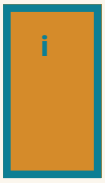
General-Purpose Register Encoding Atlas

Complete code tables for low and extended GPRs, byte-register hazards, opcode-embedded registers, and APX extensions.

CODE	REX	WIDTH	BYTE	APX
3-bit identity	extension bit	8/16/32/64	high-byte hazard	R16-R31

2.1 Base and Extended GPR Codes

Code	64-bit	32-bit	16-bit	Low 8-bit	REX.B/R/X=1
000	RAX	EAX	AX	AL	R8 / R8D / R8W / R8B
001	RCX	ECX	CX	CL	R9 / R9D / R9W / R9B
010	RDX	EDX	DX	DL	R10 / R10D / R10W / R10B
011	RBX	EBX	BX	BL	R11 / R11D / R11W / R11B
100	RSP	ESP	SP	SPL with REX	R12 / R12D / R12W / R12B
101	RBP	EBP	BP	BPL with REX	R13 / R13D / R13W / R13B
110	RSI	ESI	SI	SIL with REX	R14 / R14D / R14W / R14B
111	RDI	EDI	DI	DIL with REX	R15 / R15D / R15W / R15B



Register fields share the same three-bit code

The meaning of the extension bit depends on the field: REX.R extends ModR/M.reg, REX.X extends SIB.index, and REX.B extends ModR/M.r/m, SIB.base, or an opcode-embedded register.

2.2 Eight-Bit Register Selection

Code	Without REX	With any REX prefix
000	AL	AL
001	CL	CL
010	DL	DL
011	BL	BL
100	AH	SPL
101	CH	BPL
110	DH	SIL
111	BH	DIL

High-byte registers are incompatible with REX

AH, BH, CH, and DH cannot be encoded in an instruction that contains a REX prefix. This includes a REX prefix required only because another operand uses R8-R15 or because REX.W is set.

Byte-register boundary cases

NASM

```
; legal legacy-byte form
mov ah, bl ; 88 DC

; illegal: R8B requires REX, so AH becomes unavailable
; mov ah, r8b           ; reject

mov spl, al ; 40 88 C4
```

2.3 Opcode-Embedded Registers

Family	Base opcode	Register source	Example
`PUSH r64`	`50+rd`	opcode low 3 bits + REX.B	`41 55` = push r13
`POP r64`	`58+rd`	opcode low 3 bits + REX.B	`41 5D` = pop r13
`MOV r64, imm64`	`B8+rd`	opcode low 3 bits + REX.B	`49 BD ...` = mov r13, imm64
`XCHG RAX, r64`	`90+rd`	opcode low 3 bits + REX.B	special accumulator form

When a register is embedded in the opcode, REX.B extends the register number. REX.R does not participate because no ModR/M.reg field exists.

2.4 APX Extended GPR Namespace

Register family	Names	Encoding availability
64-bit	R16-R31	APX-enabled 64-bit mode; REX2 or extended EVEX forms
32-bit	R16D-R31D	Same logical register numbers with 32-bit write semantics
16-bit	R16W-R31W	Operand-size selection plus APX register extension
8-bit	R16B-R31B	Low-byte forms only

Intel APX doubles the architectural GPR count from 16 to 32. The exact prefix choice is form-dependent: legacy maps 0 and 1 can use REX2 for many explicit GPR or memory operands, while EVEX supports APX features such as new data destination forms. Treat APX as a feature-gated extension, not as a universal reinterpretation of old byte streams.

Prefer specification-driven generation

Do not infer REX2 or APX EVEX fields from ordinary REX rules. Generate them from the current APX architecture specification and test against a current encoder/decoder.

CHAPTER 3

Special, System, and Vector Register Atlas

Reference codes and availability rules for segment, control, debug, x87, MMX, SIMD, opmask, and tile registers.

SEG	SYS	FP	SIMD	MASK
legacy selectors	CR + DR	x87 + MMX	XMM/YMM/ZMM	K + TMM

3.1 Segment Registers

Code	Register	Long-mode use
000	ES	base treated as zero for ordinary addressing
001	CS	code selector; override generally not useful
010	SS	stack selector; base treated as zero
011	DS	base treated as zero
100	FS	FS.base remains architecturally useful
101	GS	GS.base remains architecturally useful
110-111	Reserved	not general segment-register encodings

Segment moves use dedicated forms and do not use the ordinary GPR width rules. In 64-bit mode, FS and GS overrides remain the primary segment-based address mechanisms.

3.2 Control and Debug Registers

Class	Architecturally important registers	Notes
Control	CR0, CR2, CR3, CR4, CR8	Privileged; CR8 is available in 64-bit mode. Other encodings may be reserved or feature-specific.
Debug	DR0-DR3, DR6, DR7	Hardware breakpoints and debug status/control. DR4/DR5 are legacy aliases/reserved behavior depending on control state.
Descriptor tables	GDTR, IDTR, LDTR, TR	Accessed by dedicated system instructions rather than a uniform register code table.
Model-specific	MSRs via ECX selector	Accessed with RDMSR/WRMSR and feature-specific instructions.

Privilege and existence are separate questions

An encoding can be syntactically valid yet fault because the current privilege level, CR4 state, or processor feature does not permit execution.

3.3 x87 and MMX Register Codes

Code	x87 stack name	MMX name
000	ST(0)	MM0
001	ST(1)	MM1
010	ST(2)	MM2
011	ST(3)	MM3
100	ST(4)	MM4
101	ST(5)	MM5
110	ST(6)	MM6
111	ST(7)	MM7

x87 register codes address logical stack positions relative to TOP, whereas MMX names refer to the aliased 64-bit MMX view of the x87 data registers. EMMS or FEMMS-style state transitions are relevant when mixing domains.

3.4 SIMD, Opmask, and Tile Registers

Register file	Range	Width / role	Encoding space
XMM	XMM0-XMM15 baseline; XMM16-XMM31 with EVEX in 64-bit mode	128-bit vector	Legacy SSE, VEX, EVEX
YMM	YMM0-YMM15; YMM16-YMM31 with EVEX	256-bit vector	VEX / EVEX
ZMM	ZMM0-ZMM31	512-bit vector	EVEX
Opmask	K0-K7	predicate masks; K0 often means no writemask in decorated vector forms	EVEX
Tile	TMM0-TMM7	AMX tile register names; dimensions are configured in tile state	AMX

Vector register extension uses prefix fields, not ordinary REX alone. EVEX can name 32 vector registers in 64-bit mode and carries masking, zeroing, broadcast, rounding, and vector-length controls.

CHAPTER 4

Operand-Size and Address-Size Matrix

Long-mode defaults, 66/67 overrides, REX.W, special stack forms, and the differences between encoded field width and architectural result width.

MODE	OSIZE	ASIZE	STACK	VECTOR
long/compat	16/32/64	32/64	special defaults	W/L/LL

4.1 Long-Mode Scalar Defaults

Property	Default in 64-bit mode	Override / selector
General integer operand size	32 bits	66 selects 16; REX.W selects 64 for forms that honor W
Address size	64 bits	67 selects 32-bit effective addresses
Near branch displacement	rel32 for ordinary near Jcc/JMP/CALL	short forms use rel8 where available
Stack pointer width	RSP is used for 64-bit stack addressing	67 affects address-size semantics for selected stack instructions; verify instruction-specific rules
PUSH/POP operand size	64 bits	66 selects 16 bits; ordinary 32-bit push/pop forms are not available in 64-bit mode

Default 32-bit writes zero-extend

Writing a 32-bit GPR zeroes the upper 32 bits of the corresponding 64-bit register. This is an architectural effect, not an encoding prefix.

4.2 Operand-Size Decision Table

Requested size	Typical prefix state	Example form
8-bit	No 66; REX only for SPL/BPL/SIL/DIL or R8B-R15B	`88 /r`, `80 /digit ib`
16-bit	66	`66 89 /r`
32-bit	Default, no REX.W	`89 /r`
64-bit	REX.W=1	`48 89 /r`

Some instructions have fixed or special sizes. CDQE, CQO, PUSH, POP, near CALL/JMP, MOV to/from control registers, and many system instructions do not follow a simple 66/REX.W matrix. Form metadata must be authoritative.

4.3 Address-Size Consequences

Address size	Base/index names	Displacement-only interpretation	Relative addressing
64-bit	64-bit GPRs	ModR/M special forms can encode RIP-relative or SIB no-base disp32	RIP-relative available

Address size	Base/index names	Displacement-only interpretation	Relative addressing
32-bit under 67	32-bit register names conceptually select low address bits	absolute 32-bit effective address forms are available according to ModR/M/SIB rules	No ordinary EIP-relative form

The address-size override changes address calculation, not the size of the loaded or stored data. Operand size and address size are independent axes.

4.4 Vector Width Selectors

Encoding	Width fields	Typical meanings
Legacy SSE	instruction form and prefixes	Usually 128-bit vector or scalar operation
VEX	L and sometimes W	L=0 128-bit, L=1 256-bit for forms that use L
EVEX	L'L and W	128/256/512 selection, or instruction-specific controls
AMX	tile configuration state	Dimensions come from TILECFG rather than a simple L field

W and L are form-specific

A W or L bit can be ignored, required, or repurposed. Never assign a universal semantic meaning without the selected instruction form.

CHAPTER 5

Legacy Prefix, REX, VEX, and EVEX Atlas

Byte values, prefix groups, extension fields, mandatory-prefix selection, and the prefix-order rules an encoder must enforce.

LEGACY	REX	VEX	EVEX	ORDER
four groups	W/R/X/B	map + vvvv	mask + LL	canonical bytes

5.1 Legacy Prefix Groups

Group	Bytes	Meaning
1	F0, F2, F3	LOCK; REPNE/REPNZ; REP/REPE/REPZ or mandatory opcode prefix
2	2E, 36, 3E, 26, 64, 65	CS, SS, DS, ES, FS, GS segment overrides; historical branch-hint interpretations for 2E/3E
3	66	Operand-size override or mandatory opcode prefix
4	67	Address-size override

At most one effective prefix from each legacy group should be emitted by a canonical encoder. Duplicate or conflicting prefixes may decode with implementation-specific or last-prefix behavior and should normally be rejected or normalized.

5.2 REX Byte Ledger

7	6	5	4	3	2	1	0
0	1	0	0	W	R	X	B

Byte range	Meaning
40	Empty REX: enables SPL/BPL/SIL/DIL byte namespace and suppresses AH/BH/CH/DH
41	REX.B
42	REX.X
44	REX.R
48	REX.W
4F	REX.WRXB all set

REX must immediately precede the opcode or opcode escape bytes after any legacy prefixes. A legacy prefix placed after REX is not part of the same canonical prefix sequence and can change decoding.

5.3 VEX Field Summary

Prefix	Bytes	Core fields
2-byte VEX	C5 + payload	Inverted R, inverted vvvv, L, pp; fixed map 0F and implicit W=0
3-byte VEX	C4 + two payload bytes	Inverted R/X/B, m-mmmm map, W, inverted vvvv, L, pp

Field	Purpose
R/X/B	Register extension bits, stored inverted in VEX
m-mmmm	Opcode map selector
W	Instruction-specific width or opcode extension
vvvv	Usually a source register, stored inverted
L	128/256 vector length where defined
pp	Encoded mandatory prefix: none, 66, F3, or F2

Use 2-byte VEX only when legal

The short VEX form is available only when the map, W, and extension requirements fit its fixed fields. Otherwise emit the 3-byte form.

5.4 EVEX Field Summary

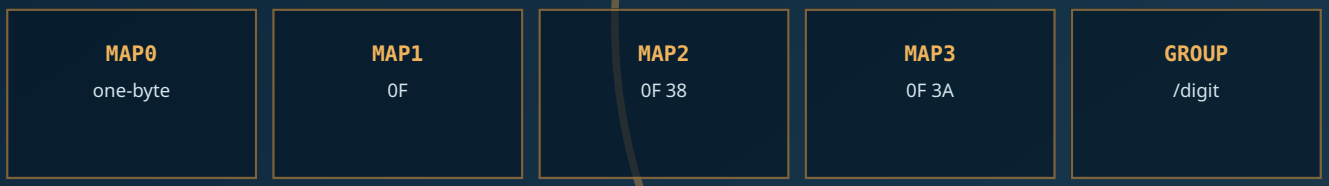
Field family	Purpose
R/R', X, B	Extended destination and address register selection; stored with EVEX-specific inversion rules
mm	Opcode map selection
W	Instruction-specific width/opcode control
vvvv/V'	Source register selection across the extended register file
pp	Mandatory-prefix encoding
z	Zeroing vs merging under a writemask
L'L	Vector length or instruction-specific control
b	Broadcast, embedded rounding, or SAE according to form
aaa	Opmask register K0-K7

EVEX is a four-byte prefix beginning with 62. Because several fields are overloaded by the instruction form, the encoder should build an EVEX semantic record first and serialize it only after all operands and decorators are validated.

CHAPTER 6

Opcode Maps and Opcode-Extension Tables

Primary and escaped maps, +rd families, /digit groups, mandatory-prefix variants, and map selection in VEX/EVEX.



6.1 Legacy Opcode Map Atlas

Map ID	Legacy byte sequence	Typical contents
0	one-byte opcode	Core integer, stack, string, branches, immediate groups
1	0F + opcode	System, conditional move/set, extended arithmetic, SSE families
2	0F 38 + opcode	SSSE3/SSE4 and later vector/integer extensions
3	0F 3A + opcode	Forms commonly carrying an immediate control byte

VEX and EVEX encode map selection in prefix fields instead of emitting 0F/0F38/0F3A escape bytes. The logical map remains part of the instruction form.

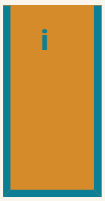
6.2 Common +rd Opcode Families

Range	Family	Register source
50-57	PUSH r64	opcode low bits + extension
58-5F	POP r64	opcode low bits + extension
90-97	XCHG accumulator, r	opcode low bits + extension
B0-B7	MOV r8, imm8	opcode low bits + extension
B8-BF	MOV r16/32/64, immediate	opcode low bits + extension

The opcode range identifies the form family; width comes from the form and prefixes. In 64-bit mode, B8+rd with REX.W carries an imm64 field, while C7 /0 can encode a sign-extended imm32 to r/m64.

6.3 High-Value /digit Groups

Opcode	Group selector	Representative operations
80/81/83	/0 ADD, /1 OR, /2 ADC, /3 SBB, /4 AND, /5 SUB, /6 XOR, /7 CMP	Immediate arithmetic/logical group
C0/C1/D0/D1/D2/D3	/0 ROL, /1 ROR, /2 RCL, /3 RCR, /4 SHL/SAL, /5 SHR, /7 SAR	Shift and rotate groups
F6/F7	/0 TEST, /2 NOT, /3 NEG, /4 MUL, /5 IMUL, /6 DIV, /7 IDIV	Unary/multiply/divide group
FE	/0 INC, /1 DEC	Byte increment/decrement
FF	/0 INC, /1 DEC, /2 CALL near, /4 JMP near, /6 PUSH	Near control and unary group
C6/C7	/0 MOV immediate to r/m	Immediate move group



Group selector is not an operand

For `/digit` forms, ModR/M.reg is a constant opcode extension. Only ModR/M.r/m identifies the register or memory operand.

6.4 Mandatory Prefix and Opcode Identity

Encoded selector	Legacy byte	VEX/EVEX pp	Common use
None	none	00	Packed single-precision or base variant
66	66	01	Packed double/integer or alternate variant
F3	F3	10	Scalar single or REP-derived variant
F2	F2	11	Scalar double or alternate variant

This table is only a selector map. The actual semantic variant comes from the instruction form. Do not assume a data type solely from pp without the opcode and map.

CHAPTER 7

ModR/M Byte Atlas

All 256 ModR/M byte values, field ownership, special r/m meanings, and the difference between register-direct and memory forms.

MOD	REG	R/M	SIB	SPECIAL
2 bits	operand/group	register/address	r/m=100	r/m=101

7.1 ModR/M Field Layout

MOD	Meaning in 64-bit address size
00	Memory, normally no displacement; special r/m cases can require RIP-relative, SIB, or disp32
01	Memory with sign-extended disp8
10	Memory with disp32
11	Register-direct; no SIB or displacement

REX.R extends REG. REX.B extends R/M when it names a register or base. REX.X is used only by the SIB index field.

7.2 MOD=00 Hex Matrix

MOD=00 / REG	R/M 000	R/M 001	R/M 010	R/M 011	R/M 100	R/M 101	R/M 110	R/M 111
000	00	01	02	03	04	05	06	07
001	08	09	0A	0B	0C	0D	0E	0F
010	10	11	12	13	14	15	16	17
011	18	19	1A	1B	1C	1D	1E	1F
100	20	21	22	23	24	25	26	27
101	28	29	2A	2B	2C	2D	2E	2F
110	30	31	32	33	34	35	36	37
111	38	39	3A	3B	3C	3D	3E	3F

With 64-bit address size, R/M=100 selects a following SIB byte. R/M=101 without SIB encodes RIP-relative addressing with disp32. Extended-base cases such as R13 must be handled from the full REX.B + MOD + R/M combination.

7.3 MOD=01 Hex Matrix

MOD=01 / REG	R/M 000	R/M 001	R/M 010	R/M 011	R/M 100	R/M 101	R/M 110	R/M 111
000	40	41	42	43	44	45	46	47
001	48	49	4A	4B	4C	4D	4E	4F
010	50	51	52	53	54	55	56	57
011	58	59	5A	5B	5C	5D	5E	5F
100	60	61	62	63	64	65	66	67
101	68	69	6A	6B	6C	6D	6E	6F
110	70	71	72	73	74	75	76	77
111	78	79	7A	7B	7C	7D	7E	7F

MOD=01 always carries disp8. The byte is sign-extended for effective-address calculation. This form is frequently the shortest legal encoding for small positive and negative offsets.

7.4 MOD=10 Hex Matrix

MOD=10 / REG	R/M 000	R/M 001	R/M 010	R/M 011	R/M 100	R/M 101	R/M 110	R/M 111
000	80	81	82	83	84	85	86	87
001	88	89	8A	8B	8C	8D	8E	8F
010	90	91	92	93	94	95	96	97
011	98	99	9A	9B	9C	9D	9E	9F
100	A0	A1	A2	A3	A4	A5	A6	A7
101	A8	A9	AA	AB	AC	AD	AE	AF
110	B0	B1	B2	B3	B4	B5	B6	B7
111	B8	B9	BA	BB	BC	BD	BE	BF

MOD=10 carries disp32. In 64-bit address size, the displacement is sign-extended as part of effective-address calculation.

7.5 MOD=11 Hex Matrix

MOD=11 / REG	R/M 000	R/M 001	R/M 010	R/M 011	R/M 100	R/M 101	R/M 110	R/M 111
000	C0	C1	C2	C3	C4	C5	C6	C7
001	C8	C9	CA	CB	CC	CD	CE	CF
010	D0	D1	D2	D3	D4	D5	D6	D7
011	D8	D9	DA	DB	DC	DD	DE	DF
100	E0	E1	E2	E3	E4	E5	E6	E7
101	E8	E9	EA	EB	EC	ED	EE	EF
110	F0	F1	F2	F3	F4	F5	F6	F7
111	F8	F9	FA	FB	FC	FD	FE	FF

MOD=11 turns R/M into a register code. SIB and displacement bytes do not follow. Both REG and R/M can be extended by prefix fields according to the selected encoding space.

CHAPTER 8

SIB Byte Atlas

All 256 Scale-Index-Base values, scale factors, no-index and no-base sentinels, RSP/R12 behavior, and indexed address recipes.

SS	INDEX	BASE	REX.X	REX.B
scale	index/no-index	base/no-base	index extension	base extension

8.1 SIB Field Layout and Semantic Codes

SS	Scale
00	x1
01	x2
10	x4
11	x8

Special code	Meaning
INDEX=100 with no extension	No index register; scale is ignored
BASE=101 and MOD=00	No base register; disp32 is required
BASE=100	RSP base; SIB is required for RSP/R12 bases
Extended INDEX/BASE	REX.X and REX.B select R8-R15 counterparts

RSP is not a normal index

The unextended INDEX=100 code is the no-index sentinel. R12 can be an index because the extension bit changes the register identity.

8.2 SS=00 (x1) Hex Matrix

SS=00 / INDEX	BASE 000	BASE 001	BASE 010	BASE 011	BASE 100	BASE 101	BASE 110	BASE 111
000	00	01	02	03	04	05	06	07
001	08	09	0A	0B	0C	0D	0E	0F
010	10	11	12	13	14	15	16	17
011	18	19	1A	1B	1C	1D	1E	1F
100	20	21	22	23	24	25	26	27
101	28	29	2A	2B	2C	2D	2E	2F
110	30	31	32	33	34	35	36	37
111	38	39	3A	3B	3C	3D	3E	3F

8.3 SS=01 (x2) Hex Matrix

SS=01 / INDEX	BASE 000	BASE 001	BASE 010	BASE 011	BASE 100	BASE 101	BASE 110	BASE 111
000	40	41	42	43	44	45	46	47
001	48	49	4A	4B	4C	4D	4E	4F
010	50	51	52	53	54	55	56	57
011	58	59	5A	5B	5C	5D	5E	5F
100	60	61	62	63	64	65	66	67
101	68	69	6A	6B	6C	6D	6E	6F
110	70	71	72	73	74	75	76	77
111	78	79	7A	7B	7C	7D	7E	7F

8.4 SS=10 (x4) Hex Matrix

SS=10 / INDEX	BASE 000	BASE 001	BASE 010	BASE 011	BASE 100	BASE 101	BASE 110	BASE 111
000	80	81	82	83	84	85	86	87
001	88	89	8A	8B	8C	8D	8E	8F
010	90	91	92	93	94	95	96	97
011	98	99	9A	9B	9C	9D	9E	9F
100	A0	A1	A2	A3	A4	A5	A6	A7
101	A8	A9	AA	AB	AC	AD	AE	AF
110	B0	B1	B2	B3	B4	B5	B6	B7
111	B8	B9	BA	BB	BC	BD	BE	BF

8.5 SS=11 (x8) Hex Matrix

SS=11 / INDEX	BASE 000	BASE 001	BASE 010	BASE 011	BASE 100	BASE 101	BASE 110	BASE 111
000	C0	C1	C2	C3	C4	C5	C6	C7
001	C8	C9	CA	CB	CC	CD	CE	CF
010	D0	D1	D2	D3	D4	D5	D6	D7
011	D8	D9	DA	DB	DC	DD	DE	DF
100	E0	E1	E2	E3	E4	E5	E6	E7
101	E8	E9	EA	EB	EC	ED	EE	EF
110	F0	F1	F2	F3	F4	F5	F6	F7
111	F8	F9	FA	FB	FC	FD	FE	FF

CHAPTER 9

Effective-Address Encoding Atlas

Base, index, scale, displacement, RIP-relative, absolute, and address-size-override recipes for x86-64 memory operands.

BASE	INDEX	SCALE	DISP	RIP
0-15	0-15 except sentinel	1/2/4/8	0/8/32	rel32

9.1 Base Register Decision Table

Base	Low code	REX.B	SIB requirement	Zero-offset form
RAX	000	0	No, unless index used	No displacement
RCX	001	0	No, unless index used	No displacement
RDX	010	0	No, unless index used	No displacement
RBX	011	0	No, unless index used	No displacement
RSP	100	0	Required	No displacement
RBP	101	0	No, unless index used	Needs disp8=0
RSI	110	0	No, unless index used	No displacement
RDI	111	0	No, unless index used	No displacement
R8	000	1	No, unless index used	No displacement
R9	001	1	No, unless index used	No displacement
R10	010	1	No, unless index used	No displacement
R11	011	1	No, unless index used	No displacement
R12	100	1	Required	No displacement
R13	101	1	No, unless index used	Needs disp8=0
R14	110	1	No, unless index used	No displacement
R15	111	1	No, unless index used	No displacement

RBP and R13 share low code 101. With MOD=00 that code has a special meaning, so a true zero-offset base uses MOD=01 with disp8=0. RSP and R12 share low code 100 and therefore require SIB.

9.2 Canonical Address Recipes

Expression	MOD	R/M	SIB	Displacement
[rbx]	00	011	none	none
[rbp]	01	101	none	00
[r12]	00	100	SS:100:100	none
[r13]	01	101 with REX.B	none	00
[rbx+16]	01	011	none	10
[rbx+rcx*4]	00	100	10:001:011	none
[r12+r13*8+32]	01	100	11:101:100 with REX.XB	20

Expression	MOD	R/M	SIB	Displacement
[rip+disp32]	00	101	none	disp32
[index*4+disp32]	00	100	10:index:101	disp32

Representative memory encodings NASM

```nasm
mov rax, [rbx] ; 48 8B 03
mov rax, [rbp] ; 48 8B 45 00
mov rax, [r12] ; 49 8B 04 24
mov rax, [r13] ; 49 8B 45 00
mov rax, [rbx + rcx*4 + 16] ; 48 8B 44 8B 10
mov r8, [r12 + r13*8 + 32] ; 4F 8B 44 EC 20
```

9.3 RIP-Relative and No-Base Forms

Form	Encoding	Use
RIP-relative	MOD=00, R/M=101, disp32	Position-independent code/data references within rel32 reach
SIB no-base	MOD=00, R/M=100, SIB.BASE=101, disp32	Index*scale + disp32 or displacement-only SIB form
Address-size 32 absolute	67 plus 32-bit address rules	Absolute 32-bit effective addresses where legal

RIP is not a normal base register

RIP-relative addressing is selected by a special encoding. RIP cannot be combined with an index register in the ordinary ModR/M/SIB address grammar.

9.4 Displacement-Width Selection

Candidate	Range / condition	Preferred when
No displacement	Base encoding has no sentinel conflict and constant offset is zero	Shortest form
disp8	Signed -128 through +127	Offset fits and form supports MOD=01
disp32	Signed 32-bit field for 64-bit address calculation	Larger constant, relocation, RIP-relative, or no-base form

A relocation may force a 32-bit field even when the current placeholder value would fit in disp8. Width selection must consider the final relocation kind, not only the provisional addend.

CHAPTER 10

Immediate, Displacement, and Relative-Offset Atlas

Little-endian serialization, sign extension, accumulator special forms, relative branch equations, and range-validation tables.

IMM	DISP	REL	SIGN	RANGE
8/16/32/64	8/32	target-nextIP	extension	diagnostics

10.1 Little-Endian Field Serialization

Value	Width	Encoded bytes
0x7F	8	7F
0x1234	16	34 12
0x12345678	32	78 56 34 12
0x1122334455667788	64	88 77 66 55 44 33 22 11

Multi-byte immediates and displacements are stored least-significant byte first. Prefix, opcode, ModR/M, and SIB bytes are not byte-swapped because they are individually serialized bytes.

10.2 Immediate Width and Extension Rules

Pattern	Field	Architectural interpretation
`83 /digit ib`	imm8	Sign-extended to operand size
`81 /digit id`	imm16 or imm32 by operand size	Used directly at selected field width
`C7 /0 id` with REX.W	imm32	Sign-extended to 64 bits
`B8+rd io` with REX.W	imm64	Full 64-bit immediate
`6A ib`	imm8	Sign-extended for PUSH
`68 id` in 64-bit mode	imm32	Sign-extended to stack operand width

Field width is not always operand width

A 64-bit operation may carry only an imm8 or imm32 field and sign-extend it. Model encoded width and semantic width separately.

10.3 Relative Offset Formula

```
relative = target_address - address_of_next_instruction
```

Field	Signed range	Typical forms
rel8	-128 to +127	short Jcc, short JMP, LOOP family

Field	Signed range	Typical forms
rel16	legacy/compatibility contexts only for selected forms	not an ordinary long-mode near branch choice
rel32	-2^31 to 2^31-1	near CALL, JMP, Jcc in 64-bit mode

The next-instruction address includes the complete length of the encoded instruction, including its displacement field. Branch relaxation is therefore a layout problem, not a single-pass subtraction.

10.4 Range-Checking Ledger

Field	Unsigned maximum	Signed range
8-bit	255	-128..127
16-bit	65535	-32768..32767
32-bit	4294967295	-2147483648..2147483647
64-bit	2^64-1	-2^63..2^63-1

Diagnostics should state the original value, selected form, encoded width, and required range. Silent truncation is acceptable only for explicitly truncating data directives or syntax with documented wrap semantics.

CHAPTER 11

RFLAGS and Condition-Code Atlas

Architectural flag bits, Jcc/CMOVcc/SETcc opcode relationships, signed and unsigned comparisons, and flag-effect classifications.

BITS	CC	JCC	CMOV	SET
RFLAGS map	16 conditions	short/near	conditional move	byte result

11.1 RFLAGS Bit Ledger

Bit	Name	Primary meaning
0	CF	Carry / borrow / unsigned condition
1	Reserved	Historically fixed
2	PF	Parity of low result byte
3	Reserved	Historically fixed
4	AF	Auxiliary carry
5	Reserved	Historically fixed
6	ZF	Zero result
7	SF	Sign bit
8	TF	Trap flag
9	IF	Interrupt enable
10	DF	String direction
11	OF	Signed overflow
12-13	IOPL	I/O privilege level
14	NT	Nested task
15	Reserved	Historically fixed
16	RF	Resume flag
17	VM	Virtual-8086 mode
18	AC	Alignment check / access control
19	VIF	Virtual interrupt flag
20	VIP	Virtual interrupt pending
21	ID	CPUID identification control

Reserved-bit read values and writable behavior are architecture-defined. Software should modify RFLAGS through documented instructions and masks rather than assuming all displayed bits are freely writable.

11.2 Condition-Code Opcode Matrix

cc	Aliases	Predicate	Jcc rel8	Jcc rel32	CMOVcc	SETcc
0	O	OF=1	70	0F 80	0F 40	0F 90
1	NO	OF=0	71	0F 81	0F 41	0F 91

cc	Aliases	Predicate	Jcc rel8	Jcc rel32	CMOVcc	SETcc
2	B/NAE/C	CF=1	72	0F 82	0F 42	0F 92
3	AE/NB/NC	CF=0	73	0F 83	0F 43	0F 93
4	E/Z	ZF=1	74	0F 84	0F 44	0F 94
5	NE/NZ	ZF=0	75	0F 85	0F 45	0F 95
6	BE/NA	CF=1 or ZF=1	76	0F 86	0F 46	0F 96
7	A/NBE	CF=0 and ZF=0	77	0F 87	0F 47	0F 97
8	S	SF=1	78	0F 88	0F 48	0F 98
9	NS	SF=0	79	0F 89	0F 49	0F 99
A	P/PE	PF=1	7A	0F 8A	0F 4A	0F 9A
B	NP/PO	PF=0	7B	0F 8B	0F 4B	0F 9B
C	L/NGE	SF != OF	7C	0F 8C	0F 4C	0F 9C
D	GE/NL	SF = OF	7D	0F 8D	0F 4D	0F 9D
E	LE/NG	ZF=1 or SF != OF	7E	0F 8E	0F 4E	0F 9E
F	G/NLE	ZF=0 and SF = OF	7F	0F 8F	0F 4F	0F 9F

All four families use the same low condition nibble. The opcode base changes: 70 for short Jcc, 0F 80 for near Jcc, 0F 40 for CMOVcc, and 0F 90 for SETcc.

11.3 Signed vs Unsigned Comparison Guide

Intent	Use flags	Conditions
Unsigned below/above	CF and ZF	B/BE/A/AE
Signed less/greater	SF, OF, and ZF	L/LE/G/GE
Equality	ZF	E/NE
Sign test	SF	S/NS
Overflow test	OF	O/NO
Parity test	PF	P/NP

CMP is SUB without storing the result

Choose signed or unsigned condition codes from the interpretation of the operands, not from a different CMP opcode.

11.4 Common Flag-Effect Classes

Class	Examples	Typical behavior
Arithmetic	ADD, SUB, ADC, SBB, CMP	Update status flags including CF, OF, SF, ZF, AF, PF
Logical	AND, OR, XOR, TEST	Set SF/ZF/PF; clear CF/OF; AF is undefined
Data movement	MOV, LEA, XCHG	Usually do not modify status flags
INC/DEC	INC, DEC	Update arithmetic flags except CF
Shifts/rotates	SHL, SHR, SAR, ROL, ROR	Count-dependent flag behavior; OF defined only for selected count cases
Multiply/divide	MUL, IMUL, DIV, IDIV	Instruction-specific flag definitions; many flags undefined

A precise instruction database should represent flags as read, written, conditionally written, cleared, set, or undefined. A single boolean "modifies flags" is insufficient.

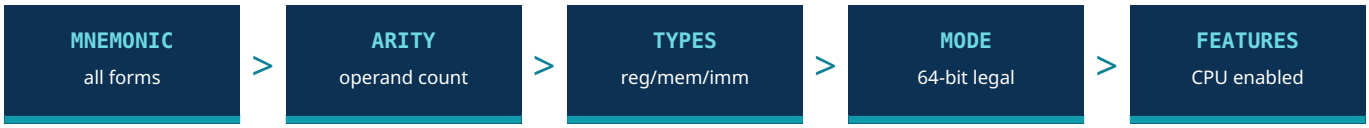
CHAPTER 12

12

Instruction-Form Selection and Shortest-Encoding Rules

Direction choices, accumulator special forms, immediate compression, branch relaxation, canonical prefixes, and deterministic tie-breaking.

12.1 Candidate Filtering Order



Only after semantic filtering should the encoder compare byte costs. A shorter encoding that changes sign-extension, memory width, flag behavior, or destination semantics is not an equivalent candidate.

12.2 Common Size Competitions

Intent	Candidate A	Candidate B	Canonical preference
Add small immediate to r/m64	REX.W + 83 /0 ib	REX.W + 81 /0 id	Use 83 if the value is representable after sign extension
Move small constant to r64	REX.W + C7 /0 id	REX.W + B8+rd io	Use C7 when the desired 64-bit value is sign-extended imm32; otherwise B8+rd
Branch to nearby target	short rel8	near rel32	Use rel8 if layout proves range and user did not force near
Register-register ADD	01 /r or 03 /r	direction-reversed equivalent	Choose deterministic canonical direction or the form minimizing extensions/prefixes
Vector VEX form	2-byte VEX	3-byte VEX	Use 2-byte VEX when map/W/extension constraints permit

A production assembler may expose syntax to force a specific width or prefix. Forced choices override size optimization but must still be legal.

12.3 Canonical Prefix Policy

Rule	Reason
Emit at most one effective prefix per legacy group	Avoid ambiguous duplicates
Emit REX only when required or explicitly requested	Preserve high-byte availability and shortest form
Place REX immediately before opcode escapes	Required canonical order
Prefer 2-byte VEX where semantically identical	Two-byte reduction
Prefer REX2 over EVEX for equivalent APX access when specified	Shorter APX encoding
Do not emit ignored W/L bits arbitrarily	Stable bytes and cleaner differential tests

12.4 Tie-Breaking and Verification

Deterministic cost record

C++

```cpp
struct EncodingCost {
 uint8_t bytes;
 uint8_t legacy_prefixes;
 uint8_t escape_or_vector_prefix_bytes;
 uint8_t displacement_bytes;
 uint8_t immediate_bytes;
 uint16_t stable_form_rank;
};

// Compare lexicographically after semantic equivalence is proven.
```

After emission, a debug build can decode the produced bytes and compare mnemonic, operand widths, register identities, memory semantics, and decorators against the selected form. This catches bit-field assembly errors early.

CHAPTER 13

SIMD Prefix and Decorator Quick Reference

Mandatory-prefix identities, VEX and EVEX operand roles, opmask rules, zeroing, broadcast, rounding, SAE, and upper-lane behavior.

PP	VVVV	L/LL	AAA	Z/B
none/66/F3/F2	source	vector width	opmask	decorators

13.1 pp Selector Table

pp bits	Legacy selector
00	None
01	66
10	F3
11	F2

In VEX and EVEX, pp replaces the emitted mandatory legacy prefix. It does not normally encode an additional standalone 66/F2/F3 byte.

13.2 Vector-Length Matrix

Encoding	Bits	Nominal width
VEX	L=0	128 bits
VEX	L=1	256 bits
EVEX	L'L=00	128 bits
EVEX	L'L=01	256 bits
EVEX	L'L=10	512 bits
EVEX	L'L=11	Reserved or instruction-specific

Scalar forms still carry vector-length fields

For many scalar or special instructions, L/LL must have a fixed value or is ignored. Follow the form definition.

13.3 EVEX Writemask and Zeroing

Decoration	Encoding concept	Effect
No mask / K0	aaa=000 in most decorated forms	Operation applies to all active destination elements
{k1}-{k7}	aaa selects mask	Only selected lanes are updated or computed
Merging	z=0	Masked-off destination elements preserve old values

Decoration	Encoding concept	Effect
Zeroing {z}	z=1	Masked-off destination elements become zero where the form permits

K0 is a real opmask register in opmask instructions, but in many EVEX vector forms aaa=000 means unmasked operation. The database must distinguish those contexts.

13.4 EVEX.b Overloads

Operand context	Possible meaning
Memory source	Broadcast of a scalar memory element
Register form with supported FP operation	Embedded rounding control and/or suppress-all-exceptions
Other forms	Reserved, ignored, or instruction-specific

The same b bit does not encode all decorators simultaneously. Operand type and instruction form determine the legal interpretation.

13.5 Upper-Lane State Rules

Family	Upper-lane behavior summary
Legacy SSE	Destination may preserve upper portions outside the architectural destination width
VEX.128	Typically zeroes upper YMM state above 128 bits for the destination register
VEX.256	Writes YMM destination and clears upper state beyond the defined width where architecturally specified
EVEX	Writes according to vector length, mask, and zero/merge semantics

Encoding family affects semantics

Legacy and VEX encodings of a similar mnemonic can differ in upper-lane behavior and operand count. Do not treat them as byte-only alternatives.

CHAPTER 14

CPU Feature and Generation Matrix

A practical chronology from the original x86 line to x86-64, SSE/AVX, AVX-512, AMX, AVX10, and APX feature-gating.

BASE	64BIT	SIMD	WIDE	NEXT
8086-386	AMD64/Intel64	SSE-AVX2	AVX-512/AMX	AVX10/APX

14.1 Major ISA Milestones

1978	**8086** - original 16-bit x86 integer, control-flow, string, and segmentation foundation.
1982	**80286** - protected-mode system architecture additions.
1985	**80386** - 32-bit GPRs, 32-bit addressing, paging, and major system extensions.
1989	**80486** - integrated x87 generation and additional system/atomic capabilities.
1997	**MMX** - packed integer media operations using the aliased MMX register file.
1999	**SSE** - XMM registers and packed/scalar single-precision operations.
2001	**SSE2** - packed/scalar double precision and broad integer SIMD in XMM registers.
2003	**AMD64 / x86-64** - 64-bit mode, 16 GPRs, RIP-relative addressing, and REX.
2006-2008	**SSSE3 and SSE4.x** - shuffle, blend, string/CRC, and arithmetic extensions.
2011	**AVX** - VEX encoding, three-operand vector syntax, and 256-bit YMM state.
2013	**AVX2 / BMI** - 256-bit integer vector expansion and bit-manipulation families.
2016+	**AVX-512** - EVEX, ZMM0-ZMM31, opmask registers, masking, and broad extension subsets.
2023+	**AMX** - tile state and matrix-oriented instruction families on supported processors.
Future/current specs	**AVX10 and APX** - converged vector evolution and expansion to 32 GPRs with new integer forms; availability must be feature-queried.

14.2 Feature-Gating Categories

Category	Examples	Assembler responsibility
Baseline mode	8086 through x86-64 base	Reject forms outside configured machine mode
CPUID feature	SSE4.2, AVX2, BMI2, SHA, AVX-512F	Require target feature or permit explicit future-target assembly
OS-enabled state	AVX, AVX-512, AMX	Assembly can encode; runtime dispatch must also verify OS state-management enablement
Privilege/system state	VMX, SGX, CR/MSR instructions	Encode only legal form; execution environment decides privilege/state faults
Future architecture specification	APX, AVX10.2 additions	Version tables and feature names must follow current primary documents

CPUID is not the whole runtime test

AVX-family execution also depends on OSXSAVE/XCR0 state. AMX requires tile state enablement and OS support. The assembler feature model and runtime dispatch model serve different purposes.

14.3 Suggested Feature-Set Hierarchy

Feature-set modeling sketch — Text

```
BASE_X86_64
  + SSE2                    ; architectural baseline for common x86-64 ABIs
  + optional scalar sets    ; POPCNT, LZCNT, BMI1/2, AES, SHA ...
  + AVX
      + AVX2
      + FMA
  + AVX512F
      + AVX512DQ/BW/VL/...  ; explicit subsets
  + AMX_TILE
      + AMX_INT8/BF16/...
  + AVX10_xxx
  + APX_F
```

Do not model AVX-512 as one undifferentiated boolean. Many instructions require specific subsets. Likewise, AVX10 and APX should carry specification-version-aware feature identities.

CHAPTER 15

Instruction-Family Quick Reference

A compact desk index of core scalar families, their dominant encoding shapes, implicit operands, and flag behavior.

DATA	ALU	SHIFT	FLOW	SYSTEM
movement	integer	bit motion	control	privileged

15.1 Core Family Summary

Family	Dominant encoding shapes	Flags / notes
ADC/ADD/SBB/SUB	r/m,r; r,r/m; accumulator immediate; group immediate	CF/OF/SF/ZF/AF/PF
AND/OR/XOR/TEST	directional /r forms and immediate groups	CF/OF cleared; SF/ZF/PF updated
MOV	reg/mem transfers, immediate forms, moffs, special registers	Usually no flags
MOVSX/MOVSXD/MOVZX	widening loads/register transfers	No flags
LEA	effective-address calculation only	No memory access; no flags
CMP	SUB flag result without destination write	Arithmetic flags
INC/DEC/NEG/NOT	group opcodes and FF/FE families	Instruction-specific; INC/DEC preserve CF
MUL/IMUL/DIV/IDIV	implicit accumulator pairs and explicit IMUL forms	Instruction-specific/undefined sets
SHL/SHR/SAR/ROL/ROR/RCL/RCR	implicit 1, CL, or imm8 count	Count-dependent flags
Jcc/JMP/CALL/RET	relative and indirect control flow	Jcc reads flags
CMOVcc/SETcc	conditional data movement	Reads flags; generally does not write them
PUSH/POP	opcode+rd, immediate, r/m groups	Stack-width special rules
MOVS/CMPS/SCAS/LODS/STOS	implicit RSI/RDI and REP semantics	String-specific
XCHG/XADD/CMPXCHG	atomic-capable read-modify-write families	Instruction-specific
CPUID/RDTSC/RDTSCP	enumeration and timestamp/system information	Fixed/implicit operands
SYSCALL/SYSRET	64-bit system-call transition	MSR-defined conventions

15.2 Implicit-Operand Register Pairs

Operation width	Unsigned multiply/divide pair	Sign-extension preparation
8-bit	AX contains dividend/product	CBW-like preparation depends on signed path
16-bit	DX:AX	CWD
32-bit	EDX:EAX	CDQ
64-bit	RDX:RAX	CQO

One-operand MUL/IMUL and DIV/IDIV use implicit accumulator pairs. Two- and three-operand IMUL forms have different encodings and do not produce the full double-width product.

15.3 String-Instruction Implicit State

Instruction family	Source	Destination	Count / direction
MOVS	DS:(E/R)SI, segment override possible	ES:(E/R)DI	REP uses RCX/ECX/CX; DF selects increment/ decrement
CMPS	source string	destination string	REPE/REPNE conditions include ZF
SCAS	accumulator	destination string	REPE/REPNE conditions include ZF
LODS	source string	accumulator	REP is rarely useful but encodable according to form
STOS	accumulator	destination string	REP fills memory

15.4 System-Instruction Safety Classification

Class	Examples	Assembly-time check
Unprivileged information	CPUID, RDTSC where enabled	Feature/form validation
Privilege-dependent	MOV CR/DR, LGDT/LIDT, RDMSR/ WRMSR	Operand legality and mode; runtime CPL/state may fault
Virtualization	VMX/SVM families	Feature and operand form
Memory ordering/cache	MFENCE, CLFLUSH-family, WBINVD	Feature/privilege according to instruction
System transition	SYSCALL/SYSRET, SYSENTER/SYSEXIT	Mode-specific form and platform convention

i

Encoding legality is not execution safety

A reference assembler can encode privileged instructions for kernels, hypervisors, firmware, and test images. Documentation should clearly separate syntactic legality from execution privilege.

CHAPTER 16

Relocation and Object-Format Cross-Reference

Neutral address intents mapped to ELF64, PE/COFF AMD64, and Mach-O x86_64 relocation families, field widths, and addend models.

INTENT	ELF	COFF	MACHO	CHECK
absolute/PC-rel	RELA	in-place	typed records	overflow

16.1 Neutral Relocation Intents

Intent	Field equation
Absolute symbol	S + A
PC-relative	S + A - P
GOT-relative	GOT(S) + A - P or format-specific variant
PLT branch	PLT(S) + A - P
Section-relative	S + A - section_base
Image-relative	S + A - image_base
Symbol difference	S1 - S2 + A

S is a symbol value, A is an addend, and P is the relocation field address. Keep this neutral algebra in the assembler core; object writers translate it to target-specific relocation records.

16.2 Common x86-64 Relocation Cross-Reference

Intent	ELF x86-64	PE/COFF AMD64	Mach-O x86_64
Absolute 64	R_X86_64_64	IMAGE_REL_AMD64_ADDR64	X86_64_RELOC_UNSIGNED, length=3
Absolute 32	R_X86_64_32 / 32S	IMAGE_REL_AMD64_ADDR32	UNSIGNED, length=2
PC-relative branch/data	R_X86_64_PC32 / PLT32	IMAGE_REL_AMD64_REL32 variants	BRANCH or SIGNED variants
Image-relative 32	format/linker policy	IMAGE_REL_AMD64_ADDR32NB	usually expressed through section/ symbol policy
Section-relative	format-specific expressions	SECREL / SECTION	section-difference or paired relocation policy
Symbol difference	fold or specialized relocation	limited / linker-specific	SUBTRACTOR + UNSIGNED pair

Names are not semantic proofs

Relocation constants and linker treatment evolve. Verify the exact field width, addend convention, relaxation policy, and overflow rule in the current format specification.

16.3 Addend Models

Format	Addend storage	Assembler implication
ELF64 RELA	Explicit addend in relocation entry	Section field can contain a neutral placeholder; writer records A separately
COFF	Addend commonly stored in the section field	Emitter must place the intended bias/value into bytes before writing relocation
Mach-O	Addend and bias are often encoded through field contents and relocation type	Branch and SIGNED_n variants require exact next-IP/ bias handling

The neutral fixup should retain expression, field width, signedness, PC-relative state, source range, and desired overflow behavior until the object writer serializes it.

16.4 Relocation Overflow Checklist

- ✓ Field width matches the instruction form.
- ✓ Signed vs unsigned range is explicit.
- ✓ PC-relative origin uses the correct P convention.
- ✓ Instruction-length bias is not applied twice.
- ✓ Linker relaxation is permitted only where the relocation type allows it.
- ✓ Undefined external symbols use the correct binding/state.
- ✓ Absolute values are folded before creating relocations.
- ✓ Error diagnostics retain the source operand range.

Appendix A - Printable GPR, Prefix, and Size Sheet

GPR low codes

Code	Base	Extended
000	RAX	R8
001	RCX	R9
010	RDX	R10
011	RBX	R11
100	RSP	R12
101	RBP	R13
110	RSI	R14
111	RDI	R15

Long-mode size selectors

Target	Selector
8-bit	byte form; REX for new byte registers
16-bit	66
32-bit	default
64-bit	REX.W
32-bit address	67
64-bit address	default

REX

7	6	5	4	3	2	1	0
0	1	0	0	W	R	X	B

Legacy prefix groups

Group	Bytes
1	F0 / F2 / F3
2	2E / 36 / 3E / 26 / 64 / 65
3	66
4	67

Appendix B - Printable ModR/M and SIB Sheet

ModR/M

MOD	Meaning
00	memory, special no-disp cases
01	memory + disp8
10	memory + disp32
11	register-direct

SIB

Special	Meaning
INDEX=100	no index without extension
BASE=101, MOD=00	no base + disp32
BASE=100	RSP/R12 base

Scale

SS	Scale
00	x1
01	x2
10	x4
11	x8

Sentinel bases

Base	Zero offset
RBP/R13	MOD=01 + disp8=0
RSP/R12	SIB required
RIP	MOD=00 R/M=101 + disp32

APPENDIX C

Golden Encoding Vectors

Representative canonical bytes for regression tests. Symbolic relative fields are shown as placeholders.

NASM / Intel syntax	Bytes
`mov rax, rbx`	`48 89 D8`
`mov r8, rax`	`49 89 C0`
`mov rax, [rbx]`	`48 8B 03`
`mov rax, [rbp]`	`48 8B 45 00`
`mov rax, [r12]`	`49 8B 04 24`
`mov rax, [r13]`	`49 8B 45 00`
`mov rax, [rbx + rcx*4 + 16]`	`48 8B 44 8B 10`
`mov r8, [r12 + r13*8 + 32]`	`4F 8B 44 EC 20`
`add rax, 1`	`48 83 C0 01`
`sub r9, -1`	`49 83 E9 FF`
`xor eax, eax`	`31 C0`
`test rax, rax`	`48 85 C0`
`shl rax, 1`	`48 D1 E0`
`shl rax, 7`	`48 C1 E0 07`
`push r13`	`41 55`
`pop r13`	`41 5D`
`call rel32`	`E8 xx xx xx xx`
`jmp short`	`EB xx`
`syscall`	`0F 05`
`ret`	`C3`

APPENDIX D

NASM Directive and Output Quick Index

High-value syntax and tool switches used while validating an assembler implementation.

Item	Purpose
`BITS 16/32/64`	Select code-generation mode
`DEFAULT REL / ABS`	Control default address interpretation for eligible memory operands
`SECTION`	Select or define an output section
`GLOBAL / EXTERN`	Control symbol linkage
`ALIGN`	Advance location with alignment policy
`DB/DW/DD/DQ`	Emit initialized data fields
`RESB/RESW/RESD/RESQ`	Reserve uninitialized storage
`-f bin/elf64/win64/macho64`	Select output format
`-l file.lst`	Generate a listing file
`ndisasm -b 64`	Disassemble raw 64-bit code

APPENDIX E

Implementation Checklists

Compact checks for encoder, address builder, and relocation writer reviews.

- ✓ Operand classes exactly match the selected form.
- ✓ Legacy prefix groups contain no conflicts.
- ✓ Register extension bits are assigned to the correct fields.
- ✓ RSP/R12 and RBP/R13 special address cases are handled.
- ✓ Displacement and immediate signedness are explicit.
- ✓ Vector decorators are legal for the form and operand type.
- ✓ All generated bytes are decoded in regression tests.
- ✓ Mode and CPU feature gates are satisfied.
- ✓ REX high-byte restrictions are checked globally.
- ✓ ModR/M.reg is distinguished from /digit opcode extension.
- ✓ RIP-relative is never combined with an ordinary index.
- ✓ Relative offsets use next-instruction address.
- ✓ Relocation intent survives until object serialization.
- ✓ Independent assemblers are used for differential validation.

APPENDIX F

Official Technical References

Primary sources used to maintain and verify this atlas.

Intel 64 and IA-32 Architectures Software Developer's Manual

Full architecture and instruction-set reference. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

AMD64 Architecture Programmer's Manual, Volumes 1-5

AMD64 architecture and instruction reference, combined Revision 4.09. https://docs.amd.com/v/u/en-US/40332_4.09_APM_PUB

NASM 3.02 Documentation

Assembler syntax, instruction list, directives, output formats, and listings. https://www.nasm.us/doc/

Intel XED Encoder Decoder

Instruction-form, encoder, decoder, and validation APIs. https://intelxed.github.io/ref-manual/

Intel Advanced Performance Extensions Architecture Specification

REX2, R16-R31, NDD, and APX encoding rules. Available through the Intel SDM portal.

Intel Advanced Vector Extensions 10.2 Architecture Specification

AVX10 feature and encoding evolution. Available through the Intel SDM portal.

System V x86-64 psABI

ELF ABI and relocation conventions. https://gitlab.com/x86-psABIs/x86-64-ABI

Microsoft PE/COFF Specification

AMD64 COFF structures and relocation constants. https://learn.microsoft.com/en-us/windows/win32/debug/pe-format

Apple Mach-O Public Headers

Mach-O structures and x86_64 relocation definitions. https://github.com/apple-oss-distributions/xnu/tree/main/EXTERNAL_HEADERS/mach-o

END OF VOLUME 5

Next: Appendices, Tools, and Developer Resources

https://ForgeVM.org