ARM64 Machine Code Translation Tables

Information parsed from **developer.arm.com/documentation/ddi0602/2025-09/Index-by-Encoding** by Caio Batista

This document was last updated on Mar 11, 2026

Changelog

- Oct 6, 2025
  - Initial version
- Oct 15, 2025
  - 4.03, 4.05, 4.06 descriptions -> added notation that imm is multiplied by 4
  - 4.10, 4.11 -> corrected order of Rn and Rm in the action
- Mar 11, 2026
  - Added changelog
  - 3.4 -> corrected ldp/stp op0 and combined the two ldp/stp entries
  - 3.4 -> combined pre-/post-index ldr/str entries
  - 4.08 -> made N bit description clearer
  - 4.11 -> corrected signed/unsigned bit of smulh/umulh
  - 4.12 -> added alignment information to imm
  - 4.13 -> added natural alignment information according to data size

Section 1 - Start here

Disclaimer

This document does not (intend to) cover all instructions in the arm64 architecture; the tables presented here only include some of the *instruction families* for each *instruction type*, and some of the specific instructions for each instruction family.

Please refer to the official docs for a complete listing: **developer.arm.com/documentation/ddi0602/2025-09/Index-by-Encoding**

Using this document

1. Going from machine code to assembly instruction
    a. Expand your instruction to 32-bits (e.g., `0x12345678 -> 0001 0010 0011 0100 0101 0110 0111 1000`)
    b. Then, start in **Section 2** and identify the type of instruction you have
    c. Go to the subsection in **Section 3** that corresponds to the type you identified and find the instruction family
    d. Go to the subsection in **Section 4** that corresponds to the instruction family you identified
    e. Find the specific instruction and its action by filling in the placeholder bits with the bits you have
2. Going from assembly instruction to machine code
    a. Find the instruction in **Section 4** and fill in any placeholder bits according to the arguments you have in assembly

Register recap

- x0 ~ x30 are 64-bit integer general purpose registers and w0 ~ w30 are 32-bit integer general purpose registers
- x29 is generally used as the frame pointer (fp) -> points to the start of subroutine's stack frame (doesn't happen automatically)
- x30 is generally reserved to be your link register (lr) -> stores the return address for your subroutine (can be set with bl or blr)
- x31 can be either a zero-filled register (xzr) or the stack pointer (sp) depending on the instruction

Understanding section 4

- The *Format* starts with the MSB (bit 31) on the left and ends with the LSB (bit 0) on the right.
- A 0 or 1 in the format string means the bit is already filled. An x means a bit to be filled based on the instruction description.
- *Count* helps you find a specific bit in the format. It has 1 below bit 31, 0 below bit 30, 9 below bit 29, 8 below bit 28, and so on.

Section 2 - Identify instruction type

Instruction format

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Value	op0			op1												
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value																

Decoding table

Decode fields		**Instruction type**
op0	**op1**	
0	0000	Reserved
1	0000	Scalable Matrix Extension (SME)
	0010	Scalable Vector Extension (SVE)
	00x1	UNALLOCATED
	100x	Data Processing -- Immediate
	101x	Branches, Exception Generating and System instructions
	x101	Data Processing -- Register
	x111	Data Processing -- Scalar Floating-Point and Advanced SIMD
	x1x0	Loads and Stores

Section 3 - Identify instruction family

3.1 - Data Processing -- Immediate

Instruction format

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Value		op0		1	0	0	op1									
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value																

Decoding table

Decode fields		**Instruction family**
op0	**op1**	
11	111x	Data-processing (1 source immediate)
	00xx	PC-rel. addressing
	010x	Add/subtract (immediate)
	0111	Min/max (immediate)
	100x	Logical (immediate)
	101x	Move wide (immediate)

3.2 - Branches, Exception Generating and System instructions

Instruction format

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Value	op0			1	0	1	op1									
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	op1 (continued)											op2				

Decoding table

Decode fields			**Instruction family**
op0	**op1**	**op2**	
010	00xxxxxxxxxxx		Conditional branch (immediate)
110	00xxxxxxxxxxx		Exception generation
110	01000000110010	11111	Hints
110	1xxxxxxxxxxxx		Unconditional branch (register)
x00			Unconditional branch (immediate)
x01	0xxxxxxxxxxxx		Compare and branch (immediate)
x01	1xxxxxxxxxxxx		Test and branch (immediate)

3.3 - Data Processing -- Register

Instruction format

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Value		op0		op1	1	0	1	op2								
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	op3															

Decoding table

Decode fields				**Instruction family**
op0	**op1**	**op2**	**op3**	
0	1	0110		Data-processing (2 source)
1	1	0110		Data-processing (1 source)
	0	0xxx		Logical (register)
	0	1xx0		Add/subtract (shifted register)
	0	1xx1		Add/subtract (extended register)
	1	0000	000000	Add/subtract (with carry)
	1	0100		Conditional select
	1	1xxx		Data-processing (3 source)

3.4 - Loads and Stores

Instruction format

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Value	op0				1	op1	0	op2								
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	op2 (continued)															

Decoding table

Decode fields			Instruction family
op0	op1	op2	
xx00	0	01x1xxxxxxxxxx	Compare and swap
xx01		0xxxxxxxxxxxxx	Load register (literal)
xx10			Load/store register pair
xx11		0xx0xxxxxxxxx1	Load/store register (immediate pre/post-indexed)
xx11		0xx1xxxxxxxx10	Load/store register (register offset)
xx11		1xxxxxxxxxxxxx	Load/store register (unsigned immediate)

Section 4 - Decode instruction

4.01 - Add/subtract (immediate): add(s) / sub(s)

```
Action: Rd = Rn ± ((imm << 12) if sh else imm)
Format:xxx1 0001 0xxx xxxx xxxx xxxx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bit 31 (sf): 0 if 32-bit regs, 1 if 64-bit
Bit 30 (op): 0 if add, 1 if sub
Bit 29 (S): 0 to not set CCs, 1 to set CCs
Bit 22 (sh): 1 if imm<<12, 0 if no shift
Bits 21 ~ 10 (imm): signed immediate
Bits 9 ~ 5 (Rn): source register number
Bits 4 ~ 0 (Rd): destination register number
```

4.02 - Move wide (immediate): mov(z)

```
Action: Rd = imm << (16 * hw)
Format:x101 0010 1xxx xxxx xxxx xxxx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bit 31 (sf): 0 if 32-bit regs, 1 if 64-bit
Bits 22 ~ 21 (hw): shift/16 (xx if sf else 0x)
Bits 20 ~ 5 (imm): signed immediate
Bits 4 ~ 0 (Rd): destination register number
```

4.03 - Unconditional branch (immediate): b(l)

```
Action: PC += (imm*4) *and* X30 = PC+4 *if* op
Format:x001 01xx xxxx xxxx xxxx xxxx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bit 31 (op): 1 for linking, 0 for no linking
Bits 25 ~ 0 (imm): signed immediate
```

4.04 - Unconditional branch (register): b(l)r

```
Action: PC = Rn *and* X30 = PC+4 *if* op
Format:1101 0110 00x1 1111 0000 00xx xxx0 0000
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bit 21 (op): 1 for linking, 0 for no linking
Bits 9 ~ 5 (Rn): address register number
```

4.05 - Conditional branch (immediate): b.cc

```
Action: PC += (imm * 4) if CC else (4)
Format:0101 0100 xxxx xxxx xxxx xxxx xxx0 xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bits 23 ~ 5 (imm): signed immediate
Bits 3~0 (CC): in order from 0000 to 1111:
  (0000) EQ, NE, CS, CC, MI, PL, VS, VC,(0111)
  (1000) HI, LS, GE, LT, GT, LE, AL, NV (1111)
```

4.06 - Compare and branch (immediate): cb(n)z

```
Action: PC += (imm * 4) if (Rt op 0) else (4)
Format:x011 010x xxxx xxxx xxxx xxxx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bit 31 (sf): 0 if 32-bit reg, 1 if 64-bit
Bit 24 (op): 0 for ==, 1 for !=
Bits 23 ~ 5 (imm): signed immediate
Bits 4 ~ 0 (Rt): source register number
```

4.07 - Data-processing (2 source): [s|u]div

```
Action: Rd = Rn / Rm
Format:x001 1010 110x xxxx 0000 1xxx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bit 31 (sf): 0 if 32-bit reg, 1 if 64-bit
Bits 20 ~ 16 (Rm): source register number
Bit 10: 1 for signed division, 0 for unsigned
Bits 9 ~ 5 (Rn): source register number
Bits 4 ~ 0 (Rd): destination register number
```

4.08 - Logical (register): and(s)/bic(s)/orr/orn/eor/eon

```
Action: Rd = Rn op (N(Rm) << uimm)
Format:xxx0 1010 00xx xxxx xxxx xxxx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bit 31 (sf): 0 if 32-bit reg, 1 if 64-bit
Bits 30 ~ 29 (op): 00 and, 01 or,
                   10 xor, 11 ands
Bit 21 (N): if 1, flip Rm (bic/eor/eon/bics)
Bits 20 ~ 16 (Rm): source register number
Bits 15 ~ 10 (uimm): unsigned immediate
Bits 9 ~ 5 (Rn): source register number
Bits 4 ~ 0 (Rd): destination register number
```

4.09 - Add/subtract (shifted register): add(s) / sub(s)

```
Action: Rd = Rn ± (Rm << uimm)
Format:xxx0 1011 000x xxxx xxxx xxxx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bit 31 (sf): 0 if 32-bit regs, 1 if 64-bit
Bit 30 (op): 0 if add, 1 if sub
Bit 29 (S): 0 to not set CCs, 1 to set CCs
Bits 20 ~ 16 (Rm) source register number
Bits 15 ~ 10 (uimm): unsigned immediate
Bits 9 ~ 5 (Rn): source register number
Bits 4 ~ 0 (Rd): destination register number
```

4.10 - Data-processing (3 source): madd / msub / [s|u]maddl / [s|u]msubl

```
Action: Rd = Ra ± Rn * Rm
Format:x001 1011 xxxx xxxx xxxx xxxx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bit 31 (sf): 0 if 32-bit result, 1 if 64-bit
Bits 23~21 (op): 000 for madd/msub,
                 001 for smaddl/smsubl,
                 101 for umaddl/umsubl,
                 x10 see 4.11 for smulh/umulh
Bits 20 ~ 16 (Rm): source register number
Bit 15 (±): 0 for add, 1 for sub
Bits 14 ~ 10 (Ra): source register number
Bits 9 ~ 5 (Rn): source register number
Bits 4 ~ 0 (Rd): destination register number
```

4.11 - Data-processing (3 source): [s|u]mulh

```
Action: Rd = (Rn * Rm) >> 64
Format:1001 1011 x10x xxxx 0111 11xx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bit 23 (s): 1 for unsigned, 0 for signed
Bits 20 ~ 16 (Rm): source register number
Bits 9 ~ 5 (Rn): source register number
Bits 4 ~ 0 (Rd): destination register number
```

4.12 - Load/store register pair: stp / ldp

```
Action: Rt:Rt2 <-dir-> Mem[Rn + imm * 4 or 8];
Format:x010 100x xxxx xxxx xxxx xxxx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bit 31 (sf): 0 if 32-bit regs, 1 if 64-bit
Bits 24 ~ 23 (mode): 10 offset, 11 pre-index,
                     01 post-index, 00 unused
Bit 22 (dir): 1 load (i.e., mem->reg), 0 store
Bits 21 ~ 15 (imm): signed immediate
Bits 14 ~ 10 (Rt2): destination reg number 2
Bits 9 ~ 5 (Rn): source register number
Bits 4 ~ 0 (Rt): destination reg number 1
```

4.13 - Load/store register (unsigned immediate): str(b|h) / ldr(b|h|sb|sh|sw)

```
Action: Rt <-(opc)-> Mem[Rn + (uimm * align)]
Format:xx11 1001 xxxx xxxx xxxx xxxx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bits 31 ~ 30 (size): 00 for byte (align=1),
                     01 for hword (align=2),
                     10 for word (align=4),
                     11 for xword (align=8)
Bits 23 ~ 22(opc): 00 for store,
                   01 for unsigned load,
                   10 for signed load into Xt,
                   11 for signed load into Wt,
Bits 21 ~ 10 (uimm): unsigned immediate
Bits 9 ~ 5 (Rn): address register number
Bits 4 ~ 0 (Rt): source/dest register number
```

4.14 - Load/store register (pre/post-index): str(b|h) / ldr(b|h|sb|sh|sw)

```
Action: Rt <-(opc)-> Mem[Rn + imm]
Format:xx11 1000 xx0x xxxx xxxx x1xx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bits 31 ~ 30 (size of data): same as 4.13
Bits 23 ~ 22 (opc): same as 4.13
Bits 20 ~ 12 (imm): signed immediate
Bit 11: 0 for post-index, 1 for pre-index
Bits 9 ~ 5 (Rn): address register number
Bits 4 ~ 0 (Rt): source/dest register number
```

4.15 - Load/store register (register offset): str(b|h) / ldr(b|h|sb|sh|sw)

```
Action: Rt <-(opc)-> Mem[Rn + Rm]
Format:xx11 1000 xx1x xxxx 0110 10xx xxxx xxxx
Count: 1098 7654 3210 9876 5432 1098 7654 3210
Bits 31 ~ 30 (size of data): same as 4.13
Bits 23 ~ 22 (opc): same as 4.13
Bits 20 ~ 16 (Rm): offset register number
Bits 9 ~ 5 (Rn): address register number
Bits 4 ~ 0 (Rt): source/dest register number
```

4.16 - Hints: nop

```
Action: literally does nothing
Format:1101 0101 0000 0011 0010 0000 0001 1111
Count: 1098 7654 3210 9876 5432 1098 7654 3210
```