LEGv8 operands

Name	Example	Comments
32 registers	X0–X30, XZR	Fast locations for data. In LEGv8, data must be in registers to perform arithmetic, register XZR always equals 0.
2^{62} memory words	Memory[0], Memory[4], . . . , Memory[4,611,686,018,427,387, 904]	Accessed only by data transfer instructions. LEGv8 uses byte addresses, so sequential doubleword addresses differ by 8. Memory holds data structures, arrays, and spilled registers.

LEGv8 assembly language

Category	InstructionExample		Meaning	Comments
Arithmetic	add	ADD X1, X2, X3	X1 = X2 + X3	Three register operands
	subtract	SUB X1, X2, X3	X1 = X2 — X3	Three register operands
	add immediate	ADDI X1, X2, 20	X1 = X2 + 20	Used to add constants
	subtract immediate	SUBI X1, X2, 20	X1 = X2 — 20	Used to subtract constants
	add and set flags	ADDS X1, X2, X3	X1 = X2 + X3	Add, set condition codes
	subtract and set flags	SUBS X1, X2, X3	X1 = X2 — X3	Subtract, set condition codes
	add immediate and set flags	ADDIS X1, X2, 20	X1 = X2 + 20	Add constant, set condition codes
	subtract immediate and set flags	SUBIS X1, X2, 20	X1 = X2 — 20	Subtract constant, set condition codes
Data transfer	load register	LDUR X1, [X2,40]	X1 = Memory[X2 + 40]	Doubleword from memory to register
	store register	STUR X1, [X2,40]	Memory[X2 + 40] = X1	Doubleword from register to memory
	load signed word	LDURSW X1,[X2,40]	X1 = Memory[X2 + 40]	Word from memory to register
	store word	STURW X1, [X2,40]	Memory[X2 + 40] = X1	Word from register to memory
	load half	LDURH X1, [X2,40]	X1 = Memory[X2 + 40]	Halfword memory to register
	store half	STURH X1, [X2,40]	Memory[X2 + 40] = X1	Halfword register to memory
	load byte	LDURB X1, [X2,40]	X1 = Memory[X2 + 40]	Byte from memory to register
	store byte	STURB X1, [X2,40]	Memory[X2 + 40] = X1	Byte from register to memory
	load exclusive register	LDXR X1, [X2,0]	X1 = Memory[X2]	Load; 1st half of atomic swap
	store exclusive register	STXR X1, X3 [X2]	Memory[X2]=X1;X3=0 or 1	Store; 2nd half of atomic swap
	move wide with zero	MOVZ X1,20, LSL 0	X1 = 20 or 20 * 2^{16} or 20 * 2^{32} or 20 * 2^{48}	Loads 16-bit constant, rest zeros
	move wide with keep	MOVK X1,20, LSL 0	X1 = 20 or 20 * 2^{16} or 20 * 2^{32} or 20 * 2^{48}	Loads 16-bit constant, rest unchanged
Logical	and	AND X1, X2, X3	X1 = X2 & X3	Three reg. operands; bit-by-bit AND
	inclusive or	ORR X1, X2, X3	X1 = X2 | X3	Three reg. operands; bit-by-bit OR
	exclusive or	EOR X1, X2, X3	X1 = X2 ^ X3	Three reg. operands; bit-by-bit XOR
	and immediate	ANDI X1, X2, 20	X1 = X2 & 20	Bit-by-bit AND reg. with constant
	inclusive or immediate	ORRI X1, X2, 20	X1 = X2 | 20	Bit-by-bit OR reg. with constant
	exclusive or immediate	EORI X1, X2, 20	X1 = X2 ^ 20	Bit-by-bit XOR reg. with constant
	logical shift left	LSL X1, X2, 10	X1 = X2 << 10	Shift left by constant
	logical shift right	LSR X1, X2, 10	X1 = X2 >> 10	Shift right by constant
Conditional branch	compare and branch on equal 0	CBZ X1, 25	if (X1 == 0) go to PC + 100	Equal 0 test; PC-relative branch
	compare and branch on not equal 0	CBNZ X1, 25	if (X1 != 0) go to PC + 100	Not equal 0 test; PC-relative branch
	branch conditionally	B.cond 25	if (condition true) go to PC + 100	Test condition codes; if true, branch
Unconditional branch	branch	B 2500	go to PC + 10000	Branch to target address; PC-relative
	branch to register	BR X30	go to X30	For switch, procedure return
	branch with link	BL 2500	X30 = PC + 4; PC + 10000	For procedure call PC-relative