ARM Instruction Set Reference

Instruction	Mnemomic	Parameters	Description	Comments
Add	ADD ADDS	**r0**, r1, r2	[r0] ⟵ [r1] + [r2]	Add source operand one to source operand two.
Add with carry	ADC	**r0**, r1, r2	[r0] ⟵ [r1] + [r2] + [C]	Add source operand one to source operand two and include any carry (carry bit = 1).
Subtract	SUB	**r0**, r1, r2	[r0] ⟵ [r1] – [r2]	Subtract the second source operand from the first.
Subtract with carry	SBC	**r0**, r1, r2	[r0] ⟵ [r1] – [r2] – [C]	Subtract the second source operand from the first and include any borrow (carry bit = 0).
Reverse Subtract	RSB no Negation -> use RSB with #0	**r0**, r1, r2	[r0] ⟵ [r2] – [r1]	Subtract first source operand from the second.
Reverse Subtract with carry	RSC	**r0**, r1, r2	[r0] ⟵ [r2] – [r1] - [C]	Subtract the first source operand from the second and include any borrow (carry bit = 0).
Multiply no Division	MUL (can't mul. to constant, only register with register)	**r0**, r1, r2	[r0] ⟵ [r1] x [r2]	Multiply the two source operands. Result is limited to 32-bits.
Multiply and add	MLA	**r0**, r1, r2, r3	[r0] ⟵ [r3] + ([r1] x [r2])	Multiply the two source operands and add the result to the value in the last register.
Multiply and subtract	MLS	**r0**, r1, r2, r3	[r0] ⟵ [r3] – ([r1] x [r2])	Multiply the two source operands and subtract the result from the value in the last register.
Multiply long signed	SMULL	**r0, r1**, r2, r3	[r1,r0] ⟵ [r2] x [r3]	Multiply the two 32-bit source operands (r2 and r3) to obtain the 64-bit result stored in (r1, r0).
Multiply long unsigned	UMULL	**r0, r1**, r2, r3	[r1,r0] ⟵ [r2] x [r3]	Multiply the two 32-bit source operands (r2 and r3) to obtain the 64-bit result stored in (r1, r0).
Multiply long signed and add	SMLAL	**r0, r1**, r2, r3	[r1, r0] ⟵ [r1,r0] + ([r2] x [r3])	Multiply the two 32-bit source operands (r2 and r3) and add the 64-bit result to the current value in (r1, r0).
Multiply long unsigned and subtract	UMLAL no SMLSL/UMLSL	**r0, r1**, r2, r3	[r1, r0] ⟵ [r1,r0] – ([r2] x [r3])	Multiply the two 32-bit source operands (r2 and r3) and subtract the 64-bit result from the current value in (r1, r0).
And	AND	**r0**, r1, r2	[r0] ⟵ [r1] & [r2]	Logic AND of the two source operands.
Or	ORR	**r0**, r1, r2	[r0] ⟵ [r1] | [r2]	Logical OR of the two source operands
Exclusive Or	EOR no NOT -> use EOR	**r0**, r1, r2	[r0] ⟵ [r1] ⊕ [r2] 0 1 0 1 1 1 1 1 1 0 1 0 (EOR result for NOT-operation)	Exclusive-OR of the two source operands.
Bit clear	BIC	**r0**, r1, r2	[r0] ⟵ [r1] & ![r2]	Clear the bits in the first source register corresponding to each bit in the second source register which is a 1.
Logical Shift Left	LSL LSLS	**r0**, r1, r2	padding with 0. Shift to the left is the same as MULTIPLY by power of 2, but be careful about magnitude. Shift and rotate can be combined with other operations. (ADD r0,r1,r2, LSL #1 ; [r0] <- [r1] + [r2]x2) (MOV r3,r3, LSL #4 ; shift a register) (MOV r4,r3, LSL r1 ; dynamic shift)	Shift the value in the first source operand left by the number of bits of the second source operand, with 0s shifted into the LSB.
Logical Shift Right	LSR LSRS no ASL -> use LSL	**r0**, r1, r2	padding with 0. Shift to the right is the same as DIVIDE by power of 2, but be careful about magnitude. Shift and rotate can be combined with other operations.	Shift the value in the first source operand right by the number of bits of the second source operand, with 0s shifted into the MSB.
Arithmetic Shift Right	ASR ASRS	**r0**, r1, r2	padding with 0 for positive number. padding with 1 for negative number. Shift and rotate can be combined with other operations.	Shift the value in the first source operand right by the number of bits of the second source operand, with value of the MSB (sign bit) preserved.
Rotate Right	no ROL ROR RORS	**r0**, r1, r2	Shift and rotate can be combined with other operations.	Shift the value in the first source operand right by the number of bits of the second source operand, with the bits rotated out the LSB wrapping around to the MSB.
Rotate Right Extended	no RLX -> use ADCS r0,r0,r0 RRX RRXS	**r0**, r1	Careful the ONE bit-in is the CARRY bit, not past LSBit. Note that this instruction only shifts ONE bit. Shift and rotate can be combined with other operations.	Shift the value in the first source operand right by one bit with the bit rotated out of the LSB going into the carry, and the current carry bit shifted into the MSB.
Compare	CMP	r1, r2	[r1] – [r2]	Compare two registers by subtracting them and update the CPSR. w/o explicitly putting "S" in instruction
Compare Negative	CMN	r1, r2	[r1] – (-[r2])	Compare two registers by subtracting the negative of the second from the first, and update the CPSR. w/o explicitly putting "S" in instruction
Test	TST	r1, r2	[r1] & [r2]	Logical AND of the two source operands and update the CPSR. w/o explicitly putting "S" in instruction
Test equivalence	TEQ	r1, r2	[r1] ⊕ [r2]	Exclusive-OR of the two source operands and update the CPSR. w/o explicitly putting "S" in instruction

Tuesday, March 10, 2020

ARM Instruction Set Reference

Instruction	Mnemomic	Parameters	Description	Comments
Move	MOV	**r0,** r1	[r0] ⟵ [r1]	Copy the contents of one register into another.
Address	ADR	**r0,** target	[r0] ⟵ address(target)	Load register with address of target memory location.
Load into register	LDR	**r0,**[address]	[r0] ⟵ [address]	Load register with the value from a memory address.
Block load into registers	LDMIA LDMFD	r13!, {r list}		Load for memory into the registers in ‘r list’, beginning at the address which the first operand points to and *incrementing* the address *after* each location is read.
	LDMIB LDMED	r13!, {r list}		Load for memory into the registers in ‘r list’, beginning at the address which the first operand points to and *incrementing* the address *before* each location is read.
	LDMDA LDMFA	r13!, {r list}		Load for memory into the registers in ‘r list’, beginning at the address which the first operand points to and *decrementing* the address *after* each location is read.
	LDMDB LDMEA	r13!, {r list}		Load for memory into the registers in ‘r list’, beginning at the address which the first operand points to and *decrementing* the address *before* each location is read.
Store from register	STR	r0,[**address**]	[r0] ⟶ [address]	Store a register’s value into a memory location.
Block store from registers	STMIA STMEA	r13!, {r list}		Store the values in the registers in ‘r list’ into memory, beginning at the address which the first operand points to and *incrementing* the address *after* each is stored.
	STMIB STMFA	r13!, {r list}		Store the values in the registers in ‘r list’ into memory, beginning at the address which the first operand points to and *incrementing* the address *before* each is stored.
	STMDA STMED	r13!, {r list}		Store the values in the registers in ‘r list’ into memory, beginning at the address which the first operand points to and *decrementing* the address *after* each is stored.
	STMDB STMFD	r13!, {r list}		Store the values in the registers in ‘r list’ into memory, beginning at the address which the first operand points to and *decrementing* the address *before* each is stored.

added to any instruction to check CPSR FIRST before executing that instruction

Branches and Conditionals (15 + 1 instructions - encoded from 0000 to 1111)

ADDEQ
ADDSEQ
(different)

Condition	Mnemonic	Branch	CSPR Bits Tested	Description
Equal	EQ	BEQ	Z = 1	Zero
Not equal	NE	BNE	Z = 0	Not zero
Carry set	CS	BCS	C = 1	Carry or no borrow
Carry clear	CC	BCC	C = 0	No carry or borrow
Negative	MI	BMI	N = 1	Negative
Positive or zero	PL	BPL	N = 0	Not negative
Overflow	VS	BVS	V = 1	Overflow
No overflow	VC	BVC	V = 0	No overflow
Higher	HI	BHI	C = 1 AND Z = 0	Unsigned higher
Lower or same	LS	BLS	C = 0 OR Z = 1	Unsigned lower or same
Greater than	GT	BGT	Z = 0 AND N = V	Signed greater than
Greater or equal	GE	BGE	N = V	Signed greater than or equal
Less than	LT	BLT	N = !V	Signed less than
Less than or equal	LE	BLE	Z = 1 OR N != V	Signed less than or equal
Branch always		B		

Always (default) AL
Never (reserved) NV

Branch-with-Link: To implement a branch-with-link, use ‘`BL{condition}`’ in place of the ‘`B{condition}`’. For example, ‘`BNE target`’ would become ‘`BLNE target`’.

Tuesday, March 10, 2020