Assembly instructions - ARM V7

Category	Instruction	Mnemonic	Meaning
Arithmetic	Add	ADD{S}{cond} dest, op1, op2 {, SHIFT_op #expression}	dest = op1 + op2
	Subtract	SUB{S}{cond} dest, op1, op2 {, SHIFT_op #expression}	dest = op1 - op2
	Add with Carry	ADC{S}{cond} dest, op1, op2 {, SHIFT_op #expression}	dest = op1 + op2 + carry
	Subtract with Carry	SBC{S}{cond} dest, op1, op2 {, SHIFT_op #expression}	dest = op1 - op2 -1 + carry
	Reverse Subtract	RSB{S}{cond} dest, op1, op2 {, SHIFT_op #expression}	dest = op2 - op1
	Reverse Subtract with Carry	RSC{S}{cond} dest, op1, op2 {, SHIFT_op #expression}	dest = op2 - op1 - 1 + carry
Logic	Bitwise And	AND{S}{cond} dest, op1, op2 {, SHIFT_op #expression}	dest = AND(op1, op2)
	Bitwise Exclusive Or	EOR{S}{cond} dest, op1, op2 {, SHIFT_op #expression}	dest = XOR(op1, op2)
	Bitwise Clear	BIC{S}{cond} dest, op1, op2 {, SHIFT_op #expression}	dest = AND(op1, NOT(op2))
	Bitwise Or	ORR{S}{cond} dest, op1, op2 {, SHIFT_op #expression}	dest = OR(op1, op2)
	Logical Shift Left	LSL{S}{cond} dest, op1, op2	dest = op1 << op2
	Logical Shift Right	LSR{S}{cond} dest, op1, op2	dest = op1 >> op2
	Arithmetic Shift Right	ASR{S}{cond} dest, op1, op2	dest = op1 >> op2 (signed extension)
	Rotate Right	ROR{S}{cond} dest, op1, op2	
	Rotate Right and Extend	RRX{S}{cond} dest, op1	
Data transfer	Move	MOV{S}{cond} dest, op1 {, SHIFT_op #expression}	dest = op1
	Move Negated	MVN{S}{cond} dest, op1 {, SHIFT_op #expression}	dest = NOT(op1)
	Address Load	ADR{S}{cond} dest, expression	
	LDR Pseudo-Instruction	LDR{S}{cond} dest, =expression	
	Load Register	LDR{B}{cond} dest, [source {, OFFSET}] Offset addressing	dest = Mem[source + OFFSET]
	Store Register	STR{B}{cond} source, [dest {, OFFSET}] Offset addressing	Mem[dest + OFFSET] = source
Comparisons and jumps	Compare	CMP{cond} op1, op2 {, SHIFT_op #expression}	
	Compare Negated	CMN{cond} op1, op2 {, SHIFT_op #expression}	
	Test Bit(s) Set	TST{cond} op1, op2 {, SHIFT_op #expression}	
	Test Equals	TEQ{cond} op1, op2 {, SHIFT_op #expression}	
	Branch	B{cond} target	
	Branch with Link	BL{cond} target	
Directives	Declare Word(s) in Memory	name DCD value_1, value_2, ... value_N	
	Declare Constant	name equ expression	
	Declare Empty Word(s) in Memory	{name} FILL N, N must be a multiple of 4	
	Stop Emulation	END{cond}	

Notes:

- For all instructions that require **dest**, **op1**, & **op2**, **dest** and **op1** must be registers.
- **expression** is a numerical constant or expression that evaluates to a 32-bit number. The operators **+**, **-** and ***** are allowed. A constant is a decimal number as a series of digits **0-9**, a hexadecimal number prefixed with **0x** or **&** or a binary number prefixed with **0b**.
- For MOV / MVN, **dest** must be a register.
- For CMP / CMN / TST / TEQ, **op1** must be a register.
- **{...}** indicates optional code.
- **{cond}** refers to the **condition code**. (See the table "Condition code sufixes")
- **{S}** is the **set bit**. If this is present, the status bits (**Flags**) will be set.
- **{, SHIFT_op #expression}** means a shift operation can be performed on **op2** as part of the same instruction. The shifted version of **op2** is then used as the second operand for the main instruction.
  - **SHIFT_op** can be any one of **LSL, LSR, ASR, ROR, RRX**.
  - With the exception of RRX, **#expression** can be a register from R0 through R15 or any numerical expression.
  - **op2** cannot be R15 or PC if a shift is being used.
  - For MOV / MVN / ADR, **op1** is used instead.
- **{, OFFSET}** refers to the offset applied to the source address or destination address for load and store instructions respectively. It can be a register, a numerical expression, or a shifted register (like the flexible second operand discussed earlier).
- **{B}** refers to **byte mode**. By default, the LDR / STR instructions load a word (32 bits) from the memory at the given address. If B is used, the byte at the given address is loaded instead.
- The **target** for a branch instruction **B** must be a label on a line. This instruction will cause the program to "jump", i.e. branch, to this line of code.
- Branch with link **BL** is identical to branch, with the additional function that the link register is set to point to the next line of code before the branch is performed. This can be used to return from a subroutine.
- For **EQU**, **expression** can be any numerical expression

Condition code suffixes					
Suffix	**Flags**	**Meaning**	**Suffix**	**Flags**	**Meaning**
EQ	Z = 1	Equal	VC	V = 0	No overflow
NE	Z = 0	Not equal	HI	C = 1 and Z = 0	Higher, unsigned
CS or HS	C = 1	Higher or same, unsigned	LS	C = 0 or Z = 1	Lower or same, unsigned
CC or LO	C = 0	Lower, unsigned	GE	N = V	Greater than or equal, signed
MI	N = 1	Negative	LT	N != V	Less than, signed
PL	N = 0	Positive or zero	GT	Z = 0 and N = V	Greater than, signed
VS	V = 1	Overflow	LE	Z = 1 and N != V	Less than or equal, signed

The condition flags	
Name	**Behavior**
N	Set to 1 when the result of the operation was negative, cleared to 0 otherwise.
Z	Set to 1 when the result of the operation was zero, cleared to 0 otherwise.
C	Set to 1 when the operation resulted in a carry, cleared to 0 otherwise.
V	Set to 1 when the operation caused overflow, cleared to 0 otherwise.

Registers	
Name	**Description**
R0-R12	General-purpose registers. By convention, registers R0 to R3 are used to pass arguments to subroutines, and R0 is used to pass a result back to the callers. A subroutine that needs more than 4 inputs uses the stack for the additional inputs.
SP	*Stack Pointer*
LR	*Link Register*
PC	Program Counter