ARM32 Information:

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3.
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

The first four registers r0-r3 (a1-a4) are used to pass argument values into a subroutine and to return a result value from a function. They may also be used to hold intermediate values within a routine (but, in general, only *between* subroutine calls).

Register r12 (IP) may be used by a linker as a scratch register between a routine and any subroutine it calls (for details, see *§5.3.1.1, Use of IP by the linker*). It can also be used within a routine to hold intermediate values *between* subroutine calls.

The role of register r9 is platform specific. A virtual platform may assign any role to this register and must document this usage. For example, it may designate it as the static base (SB) in a position-independent data model, or it may designate it as the thread register (TR) in an environment with thread-local storage. The usage of this register may require that the value held is persistent across all calls. A virtual platform that has no need for such a special register may designate r9 as an additional callee-saved variable register, v6.

Typically, the registers r4-r8, r10 and r11 (v1-v5, v7 and v8) are used to hold the values of a routine's local variables. Of these, only v1-v4 can be used uniformly by the whole Thumb instruction set, but the AAPCS does not require that Thumb code only use those registers.

A subroutine must preserve the contents of the registers r4-r8, r10, r11 and SP (and r9 in PCS variants that designate r9 as v6).

ARM64 information:

Register	Special	Role in the procedure call standard
SP		The Stack Pointer.
r30	LR	The Link Register.
r29	FP	The Frame Pointer
r19 r28		Callee-saved registers
r18		The Platform Register, if needed; otherwise a temporary register. See notes.
r17	IP1	The second intra-procedure-call temporary register (can be used by call veneers and PLT code); at other times may be used as a temporary register.
r16	IP0	The first intra-procedure-call scratch register (can be used by call veneers and PLT code); at other times may be used as a temporary register.
r9 r15		Temporary registers
r8		Indirect result location register
r0 r7		Parameter/result registers

ARM assembly language reference card

MOVcdS	*reg*, *arg*	copy argument (S = set flags)	Bcd	imm_{12}	branch to imm_{12} words away
MVNcdS	*reg*, *arg*	copy bitwise NOT of argument	BLcd	imm_{12}	copy PC to LR, then branch
ANDcdS	*reg*, *reg*, *arg*	bitwise AND	BXcd	*reg*	copy *reg* to PC
ORRcdS	*reg*, *reg*, *arg*	bitwise OR	SWIcd	imm_{24}	software interrupt
EORcdS	*reg*, *reg*, *arg*	bitwise exclusive-OR	LDRcdB	*reg*, *mem*	loads word/byte from memory
BICcdS	*reg*, reg_a, arg_b	bitwise reg_a AND (NOT arg_b)	STRcdB	*reg*, *mem*	stores word/byte to memory
ADDcdS	*reg*, *reg*, *arg*	add	LDMcd*um*	*reg*!, *mreg*	loads into multiple registers
SUBcdS	*reg*, *reg*, *arg*	subtract	STMcd*um*	*reg*!, *mreg*	stores multiple registers
RSBcdS	*reg*, *reg*, *arg*	subtract reversed arguments	SWPcdB	reg_d, reg_m, [reg_n]	copies reg_m to memory at reg_n, old value at address reg_n to reg_d
ADCcdS	*reg*, *reg*, *arg*	add with carry flag			
SBCcdS	*reg*, *reg*, *arg*	subtract with carry flag			
RSCcdS	*reg*, *reg*, *arg*	reverse subtract with carry flag			
CMPcd	*reg*, *arg*	update flags based on subtraction			
CMNcd	*reg*, *arg*	update flags based on addition			
TSTcd	*reg*, *arg*	update flags based on bitwise AND			
TEQcd	*reg*, *arg*	update flags based on bitwise exclusive-OR			

MULcdS	reg_d, reg_a, reg_b	multiply reg_a and reg_b, places lower 32 bits into reg_d
MLAcdS	reg_d, reg_a, reg_b, reg_c	places lower 32 bits of $reg_a \cdot reg_b + reg_c$ into reg_d
UMULLcdS	reg_ℓ, reg_u, reg_a, reg_b	multiply reg_a and reg_b, place 64-bit unsigned result into $\{reg_u, reg_\ell\}$
UMLALcdS	reg_ℓ, reg_u, reg_a, reg_b	place unsigned $reg_a \cdot reg_b + \{reg_u, reg_\ell\}$ into $\{reg_u, reg_\ell\}$
SMULLcdS	reg_ℓ, reg_u, reg_a, reg_b	multiply reg_a and reg_b, place 64-bit signed result into $\{reg_u, reg_\ell\}$
SMLALcdS	reg_ℓ, reg_u, reg_a, reg_b	place signed $reg_a \cdot reg_b + \{reg_u, reg_\ell\}$ into $\{reg_u, reg_\ell\}$

reg: register

R0 to R15	register according to number
SP	register 13
LR	register 14
PC	register 15

um: update mode

IA	increment, starting from *reg*
IB	increment, starting from *reg* + 4
DA	decrement, starting from *reg*
DB	decrement, starting from *reg* − 4

cd: condition code

AL or omitted	always
EQ	equal (zero)
NE	nonequal (nonzero)
CS	carry set (same as HS)
CC	carry clear (same as LO)
MI	minus
PL	positive or zero
VS	overflow set
VC	overflow clear
HS	unsigned higher or same
LO	unsigned lower
HI	unsigned higher
LS	unsigned lower or same
GE	signed greater than or equal
LT	signed less than
GT	signed greater than
LE	signed less than or equal

arg: right-hand argument

#imm_{8*}	immediate (rotated into 8 bits)
reg	register
reg, *shift*	register shifted by distance

mem: memory address

[*reg*, #±imm_{12}]	*reg* offset by constant
[*reg*, ±*reg*]	*reg* offset by variable bytes
[reg_a, ±reg_b, *shift*]	reg_a offset by shifted variable reg_b†
[*reg*, #±imm_{12}]!	update *reg* by constant, then access memory
[*reg*, ±*reg*]!	update *reg* by variable bytes, access memory
[*reg*, ±*reg*, *shift*]!	update *reg* by shifted variable†, access memory
[*reg*], #±imm_{12}	access address *reg*, then update *reg* by offset
[*reg*], ±*reg*	access address *reg*, then update *reg* by variable
[*reg*], ±*reg*, *shift*	access address *reg*, update *reg* by shifted variable†

† shift distance must be by constant

shift: shift register value

LSL #imm_5	shift left 0 to 31
LSR #imm_5	logical shift right 1 to 32
ASR #imm_5	arithmetic shift right 1 to 32
ROR #imm_5	rotate right 1 to 31
RRX	rotate carry bit into top bit
LSL *reg*	shift left by register
LSR *reg*	logical shift right by register
ASR *reg*	arithmetic shift right by register
ROR *reg*	rotate right by register