G51CSA ARM Instruction Set Summary

This listing is sufficient for use with the G51CSA module. However it is not an exhaustive listing so for more general use you may need to consult other documentation.

Data processing

ADD	Rd, Rn, Op	Rd = Rn + Op
SUB	Rd, Rn, Op	Rd = Rn – Op
RSB	Rd, Rn, Op	Rd = Op – Rn ("reverse subtract")
AND	Rd, Rn, Op	Rd = Rn AND Op
ORR	Rd, Rn, Op	Logical OR
EOR	Rd, Rn, Op	Exclusive OR
BIC	Rd, Rn, Op	Bit Clear; Rd = Rn AND NOT Op
MOV	Rd, Op	Rd = Op
MVN	Rd, Op	Rd = NOT Op
CMP	Rn, Op	set status on Rn – Op
CMN	Rn, Op	set status on Rn + Op
TST	Rn, Op	set status on Rn AND Op
TEQ	Rn, Op	set status on Rn EOR Op
MUL	Rd, Rm, Rs	Rd = Rm * Rs
MLA	Rd, Rm, Rs, Rn	Rd = (Rm * Rs) + Rn

An "Op" in the table above can be a literal (e.g. **`#10`**) or a register (e.g. **`R1`**) or a shifted register.

If a shift (by a literal e.g. **`#3`** or a register e.g. **`R6`**) is required it is specifed as the fourth parameter e.g.
`ADD R1, R4, R5, LSL #3` ; R1 = R4 + R5 * 8
`ADD R1, R4, R5, LSL R6` ; R1 = R4 + R5 * 2^{R6}

LSL Shift	logical shift left by 0 ≤ Shift ≤ 31 places, putting 0s into least significant end
LSR Shift	logical shift right by 0 ≤ Shift ≤ 32 places, putting 0s in most significant end
ASR Shift	arithmetic shift right by 0 ≤ Shift ≤ 32 places, copying the sign bit into the most significant end
ROR Shift	circular rotate right by 0 ≤ Shift ≤ 32 places, moving bits lost from one end into other end
RRX	One place shift right. Carry from status reg. shifts into most significant bit. If status reg. set by instruction, carry bit = least significant bit. This gives a 1-bit circular rotate through the carry bit

Loads and Stores

LDR	Rd, Address	loads a 32-bit word into Rd from memory location
LDRB	Rd, Address	loads 8-bits into Rd; top 24 bits are 0s
STR	Rd, Address	stores Rd at memory location
STRB	Rd, Address	stores bottom byte of Rd at memory location
LDMFD	Rd, register list	multiple reg. load,load from addr. Rd
STMFD	Rd!, register list	multiple reg. store, Rd is updated after each store operation

An "Address" in the table above can be a numerical address (e.g. **`100`**) or a named memory location (e.g. **`fred`**) or calculated from the contents of registers and literals e.g.:

operand form	address	final value of R0
[R0]	R0	(unchanged)
[R0, R1]	R0 + R1	(unchanged)
[R0, #1]	R0 + 1	(unchanged)
[R0, R1]!	R0 + R1	R0 + R1
[R0, #1]!	R0 + 1	R0 + 1
[R0], R1	R0	R0 + R1
[R0], #1	R0	R0 + 1

In the examples above the value copied from **`R1`** can be modified by a shift (but **`R1`** itself is unchanged) e.g.
`LDR R2, [R0, R1, LSL #3]` ; address = R0 + 8 * R1
The possible shifts are **`LSL`**, **`LSR`**, **`ASL`** and **`ROR`** as described for Data Processing instructions above.

G51CSA ARM Instruction Set Summary – continued

Control Transfer

B	Label	branch to label: R15 = label
BL	Function	branch and link: R14 = R15, R15 = start address of function
SWI	Number	software interrupt

Condition Codes

Any instruction can be made conditional e.g. **`ADD`** can become **`ADDEQ`** etc.

Code	Meaning	Flag condition
EQ	Equal	Z
NE	Not Equal	!Z
GE	Greater than or Equal (signed)	N = V
GT	Greater Than (signed)	(N = V) . !Z
LT	Less Than (signed)	N != V
LE	Less than or Equal (signed)	(N != V) + Z
HI	Higher (unsigned)	C . !Z
LS	Lower or Same (unsigned)	!C + Z
CS/HS	Carry Set/Higher or Same (unsigned)	C
CC/LO	Carry Clear/Lower (unsigned)	!C
MI	Minus (negative)	N
PL	Plus (positive)	!N
VS	Overflow Set	V
VC	Overflow Clear	!V
AL	Always	TRUE

Also, any Data Processing instruction can be followed by an “S” e.g. **`ADD`** can become **`ADDS`** meaning that the result of the instruction is to be compared with zero. Note that the “S” comes after any condition code, e.g. **`ADDNES`**

Assembler Supplied ‘Pseudo Operations’

NOP		no operation
MOV	Rd, #–nn	replaced by MVN
CMP	Rn, #–nn	replaced by CMN
ADR	Rd, label	Rd = label address. Replaced by ADD Rd, R15, #nn
ADRL	Rd, label	Rd = label address. Used when #nn in ADR is out of range
LDR	Rd, =nnnnnnnn	Load constant

Assembler Directives

ENTRY		Mark the code entry point
ORIGIN	0xaddress	Set start address of the code or data that follows
ALIGN		Align to next 4-byte boundary
EQU	0x2A	Define a compile-time constant
DEFW	0x12345678	Define the value of the next word(s)
DEFB	0, 2, "bytes"	Define the value of the next byte(s)
DEFS	0x20	Reserve the next 20 (hex) bytes of storage

DEFB, DEFW and DEFS allow data to be planted amongst the instructions in a program.

SWI options under KoMoDo

The following SWI options are supported by KoMoDo in emulator mode

SWI 0	Output the single character in R0 to the ‘Features’ window
SWI 1	Read in, to R0, a single character typed at the ‘Features’ window
SWI 2	Halt KoMoDo
SWI 3	Print a string, whose start address is in R0, to the ‘Features’ window
SWI 4	Print out, to the ‘Features’ window, in decimal, the integer stored in R0