Keil Arm Assembly Language NOTES Commands:

-LABEL: Name of a Function
-REGISTERS: R0-R15 **R13**=SP(Stack Pointer) **R14**=LR(Link Register[Return Address]) **R15**=PC(Program Counter)
-ARM is a **load-store** architecture language **WORD** = 4 Bytes
-Data Definition Directive:
-DCD (Define Constant Data) allocates 1+ words of memory, aligned on 4-byte boundaries, & defines initial runtime contents of memory. '**&**' is a synonym for DCD.
-DCB: 1+ bytes of memory & defines initial runtime contents of the memory.
-**Load**: Data transfer instruction. Copies data from memory to a register.
-**base address**: Starting address of an array in memory
-**base register**: holds array's base address
-**offset**: a constant value added to a base address. Locates a particular array element
-Instruction Types: **R**-Type / **D**-Type / **I**-Type (Immediate Constant)
R-Type (Register type) = Jumps/Arithmetic/System Calls D-Type(Load & Store)

COMMANDS: Arithmetic / Suffixes / Load&Store / Multiplication

Arithmetic

Instruction	Example	Meaning	Comments
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

MOV / MOVN??

Suffixes

Table 5-1 Condition code suffixes and related flags

Suffix	Flags	Meaning
`EQ`	`Z` set	Equal
`NE`	`Z` clear	Not equal
`CS` or `HS`	`C` set	Higher or same (unsigned >=)
`CC` or `LO`	`C` clear	Lower (unsigned <)
`MI`	`N` set	Negative
`PL`	`N` clear	Positive or zero
`VS`	`V` set	Overflow
`VC`	`V` clear	No overflow
`HI`	`C` set and `Z` clear	Higher (unsigned >)
`LS`	`C` clear or `Z` set	Lower or same (unsigned <=)
`GE`	`N` and `V` the same	Signed >=
`LT`	`N` and `V` differ	Signed <
`GT`	`Z` clear, `N` and `V` the same	Signed >
`LE`	`Z` set, `N` and `V` differ	Signed <=
`AL`	Any	Always. This suffix is normally omitted.

The optional condition code is shown in syntax descriptions as `{cond}`. This condition is encoded in a preceding `IT` instruction. An instruction with a condition code is only executed if the condition flags in the APSR meet the

Load & Store

LDR -> Load word to register
STR ->Save word from register
LDRB -> Load Byte to register
STRB ->Save Byte from register
LDRH ->Load Halfword to register
STRH ->Save Halfword from register

Multiply Instructions

Mnemonic	Meaning	Register Operations
MLA	Multiply and Accumulate	Rd = (Rm*Rs)+Rn
MUL	Multiply	Rd = (Rm*Rs)
UMLAL	Multiply and Accumulate Long (unsigned)	[RdHi,RdLo] = [RdHi,RdLo]+ (Rm*Rs)
UMULL	Multiply long (unsigned)	[RdHi,RdLo] = (Rm*Rs)
SMLAL	Multiply and Accumulate Long (signed)	[RdHi,RdLo] = [RdHi,RdLo]+ (Rm*Rs)
SMULL	Multiply long (signed)	[RdHi,RdLo] = (Rm*Rs)

Division

Mnemonic	Meaning	Register Operations
UDIV	Unsigned division	UDIV R0,R1,R2 => R0=R1/R2(Unsigned)
SDIV	Signed division	SDIV R0,R1,R2 => R0=R1/R2 (Signed)

Branching - CBZ / CBNZ

Conditional Branch (NOT) Zero

- **CBZ** register, LabelName
  - **CBZ** R0, goto1
- **CBNZ** register, LabelName
  - **CBNZ** R0, goto1

Looping

Example Loops

```
while(a<5)
{
 a=a+1
}
```

```
        LDR R4,=a
        LDR R1,[R4]
loop1
        SUB R2,R1,#5
        CBZ R2,exit
        ADD R1,R1,#1
        B loop1
exit
stop    B stop

a DCD 1
    END
```

Branching with Conditions

Conditional Branching

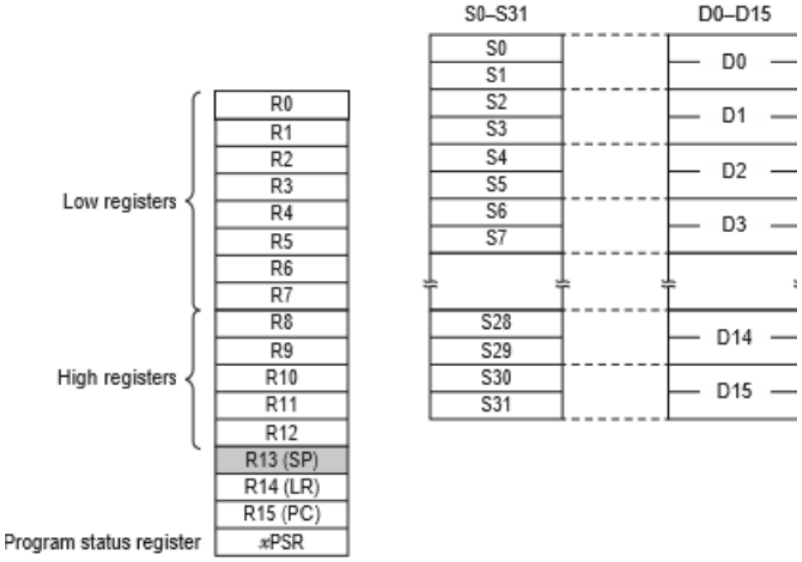
	Signed		Unsigned	
Comparison	Instruction	Flag Test	Instruction	Flag Test
=	BEQ	$Z = 1$	BEQ	$Z = 1$
≠	BNE	$Z = 0$	BNE	$Z = 0$
<	BLT	$N! = V$	BLO	$C = 0$
≤	BLE	$\sim(Z = 0 \,\&\, N = V)$	BLS	$\sim(Z = 0 \,\&\, C = 1)$
>	BGT	$(Z = 0 \,\&\, N = V)$	BHI	$(Z = 0 \,\&\, C = 1)$
≥	BGE	$N = V$	BHS	$C = 1$

Compare 1=True

Mnemonic	Meaning	Register Operations
CMN	Compare Negated	Flags Set -> Rn+Operand 2
CMP	Compare	Flags Set -> Rn-Operand2
TEQ	Test for quality-two 32 bit values	Flags Set -> Rn ^ Operand2

Floating Point Numbers

```
LDR r3, =0x3F800000 ; single precision 1.0
VMOV.F32 s3, r3 ; transfer contents from ARM to FPU
VLDR.F32 s4, =6.02
VMOV.F32 r4, s4 ; transfer contents from FPU to ARM
```




```
LDR r0, ='!'
TEQ r0,#'!'
TEQNE r0,#'?'
ADDEQ r1,r1,#1
```
