Cortex-M4 Thumb-2 Instruction Set Summary

**<Operand2>** may be one of the following:

- #imm8: One byte, zero-extended to 32 bits (a few other formats can also be produced)
- Rm: Normal register operation
- Rm, <LSL|LSR|ASR|ROR> #imm5: Register operation with constant shift
- Rm, RRX: Register operation with rotate right with extend

**<S>** (instruction mnemonic suffix) => Update the condition flags after the instruction has executed

- MOV{S} Rd, <Operand2>: Move Rd = Operand2
- MVN{S} Rd, <Operand2>: Move not Rd = 0xFFFFFFFF EOR Operand2
- MOV Rd, #<imm16>: Move wide Rd = imm16 (zero-extended)
- MOVT Rd, #<imm16>: Move top Rd[31:16] = imm16, bits Rd[15:0] are unaffected
- ADD{S} Rd, Rn, <Operand2>: Add Rd = Rn + Operand2
- ADD Rd, Rn, #<imm12>: Add wide Rd = Rn + Imm12
- ADC{S} Rd, Rn, <Operand2>: Add with carry Rd = Rn + Operand2 + Carry
- SUB{S} Rd, Rn, <Operand2>: Subtract Rd = Rn - <Operand 2>
- SBC{S} Rd, Rn, <Operand2>: Subtract with carry Rd = Rn – Operand2 - (1 - Carry)
- SUB Rd, Rn, #<imm12>: Subtract wide Rd = Rn - imm12
- RSB{S} Rd, Rn, <Operand2>: Reverse subtract Rd = <Operand 2> - Rn
- RSC{S} Rd, Rn, <Operand2>: Reverse subtract with carry Rd = Operand2 – Rn – (1 - Carry)
- MUL{S} Rd, Rm, Rs: Multiply Rd = Rn * Rm Returns the 32 least significant bits of the result
- MLA Rd, Rm, Rs, Rn: Multiply and accumulate Rd = (Rn + (Rm * Rs)) Returns the 32 least significant bits of the result
- MLS Rd, Rm, Rs, Rn: Multiply and subtract Rd = (Rn - (Rm * Rs)) Returns the 32 least significant bits of the result
- UMULL RdLo, RdHi, Rm, Rs: Multiply unsigned long, 64 bit result
- SMULL RdLo, RdHi, Rm, Rs: Multiply signed long, 64 bit result
- SDIV Rd, Rn, Rm: Signed division Rd = Rn/Rm (0x80000000 / 0xFFFFFFFF = 0x80000000, Rn / 0 = 0)
- UDIV Rd, Rn, Rm: Unsigned division Rd = Rn/Rm (Rn / 0 = 0)
- ASR{S} Rd, Rm, <Rs|#imm5>: Arithmetic shift right, canonical form of MOV{S} Rd, Rm, ASR <Rs|#imm5>
- LSL{S} Rd, Rm, <Rs|#imm5>: Logical shift left
- LSR{S} Rd, Rm, <Rs|#imm5>: Logical shift right
- ROR{S} Rd, Rm: Rotate right
- RRX{S} Rd, Rm: Rotate right with extent, uses Carry as a 33rd bit

- CMP Rn, <Operand2>: Same as SUBS Rd, Rn, <Operand2>, but result not written to Rd, only the condition flags are updated
- CMN Rn, <Operand2>: Rn + <Operand2>
- TST Rn, <Operand2>: Rn AND <Operand2>
- TEQ Rn, <Operand2>: Rn EOR <Operand2>

**AND{S} Rd, Rn, <Operand2>:** Bitwise AND, Rd = Rn AND <Operand2>
- ORR{S} Rd, Rn, <Operand2>: Bitwise OR, Rd = Rn OR <Operand2>
- EOR{S} Rd, Rn, <Operand2>: Bitwise Exclusive-OR. Rd = Rn EOR <Operand2>
- ORN{S} Rd, Rn, <Operand2>: Or not, Rd = Rn OR NOT <Operand2>
- BIC{S} Rd, Rn, <Operand2>: Bit clear, Rd = Rn AND NOT <Operand2>

**B <label>:** Unconditional jump
- BL <label>: R14 = address of next instruction, then jump to label
- BX Rm: Branch and exchange (jump to address in Rm), use it to return from a function (BX LR)
- BLX Rm: R14 = address of next instruction, then jump to Rm

**Bcc <label>:** Conditional jump, where cc is one of {EQ,NE,GE,GT,LE,LT,HS,HI,LS,LO,VS,VC,CS,CC,MI,PL}
### <Address> can be one of the following

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Rn]</td>
<td>LDR R0,[R1]</td>
<td>R0 = [R1 + 0]</td>
</tr>
<tr>
<td>[Rn {, #&lt;imm8</td>
<td>+imm12&gt;}]</td>
<td>LDR R0, [R1, #8]</td>
</tr>
<tr>
<td>[Rn {, #&lt;+-imm8&gt;}]</td>
<td>LDR R0, [R1, #8]!</td>
<td>R1 = R1 + 8, R0 = [R1]</td>
</tr>
<tr>
<td>[Rn], #&lt;+-imm8&gt;</td>
<td>LDR R0, [R1], #4</td>
<td>R0 = [R1], R1 = R1 + 4</td>
</tr>
<tr>
<td>[Rn, Rm {, &lt;LSL #0-3&gt;}]</td>
<td>STR R0, [R1, R2, LSL #2]</td>
<td>R0 = [R1 + (R2 * 4)]</td>
</tr>
</tbody>
</table>

- **LDR Rd, <Address>**  
  Load 32 bit word from memory
- **LDRH Rd, <Address>**  
  Load 16 bit half-word from memory
- **LDRSH Rd, <Address>**  
  Load signed 16 bit half-word from memory
- **LDRB Rd, <Address>**  
  Load 8 bit byte from memory
- **LDRSB Rd, <Address>**  
  Load signed 8 bit byte from memory
- **STR Rd, <Address>**  
  Store 32 bit word to memory
- **STRH Rd, <Address>**  
  Store 16-bit halfword to memory
- **STRB Rd, <Address>**  
  Store 8-bit byte to memory

- **LDR Rd,=Label**  
  Pseudo-Op: Load Rd with 32-bit address/constant equivalent to Label
- **LDR Rd,=Constant**  
  Pseudo-Op: Load Rd with 32-bit constant

#### PUSH <reglist>

Push registers onto stack pointed to by SP, decrement address before each store; lowest-numbered register to the lowest memory address

#### POP <reglist>

Restore registers from stack, increment address after each load. Be careful with registers SP and PC.

#### LDM{IA|IB|DA|DB} Rn{!}, <reglist>

Load/store multiple, can transfer any list of registers, ! will update Rn to point to the address after/before the last register

#### STM{IA|IB|DA|DB} Rn{!}, <reglist>

IA = increment after (default), IB = increment before, DA = decrement after, DB = decrement before (Action on address)