ARM Assembly Handy Reference

Labels are non-reserved symbols followed by a colon. If on a separate line they apply to the next line with an instruction

Instruction mnemonics begin after column 1, usually UPPER CASE

Registers: PC, LR, SP, R0 - R12.
Parameters are passed in R0 - R3 as needed.

Assembly (.S) file begins with:

```
.syntax unified
.section .text
.global name_of_entry_label
```

File ends with: .end

Comment to end of line: @

To save incoming registers and create working space, push a copy on the stack. Include LR if you will call a function within your function.

```
PUSH    {R0, R1, R2, R3, R4, R5, LR}
```

Before returning, pop the registers you have saved. List them in the same order:

```
POP     {R0, R1, R2, R3, R4, R5, LR}
```

To return:

```
BX      LR
```

Each lower level function should follow the PUSH, POP, BX LR protocol to free up working registers and restore them before returning.

Instruction mnemonic extensions:

Append S to the mnemonic to set flags.
Append condition code to execute conditionally.

```
SUBS    R1, R1, #1  @Decrement R1, set flags
BGT     looptop     @Branch to loop top if >0
```

Constants (imm):

If a small value, just precede with #
Longer (e.g. address) values use =
0x precedes hexadecimal values

```
SUB     R5, R1, #42
LDR     R1, =0x2009C020
```

Commonly Used Instructions:

```
LDR     dest, [base]          @load register
LDR     dest, [base, index]
LDR     dest, [base, index, LSL #imm]
LDR     = imm
STR     src, [base]           @store register
STR     src, [base, index]
STR     src, [base, index, LSL #imm]
ADD     dest, src1, src2    @dest = src1+src2
ADD     dest, src1, #imm
SUB     dest, src1, src2    @dest = src1 - src2
SUB     dest, src1, #imm
RSB     dest, src1, src2    @dest = src2 - src1
RSB     dest, src1, #imm
MUL     dest, src1, src2    @low order 32 bits
SDIV    dest, src1, src2    @dest = src1/src2
CMP     src1, src2          @set flags for src1-src2
CMP     src1, #imm          @set flags for src1-#imm
LSL     dest, src, #imm     @logical shift left
LSL     reg, #imm
LSR     dest, src, #imm     @logical shift right
LSR     reg, #imm
ASR     dest, src, #imm     @arithmetic shift right
ASR     reg, #imm
ROR     reg, #imm           @rotate right
AND     reg, #imm           @bitwise AND
AND     dest, src, #imm
ORR     reg, #imm           @bitwise OR
ORR     dest, src, #imm
EOR     reg, #imm           @bitwise exclusive OR
EOR     dest, src, #imm
MOV     dest, src           @move reg to reg
MVN     dest, src           @bitwise NOT
B       label               @unconditional branch
BL      label               @subroutine call
BX      reg                 @branch indirect
```

conditions: EQ, NE, GT, LT, GE, LE, PL, MI

IF block has a condition followed by up to four instructions. The block opens with I followed by a pattern of Ts and Es indicating the IF/ELSE cases in relation to the condition. Each instruction must have the corresponding condition. Condition flags must be set previously. Example:

```
CMP     R1, R2          @set flags
ITTEE   GT              @IF block begin
ADDGT   R3, R4, R5      @then
SUBGT   R6, R7, R8      @then
ADDLE   R6, R7, R8      @else
SUBLE   R3, R4, R5      @else
```