ARM Instruction Set Architecture

Subset for Fundamentals of Computer Science students. The following subset is based on GNU as Assembler, and includes pseudo-operations.

Instruction Format (syntax)	Instruction effects (semantics)
`ADD Rd Rn Rm/#imm`	Rd = Rn + Rm/#imm
`SUB Rd Rn Rm/#imm`	Rd = Rn - Rm/#imm
`MUL Rd Rn Rm`	Rd = Rn * Rm/#imm capped at 32 bits!
`AND Rd Rn Rm/#imm`	Rd = Rn & Rm/#imm
`EOR Rd Rn Rm/#imm`	Rd = Rn ^ Rm/#imm
`ORR Rd Rn Rm/#imm`	Rd = Rn | Rm/#imm
`MVN Rd Rm/#imm`	Rd = ~ Rm/#imm
`CMP Rn Rm/#imm`	Status ← comparison (<, >, ==) of Rn and Rm/#imm
`MOV Rd Rm/#imm`	Rd = Rm/#imm
`B` label	jump to label (unconditional)
`BEQ` label	jump to label if previously compared values were equal
`BGT` label	jump to label if previously compared Rn > Rn/#imm
`BLT` label	jump to label if previously compared Rn < Rn/#imm
`BL` label	function call (label is the function name/entry point)
`BX Rd`	return from function (always as BX lr)
`LDR Rd [Rn, Rm/#imm]`	Rd = mem[Rn+Rm/#imm] (32 bit copy)
`STR Rd [Rn, Rm/#imm]`	mem[Rn+Rm/#imm] = Rd (32 bit copy)
`LDRB Rd [Rn, Rm/#imm]`	Rd = mem[Rn+Rm/#imm] (8 bit copy)
`STRB Rd [Rn, Rm/#imm]`	mem[Rn+Rm/#imm] = Rd (8 bit copy)
`PUSH {` register list `}`	Push registers onto stack
`POP {` register list `}`	Pop registers from stack

Directive Format (syntax)	Directive effects (semantics)
`.arm`	Specify ARM assembly
`.align` number	Specify alignment to <number> bytes; defaults to 4
`.data`	Begin data section
`.end`	End of assembly program
`.global` name	Declare globally visible identifier
`.text`	Begin code section
`.type` name, `%function`	Specify that identifier <name> is a function name (you also need to declare a label with that name!)
`label:`	Specify a position which can be referred to in jump instructions
`.int` number	Define a 32-bit integer constant
`.float` number	Define a floating point constant
`.string "string of ASCII chars"`	Define a character string constant