Opcode	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC Rd, Rm	0	1	0	0	0	0	0	1	0	1	Rm			Rd		
ADD Rd, #	0	0	1	1	0	Rd			#							
ADD Rd, [PC, #]	1	0	1	0	0	Rd			#							
ADD Rd, [SP, #]	1	0	1	0	1	Rd			#							
ADD Rd, Rm	0	1	0	0	0	1	0	0	H1	H2	Rm			Rd		
ADD Rd, Rn, #	0	0	0	1	1	1	0	#			Rn			Rd		
ADD Rd, Rn, Rm	0	0	0	1	1	0	0	Rm			Rn			Rd		
AND Rd, Rm	0	1	0	0	0	0	0	0	0	0	Rm			Rd		
ASR Rd, Rm, #	0	0	0	1	0	#					Rm			Rd		
ASR Rd, Rs	0	1	0	0	0	0	0	1	0	0	Rs			Rd		
B <Target Addr>	1	1	1	0	0	offset										
B{<cond>} offset	1	1	0	1	cond				offset							
BIC Rd, Rm	0	1	0	0	0	0	1	1	1	0	Rm			Rd		
BL <Target Addr>	1	1	1	1	1	offset										
BL <Target Addr> (+)	1	1	1	1	0	offset										
BX Rm	0	1	0	0	0	1	1	1	0	H2	Rm			SBZ		
CMN Rn, Rm	0	1	0	0	0	0	1	0	1	1	Rm			Rn		
CMP Rn, #	0	0	1	0	0	Rn			#							
CMP Rn, Rm	0	1	0	0	0	0	1	0	1	0	Rm			Rn		
CMP Rn, Rm	0	1	0	0	0	1	0	1	H1	H2	Rm			Rn		
EOR Rd, Rm	0	1	0	0	0	0	0	0	0	1	Rm			Rd		
LDMIA Rn!, <reg list>	1	1	0	0	1	Rn			Register List							
LDR Rd, [PC, #]	0	1	0	0	1	Rd			PC Relative Offset							
LDR Rd, [PC, #]	1	0	0	1	0	Rd			SP Relative Offset							
LDR Rd, [Rn, #]	0	1	1	0	1	#					Rn			Rd		
LDR Rd, [Rn, Rm]	0	1	0	1	1	0	0	Rm			Rn			Rd		
LDRB Rd, [Rn, #]	0	1	1	1	1	#					Rn			Rd		
LDRB Rd, [Rn, Rm]	0	1	0	1	1	1	0	Rm			Rn			Rd		
LDRH Rd, [Rn, #]	1	0	0	0	1	#					Rn			Rd		
LDRSB Rd, [Rn, Rm]	0	1	0	1	1	1	1	Rm			Rn			Rd		
LDRSH Rd, [Rn, Rm]	0	1	0	1	1	0	1	Rm			Rn			Rd		
LSL Rd, Rm, #	0	0	0	0	0	#					Rm			Rd		
LSL Rd, Rs	0	1	0	0	0	0	0	0	1	0	Rs			Rd		
LSR Rd, Rm, #	0	0	0	0	1	#					Rm			Rd		
LSR Rd, Rs	0	1	0	0	0	0	0	0	1	1	Rs			Rd		
MOV Rd, #	0	0	1	0	1	Rd			#							
MOV Rd, Rm	0	1	0	0	0	1	1	0	H1	H2	Rm			Rd		
MUL Rd, Rm	0	1	0	0	0	0	1	1	0	1	Rm			Rd		
MVN Rd, Rm	0	1	0	0	0	0	1	1	1	1	Rm			Rd		
NEG Rd, Rm	0	1	0	0	0	0	1	0	0	1	Rm			Rd		
ORR Rd, Rm	0	1	0	0	0	0	1	1	0	0	Rm			Rd		
POP {<reg list>, <PC>}	1	0	1	1	1	1	Z	PC	Register List							
PUSH {<reg list>, <LR>}	1	0	1	1	0	1	Z	LR	Register List							
ROR Rd, Rs	0	1	0	0	0	0	0	1	1	1	Rs			Rd		
SBC Rd, Rm	0	1	0	0	0	0	0	1	1	0	Rm			Rd		
STMIA Rn!, <reg list>	1	1	0	0	0	Rn			Register List							
STR Rd, [PC, #]	1	0	0	1	1	Rd			SP Relative Offset							
STR Rd, [Rn, #]	0	1	1	0	0	#					Rn			Rd		
STR Rd, [Rn, Rm]	0	1	0	1	0	0	0	Rm			Rn			Rd		
STRB Rd, [Rn, #]	0	1	1	1	0	#					Rn			Rd		
STRB Rd, [Rn, Rm]	0	1	0	1	0	1	0	Rm			Rn			Rd		
STRH Rd, [Rn, #]	1	0	0	0	0	#					Rn			Rd		
STRSB Rd, [Rn, Rm]	0	1	0	1	0	1	1	Rm			Rn			Rd		
STRSH Rd, [Rn, Rm]	0	1	0	1	0	0	1	Rm			Rn			Rd		
SUB Rd, #	0	0	1	1	1	Rd			#							
SUB Rd, Rn, #	0	0	0	1	1	1	1	#			Rn			Rd		
SUB Rd, Rn, Rm	0	0	0	1	1	0	1	Rm			Rn			Rd		
SUB SP, SP, #	1	0	1	1	Z	0	SBZ		1	#						
SWI #	1	1	0	1	1	1	1	1	#							
TST Rn, Rm	0	1	0	0	0	0	1	0	0	0	Rm			Rn		
Unpredictable	0	1	0	0	0	1	0	0	0	0	x	x	x	x	x	x
Unpredictable	0	1	0	0	0	1	0	1	0	0	x	x	x	x	x	x
Unpredictable	0	1	0	0	0	1	1	0	0	0	x	x	x	x	x	x
Undefined Instruction	0	1	0	0	0	1	1	1	1	x	x	x	x	x	x	x
Undefined Instruction	1	1	1	0	1	x	x	x	x	x	x	x	x	x	x	x

Mnemonic	Description	Variants	Work	Notes	Z	C	N	V
ADC	Add with Carry	ADC Rd, Rm	Rd = Rd + Rm + C	-	x	x	x	x
ADD	Add	ADD Rd, #	Rd = Rd + #	-	x	x	x	x
		ADD Rd, [PC, #]	Rd = (PC & 0xFFFFFFFC) + #	-				
		ADD Rd, [SP, #]	Rd = SP + #	-				
		ADD Rd, Rm	Rd = Rd + Rm	Rd or Rm must be High Registers				
		ADD Rd, Rn, #	Rd = Rn + #	-	x	x	x	x
		ADD Rd, Rn, Rm	Rd = Rn + Rm	Use this for Low Reg Addition	x	x	x	x
		ADD SP, SP, #	SP = SP + #	-				
AND	Logical And	AND Rd, Rm	Rd = Rd & Rm	-	x	x	x	x
ASR	Arithmetic Shift Right	ASR Rd, Rm, #	Rd = Rm >> #	Signed	x	x	x	
		ASR Rd, Rs	Rd = Rd >> Rs	Signed		x		
B	Branch	B <Target Addr>	PC = PC + (Offset << 1)	-				
		B{<cond>} <Target Addr>	*if <cond> then* PC = PC +(Offset << 1)	-				
BIC	Bit Clear	BIC Rd, Rm	Rd = Rd & !(Rm)	-	x	x	x	
BL	Branch with Link	BL <Target Addr>	LR = (PC + 2); PC = PC + (Offset << 1)	Short Version				
		BL <Target Addr>	LR = (PC + 4); PC = PC + ((Offset<<12) | (Next_HalfWord))	Long Version				
BX	Branch and Exchange	BX Rm	PC = Rm[31..1] << 1; Mode = ARM	Changes Instruction set to ARM. Rm Can be High Reg				
CMN	Compare Negative	CMN Rn, Rm	<Flags> = Rn + Rm	-	x	x	x	x
CMP	Compare	CMP Rn, #	<Flags> = Rn - #	-	x	x	x	x
		CMP Rn, Rm	<Flags> = Rn - Rm	-	x	x	x	x
		CMP Rn, Rm	<Flags> = Rn - Rm	Rn or Rm must be a High Reg	x	x	x	x
EOR	Logical Exclusive Or (XOR)	EOR Rd, Rm	Rd = Rd ^ Rm	-	x	x	x	x
LDM	Load Multiple	LDMIA Rn!, <reg list>	for each in <reglist> = [Rn+=4]	-				
LDR	Load Register (word)	LDR Rd, [PC, #]	Rd = [PC + (#<<2)]	-				
		LDR Rd, [SP, #]	Rd = [SP + (#<<2)]	-				
		LDR Rd, [Rn, #]	Rd = [Rn + (#<<2)]	-				
		LDR Rd, [Rn, Rm]	Rd = [Rn + Rm]	-				
LDRB	Load Register (unsigned byte)	LDRB Rd, [Rn, #]	Rd = [Rn + #]	-				
		LDRB Rd, [Rn, Rm]	Rd = [Rn + Rm]	-				
LDRH	Load Register (unsigned halfword)	LDRH Rd, [Rn, #]	Rd = [Rn + #]	-				
		LDRH Rd, [Rn, Rm]	Rd = [Rn + Rm]	-				
LDRSB	Load Register (signed byte)	LDRSB Rd, [Rn, Rm]	Rd = [Rn + Rm]	-				
LDRSH	Load Register (signed halfword)	LDRSH Rd, [Rn, Rm]	Rd = [Rn + Rm]	-				
LSL	Logical Shift Left	LSL Rd, Rm, #	Rd = Rm << #	-	x	x	x	
		LSL Rd, Rs	Rd = Rd << Rs	-	x	x	x	
LSR	Logical Shift Right	LSR Rd, Rm, #	Rd = Rm >> #	-	x	x	x	
		LSR Rd, Rs	Rd = Rd >> Rs	-	x	x	x	

Meaning	Mnemonic	Opcode	Status Flags
Equal	EQ	0 0 0 0	Z = 1
Not Equal	NE	0 0 0 1	Z = 0
Carry Set	CS	0 0 1 0	C = 1
Carry Clear	CC	0 0 1 1	C = 0
Unsigned Higher or Same	HS	0 0 1 0	C = 1
Unsigned Lower	LO	0 0 1 1	C = 0
Minus/Negative	MI	0 1 0 0	N = 1
Plus/Positive or Zero	PL	0 1 0 1	N = 0
Overflow	VS	0 1 1 0	V = 1
No Overflow	VC	0 1 1 1	V = 0
Unsigned Higher	HI	1 0 0 0	C = 1, Z = 0
Unsigned Lower or Same	LS	1 0 0 1	C = 0, Z = 1
Signed Greater than or Equal	GE	1 0 1 0	N = V
Signed Less than	LT	1 0 1 1	N != V
Signed Greater than	GT	1 1 0 0	Z = 0, N = V
Signed Less than or Equal	LE	1 1 0 1	Z = 1, N != V
Always	AL	1 1 1 0	-
Never	NE	1 1 1 1	-

rE Ejected

ARM Thumb Reference

Opcode	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LSL Rd, Rm, #	0	0	0	0	0	#					Rm			Rd		
LSR Rd, Rm, #	0	0	0	0	1	#					Rm			Rd		
ASR Rd, Rm, #	0	0	0	1	0	#					Rm			Rd		
ADD Rd, Rn, Rm	0	0	0	1	1	0	0	Rm			Rn			Rd		
SUB Rd, Rn, Rm	0	0	0	1	1	0	1	Rm			Rn			Rd		
ADD Rd, Rn, #	0	0	0	1	1	1	0	#			Rn			Rd		
SUB Rd, Rn, #	0	0	0	1	1	1	1	#			Rn			Rd		
CMP Rn, #	0	0	1	0	0	Rn			#							
MOV Rd, #	0	0	1	0	1	Rd			#							
ADD Rd, #	0	0	1	1	0	Rd			#							
SUB Rd, #	0	0	1	1	1	Rd			#							
AND Rd, Rm	0	1	0	0	0	0	0	0	0	0	Rm			Rd		
EOR Rd, Rm	0	1	0	0	0	0	0	0	0	1	Rm			Rd		
LSL Rd, Rs	0	1	0	0	0	0	0	0	1	0	Rs			Rd		
LSR Rd, Rs	0	1	0	0	0	0	0	0	1	1	Rs			Rd		
ASR Rd, Rs	0	1	0	0	0	0	0	1	0	0	Rs			Rd		
ADC Rd, Rm	0	1	0	0	0	0	0	1	0	1	Rm			Rd		
SBC Rd, Rm	0	1	0	0	0	0	0	1	1	0	Rm			Rd		
ROR Rd, Rs	0	1	0	0	0	0	0	1	1	1	Rs			Rd		
TST Rn, Rm	0	1	0	0	0	0	1	0	0	0	Rm			Rn		
NEG Rd, Rm	0	1	0	0	0	0	1	0	0	1	Rm			Rd		
CMP Rn, Rm	0	1	0	0	0	0	1	0	1	0	Rm			Rn		
CMN Rn, Rm	0	1	0	0	0	0	1	0	1	1	Rm			Rn		
ORR Rd, Rm	0	1	0	0	0	0	1	1	0	0	Rm			Rd		
MUL Rd, Rm	0	1	0	0	0	0	1	1	0	1	Rm			Rd		
BIC Rd, Rm	0	1	0	0	0	0	1	1	1	0	Rm			Rd		
MVN Rd, Rm	0	1	0	0	0	0	1	1	1	1	Rm			Rd		
ADD Rd, Rm	0	1	0	0	0	1	0	0	H1	H2	Rm			Rd		
Unpredictable	0	1	0	0	0	1	0	0	0	0	x	x	x	x	x	x
CMP Rn, Rm	0	1	0	0	0	1	0	1	H1	H2	Rm			Rn		
Unpredictable	0	1	0	0	0	1	0	1	0	0	x	x	x	x	x	x
MOV Rd, Rm	0	1	0	0	0	1	1	0	H1	H2	Rm			Rd		
Unpredictable	0	1	0	0	0	1	1	1	0	0	x	x	x	x	x	x
BX Rm	0	1	0	0	0	1	1	1	0	H2	Rm			SBZ		
Undefined Instruction	0	1	0	0	0	1	1	1	1	x	x	x	x	x	x	x
LDR Rd, [PC, #]	0	1	0	0	1	Rd			PC Relative Offset							
STR Rd, [Rn, Rm]	0	1	0	1	0	0	0	Rm			Rn			Rd		
STRSH Rd, [Rn, Rm]	0	1	0	1	0	0	1	Rm			Rn			Rd		
STRB Rd, [Rn, Rm]	0	1	0	1	0	1	0	Rm			Rn			Rd		
STRSB Rd, [Rn, Rm]	0	1	0	1	0	1	1	Rm			Rn			Rd		
LDR Rd, [Rn, Rm]	0	1	0	1	1	0	0	Rm			Rn			Rd		
LDRSH Rd, [Rn, Rm]	0	1	0	1	1	0	1	Rm			Rn			Rd		
LDRB Rd, [Rn, Rm]	0	1	0	1	1	1	0	Rm			Rn			Rd		
LDRSB Rd, [Rn, Rm]	0	1	0	1	1	1	1	Rm			Rn			Rd		
STR Rd, [Rn, #]	0	1	1	0	0	#					Rn			Rd		
LDR Rd, [Rn, #]	0	1	1	0	1	#					Rn			Rd		
STRB Rd, [Rn, #]	0	1	1	1	0	#					Rn			Rd		
LDRB Rd, [Rn, #]	0	1	1	1	1	#					Rn			Rd		
STRH Rd, [Rn, #]	1	0	0	0	0	#					Rn			Rd		
LDRH Rd, [Rn, #]	1	0	0	0	1	#					Rn			Rd		
LDR Rd, [PC, #]	1	0	0	1	0	Rd			SP Relative Offset							
STR Rd, [PC, #]	1	0	0	1	1	Rd			SP Relative Offset							
ADD Rd, [PC, #]	1	0	1	0	0	Rd			#							
ADD Rd, [SP, #]	1	0	1	0	1	Rd			#							
SUB SP, SP, #	1	0	1	1	Z	0	SBZ		1	#						
PUSH {<reg list>, <LR>}	1	0	1	1	0	1	Z	LR	Register List							
POP {<reg list>, <PC>}	1	0	1	1	1	1	Z	PC	Register List							
STMIA Rn!, <reg list>	1	1	0	0	0	Rn			Register List							
LDMIA Rn!, <reg list>	1	1	0	0	1	Rn			Register List							
B{<cond>} offset	1	1	0	1	cond				offset							
SWI #	1	1	0	1	1	1	1	1	#							
B <Target Addr>	1	1	1	0	0	offset										
Undefined Instruction	1	1	1	0	1	x	x	x	x	x	x	x	x	x	x	x
BL <Target Addr> (+)	1	1	1	1	0	offset										
BL <Target Addr>	1	1	1	1	1	offset										

Symbol	Meaning
Rd	Destination Register
Rn	Register
Rm	Register
Rs	Register (shift ammount)
#	Immediate Value (a number)
C	Carry Bit
PC	Program Counter
SP	Stack Pointer
LR	Link Register
<flags>	Result affects only flags

Symbol	Meaning
=	equals
+	plus/add
-	minus/subtract/negate
*	multiply/multiplication
<<	left shift (similar to multiply)
>>	right shift (similar to a divide)
&	Boolean Operator AND
|	Boolean Operator OR
^	Boolean Operator XOR
!	Not Operator/Update Register

Flag	Description
Z	**Z**ero Flag
C	**C**arry Flag
N	**N**egative Flag
V	Overflow Flag

Mnemonic	Description	Variants	Work	Notes	Z	C	N	V
MOV	Move	MOV Rd, #	Rd = #	-	x		x	
		MOV Rd, Rm	Rd = Rm	Rd or Rm must be High Registers				
MUL	Multiply	MUL Rd, Rm	Rd = Rd * Rm	-	x	x	x	
MVN	Move Negative	MVN Rd, Rm	Rd = ! (Rm)	Ones Compliment	x		x	
NEG	Negate	NEG Rd, Rm	Rd = -(Rm)	Twos Compliment	x	x	x	x
ORR	Logical OR	ORR Rd, Rm	Rd = Rd | Rm	-	x	x	x	
POP	Pop Multiple Registers	POP {<reg list>, <PC>}	get <reg list> and/or <PC> from stack	-				
PUSH	Push Multiple Registers	PUSH {<reg list>, <LR>}	put <reg list> and/or <LR> on stack	-				
ROR	Rotate Right Register	ROR Rd, Rm	Rd = Rd Rotate_Right Rm	-		x		
SBC	Subtract with Carry	SBC Rd, Rm	Rd = Rd - Rm + C	-	x	x	x	x
STM	Store Multiple	STMIA Rn!, <reg list>	[Rn+=4] = for each in <reglist>	-				
STR	Store Register (word)	STR Rd, [SP, #]	[SP + (#<<2)] = Rd	-				
		STR Rd, [Rn, #]	[Rn + (#<<2)] = Rd	-				
		STR Rd, [Rn, Rm]	[Rn + Rm] = Rd	-				
STRB	Store Register (unsigned byte)	STRB Rd, [Rn, #]	[Rn + (#<<2)] = Rd	-				
		STRB Rd, [Rn, Rm]	[Rn + Rm] = Rd	-				
STRH	Store Register (unsigned halfword)	STRH Rd, [Rn, #]	[Rn + (#<<2)] = Rd	-				
		STRH Rd, [Rn, Rm]	[Rn + Rm] = Rd	-				
SUB	Subtract	SUB Rd, #	Rd = Rd - #	-	x	x	x	x
		SUB Rd, Rn, #	Rd = Rn - #	-	x	x	x	x
		SUB Rd, Rn, Rm	Rd = Rn - Rm	-	x	x	x	x
		SUB SP, SP, #	SP = SP - #	-				
SWI	Software Interrupt	SWI #	Run BIOS Function	-				
TST	Test	TST Rn, Rm	<Flags> = Rn & Rm	-	x		x	