

Thumb code quick reference

	Syntax	Action	Flags	Notes
Move				
Immediate	<code>movs rd, #imm</code>	$Rd := imm$	NZ	Range 0-255
Reg to reg	<code>movs rd, rm</code>	$Rd := Rm$	NZ	Synonym for <code>lsls rd, rm, #0</code>
High regs	<code>mov rd, rm</code>	$Rd := Rm$		Regs R8-R12, <i>sp</i> , <i>lr</i> , <i>pc</i> allowed
Add				
Register	<code>adds rd, rn, rm</code>	$Rd := Rn + Rm$	NZCV	
Immediate	<code>adds rd, rn, #imm</code>	$Rd := Rn + imm$	NZCV	Range 0-7 or 0-255 if $Rn \equiv Rd$
Value to <i>sp</i>	<code>add sp, sp, #imm</code>	$sp := sp + imm$		Range 0-508 (word aligned)
Form addr from <i>sp</i>	<code>add rd, sp, #imm</code>	$Rd := sp + imm$		Range 0-1020 (word aligned)
Form addr from <i>pc</i>	<code>adr rd, label</code>	$Rd := label$		Range <i>pc</i> to <i>pc</i> + 1020
Subtract				
Register	<code>subs rd, rn, rm</code>	$Rd := Rn - Rm$	NZCV	
Immediate	<code>subs rd, rn, #imm</code>	$Rd := Rn - imm$	NZCV	Range 0-7 or 0-255 if $Rn \equiv Rd$
Value from <i>sp</i>	<code>subs sp, sp, #imm</code>	$sp := sp - imm$		Range 0-508 (word aligned)
Negate	<code>negs rd, rm</code>	$Rd := -Rm$	NZCV	
Multiply				
Register	<code>muls rd, rn, rd</code>	$Rd := Rn * Rm$	NZ	
Compare				
Register	<code>cmp rn, rm</code>	$Rn - Rm$	NZCV	Updates flags from result
Immediate	<code>cmp rn, #imm</code>	$Rn - imm$	NZCV	Range 0-255
Bitwise				
AND	<code>ands rd, rd, rm</code>	$Rd := Rd \text{ AND } Rm$	NZ	
Exclusive OR	<code>eors rd, rd, rm</code>	$Rd := Rd \text{ XOR } Rm$	NZ	
OR	<code>orrs rd, rd, rm</code>	$Rd := Rd \text{ OR } Rm$	NZ	
Bit clear	<code>bics rd, rd, rm</code>	$Rd := Rd \text{ AND NOT } Rm$	NZ	
Move NOT	<code>mvns rd, rm</code>	$Rd := \text{NOT } Rm$	NZ	
Test bits	<code>tst rn, rm</code>	$Rn \text{ AND } Rm$	NZ	Updates flags from result
Shift				
Logical shift left	<code>lsls rd, rm, #imm</code>	$Rd := Rm \ll imm$	NZC	} C flag set to last bit shifted out, or unchanged if shift is zero
	<code>lsls rd, rd, rm</code>	$Rd := Rd \ll Rm$	NZC	
Logical shift right	<code>lsrs rd, rm, #imm</code>	$Rd := Rm \gg imm$	NZC	
	<code>lsrs rd, rd, rm</code>	$Rd := Rd \gg Rm$	NZC	
Arith shift right	<code>asrs rd, rm, #imm</code>	$Rd := Rm \text{ ASR } imm$	NZC	
	<code>asrs rd, rd, rm</code>	$Rd := Rd \text{ ASR } Rm$	NZC	
Rotate right	<code>rors rd, rd, rm</code>	$Rd := Rd \text{ ROR } Rm$	NZC	
Load				
Word, imm offset	<code>ldr rt, [rn, #imm]</code>	$Rt := \text{Mem}_4[Rn+imm]$		Range 0-124, mult of 4
Word, reg offset	<code>ldr rt, [rn, rm]</code>	$Rt := \text{Mem}_4[Rn+Rm]$		
Halfword, immed	<code>ldrh rt, [rn, #imm]</code>	$Rt := \text{Mem}_2[Rn+imm]$		Range 0-62, mult of 2
Halfword, register	<code>ldrh rt, [rn, rm]</code>	$Rt := \text{Mem}_2[Rn+Rm]$		
Signed halfword	<code>ldrsh rt, [rn, rm]</code>	$Rt := \text{sext}(\text{Mem}_2[Rn+Rm])$		
Byte, imm offset	<code>ldrb rt, [rn, #imm]</code>	$Rt := \text{Mem}_1[Rn+imm]$		Range 0-31

Thumb code quick reference (*continued*)

	Syntax	Action	Notes
Load (<i>continued</i>)			
Byte, reg offset	<code>ldrb <i>rt</i>, [<i>rn</i>, <i>rm</i>]</code>	$Rt := Mem_1[Rn+Rm]$	
Signed byte	<code>ldrsh <i>rt</i>, [<i>rn</i>, <i>rm</i>]</code>	$Rt := sext(Mem_1[Rn+Rm])$	
PC-relative	<code>ldr <i>rt</i>, <i>label</i></code>	$Rt := Mem_4[label]$	Range <i>pc</i> to <i>pc</i> + 1020
SP-relative	<code>ldr <i>rt</i>, [<i>sp</i>, #<i>imm</i>]</code>	$Rt := Mem_4[sp+imm]$	Range 0-1020, mult of 4
Store			
Word, imm offset	<code>str <i>rt</i>, [<i>rn</i>, #<i>imm</i>]</code>	$Mem_4[Rn+imm] := Rt$	Range 0-124, mult of 4
Word, reg offset	<code>str <i>rt</i>, [<i>rn</i>, <i>rm</i>]</code>	$Mem_4[Rn+Rm] := Rt$	
Halfword, immed	<code>strh <i>rt</i>, [<i>rn</i>, #<i>imm</i>]</code>	$Mem_2[Rn+imm] := Rt[15:0]$	
Halfword, register	<code>strh <i>rt</i>, [<i>rn</i>, <i>rm</i>]</code>	$Mem_2[Rn+Rm] := Rt[15:0]$	
Byte, imm offset	<code>strb <i>rt</i>, [<i>rn</i>, #<i>imm</i>]</code>	$Mem_1[Rn+imm] := Rt[7:0]$	Range 0-31
Byte, reg offset	<code>strb <i>rt</i>, [<i>rn</i>, <i>rm</i>]</code>	$Mem_1[Rn+Rm] := Rt[7:0]$	
SP-relative	<code>str <i>rt</i>, [<i>sp</i>, #<i>imm</i>]</code>	$Mem_4[sp+imm] := Rt$	Range 0-1020, mult of 4
Push and pop			
Push	<code>push {<i>regset</i>}</code>		} Subset of R0-R7
Push with link	<code>push {<i>regset</i>, <i>lr</i>}</code>		
Pop	<code>pop {<i>regset</i>}</code>		
Pop and return	<code>pop {<i>regset</i>, <i>pc</i>}</code>		
Branch			
—if equal	<code>beq <i>label</i></code>	$pc := label$ —if Z	Range –252 to +258 bytes
—if not equal	<code>bne <i>label</i></code>	—if !Z	
—if higher or same	<code>bhs <i>label</i></code>	—if C	Synonym for bcs
—if lower	<code>blo <i>label</i></code>	—if !C	Synonym for bcc
—if minus	<code>bmi <i>label</i></code>	—if N	
—if plus	<code>bpl <i>label</i></code>	—if !N	
—if overflow	<code>bvs <i>label</i></code>	—if V	
—if not overflow	<code>bvc <i>label</i></code>	—if !V	
—if higher	<code>bhi <i>label</i></code>	—if C && !Z	
—if lower or same	<code>bls <i>label</i></code>	—if !C Z	
—if greater or eq	<code>bge <i>label</i></code>	—if N == V	
—if less than	<code>blt <i>label</i></code>	—if N != V	
—if greater than	<code>bgt <i>label</i></code>	—if !Z && N == V	
—if less or eq	<code>ble <i>label</i></code>	—if Z N != V	
Unconditional	<code>b <i>label</i></code>	$pc := label$	Range ±2 KB
Branch with link	<code>bl <i>label</i></code>	$lr := next; pc := label$	
Branch to reg	<code>bx <i>rm</i></code>	$pc := Rm$	} High regs allowed
Branch reg & link	<code>blx <i>rm</i></code>	$lr := next; pc := Rm$	
No operation	<code>nop</code>		
Extend			
Signed byte	<code>sxtb <i>rd</i>, <i>rm</i></code>	$Rd := sext(Rm[7:0])$	
Unsigned byte	<code>uxtb <i>rd</i>, <i>rm</i></code>	$Rd := Rm[7:0]$	
Signed halfword	<code>sxth <i>rd</i>, <i>rm</i></code>	$Rd := sext(Rm[15:0])$	
Unsigned halfword	<code>uxth <i>rd</i>, <i>rm</i></code>	$Rd := Rm[15:0]$	