

For each instruction the information contains columns for "Instruction Syntax"|"ARM Standard"|"Description"|"Operation"

ARM standard is (v7 supports v6, and v7r supports v7 and v6)

v6 => e.g. ARM 11 (e.g. Raspberry Pi)

v7 => e.g. Cortex A8/A9 (e.g. BeagleBoard, PandaBoard)

v7r => e.g. Cortex A15 (e.g. ISSE IgepV5)

For more detailed informations get the latest ARM Architecture Reference Manual from www.arm.com

For a complete instruction set overview get the latest ARM and Thumb-2 Instruction Set Quick Reference Card from www.arm.com

I recommend to print this list at landscape orientation and zoom to fit full lines.

Parallel Arithmetic

<p>ADD16 Rd, Rn, Rm	v6	Parallel Halfword-wise addition	Rd[31:16] := Rn[31:16] + Rm[31:16], Rd[15: 0] := Rn[15: 0] + Rm[15: 0]
<p>SUB16 Rd, Rn, Rm	v6	Halfword-wise subtraction	Rd[31:16] := Rn[31:16] - Rm[31:16], Rd[15: 0] := Rn[15: 0] - Rm[15: 0]
<p>ADD8 Rd, Rn, Rm	v6	Byte-wise addition	Rd[31:24] := Rn[31:24] + Rm[31:24], Rd[23:16] := Rn[23:16] + Rm[23:16], Rd[15: 8] := Rn[15: 8] + Rm[15: 8], Rd[7: 0] := Rn[7: 0] + Rm[7: 0]
<p>SUB8 Rd, Rn, Rm	v6	Byte-wise subtraction	Rd[31:24] := Rn[31:24] - Rm[31:24], Rd[23:16] := Rn[23:16] - Rm[23:16], Rd[15: 8] := Rn[15: 8] - Rm[15: 8], Rd[7: 0] := Rn[7: 0] - Rm[7: 0]
<p>ASX Rd, Rn, Rm	v6	Halfword-wise exchange, add, subtract	Rd[31:16] := Rn[31:16] + Rm[15: 0], Rd[15: 0] := Rn[15: 0] - Rm[31:16]
<p>SAX Rd, Rn, Rm	v6	Halfword-wise exchange, subtract, add	Rd[31:16] := Rn[31:16] - Rm[15: 0], Rd[15: 0] := Rn[15: 0] + Rm[31:16]
USAD8 Rd, Rm, Rs	v6	Unsigned sum of absolute differences	Rd := Abs(Rm[31:24] - Rs[31:24]) + Abs(Rm[23:16] - Rs[23:16]) + Abs(Rm[15: 8] - Rs[15: 8]) + Abs(Rm[7: 0] - Rs[7: 0])
USADA8 Rd, Rm, Rs, Rn	v6	Unsigned sum of absolute differences and accumulate	Rd := Rn + Abs(Rm[31:24] - Rs[31:24]) + Abs(Rm[23:16] - Rs[23:16]) + Abs(Rm[15: 8] - Rs[15: 8]) + Abs(Rm[7: 0] - Rs[7: 0])

Six prefixes <p> are available for the Parallel Arithmetic Instructions:

S Signed arithmetic modulo 28 or 216, sets CPSR GE bits	U Unsigned arithmetic modulo 28 or 216, sets CPSR GE bits
Q Signed saturating arithmetic	UQ Unsigned saturating arithmetic
SH Signed arithmetic, halving results	UH Unsigned arithmetic, halving results

Saturate

SSAT Rd, #<sat>, Rm{,ASR <sh>}	v6	Signed saturate word, right shift	Rd := SignedSat((Rm ASR sh), sat). <sat> range 1-32, <sh> range 1-31
SSAT Rd, #<sat>, Rm{,LSL <sh>}	v6	Signed saturate word, left shift	Rd := SignedSat((Rm LSL sh), sat). <sat> range 1-32, <sh> range 0-31
SSAT16 Rd, #<sat>, Rm	v6	Signed saturate two halfwords	Rd[31:16] := SignedSat(Rm[31:16], sat), Rd[15:0] := SignedSat(Rm[15:0], sat). <sat> range 1-16
USAT Rd, #<sat>, Rm{,ASR <sh>}	v6	Unsigned saturate word, right shift	Rd := UnsignedSat((Rm ASR sh), sat). <sat> range 0-31, <sh> range 1-31
USAT Rd, #<sat>, Rm{,LSL <sh>}	v6	Unsigned saturate word, left shift	Rd := UnsignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 0-31
USAT16 Rd, #<sat>, Rm	v6	Unsigned saturate two halfwords	Rd[31:16] := UnsignedSat(Rm[31:16], sat), Rd[15:0] := UnsignedSat(Rm[15:0], sat). <sat> range 0-15

Multiply

MLS	Rd, Rm, Rs, Rn	v7	Multiply and subtract	Rd := (Rn - (Rm * Rs))[31:0]
UMAAL	RdLo, RdHi, Rm, Rs	v6	Multiply unsigned double accumulate long	RdHi, RdLo := unsigned(RdHi + RdLo + Rm * Rs)
SMUAD{X}	Rd, Rm, Rs	v6	Dual signed multiply, add	Rd := Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]
SMLAD{X}	Rd, Rm, Rs, Rn	v6	Dual signed multiply, add and accumulate	Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]
SMLALD{X}	RdLo, RdHi, Rm, Rs	v6	Dual signed multiply, add and accumulate long	RdHi, RdLo := RdHi, RdLo + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]
SMUSD{X}	Rd, Rm, Rs	v6	Dual signed multiply, subtract	Rd := Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]
SMLSD{X}	Rd, Rm, Rs, Rn	v6	Dual signed multiply, subtract and accumulate	Rd := Rn + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]
SMLSLD{X}	RdLo, RdHi, Rm, Rs	v6	Dual signed multiply, subtract and accumulate long	RdHi, RdLo := RdHi, RdLo + Rm[15:0] * RsX[15:0] - Rm[31:16] * RsX[31:16]
SMMUL{R}	Rd, Rm, Rs	v6	Signed top word multiply	Rd := (Rm * Rs)[63:32]
SMMLA{R}	Rd, Rm, Rs, Rn	v6	Signed top word multiply and accumulate	Rd := Rn + (Rm * Rs)[63:32]
SMMLS{R}	Rd, Rm, Rs, Rn	v6	Signed top word multiply and subtract	Rd := Rn - (Rm * Rs)[63:32]

{X} := IF X is present the multiplications are "bottom x top" and "top x bottom", otherwise they are "bottom x bottom" and "top x top"

{R} := IF R is present the multiplication is rounded, otherwise the multiplication is truncated

Divide

SDIV Rd, Rn, Rm	v7r	Signed divide	Rd := Rn / Rm
UDIV Rd, Rn, Rm	v7r	Unsigned divide	Rd := Rn / Rm

Move

MOVW Rd, #<imm16>	v7	Move wide	Rd[31:16] := imm16, Rd[15: 0] unaffected, imm16 range 0-65535
MOVT Rd, #<imm16>	v7	Move top	Rd[15: 0] := imm16, Rd[31:16] = 0, imm16 range 0-65535

Bit field

BFC Rd, #<lsb>, #<width>	v7	Bit Field Clear	Rd[(width+lsb-1):lsb] := 0, other bits of Rd unaffected
BFI Rd, Rn, #<lsb>, #<width>	v7	Bit Field Insert	Rd[(width+lsb-1):lsb] := Rn[(width-1):0], other bits of Rd unaffected
SBFX Rd, Rn, #<lsb>, #<width>	v7	Signed Bit Field Extract	Rd[(width-1):0] = Rn[(width+lsb-1):lsb], Rd[31:width] = Replicate(Rn[width+lsb-1])
UBFX Rd, Rn, #<lsb>, #<width>	v7	Unsigned Bit Field Extract	Rd[(width-1):0] = Rn[(width+lsb-1):lsb], Rd[31:width] = Replicate(0)

Pack

PKHBT Rd, Rn, Rm{,LSL #<sh>}	v6	Pack halfword bottom + top	Rd[15: 0] := Rn[15: 0], Rd[31:16] := (Rm LSL sh)[31:16]. sh 0-31
PKHTB Rd, Rn, Rm{,ASR #<sh>}	v6	Pack halfword top + bottom	Rd[31:16] := Rn[31:16], Rd[15: 0] := (Rm ASR sh)[15: 0]. sh 1-32

Signed extend

SXTH Rd, Rm{,ROR #<sh>}	v6	Halfword to word	Rd[31: 0] := SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3
SXTB16 Rd, Rm{,ROR #<sh>}	v6	Two bytes to halfwords	Rd[31:16] := SignExtend((Rm ROR (8 * sh))[23:16]), Rd[15: 0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3
SXTB Rd, Rm{,ROR #<sh>}	v6	Byte to word	Rd[31: 0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3

Unsigned extend

```
-----
UXTH    Rd, Rm{,ROR #<sh>}      v6  Halfword to word
UXTB16  Rd, Rm{,ROR #<sh>}      v6  Two bytes to halfwords

UXTB    Rd, Rm{,ROR #<sh>}      v6  Byte to word
```

Rd[31: 0] := ZeroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3
Rd[31:16] := ZeroExtend((Rm ROR (8 * sh))[23:16]),
Rd[15: 0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3
Rd[31: 0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3

Signed extend with add

```
-----
SXTAH   Rd, Rn, Rm{,ROR #<sh>}  v6  Halfword to word, add
SXTAB16 Rd, Rn, Rm{,ROR #<sh>}  v6  Two bytes to halfwords, add

SXTAB   Rd, Rn, Rm{,ROR #<sh>}  v6  Byte to word, add
```

Rd[31: 0] := Rn[31: 0] + SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3
Rd[31:16] := Rn[31:16] + SignExtend((Rm ROR (8 * sh))[23:16]),
Rd[15: 0] := Rn[15: 0] + SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3
Rd[31: 0] := Rn[31: 0] + SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3

Unsigned extend with add

```
-----
UXTAH   Rd, Rn, Rm{,ROR #<sh>}  v6  Halfword to word, add
UXTAB16 Rd, Rn, Rm{,ROR #<sh>}  v6  Two bytes to halfwords, add

UXTAB   Rd, Rn, Rm{,ROR #<sh>}  v6  Byte to word, add
```

Rd[31: 0] := Rn[31: 0] + ZeroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3
Rd[31:16] := Rn[31:16] + ZeroExtend((Rm ROR (8 * sh))[23:16]),
Rd[15: 0] := Rn[15: 0] + ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3
Rd[31: 0] := Rn[31: 0] + ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3

Reverse

```
-----
RBIT    Rd, Rm      v7  Bits in word
REV     Rd, Rm      v6  Bytes in word
REV16   Rd, Rm      v6  Bytes in both halfwords
REVSH   Rd, Rm      v6  Bytes in low halfword, sign extend
```

For (i = 0; i < 32; i++) : Rd[i] = Rm[31- i]
Rd[31:24] := Rm[7:0], Rd[23:16] := Rm[15:8], Rd[15: 8] := Rm[23:16], Rd[7:0] := Rm[31:24]
Rd[15: 8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:24] := Rm[23:16], Rd[23:16] := Rm[31:24]
Rd[15: 8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:16] := Rm[7] * &FFFF

Select

```
-----
SEL Rd, Rn, Rm v6  Select bytes
```

Rd[7:0] := Rn[7:0] if GE[0] = 1, else Rd[7:0] := Rm[7:0],
Bits[15:8], [23:16], [31:24] selected similarly by GE[1], GE[2], GE[3]