**Appendix 1: Conditional Codes**

|  |  |  |
| --- | --- | --- |
| **Compare** | **Signed** | **Unsigned** |
| == | BEQ | BEQ |
| != | BNE | BNE |
| > | BGT | BHI |
| >= | BGE | BHS |
| < | BLT | BLO |
| <= | BLE | BLS |

**Appendix 2: Basic Instructions (excluding FPU and DSP instructions)**

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Operands** | **Description and Action** |
| ADC, ADCS | {Rd,} Rn, Op2 | Add with Carry, Rd ← Rn + Op2 + Carry, ADCS updates N,Z,C,V |
| ADD, ADDS | {Rd,} Rn, Op2 | Add, Rd ← Rn + Op2, ADDS updates N,Z,C,V |
| ADD, ADDS | {Rd,} Rn, #imm12 | Add Immediate, Rd ← Rn + imm12, ADDS updates N,Z,C,V |
| ADR | Rd, label | Load PC-relative Address, Rd ← <label> |
| AND, ANDS | {Rd,} Rn, Op2 | Logical AND, Rd ← Rn AND Op2, ANDS updates N,Z,C |
| ASR, ASRS | Rd, Rm, <Rs|#n> | Arithmetic Shift Right, Rd ← Rm>>(Rs|n), ASRS updates N,Z,C |
| B | label | Branch, PC ← label |
| BFC | Rd, #lsb, #width | Bit Field Clear, Rd[(width+lsb–1):lsb] ← 0 |
| BFI | Rd, Rn, #lsb, #width | Bit Field Insert, Rd[(width+lsb–1):lsb] ← Rn[(width-1):0] |
| BIC, BICS | {Rd,} Rn, Op2 | Bit Clear, Rd ← Rn AND NOT Op2, BICS updates N,Z,C |
| BKPT | #imm | Breakpoint, prefetch abort or enter debug state |
| BL | label | Branch with Link,  LR ← address of next instruction, PC ← label |
| BLX | Rm | Branch register with link,  LR ← address of next instruction, PC ← Rm[31:1] |
| BX | Rm | Branch register, PC ← Rm |
| CBNZ | Rn, label | Compare and Branch if Non-zero; PC ← label if Rn != 0 |
| CBZ | Rn, label | Compare and Branch if Zero; PC ← label if Rn == 0 |
| CLREX | - | Clear local processor exclusive tag |
| CLZ | Rd, Rm | Count Leading Zeroes, Rd ← number of leading zeroes in Rm |
| CMN | Rn, Op2 | Compare Negative, Update N,Z,C,V flags on Rn + Op2 |
| CMP | Rn, Op2 | Compare, Update N,Z,C,V flags on Rn ̶ Op2 |
| CPSID | i | Disable specified (i) interrupts, optional change mode |
| CPSIE | i | Enable specified (i) interrupts, optional change mode |
| DMB | - | Data Memory Barrier, ensure memory access order |
| DSB | - | Data Synchronization Barrier, ensure completion of access |
| EOR, EORS | {Rd,} Rn, Op2 | Exclusive OR, Rd ← Rn XOR Op2, EORS updates N,Z,C |
| ISB | - | Instruction Synchronization Barrier |
| IT | - | If-Then Condition Block |
| LDM | Rn{!}, reglist | Load Multiple Registers increment after, <reglist> = mem[Rn], Rn increments after each memory access |
| LDMDB, LDMEA | Rn{!}, reglist | Load Multiple Registers Decrement Before, <reglist> = mem[Rn], Rn decrements before each memory access |
| LDMFD, LDMIA | Rn{!}, reglist | <reglist> = mem[Rn], Rn increments after each memory access |
| LDR | Rt, [Rn, #offset] | Load Register with Word, Rt ← mem[Rn + offset] |
| LDRB, LDRBT | Rt, [Rn, #offset] | Load Register with Byte, Rt ← mem[Rn + offset] |
| LDRD | Rt, Rt2, [Rn,#offset] | Load Register with two words,  Rt ← mem[Rn + offset], Rt2 ← mem[Rn + offset + 4] |
| LDREX | Rt, [Rn, #offset] | Load Register Exclusive, Rt ← mem[Rn + offset] |
| LDREXB | Rt, [Rn] | Load Register Exclusive with Byte, Rt ← mem[Rn] |
| LDREXH | Rt, [Rn] | Load Register Exclusive with Halfword, Rt ← mem[Rn] |
| LDRH, LDRHT | Rt, [Rn, #offset] | Load Register with Halfword, Rt ← mem[Rn + offset] |
| LDRSB, LDRSBT | Rt, [Rn, #offset] | Load Register with Signed Byte, Rt ← mem[Rn + offset] |
| LDRSH, LDRSHT | Rt, [Rn, #offset] | Load Register with Signed Halfword, Rt ← mem[Rn + offset] |
| LDRT | Rt, [Rn, #offset] | Load Register with Word, Rt ← mem[Rn + offset] |
| LSL, LSLS | Rd, Rm, <Rs|#n> | Logic Shift Left, Rd ← Rm << Rs|n, LSLS update N,Z,C |
| LSR, LSRS | Rd, Rm, <Rs|#n> | Logic Shift Right, Rd ← Rm >> Rs|n, LSRS update N,Z,C |
| MLA | Rd, Rn, Rm, Ra | Multiply with Accumulate, Rd ← (Ra + (Rn\*Rm))[31:0] |
| MLS | Rd, Rn, Rm, Ra | Multiply with Subtract, Rd ← (Ra – (Rn\*Rm))[31:0] |
| MOV, MOVS | Rd, Op2 | Move, Rd ← Op2, MOVS updates N,Z,C |
| MOVT | Rd, #imm16 | Move Top, Rd[31:16] ← imm16, Rd[15:0] unaffected |
| MOVW, MOVWS | Rd, #imm16 | Move 16-bit Constant, Rd ← imm16, MOVWS updates N,Z,C |
| MRS | Rd, spec\_reg | Move from Special Register, Rd ← spec\_reg |
| MSR | spec\_reg, Rm | Move to Special Register, spec\_reg ← Rm, Updates N,Z,C,V |
| MUL, MULS | {Rd,} Rn, Rm | Multiply, Rd ← (Rn\*Rm)[31:0], MULS updates N,Z |
| MVN, MVNS | Rd, Op2 | Move NOT, Rd ← 0xFFFFFFFF EOR Op2, MVNS updates N,Z,C |
| NOP | - | No Operation |
| ORN, ORNS | {Rd,} Rn, Op2 | Logical OR NOT, Rd ← Rn OR NOT Op2, ORNS updates N,Z,C |
| ORR, ORRS | {Rd,} Rn, Op2 | Logical OR, Rd ← Rn OR Op2, ORRS updates N,Z,C |
| POP | reglist | Canonical form of LDM SP!, <reglist> |
| PUSH | reglist | Canonical form of STMDB SP!, <reglist> |
| RBIT | Rd, Rn | Reverse Bits, for (i = 0; i < 32; i++): Rd[i] = RN[31–i] |
| REV | Rd, Rn | Reverse Byte Order in a Word, Rd[31:24]←Rn[7:0], Rd[23:16]←Rn[15:8], Rd[15:8]←Rn[23:16], Rd[7:0]←Rn[31:24] |
| REV16 | Rd, Rn | Reverse Byte Order in a Halfword, Rd[15:8]←Rn[7:0], Rd[7:0]←Rn[15:8], Rd[31:24]←Rn[23:16], Rd[23:16]←Rn[31:24] |
| REVSH | Rd, Rn | Reverse Byte order in Low Halfword and sign extend, Rd[15:8]←Rn[7:0], Rd[7:0]←Rn[15:8], Rd[31:16]←Rn[7]\*&FFFF |
| ROR, RORS | Rd, Rm, <Rs|#n> | Rotate Right, Rd ← ROR(Rm, Rs|n), RORS updates N,Z,C |
| RRX, RRXS | Rd, Rm | Rotate Right with Extend, Rd ← RRX(Rm), RRXS updates N,Z,C |
| RSB, RSBS | {Rd,} Rn, Op2 | Reverse Subtract, Rd ← Op2 ̶ Rn, RSBS updates N,Z,C,V |
| SBC, SBCS | {Rd,} Rn, Op2 | Subtract with Carry, Rd ← Rn–Op2–NOT(Carry), updates NZCV |
| SBFX | Rd, Rn, #lsb, #width | Signed Bit Field Extract, Rd[(width–1):0] = Rn[(width+lsb–1):lsb], Rd[31:width] = Replicate(Rn[width+lsb–1]) |
| SDIV | {Rd,} Rn, Rm | Signed Divide, Rd ← Rn/Rm |
| SEV | - | Send Event |
| SMLAL | RdLo, RdHi, Rn, Rm | Signed Multiply with Accumulate,  RdHi,RdLo ← signed(RdHi,RdLo + Rn\*Rm) |
| SMULL | RdLo, RdHi, Rn, Rm | Signed Multiply, RdHi,RdLo ← signed(Rn\*Rm) |
| SSAT | Rd, #n, Rm{,shift #s} | Signed Saturate, Rd ← SignedSat((Rm shift s), n). Update Q |
| STM | Rn{!}, reglist | Store Multiple Registers |
| STMDB, STMEA | Rn{!}, reglist | Store Multiple Registers Decrement Before |
| STMFD, STMIA | Rn{!}, reglist | Store Multiple Registers Increment After |
| STR | Rt, [Rn, #offset] | Store Register with Word, mem[Rn+offset] = Rt |
| STRB, STRBT | Rt, [Rn, #offset] | Store Register with Byte, mem[Rn+offset] = Rt |
| STRD | Rt, Rt, [Rn,#offset] | Store Register with two Words,  mem[Rn+offset] = Rt, mem[Rn+offset+4] = Rt2 |
| STREX | Rd, Rt, [Rn,#offset] | Store Register Exclusive, If allowed, mem[Rn + offset] ← Rt, clear exclusive tag, Rd ← 0. Else Rd ← 1. |
| STREXB | Rd, Rt, [Rn] | Store Register Exclusive Byte, mem[Rn] ← Rt[15:0] or mem[Rn] ← Rt[7:0], clear exclusive tag, Rd ← 0. Else Rd ← 1 |
| STREXH | Rd, Rt, [Rn] | Store Register Exclusive Halfword, mem[Rn] ← Rt[15:0] or mem[Rn] ← Rt[7:0], clear exclusive tag, Rd ← 0. Else Rd ← 1 |
| STRH, STRHT | Rt, [Rn, #offset] | Store Halfword, mem[Rn + offset] ← Rt[15:0] |
| STRT | Rt, [Rn, #offset] | Store Register with Translation, mem[Rn + offset] = Rt |
| SUB, SUBS | {Rd,} Rn, Op2 | Subtraction, Rd ← Rn ̶ Op2, SUBS updates N,Z,C,V |
| SUB, SUBS | {Rd,} Rn, #imm12 | Subtraction, Rd ← Rn-imm12, SUBS updates N,Z,C,V |
| SVC | #imm | Supervisor Call |
| SXTB | {Rd,} Rm {,ROR #n} | Sign Extend Byte, Rd ← SignExtend((Rm ROR (8\*n))[7:0]) |
| SXTH | {Rd,} Rm {,ROR #n} | Sign Extend Halfword, Rd ← SignExtend((Rm ROR (8\*n))[15:0]) |
| TBB | [Rn, Rm] | Table Branch Byte, PC ← PC+ZeroExtend(Memory(Rn+Rm,1)<<1) |
| TBH | [Rn, Rm, LSL #1] | Table Branch Halfword,  PC ← PC + ZeroExtend(Memory(Rn+Rm<<1, 2)<<1) |
| TEQ | Rn, Op2 | Test Equivalence, Update N,Z,C,V on Rn EOR Operand2 |
| TST | Rn, Op2 | Test, Update N,Z,C,V on Rn AND Op2 |
| UBFX | Rd, Rn, #lsb, #width | Unsigned Bit Field Extract, Rd[(width–1):0] = Rn[(width+lsb–1):lsb], Rd[31:width] = Replicate(0) |
| UDIV | {Rd,} Rn, Rm | Unsigned Divide, Rd ← Rn/Rm |
| UMLAL | RdLo, RdHi, Rn, Rm | Unsigned Multiply with Accumulate,  RdHi,RdLo ← unsigned(RdHi,RdLo + Rn\*Rm) |
| UMULL | RdLo, RdHi, Rn, Rm | Unsigned Multiply, RdHi,RdLo ← unsigned(Rn\*Rm) |
| USAT | Rd, #n, Rm{,shift #s} | Unsigned Saturate, Rd←UnsignedSat((Rm shift s),n), Update Q |
| UXTB | {Rd,} Rm {,ROR #n} | Unsigned Extend Byte, Rd ← ZeroExtend((Rm ROR (8\*n))[7:0]) |
| UXTH | {Rd,} Rm {,ROR #n} | Unsigned Extend Halfword,  Rd ← ZeroExtend((Rm ROR (8\*n))[15:0]) |
| WFE | - | Wait For Event |
| WFI | - | Wait for Interrupt |