Memory access and register move instructions

**LDR Rt, [Rn] // 32-bit load, EA=Rn**

**LDR Rt, [Rn,#n5] // 32-bit load, EA=Rn+n5**

**LDR Rt, [SP,#n8] // 32-bit load, EA=SP+n8**

**LDR Rt, [Rn,Rm] // 32-bit load, EA=Rn+Rm**

**LDR Rt, label2 // read contents at label2, PC rel, EA=PC+relative**

**LDR Rt, =number // Rt=number, PC relative, EA=PC+relative**

**LDRH Rt, [Rn] // 16-bit unsigned load, EA=Rn**

**LDRH Rt, [Rn,#h5] // 16-bit unsigned load, EA=Rn+h5**

**LDRH Rt, [Rn,Rm] // 16-bit unsigned load, EA=Rn+Rm**

**LDRSH Rt, [Rn,Rm] // 16-bit signed load, EA=Rn+Rm**

**LDRB Rt, [Rn] // 8-bit unsigned load, EA=Rn**

**LDRB Rt, [Rn,#imm5] // 8-bit unsigned load, EA=Rn+imm5**

**LDRB Rt, [Rn,Rm] // 8-bit unsigned load, EA=Rn+Rm**

**LDRSB Rt, [Rn,Rm] // 8-bit signed load, EA=Rn+Rm**

**STR Rt, [Rn] // 32-bit store, EA=Rn**

**STR Rt, [Rn,#n5] // 32-bit store, EA=Rn+n5**

**STR Rt, [SP,#n8] // 32-bit store, EA=SP+n8**

**STR Rt, [Rn,Rm] // 32-bit store, EA=Rn+Rm**

**STRH Rt, [Rn] // 16-bit store, EA=Rn**

**STRH Rt, [Rn,#h5] // 16-bit store, EA=Rn+h5**

**STRH Rt, [Rn,Rm] // 16-bit store, EA=Rn+Rm**

**STRB Rt, [Rn] // 8-bit store, EA=Rn**

**STRB Rt, [Rn,#imm5] // 8-bit store, EA=Rn+imm5**

**STRB Rt, [Rn,Rm] // 8-bit store, EA=Rn+Rm**

**MOV Rd2, Rm2 // move contents of Rm2 into Rd2**

**MOVS Rd, Rm // move contents of Rm into Rd, set flags**

**MOVS Rd, #imm8 // move contents of imm8 into Rd, set flags**

**MVNS Rd, Rm // set Rd equal to ~Rm (logical NOT)**

Compare and Branch instructions

**CMP Rd, #imm8 // Rd – imm8, set flags**

**CMP Rn, Rm // Rn – Rm, set flags**

**CMN Rn, Rm // Rn - (-Rm), set flags**

**B label0 // branch to label0 Always**

**BEQ label // branch if Z == 1 Equal**

**BNE label // branch if Z == 0 Not equal**

**BCS/BHS label // branch if C == 1 Higher or same, unsigned ≥**

**BCC/BLO label // branch if C == 0 Lower, unsigned <**

**BMI label // branch if N == 1 Negative**

**BPL label // branch if N == 0 Positive or zero**

**BVS label // branch if V == 1 Overflow**

**BVC label // branch if V == 0 No overflow**

**BHI label // branch if C==1 and Z==0 Higher, unsigned >**

**BLS label // branch if C==0 or Z==1 Lower or same, unsigned ≤**

**BGE label // branch if N == V Greater than or equal, signed ≥**

**BLT label // branch if N != V Less than, signed <**

**BGT label // branch if Z==0 and N==V Greater than, signed >**

**BLE label // branch if Z==1 or N!=V Less than or equal, signed ≤**

Function call, function return, stack, and interrupt instructions

**PUSH {reglist} // push 32-bit registers onto stack, R0-R7,LR**

**POP {reglist} // pop 32-bit from stack into registers, R0-R7,PC**

**ADD Rd, SP, #n8 // Rd = SP+n8**

**ADD SP, SP, #n7 // SP = SP+n7**

**SUB SP, SP, #imm7w // SP = SP-imm7w**

**BL label1 // branch to subroutine at label1, anywhere**

**BLX Rm4 // branch to subroutine specified by Rm4, R0-R12**

**BX Rm3 // branch to location specified by Rm3, R0-R12,LR**

**CPSIE I // enable interrupts (I=0)**

**CPSID I // disable interrupts (I=1)**

**WFI // sleep and wait for interrupt**

**SVC #imm8 // software interrupt**

Logical and shift instructions

**ANDS Rdn, Rm // Rdn = Rdn&Rm**

**ORRS Rdn, Rm // Rd=Rn|Rm**

**EORS Rdn, Rm // Rd=Rn^Rm**

**BICS Rdn, Rm // Rd=Rn&(~Rm) (op2 is 32 bits)**

**LSRS Rd, Rd, Rs // logical shift right Rd=Rd>>Rs (unsigned)**

**LSRS Rd, Rm, #n // logical shift right Rd=Rm>>n (unsigned), 0 to 31**

**ASRS Rd, Rm, Rs // arithmetic shift right Rd=Rd>>Rs (signed)**

**ASRS Rd, Rm, #n // arithmetic shift right Rd=Rm>>n (signed), 1 to 32**

**LSLS Rd, Rd, Rs // shift left Rd=Rd<<Rs (signed or unsigned)**

**LSLS Rd, Rm, #n // shift left Rd=Rm<<n (signed or unsigned), 1 to 32**

*Arithmetic instructions*

**ADDS Rd, Rn, #imm3 // Rd = Rn+imm3, set flags**

**ADDS Rdn, #imm8 // Rdn = Rdn+imm8, set flags**

**ADDS Rd, Rn, Rm // Rd = Rm+Rn, set flags**

**ADD Rd2, Rm // Rd2 = Rd2+Rm**

**SUBS Rd, Rn, #imm3 // Rd = Rn-imm3, set flags**

**SUBS Rdn, #imm8 // Rdn = Rdn-imm8, set flags**

**SUBS Rd, Rn, Rm // Rd = Rm-Rn**

**RSBS Rd, Rn, #0 // Rd = 0-Rn, set flags**

**MULS Rdn, Rdn, Rm // Multiply Rdn = Rdn\*Rm, set flags**

Notes

**Rd Rdn Rm Rn Rt represent 32-bit registers R0 to R7**

**Rd2 Rm2 represent 32-bit registers R0 to R15**

**number any 32-bit value: signed, unsigned, or address**

**label0 -2048 to 2046, in multiples of 2, from PC**

**label -256 to 254, in multiples of 2, from PC**

**label2 any address within 0 to 1020, in multiples of 4, from PC**

**#h5 any value from 0 to 62 in multiples of 2**

**#n5 any value from 0 to 124 in multiples of 4**

**#n7 any value from 0 to 508 in multiples of 4**

**#n8 any value from 0 to 1020 in multiples of 4**

**#imm3 any value from 0 to 7**

**#imm5 any value from 0 to 31**

**#imm8 any value from 0 to 255**

**.data // places following lines in RAM**

**.text // places following lines in ROM**

**.align 2 // skips 0-3 bytes so the address of next line is divisible by 4**

**.equ size,10 // defines an assembly constant size with value 10**

**.byte 1,2,3 // allocates three 8-bit byte(s)**

**.short 1,2,3 // allocates three 16-bit halfwords**

**.long 1,2,3 // allocates three 32-bit words**

**.space 4 // reserves 4 bytes**

**GPIOB\_DIN31\_0** (assembly) or **GPIOB->DIN31\_0** (C code)

Read from gets the current values of the input pins of port B

**GPIOB\_DOUT31\_0** (assembly) or **GPIOB->DOUT31\_0** (C code)

Write to sets the output pins of port B; read from gets the last value written

**GPIOB\_DOUTSET31\_0** (assembly) or **GPIOB->DOUTSET31\_0** (C code)

Write 1 to bit **n** to make the output pin **n** go high

**GPIOB\_DOUTCLR31\_0** (assembly) or **GPIOB->DOUTCLR31\_0** (C code)

Write 1 to bit **n** to make the output pin **n** go low

**GPIOB\_DOUTTGL31\_0** (assembly) or **GPIOB->DOUTTGL31\_0** (C code)

Write 1 to bit **n** to toggle the output pin **n** (invert from 0 to 1 or 1 to 0)