Aarch64 V8 most common instructions (Work in Progress - V3.0)

General conventions

`rd, rn, rm: W or X registers; op2: register or immediate`	`if {S} is present flags will be affected`

Addition	`ADD{S}`	`ADD{S} rd, rn, op2`	`rd = rn + op2`	{Yes}
Subtraction	`SUB{S}`	`SUB{S} rd, rn, op2`	`rd = rn - op2`	{Yes}
Negation	`NEG{S}`	`NEG{S} rd, op2`	`rd = –op2`	{Yes}
with carry	`NGC{S}`	`NGC{S} rd, rm`	`rd = –rm – ~C`	{Yes}

Multiply	`MUL`	`MUL rd, rn, rm`		
Multiply and add	`MADD`	`MADD rd, rn, rm, ra`	`rd = ra + (rn x rm)`	
Multiply and sub	`MSUB`	`MSUB rd, rn, rm, ra`	`rd = ra - (rn x rm)`	
Multiply and neg	`MNEG`	`MNEG rd, rn, rm`	`rd = - (rn x rm)`	

Integer divide unsigned	`UDIV`	`UDIV rd, rn, rm`	`rd = rn / rm`	
Integer divide signed	`SDIV`	`UDIV rd, rn, rm`	`rd = rn / rm`	
Note: the remainder may be computed using the MSUB instruction as numerator – (quotient x denominator)				

Logical	`AND`	`AND{S} rd, rn, op2`	`rd = rn & op2`	{Yes}
	`BIC`	`BIC{S} rd, rn, op2`	`rd = rn & ~op2`	{Yes}
	`EON`	`EON rd, rn, op2`	`rd = rn ⊕ ~op2`	
	`EOR`	`EOR rd, rn, op2`	`rd = rn ⊕ op2`	
	`LSL`	`LSL rd, rn, op2`		
	`LSR`	`LSR rd, rn, rm`		
	`ASR`	`ASR rd, rn, op2`		
	`MOV`	`MOV rd, op2`	`rd = op2`	Yes
	`MVN`	`MVN rd, op2`	`rd = ~op2`	
	`ORN`	`ORN rd, rn, op2`	`rd = rn | ~op2`	
	`ORR`	`ORR rd, rn, op2`	`rd = rn | op2`	
	`ROR`	`ROR  rd, rn, op2`		
	`TST`	`TST rn, op2`	`rn & op2`	Yes

Load single register	`LDR`	`LDR rt, [addr]`	`rt = Mem[addr]`	
Sub-type byte	`LDRB`	`LDRB wt, [addr]`	`wt = byte[addr] (only 32-byte containers)`	
Sub-type signed byte	`LDRSB`	`LDRSB rt, [addr]`	`rt = sbyte[addr]`	
Sub-type half word	`LDRH`	`LDRH wt, [addr]`	`wt = hword[addr] (only 32-byte containers)`	
Sub-type signed half word	`LDRSH`	`LDRSH rt, [addr]`	`rt = Mem[addr] (load one half word, signed)`	
Sub-type signed word	`LDRSW`	`LDRSW xt, [addr]`	`xt = sword[addr] (only for 64-byte containers)`	
Store single register	`STR`	`rt, [addr]`	`Mem[addr] = rt`	
Subtype byte	`STRB`	`wt, [addr]`		
Subtype half word	`STRH`	`wt, [addr]`		
Load register pair				
Store register pair				

Branch	`B`	`B target`	`Jump to target`	
Conditional branch	`B.CC`	`B.cc target`	`If (cc) jump to target`	
Compare and branch if zero	`CBZ`	`CBZ rd, target`	`If (rd=0) jump to target`	
Compare and branch if not zero	`CBNZ`	`CBNZ rd, target`	`If (rd≠0) jump to target`	

Conditional select	`CSEL`	`CSEL rd, rn, rm, cc`	`If (cc) rd = rn else rd = rm`	
with increment,	`CSINC`	`CSINC rd, rn, rm, cc`	`If (cc) rd = rn else rd = rm+1`	
with negate,	`CSNEG`	`CSNEG rd, rn, rm, cc`	`If (cc) rd = rn else rd = -rm`	
with invert	`CSINV`	`CSINV rd, rn, rm, cc`	`If (cc) rd = rn else rd = ~rm`	

Conditional set	`CSET`	`CSET rd, cc`	`If (cc) rd = 1 else rd = 0`	
with mask,	`CSETM`	`CSETM rd, cc`	`If (cc) rd = -1 else rd = 0`	
with increment,	`CINC`	`CINC rd, rn, cc`	`If (cc) rd = rn+1 else rd = rn`	
with negate,	`CNEG`	`CNEG rd, rn, cc`	`If (cc) rd = -rn else rd = rn`	
with invert	`CINV`	`CINV rd, rn, cc`	`If (cc) rd = ~rn else rd = rn`	

Compare	`CMP`	`CMP  rd, op2`	`rd - op2`	Yes
with negative	`CMN`	`CMN  rd, op2`	`rd - (-op2)`	Yes
Conditional compare	`CCMP`	`CCMP rd, rn, #im4, cc`	`If (cc) NZCV = CMP(rd,rn) else NZCV = #imm4`	Yes
with negative	`CCMN`	`CCMP rd, rn, #im4, cc`	`If (cc) NZCV = CMP(rd,-rn) else NZCV = #imm4`	Yes
`Note: for these instructions rn can also be a 5-bit unsigned immediate value (0..32)`				

Aarch64 V8 accessory information

Condition codes (magnitude of operands)

LO	Lower, unsigned	C = 0
HI	Higher, unsigned	C = 1 and Z = 0
LS	Lower or same, unsigned	C = 0 or Z = 1
HS	Higher or same, unsigned	C = 1
LT	Less than, signed	N != V
GT	Greater than, signed	Z = 0 and N = V
LE	Less than or equal, signed	Z = 1 and N != V
GE	Greater than or equal, signed	N = V

Condition codes (direct flags)

EQ	Equal	Z = 1
NE	Not equal	Z = 0
MI	Negative	N = 1
PL	Positive or zero	N = 0
VS	Overflow	V = 1
VC	No overflow	V = 0
CS	Carry	C = 0
CC	No carry	C = 1

Sub types (suffix of some instructions)

B	byte	8 bits
SB	signed byte	8 bits
H	Half word	16 bits
SH	signed half word	16 bits
W	word	32 bits
SW	signed word	32 bits

Flags set to 1 when:

N	the result of the last operation was negative, cleared to 0 otherwise
Z	the result of the last operation was zero, cleared to 0 otherwise
C	the last operation resulted in a carry, cleared to 0 otherwise
V	the last operation caused overflow, cleared to 0 otherwise

Addressing modes (base: register; offset: register or immediate)

[base]	MEM[base]	
[base, offset]	MEM[base+offset]	
[base, offset]!	MEM[base+offset] then base = base + offset	(pre indexed)
[base], offset	MEM[base] then base = base + offset	(post indexed)

Containers

Xn	64 bit register
Wn	32 LSB of Xn

Sizes, Assembly and C

8	byte	char
16	Half word	short int
32	word	int
64	double word	long int
128	quad word	-

Op2 processing

```
LSL|LSR|ASR #imm
SXTW
SXTB {#imm}
```

Range of immediate values…

Calling convention(params and saved registers)