3-Register Format							
			Machine-code encoding (big-endian)				
Instruction	**Assembly syntax**	**Behaviour**	**Opcode**	**xm**	**Flags**	**xn**	**xd**
Add	`add xd, xn, xm`	`xd = xn + xm`	`10001011 001`	`mmmmm`	`011000`	`nn nnn`	`ddddd`
Subtract	`sub xd, xn, xm`	`xd = xn - xm`	`11001011 001`	`mmmmm`	`011000`	`nn nnn`	`ddddd`
Multiply	`mul xd, xn, xm`	`xd = (xn * xm) % 2^64`	`10011011 000`	`mmmmm`	`011111`	`nn nnn`	`ddddd`
Multiply Overflow Signed	`smulh xd, xn, xm`	`xd = (xn * xm) / 2^64`	`10011011 010`	`mmmmm`	`011111`	`nn nnn`	`ddddd`
Multiply Overflow Unsigned	`umulh xd, xn, xm`	`xd = (xn * xm) / 2^64`	`10011011 110`	`mmmmm`	`011111`	`nn nnn`	`ddddd`
Divide Signed	`sdiv xd, xn, xm`	`xd = xn / xm`	`10011010 110`	`mmmmm`	`000011`	`nn nnn`	`ddddd`
Divide Unsigned	`udiv xd, xn, xm`	`xd = xn / xm`	`10011010 110`	`mmmmm`	`000010`	`nn nnn`	`ddddd`
Compare	`cmp xn, xm`	`compare xn and xm`	`11101011 001`	`mmmmm`	`011000`	`nn nnn`	`11111`
Branch to Register	`br xn`	`NPC = xn`	`11010110 000`	`11111`	`000000`	`nn nnn`	`00000`
Branch & Link Register	`blr xn`	`NPC = xn; x30 = PC + 4`	`11010110 001`	`11111`	`000000`	`nn nnn`	`00000`

2-Register Format							
			Machine-code encoding (big-endian)				
Instruction	**Assembly syntax**	**Behaviour**	**Opcode**	**Immediate**	**Unused**	**xn**	**xd**
Load Register	`ldur xd, [xn, i]`	`xd = MEM[xn + i]`	`11111000 010`	`iiiii iiii`	`00`	`nn nnn`	`ddddd`
Store Register	`stur xd, [xn, i]`	`MEM[xn + i] = xd`	`11111000 000`	`iiiii iiii`	`00`	`nn nnn`	`ddddd`

1-Register Format					
			Machine-code encoding (big-endian)		
Instruction	**Assembly syntax**	**Behaviour**	**Opcode**	**Immediate**	**xd**
PC-Relative Load	`ldr xd, i`	`xd = MEM[PC + i(*4)]`	`01011000`	`iiiiiiii iiiiiiii iii`	`ddddd`

Branching Format				
			Machine-code encoding (big-endian)	
Instruction	**Assembly syntax**	**Behaviour**	**Opcode**	**Immediate (& Condition)**
Branch	`b i`	`NPC = PC + i(*4)`	`000101`	`ii iiiiiiii iiiiiiii iiiiiiii`
Conditional Branch	`b.`*`cond`* `i`	`if (cond) NPC = PC + i(*4)`	`01010100`	`iiiiiiii iiiiiiii iiiccccc`

Condition codes		
Condition	**Assembly syntax**	**Bits**
Equals	`eq`	`00000`
Not equals	`ne`	`00001`
Greater than or equals (unsigned)	`hs`	`00010`
Less than (unsigned)	`lo`	`00011`
Greater than (unsigned)	`hi`	`01000`
Less than or equals (unsigned)	`ls`	`01001`
Greater than or equals (signed)	`ge`	`01010`
Less than (signed)	`lt`	`01011`
Greater than (signed)	`gt`	`01100`
Less than or equals (signed)	`le`	`01101`

- `ldr`, `b`, `b.`*`cond`*: In assembly code, `i` represents bytes, but in machine code, `i` represents halfwords.
- I/O:
  - When a word is stored to memory location `0xc000 0000 0001 0008`, the least-significant byte of the word is sent to standard output.
  - When a word is read from memory location `0xc000 0000 0001 0000`, a byte is read from standard input, with EOF represented as -1.