

Porting - ARMV8 Neon

LCE13, Dublin. 8-12 July 2013



Contents

- ARMV8 Neon
- Vector operations on ARMV8
- Scalar operation on ARMV8
- Neon load/store instructions
- Load/Store illustration
- Widening/Narrow Arithemetic
- Vector Reduction
- ARMV7 vs ARMV8 Neon Instructions
- Armv8 Porting (Jpeg Turbo IDCT code)



ARMV8 Neon

- Neon is an optional extension to ARMV8
- Neon performs packed SIMD operations on either integer or floating point (Single/double precision) data.
- Neon extension registers are distinct from ARM core register set and contains
 - 32 X 128 bit wide vector registers
 - 32 X 64 bit wide vector registers which are held in lower 64 bit of each 128 bit registers

Shape (bits×lanes)	8b×8	8b×16	16b×4	16b×8	32b×2	32b×4	64b×1	64b×2
Name	Vn.8B	Vn.16B	Vn.4H	Vn.8H	Vn.2S	Vn.4S	Vn.1D	Vn.2D



Vector Operations on ARMV8

Vector Operations (Data type - byte)

```
- Char array1[8] = \{1,2,3,4,5,6,7,8\}
```

$$-$$
 Char array2[8] = {1,2,3,4,5,6,7,8}



Scalar Operation on Neon

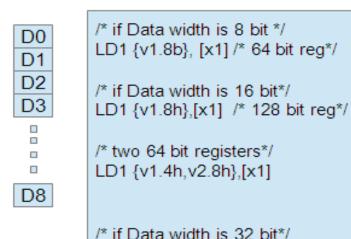
- Scalar Operations
 - Some SIMD instruction operate on scalar data and registers holding these data behaves similar to main general purpose registers.

Size (bits)	8b	16b	32b	64b	128b
Name	Bn	Hn	Sn	Dn	Qn

- Suffix n ->{registers from 0 to 31}
- Unused higher bits are not accessed during read and set to zero during write.

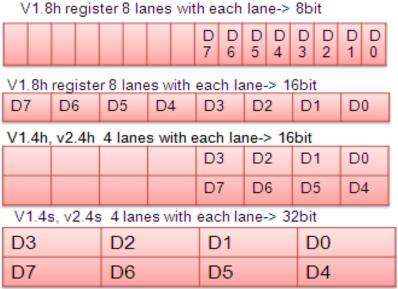


- LD1,LD2,LD3,LD4 and ST1,ST2,ST3 & ST4 are some of the vector load and store instructions
- The format of this instruction is -> it can contain $1 \sim 4$ consecutive registers to/from which consecutive 8,16,32,64 bit elements can be transferred.
- LD1 transfers data from memory to consecutive registers without any interleave



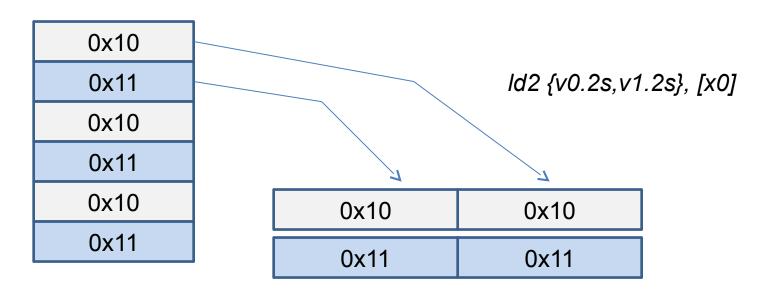
LD1 {v1.4s, v2.4s},[x1]







- LD2/St2 instruction loads elements to the specified registers/memory in an interleaved fashion.
- The format of the instruction ld2 {vn.<t>, vm.<t>}, [x1] t-> 8b,16b, 4h,8h,2s,4s,d,2d for this particular instruction the rules are the registers should be consecutive and the number of registers has to be 2 or 4.





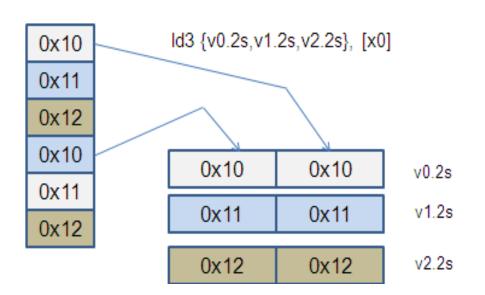
LD3/St3 instruction loads elements to the specified registers / memory with 2 data element interleaved .

The format of the instruction

ld3 {vi.<t>, vj.<t>,vk}, [x1] t-> 8b,16b, 4h,8h,2s,4s,d,2d

For this particular instruction the rules are registers should be consecutive and the number of registers has to be 3.

optional space is allowed to support 128bit vector



ld3	{v0.2s,v2.2s,v4.2s},	[x0]	
-----	----------------------	------	--

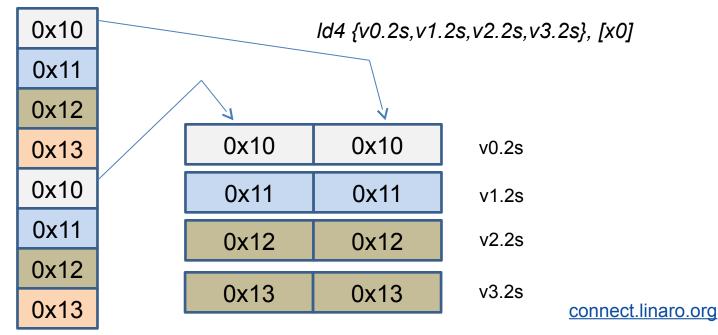
0x10	0x10	v0.2s
		v1.2s
0x11	0x11	v2.2s
		v3.2s
0x11	0x11	v4.2s
		v5.2s



- LD4/St4 instruction loads elements to the specified registers/memory with 3 data element interleaved.
- The format of the instruction-> ld3 {vi.<t>, vj.<t>,vk}, [x1] t-> 8b,16b, 4h,8h,2s,4s,d,2d

Instruction rules:

- 1. the registers should be consecutive and the number of registers has to be 4.
- 2. optional space is allowed to support 128bit vector

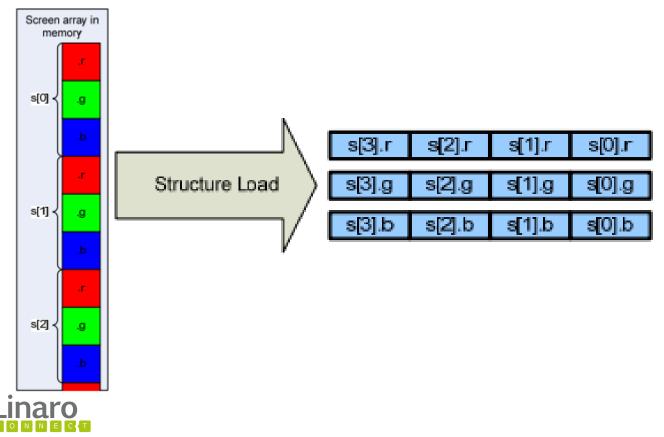




Example RGB channel:

Example: Load 12 16 bit values from the address stored in R0, and split them over 64 bit registers V0.8B, V1.8B and V2.8B.

LD3 {v0.8B,V1.16B,V2.16B}, [X0]



Narrow/Wide Neon Instructions

Narrow,Long and Wide Instructions support

- Long Instructions SADDL Vd.<Td>, Vn.<Ts>, Vm.<Ts>
 <Td>/<Ts> is 8H/8B, 4S/4H or 2D/2S.
- 2. Wide Instruction
 SADDW Vd.<Td>, Vn.<Td>, Vm.<Ts>
 <Td>/<Ts> is 8H/8B, 4S/4H or 2D/2S
- 3. Narrow Instruction -> add and narrow higher half ADDHN Vd.<Td>, Vn.<Ts>, Vm.<Ts>



Vector Reduction

Vector Reduce

- addv <v>d, vn<T> (eg:- addv v15.[5]b, v5.8b)
 <v>d is the destination scalar register (b,h,s,d)
 Vn<T> is vector register of size T (8b/16b, 4h/8h, 2s/4s,d/2d)
- saddlv <v>d, vn<T> (eg:- addv v15.[5]h, v5.8b)
 <v>d is the destination scalar register (h,s,d)
 Vn<T> is vector register of size T (8b/16b, 4h/8h, 2s/4s,d/2d)



Scalar Operation on Neon

Difference in the mnemonics

ARMV7	ARMV8
1. vld1.16 {d1,d2,d3,d4}, [r0]	1. ld1 {v2.4h,v3.4h,v4.4h}, [x0]
2. vsub.s32 q1,q1,q2	2. ssub v1.4s,v1.4s,v1.4s

- 1. ARMV7 mnemonic prefix 'v' has been removed and replaced by s/u/f/p which indicates the data type as signed/unsigned/float/polynomial
- 2. Vector organization (element Size/ number of lanes) is actually described by the register qualifiers and not by the mnemonics as in ARMV7.
- 3. suffix like V has been added to the mnemonic to indicate it's a reduction type operation, suffix 2 indicates the instruction is a widening/narrowing type operating on the second half of the register.



Armv8 Porting (Jpeg Turbo IDCT code)

- 1. IDCT (integer & Fast algorithms) ARMV7 neon code availability in Jpeg Turbo.
- 2. Some of the instruction of ARMV7 doesnot exist in ARMV8 like vswap, push, pull and has partial load pair instructions.
- 3. IDCT ARMV7 code takes advantage of complete register overlap of 64bit & 128bit registers and this can increase the effort of porting.

Other methods of porting if you dont have an ARMv7 code

- 1. use of Intrinisics can be a good option but bit risky as it doesnot guarntee the effective neon code generation as intended
- 2. can seperate the function of interest and generate armv8 neon code through vectorizaiton option and start optimizing that.





Questions?





More about Linaro: http://www.linaro.org/about/

More about Linaro engineering: http://www.linaro.org/engineering/

How to join: http://www.linaro.org/about/how-to-join

Linaro members: www.linaro.org/members