3.5 Parallelism and computer arithmetic: Subword parallelism

Since every microprocessor in a phone, tablet, or laptop by definition has its own graphical display, as transistor budgets increased it was inevitable that support would be added for graphics operations.

Many graphics systems originally used 8 bits to represent each of the three primary colors plus 8 bits for a location of a pixel. The addition of speakers and microphones for teleconferencing and video games suggested support of sound as well. Audio samples need more than 8 bits of precision, but 16 bits are sufficient.

Every microprocessor has special support so that bytes and halfwords take up less space when stored in memory (see COD Section 2.9 (Communicating with people)), but due to the infrequency of arithmetic operations on these data sizes in typical integer programs, there was little support beyond data transfers. Architects recognized that many graphics and audio applications would perform the same operation on vectors of these data. By partitioning the carry chains within a 128-bit adder, a processor could use **parallelism** to perform simultaneous operations on short vectors of sixteen 8-bit operands, eight 16-bit operands, four 32-bit operands, or two 64-bit operands. The cost of such partitioned adders was small yet the speedups could be large.

Given that the parallelism occurs within a wide word, the extensions are classified as *subword parallelism*. It is also classified under the more general name of *data level parallelism*. They are known as well as vector or SIMD, for single instruction, multiple data (see COD Section 6.6 (Introduction to graphics processing units)). The rising popularity of multimedia applications led to arithmetic instructions that support narrower operations that can easily compute in parallel.

PARTICIPATION ACTIVITY 3.5.1: Subword parallelism.

1) Subword parallelism takes advantage of byte- and halfword-sized data by _____.
   - ○ turning off the unused bits of a 128-bit adder
   - ○ splitting the add operation to execute across multiple cycles
   - ○ partitioning the adder to perform multiple operations in parallel

For example, ARMv8 added 32 128-bit registers (`V0, V1,…,V31`) and more than 500 machine-language instructions to support subword parallelism. It supports all the subword data types you can imagine:

- 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit signed and unsigned integers
- 32-bit and 64-bit floating point numbers

The figure below gives a quick summary of some basic ARMv8 SIMD instructions, while COD Section 3.8 (Real stuff: The rest of the ARMv8 arithmetic instructions) surveys the full SIMD architecture.

Figure 3.5.1: Example of ARMv8 SIMD instructions for subword parallelism (COD Figure 3.18).

COD Figure 3.21 (Full ARMv8 assembly language for SIMD instructions …) shows the full set of SIMD instructions.

Type	Description	Name	Size (bits)					FP Precision	
			8	16	32	64	128	SP	DP
Add/ Subtract	Integer add	ADD	✓	✓	✓	✓	✓		
	FP add	FADD						✓	✓
	Integer subtract	SUB	✓	✓	✓	✓	✓		
	FP subtract	FSUB						✓	✓
Multiply	Unsigned integer multiply	UMUL	✓	✓	✓	✓	✓		
	Signed integer multiply	SMUL	✓	✓	✓	✓	✓		
	FP multiply	FMUL						✓	✓
Compare	Integer compare equal	CMEQ	✓	✓	✓	✓	✓		
	FP compare equal	FCMEQ						✓	✓
Min/Max	Unsigned integer minimum	UMIN	✓	✓	✓	✓	✓		
	Signed integer minimum	SMIN	✓	✓	✓	✓	✓		
	FP minimum	FMIN						✓	✓
	Unsigned integer maximum	UMAX	✓	✓	✓	✓	✓		
	Signed integer maximum	SMAX	✓	✓	✓	✓	✓		
	FP maximum	FMAX						✓	✓
Shift	Integer shift left	SHL	✓	✓	✓	✓	✓		
	Unsigned integer shift right	USHR	✓	✓	✓	✓	✓		
	Signed integer shift right	SSHR	✓	✓	✓	✓	✓		
Logical	Bitwise AND	AND	✓	✓	✓	✓	✓		
	Bitwise OR	ORR	✓	✓	✓	✓	✓		
	Bitwise exclusive OR	EOR	✓	✓	✓	✓	✓		
Data Transfer	Load register	LDR	✓	✓	✓	✓	✓	✓	✓
	Store register	STR	✓	✓	✓	✓	✓	✓	✓

Rather than have a different assembly mnemonic for each data width, the ARMv8 assembler uses different suffixes for the SIMD registers to represent different widths, which allows it to pick the proper machine language opcode. The suffixes are B (*byte*) for 8-bit operands, H (*half*) for 16-bit operands, S (*single*) for 32-bit operands, D (*double*) for 64-bit operands, and Q (*quad*) for 128-bit operands. The programmer also specifies the number of subword operations for that data width with a number before the register name. Thus, to perform 16 parallel integer adds between the 8-bit elements in registers `V2` and `V3` and place the sixteen 8-bit sums in the elements of `V1`, we write:

```
ADD V1.16B, V2.16B, V3.16B        // 16 8-bit integer adds
```

To perform four parallel 32-bit floating-point additions, we write:

```
FADD V1.4S, V2.4S, V3.4S          // 4 32-bit floating-point adds
```

PARTICIPATION ACTIVITY 3.5.2: Support for subword parallelism.

1) ARMv8 added _____-bit registers to support subword parallelism.

- 64
- 128

2) ARMv8 has little support for subword parallelism beyond data transfer instructions.

- True
- False

3) The add instruction, ADD, includes support for 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit numbers. Operands can be integers, but not floating point numbers.

- True
- False

Provide feedback on this section