10.4 Instructions: Multimedia extensions of the desktop/server RISCs

(Original section[1])

Since every desktop microprocessor by definition has its own graphical displays, as transistor budgets increased it was inevitable that support would be added for graphics operations. Many graphics systems use 8 bits to represent each of the three primary colors plus 8 bits for the 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, but due to the infrequency of arithmetic operations on these data sizes in typical integer programs, there is little support beyond data transfers. The architects of the Intel i860, which was justified as a graphical accelerator within the company, recognized that many graphics and audio applications would perform the same operation on vectors of these data. Although a vector unit was beyond the transistor budget of the i860 in 1989, by partitioning the carry chains within a 64-bit ALU, it could perform simultaneous operations on short vectors of eight 8-bit operands, four 16-bit operands, or two 32-bit operands. The cost of such partitioned ALUs was small. Applications that lend themselves to such support include MPEG (video), games like DOOM (3-D graphics), Adobe Photoshop (digital photography), and teleconferencing (audio and image processing).

Like a virus, over time such multimedia support has spread to nearly every desktop microprocessor. HP was the first successful desktop RISC to include such support. As we shall see, this virus spread unevenly. The PowerPC is the only holdout, and rumors are that it is "running a fever."

These extensions have been called subword parallelism, vector, or SIMD (single-instruction, multiple data) (see COD Chapter 6 (Parallel Processor from Client to Cloud)). Since Intel marketing uses SIMD to describe the MMX extension of the 8086, that has become the popular name. The figure below summarizes the support by architecture.

Figure 10.4.1: Summary of multimedia support for desktop RISCs (COD Figure D.4.1).

B stands for byte (8 bits), H for half word (16 bits), and W for word (32 bits). Thus 8B means an operation on eight bytes in a single instruction. Pack and unpack use the notation 2*2W to mean two operands each with two words. Note that MDMX has vector/scalar operations, where the scalar is specified as an element of one of the vector registers. This table is a simplification of the full multimedia architectures, leaving out many details. For example, MIPS MDMX includes instructions to multiplex between two operands, HP MAX2 includes an instruction to calculate averages, and SPARC VIS includes instructions to set registers to constants. Also, this table does not include the memory alignment operation of MDMX, MAX, and VIS.

Instruction category	Alpha MAX	MIPS MDMX	PA-RISC MAX2	PowerPC	SPARC VIS
Add/subtract		8B, 4H	4H		4H, 2W
Saturating add/sub		8B, 4H	4H		
Multiply		8B, 4H			4B/H
Compare	8B (>=)	8B, 4H (=,<,<=)			4H, 2W (=, not=, >, <=)
Shift right/left		8B, 4H	4H		
Shift right arithmetic		4H	4H		
Multiply and add		8B, 4H			
Shift and add (saturating)			4H		
And/or/xor	8B, 4H, 2W	8B, 4H, 2W	8B, 4H, 2W		8B, 4H, 2W
Absolute difference	8B				8B
Max/min	8B, 4W	8B, 4H			
Pack (2*n* bits –> *n* bits)	2W->2B, 4H->4B	2*2W->4H, 2*4H->8B	2*4H->8B		2W->2H, 2W->2B, 4H->4B
Unpack/merge	2B->2W, 4B->4H	2*4B->8B, 2*2H->4H			4B->4H, 2*4B->8B
Permute/shuffle		8B, 4H	4H		
Register sets	Integer	Fl. Pt. + 192b Acc.	Integer		Fl. Pt.

From the figure above, you can see that in general, MIPS MDMX works on eight bytes or four halfwords per instruction, HP PA-RISC MAX2 works on four halfwords, SPARC VIS works on four halfwords or two words, and Alpha doesn't do much. The Alpha MAX operations are just byte versions of compare, min, max, and absolute difference, leaving it up to software to isolate fields and perform parallel adds, subtracts, and multiplies on bytes and halfwords. MIPS also added operations to work on two 32-bit floating-point operands per cycle, but they are considered part of MIPS V and not simply multimedia extensions (see COD Section D.7 (Instructions Unique to MIPS-64)).

One feature not generally found in general-purpose microprocessors is saturating operations. Saturation means that when a calculation overflows, the result is set to the largest positive number or most negative number, rather than a modulo calculation as in two's complement arithmetic. Commonly found in digital signal processors (see COD Section D.5 (Instructions: Digital Signal-Processing Extensions of the Embedded RISCs)), these saturating operations are helpful in routines for filtering.

These machines largely used existing register sets to hold operands: integer registers for Alpha and HP PA-RISC and floating-point registers for MIPS and Sun. Hence data transfers are accomplished with standard load and store instructions. MIPS also added a 192-bit (3*64) wide register to act as an accumulator for some operations. By having three times the native data width, it can be partitioned to accumulate either eight bytes with 24 bits per field or four halfwords with 48 bits per field. This wide accumulator can be used for add, subtract, and multiply/ add instructions. MIPS claims performance advantages of two to four times for the accumulator.

Perhaps the surprising conclusion of this table is the lack of consistency. The only operations found on all four are the logical operations (AND, OR, XOR), which do not need a partitioned ALU. If we leave out the frugal Alpha, then the only other common operations are parallel adds and subtracts on four halfwords.

Each manufacturer states that these are instructions intended to be used in hand-optimized subroutine libraries, an intention likely to be followed, as a compiler that works well with multimedia extensions of all desktop RISCs would be challenging.

(*1) This section is in original form.

Provide feedback on this section