UltraSparc Adds Multimedia Instructions

Other New Instructions Handle Unaligned and Little-Endian Data

by Linley Gwennap

Although a few others have dabbled in this area, Sun's UltraSparc processor *(see **081301.PDF**)* offers the most extensive set of features aimed at improving performance on multimedia data types. With just a small amount of special hardware, UltraSparc increases the speed of some calculations by eight times or more. This new hardware will give the forthcoming Sun chip an advantage when competing against other next-generation processors, an advantage that is not apparent from basic SPEC measurements. As multimedia applications become more prevalent, other manufacturers are likely to include similar features in future designs.

The idea of special instructions for graphics data go as far back as the Intel i860 (see MPR 3/89, p. 1), which has instructions for Z-buffer checks. Motorola's 88110 (see MPR 12/4/91, p. 1) can pack pixel data and perform add, subtract, and multiply operations in parallel. Hewlett-Packard has implemented generalized multimedia functions in its processors *(see **080103.PDF**)* without adding special function units.

UltraSparc's feature set is similar to that of the 88110 but adds some highly specialized instructions that go beyond any previous chip. UltraSparc's new instructions are not included as part of the SPARC V9 architecture; they are implementation-dependent instructions that, so far, are not part of any SPARC processor other than UltraSparc. Sun plans to include these features in future designs and has offered to make them available to other SPARC processor designers.

Opcode	Operands	Description
FPADD16/32(S) FPSUB16/32(S)	fsrc1,fsrc2,fdest	Four 16-bit or two 32-bit partitioned add or subtract
FPACK16	fsrc2,fdest	Pack four 16-bit pixels into fdest
FPACK32	fsrc1,fsrc2,fdest	Add two 32-bit pixels into fdest
FPACKFIX	fsrc2,fdest	Pack two 32-bit pixels into fdest
FEXPAND	fsrc2,fdest	Expand four 8-bit pixels into fdest
FPMERGE	fsrc1,fsrc2,fdest	Merge two sets of four 8-bit pixels
FMUL8x16(opt)	fsrc1,fsrc2,fdest	Multiply four 8-bit pixels by four 16-bit constants
ALIGNADDR(L)	src1,src2,dest	Set up for unaligned access
FALIGNDATA	fsrc1,fsrc2,fdest	Align data from unaligned access
FZERO(S)	fdest	Fill fdest with zeroes
FONE(S)	fdest	Fill fdest with ones
FSRC(S)	fsrc,fdest	Copy fsrc to fdest
FNOT(S)	fsrc,fdest	Negate fsrc in fdest
Flogical(S)	fsrc1,fsrc2,fdest	Perform one of 10 logical operations (AND, OR, etc.)
FCMPcc16/32	fsrc1,fsrc2,dest	Perform four 16-bit or two 32-bit compares with results in dest
EDGE8/16/32(L)	src1,src2,dest	Edge boundary processing
PDIST	fsrc1,fsrc2,fdest	Pixel distance calculation
ARRAY8/16/32	src1,src2,dest	Convert 3D address to blocked byte address
SHUTDOWN	none	Prepare CPU for shutdown
PST	fsrc,[address]	Partial store
FLD,STF	[address],fdest	8- or 16-bit load/store to FP reg
QLDA	[address],dest	128-bit atomic load
BLD,BST	[address],dest	64-byte block load/store

Table 1. Summary of new UltraSparc instructions. (S) single-precision option; (L) little-endian option; "opt," "logical," and "cc" explained in text.

Graphics Implemented in FPU

The UltraSparc FPU includes two special graphics units. The designers chose to store graphics data in the FP registers for three reasons. First, graphics applications can use the integer registers for addresses, loop counts, and similar data, leaving all 32 FP registers for graphics data. Since many of these programs use a large number of registers, this is an important advantage.

Similarly, this decision makes better use of UltraSparc's instruction-issue rules, which permit only three integer instructions per cycle but allow four if at least one is an FP instruction. Graphics code can ideally issue two graphics instructions per cycle to the FPU along with one load or store and one integer arithmetic instruction, possibly for loop control or address generation.

Third, a few of the graphics instructions—multiplies, packs, and PDIST—take multiple cycles to execute, although they are fully pipelined and can be issued one per cycle. These instructions fit well within UltraSparc's FP pipeline, which handles three-cycle instructions such as FP adds and multiplies.

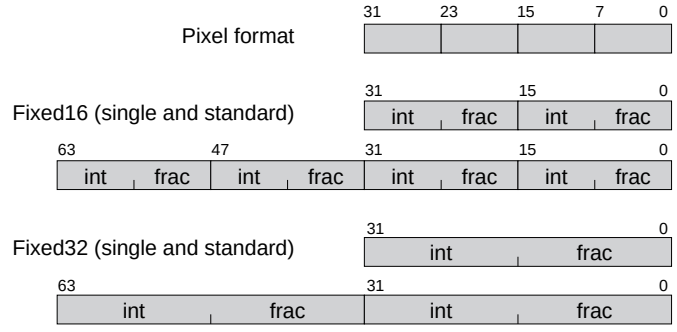

Figure 1. UltraSparc packs 8-, 16-, or 32-bit data into single-precision (32-bit) or standard (64-bit) floating-point registers.

The new graphics instructions, listed in Table 1, are optimized for short integer arithmetic. Graphics and video data are typically represented using 8-bit values (24-bit "true color" uses three 8-bit values for red, green, and blue). Audio data can be encoded as 8-, 12-, or 16-bit values, depending on the desired sound quality. High-end "CD-quality" audio uses 16-bit data. As with most processors, UltraSparc's 64-bit integer data path is wasted on these short values.

Figure 1 shows the new data formats implemented in the graphics units. The Fixed16 format can be used for 8-bit data, providing adequate precision and dynamic range for intermediate values. The Fixed32 format is available for operations on 12- or 16-bit data, or for 8-bit calculations, such as interpolation, that require higher resolution. A 32-bit pixel format, consisting of four 8-bit values, is used for graphics input and output data.

The EXPAND instruction converts 8-bit data to the Fixed16 format, which allows four bits at the high end for overflow and four bits at the low end for added resolution. This format avoids the need for all intermediate results to be rounded and clipped. The FPACK instructions converts Fixed16 or Fixed32 back to 8- or 16-bit data, scaling, rounding, and clipping as needed.

The new partial store (PST) instruction stores any number of bytes from a register without disturbing adjacent data, avoiding the need for an explicit read-modify-write operation. A single 8- or 16-bit datum can be loaded or stored using FLD and STF; these are useful if the pixel data being accessed is not contiguous.

One ALU Does Four Operations

Using 64-bit data paths, the graphics units can perform two calculations per cycle on Fixed32 data or four calculations per cycle on Fixed16 data, greatly increasing throughput over a standard processor. For example, the FPADD and FPSUB instructions perform partitioned addition or subtraction on these two data types. "Partitioned" means that each pair of components is added (or subtracted) separately; the carry from one add does not flow into the next, as Figure 2(a) shows.

Similarly, a set of compare instructions (FCMPcc) compares two or four sets of values in a single cycle. A full set of signed comparisons is available. These instructions place a 2- or 4-bit result into an integer register, with each bit corresponding to one comparison.

Because graphics data is kept in the FP registers, the standard logical instructions (AND, OR, etc.) cannot be used, as these operate on the integer registers. Thus, UltraSparc implements a full set of logical operations in the graphics unit as well. All 16 possible Boolean operations can be performed to combine two source registers. No partitioning is required, as these operations are performed on a bit-by-bit basis. As with FPADD and FPSUB, single-precision and standard versions are available.

Support for Windows NT

UltraSparc is designed to operate natively in either big- or little-endian modes, making it the first SPARC processor able to support the little-endian Windows NT. The designers added a few new instructions to assist with accessing data structures that may have been generated by x86 processors.

These data structures frequently contain unaligned data, which are not a problem for x86 processors. Like most RISCs, UltraSparc does not have hardware support for unaligned loads and stores. The following code sequence, however, performs an unaligned load:

```
ALIGNADDR   Addr, Offset, Addr   ;Calculate mask data
LDD         [Addr], %D0          ;Load lower bytes
LDD         [Addr+8], %D2        ;Load upper bytes
FALIGNDATA  %D0, %D2, %D4        ;Combine needed bytes
```

Note that these instructions operate on integer registers. The ALIGNADDRL instruction performs a similar function for data in little-endian format.

The add, subtract, and logical operations can be used to combine pixel data from different images, making objects appear transparent or opaque. In combination with the multiply instructions, FPADD is used for audio and image filtering.

Pixels Can Be Multiplied and Scaled

The FMUL instructions perform four partitioned multiply operations in parallel. Figure 2(b) shows the basic FMUL8x16 operation. The four 8-bit values in a single 32-bit pixel are each multiplied by a 16-bit multiplier; the four products are rounded and stored in Fixed16 format. Only the upper 16 bits of the product are stored. This operation is typically used to combine pixel data with constant filter coefficients. These constants can be

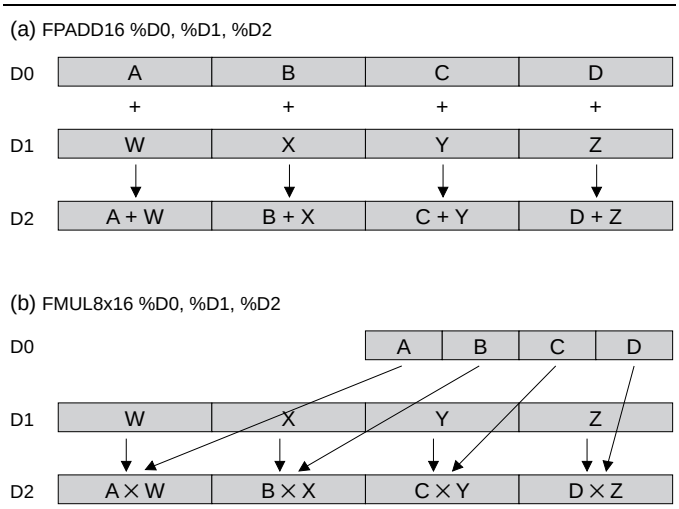

Figure 2. The partitioned addition (a) and multiplication (b) functions perform four parallel operations in a single cycle and generate four separate results in Fixed16 format.

For More Information

Sun has not announced pricing for the UltraSparc processor, which is expected to begin shipping in 3Q95. For more information on UltraSparc or its multimedia extensions, call SPARC Technology Business (Sunnyvale, Calif.) at 408.774.8119; fax 408.774.8537.

scaled ahead of time to allow scaling of the pixel data during the multiplication.

This operation has several variations. Instead of using a different coefficient for each multiply, the same multiplier can be used for all four calculations. The multiplier is typically an α (intensity) value used to scale the pixel data; it can be held in either the upper or lower 16 bits of a single-precision register. This instruction is also used for filter functions with fixed multipliers.

To avoid the cost of a 16-bit multiplier, the graphics unit includes only an 8-bit multiplier. UltraSparc includes instructions to synthesize a $16 \times 16 \rightarrow 16$-bit multiply on Fixed16 data:

```
FMUL8SUx16    %D0, %D1, %D2    ;Calculate upper bytes
FMUL8ULx16    %D0, %D1, %D3    ;Calculate lower bytes
FPADD16       %D2, %D3, %D4    ;Combine upper and lower
```

Because of the interlocks in this sequence, a 16-bit multiply has a five-cycle latency, although a new multiply can be issued every two cycles. Two "double" instructions (FMULD8SUx16 and FMULD8ULx16) can synthesize a $16 \times 16 \rightarrow 32$-bit multiply, with the output in Fixed32 format.

Highly Specialized Instructions

In addition to instructions for general filter functions and pixel operations, UltraSparc has some unique instructions aimed at very specific situations. One of the most complex is PDIST, which is used for motion estimation in video-compression schemes like MPEG *(see **080204.PDF**)*. The inner loop for motion estimation must compare 16×16 blocks of pixels to find the best match, a laborious process using a standard instruction set.

A single PDIST instruction compares two sets of eight 8-bit values in parallel, performing a partitioned subtraction of two 64-bit registers. The processor then takes the absolute value of each of the eight results, and finally computes the sum of the differences, adding it to the accumulated difference value. This instruction is similar to TriMedia's me8 operation *(see **081603.PDF**)* but handles twice as much data. Assuming all data is in registers, the inner loop for motion estimation can consist of a single PDIST instruction, compared with dozens of instructions in a standard processor.

Another useful instruction is PMERGE, which takes two sets of four bytes and interleaves them into a 64-bit value. This instruction can convert pixels from standard (αRGB,αRGB) format to packed (ααRRGGBB) format. It can also be used in discrete cosine transform (DCT) functions to transpose matrix values. DCT is a key part of most popular video-compression algorithms.

Many graphics rendering loops handle two, four, or even eight pixels at a time, speeding situations where a block of pixels has the same data. These loops, however, typically have complex, slower code to handle the edge of an object, which may occur in the middle of a block of pixels. UltraSparc's EDGE instructions compare the address of the edge with that of the current pixel block, calculating the appropriate byte mask. This mask can then be used by the PST instruction to store the proper bytes to memory. This instruction sequence eliminates the need for special-case code.

The ARRAY instructions address another problem. Visualization of a 3D data set requires displaying a 2D slice of arbitrary orientation. If the data set is stored linearly, many slices will have poor locality, reducing the performance of UltraSparc's cache. The ARRAY instructions convert a linear array address to a special blocked address. The conversion function hashes the address bits so each 64-byte block contains a contiguous group of pixels; this hashing ensures that, no matter how the slice is oriented, each cache line will contain an average of at least eight useful pixels. A second level of hashing improves the TLB hit rate for large arrays.

UltraSparc in a Class by Itself

UltraSparc's multimedia feature set requires some complex hardware to implement but, given the overall complexity of next-generation processors, even a large number of new instructions makes only a small increment to the design. Sun says that the extra multimedia hardware uses about 3% of the UltraSparc's 315-mm^2 die area, a modest increase.

Adding so many specialized instructions violates the strict RISC philosophy of leaving complex functions for the compiler to synthesize, making UltraSparc somewhat CISC-like. Sun's designers took care to avoid the common pitfalls of CISC designs. For example, they pipelined the new operations (including their instruction decoding) when necessary to avoid creating critical timing paths, and they kept the die-area impact to a minimum. If compression algorithms change, however, instructions such as PDIST may be less useful in the future.

Sun expects that the new instruction set will allow UltraSparc to decode MPEG-2 video at 30 frames per second and even perform video compression adequate for video conferencing. These tasks are well beyond the capabilities of current microprocessors, requiring special-purpose chips. In fact, it is doubtful that any other next-generation microprocessor will achieve these feats, putting UltraSparc in a class by itself. As multimedia becomes an integral part of general-purpose computing *(see **0816ED.PDF**)*, this will give Sun a badly needed edge. ♦

