V830 Gunning for Home Media Boxes

Saturating Arithmetic, Multiply-Accumulate, On-Chip RAM Aid DSP Tasks

by Jim Turley

In a previous issue (see MPR 6/2/97, p. 22), we covered NEC's newest V830 and V831 embedded microprocessors. This article examines the V830 family's instruction set more closely.

To meet the needs of emerging consumer-electronics devices (set-top boxes, Internet terminals, etc.), NEC's V830 architecture extends the capabilities of the earlier V810 and V850 chips in significant ways. It does this mostly through new single-cycle multiplication and multiply-accumulate instructions. The V830 also adds on-chip data and instruction RAM and a nonblocking burst-fill instruction, further enhancing the family's signal-processing capability.

Although it still bears a strong similarity to other V800 family members—warts and all—the V830 instruction set gives ARM, MIPS, and ColdFire some strong competition for the emerging class of media-processing microprocessors. NEC's "other" 32-bit embedded architecture may not have the third-party industry support of its better-known MIPS line, but in performance and capability, the V830 makes a strong showing. It's faster than ColdFire or ARM and has better signal processing than MIPS or SuperH.

V830 Bears Strong Similarity to V810, V850

Like NEC's V810 and V850 chips, the V830 uses a combination of 16-bit and 32-bit instructions, which may be freely intermixed. Most operations can be encoded in just 16 bits, which is unusual for an architecture with 32 general-purpose registers. Consequently, the V830 enjoys good code density, on a par with the 68K or SuperH. Despite the short instruction word, any instruction can use any register; there are no register limitations like those in Atmel's AVR family (see MPR 7/14/97, p. 10).

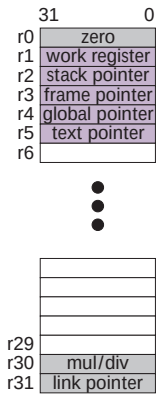

Figure 1. Only three of the V830's 32 registers are dedicated for particular uses; five more are reserved by software convention.

As Figure 1 shows, the V830 follows the popular RISC convention of hardwiring r0 to the value zero and dedicating a few other registers to the stack pointer and return-address (subroutine) pointer. Historically, V800 software relies on a few other registers, but these conventions are not binding.

Overall, the V830 is much like its predecessors but, as with Motorola's ColdFire, is not binary compatible with them. The V830 loses the V810's floating-point and bit-string instructions and the V850's bit-manipulation instructions, but it adds pipelined multiply-accumulate features. Programmers who are already familiar with earlier V800 generations will have little trouble getting accustomed to the V830.

Conditional Branches Must Be Short

The short instruction word squeezes the displacement of conditional branches to just 9 bits (8 significant bits plus a sign bit). Unconditional jumps (JR and JAL) have a 26-bit displacement when they're encoded in 32-bit form. The now-defunct V810 series used a similar format; the V850 takes only 22-bit offsets, which is sufficient for its 16M code space.

The small offset limits conditional branches to just 128 instructions (or fewer, depending on the number of intervening 32-bit instructions), which is fine for most loops but restricts function calls and major decision trees. In these cases, the sense of the conditional branch must be reversed, to skip over an unconditional jump.

Branch History Has Short Memory

The most unusual enhancement to the instruction set, shown in Table 1, is the conditional "advanced branch" (ABcc) instruction. This takes advantage of branch history to predict the outcome of the branch, a feature commonly found in high-end processors. Correctly predicted branches take only a single clock cycle to execute whether they're taken or not, versus three cycles for most taken branches.

What makes ABcc peculiar is the depth of its branch history: one bit. There is only one branch-history bit for the entire machine. This lone bit reflects the outcome of the most recent iteration of the most recent ABcc instruction. The V830 is not smart enough to know if the branch-history bit was set by the current ABcc instruction; each different ABcc instruction will use and then overwrite this bit.

Because the normal conditional-branch instructions (Bcc) cost three cycles if taken, using an ABcc makes the most sense at the bottom of a loop. It would repeat often enough for the two-cycle advantage to pay off, without intervening ABcc instructions to upset the delicate branch history.

That history, such as it is, is extremely fragile. Apart from being changed by every ABcc instruction, the branch-history bit is extinguished by any write into the instruction RAM (via the BILD instruction), the instruction cache, or the instruction tags.

Nested loops using ABcc are possible but defeat the purpose. Every time code falls through the inner loop, it resets the history bit and prevents accurate prediction of the outer loop. In such cases, only the innermost loop benefits from the ABcc instruction.

NEC's approach is unique. Other vendors have optimized loops with branch-likely instructions, such as the

Mnemonic	Description	Mnemonic	Description	Mnemonic	Description
Load/Store		**Arithmetic**		**Logical**	
LD.B	Load byte	ADD	Add with carry	**OR/I**	Logical OR/with immediate 16
LD.H	Load halfword (16 bits)	**ADDI**	Add with immediate 16	**AND/I**	Logical AND/with immediate 16
LD.W	Load word	**MOVEA**	Add register, immediate 16	**XOR/I**	Logical XOR/with immed 16
ST.B	Store byte	**MOVHI**	Add register, immediate 16	NOT	One's complement register
ST.H	Store halfword	SUB	Subtract with borrow	SHL	Logical shift left 1–32 bits
ST.W	Store word	SATADD3	Add three operands, saturating	SHR	Logical shift right 1–32 bits
IN.B	Load byte from I/O space	SATSUB3	Subtract three ops, saturating	SAR	Arithmetic shift right 1–32 bits
IN.H	Load halfword from I/O space	MUL	Multiply, signed	SHLD3	Shift left doubleword 1–32 bits
IN.W	Load word from I/O space	MULU	Multiply, unsigned	SHRD3	Shift right doubleword 1–32 bits
OUT.B	Store byte to I/O space	**MULI**	Multiply with immediate	CMP	Compare two registers
OUT.H	Store halfword to I/O space	MUL3	Multiply three operands	**CAXI**	Atomic compare and exchange
OUT.W	Store word to I/O space	MULT3 ●	MUL3 with truncate	SETF	Set flag on condition
MOV	Move register to register	**MACI** ●	Multiply-add with immediate	**Flow Control**	
BILD ●	Load 4 words to instruction RAM	MAC3 ●	Multiply-add three operands	JMP	Jump indirect via register
BDLD ●	Load 4 words to data RAM	MACT3 ●	MAC3 with truncate	**JR**	Jump relative
BIST ●	Store 4 words from instr RAM	DIV	Divide, signed	**JAL**	Subroutine call (jump and link)
BDST ●	Store 4 words from data RAM	DIVU	Divide, unsigned	Bcc	Branch on condition
LDSR	Load to system register	MIN3	Minimum of three operands	ABcc ●	Advanced branch on condition
STSR	Store from system register	MAX3	Maximum of three operands	BR	Branch always
Miscellaneous		**Miscellaneous**		TRAP	Software trap
EI/DI	Enable/disable interrupts	STBY	Enter Stop Mode	RETI	Return from interrupt
NOP	No operation	HALT	Enter Sleep Mode	BRKRET	Return from exception

Table 1. The V830 instruction set, while not binary-compatible with earlier V800 chips, is similar in many ways. All instructions can operate on any of the CPU's 32 general-purpose registers. • indicates new to the V830 family; **highlighted** instructions are 32 bits long.

MIPS BccL or i960's Bcc.T. These are compile-time optimizations that assume the branch will be taken more often than not. In practice, they would generally perform better than NEC's ABcc because ABcc always mispredicts the first and last iteration of a loop, while branch-likely instructions mispredict only the last iteration.

The one advantage ABcc has is that its prediction is dynamic. This could be valuable if the outcome of the branch is dependent on some condition that cannot be foreseen at compile time, such as an input variable.

Although its branch history is not as well endowed as that of bigger RISC processors, ABcc never performs worse than the normal Bcc instruction. Even if its prediction is incorrect, ABcc takes no longer than Bcc whether the branch is taken (three clocks) or not taken (one clock). In the worst case, the program is no worse off; used judiciously, ABcc can shave two clocks from nearly every loop iteration.

Some Immediate Forms, If They're Small

The most basic operations (e.g., add, shift, move) accept either two registers or a register and a 5-bit immediate value (the 5-bit immediate taking the place of the second register specifier). The sign-extended 5-bit immediate, of course, allows values only from –16 to 15. Anything other than that must be loaded into a register first. This method is carried over from previous incarnations of the V800 family and is typical of RISC chips.

A few arithmetic instructions (ADDI, MULI, MACI, and MOVHI) and logic instructions (ORI, ANDI, and XORI) can take bigger immediate values by extending the instruction to 32 bits, with the extra 16 bits devoted to a signed immediate. Since the base instruction word still has room for two registers, these instructions also have the nice side effect of being nondestructive.

Like any RISC design, there are no instructions that allow 32-bit immediate values or that load 32-bit constants. For values larger than ±32,767, the programmer must rely on register-to-register operations. To load a register, the programmer must use the MOVHI instruction created for the purpose. MOVHI is like ADDI, except that it adds a 16-bit immediate to the upper half of the source register before depositing the result in the destination register, and it doesn't affect the status flags. If the source register already holds the low-order half of the desired value (loaded with MOVEA), MOVHI can fold in the upper half in a single operation.

A Multiplicity of Multiplication Options

Previous generations of NEC's V800 family had saturating addition and subtraction, but only the V830 does saturating multiplication. (Saturating multiplies are among the enhancements in MMX, MDMX, and Piccolo, and they have long been a feature of SuperH processors.) As Table 2 shows, the instruction set supports saturating multiplication, nondestructive (using three operands) multiplication, and multiplication with 64-bit results, but never all of these at once.

The basic 32 × 32-bit multiply instruction is MUL. It multiplies any two registers and stores a signed 64-bit result, with the upper half always going into r30. Thus, MUL is a

	Saturates	Truncates	64-bit result	Three-operand	Nondestructive	16-bit immediate	Updates flags
MUL			●				●
MULU			●				●
MULI	●				●	●	●
MUL3	●			●	●		●
MULT3		●		●	●		●
MACI	●					●	●
MAC3	●			●	●		●
MACT3	●	●		●	●		●
MOVEA					●	●	
MOVHI					●	●	
SATADD3	●			●	●		●
SATSUB3	●			●	●		●

Table 2. The V830 includes several permutations of multiply and multiply-accumulate that offer 32- or 64-bit precision, saturation, truncation, or nondestructive operand addressing. There are also two forms of saturating addition and subtraction.

destructive, extended-precision operation useful for general-purpose arithmetic. MULU is essentially the same, but treats the sources as unsigned operands. MUL3 is the saturating, three-operand version of MUL (there is no unsigned MUL3).

By extending the instruction to 32 bits and adding an immediate multiplier, MULI performs a $32 \times 16 \rightarrow 32$-bit operation. Like ADDI, MULI is nondestructive, taking both a source and a destination register. It also saturates: if the result overflows 32 bits, the destination register is set to its largest (or smallest) extent and the saturation flag is set.

Finally, MULT3 offers three-operand addressing for bigger numbers. The lower 32 bits of the product are discarded, and only the upper half is stored in the destination register. Rather than saturating, MULT3 truncates (hence the mnemonic) the result. Obviously, MULT3 is useful only when the results of the multiplication will significantly overflow 32 bits. This is useful for handling scaled (Q-format) numbers, where digits to the left of the "binary point" are significant.

Multiply performance is middling for a RISC design but far ahead of chips from the 68K or x86 families. Both MULI and MUL3 take three cycles to execute, with a single-cycle repeat rate. Oddly, MUL and MULU are one clock slower, with four-cycle latency and two-cycle throughput. Many RISCs (e.g., MIPS, PowerPC) have either single-cycle hardware multipliers or, like ARM, have iterative multipliers that take 10–15 cycles for 32×32-bit operations.

Saturating MUL, MAC Intended for Video

NEC's other change for media processing is a set of multiply-accumulate (MAC) instructions. These are, for the most part, orthogonal with the multiply instructions: MACI works like MULI, MAC3 like MUL3, and MACT3 like MULT3. The only difference among these three pairs is whether the destination is overwritten or accumulated. The MULI and MACI instructions are a convenient way to accumulate results with a constant scaling factor, a common construct when scaling the brightness or size of a polygon.

Of all the multiplication operations, only MUL3 and MULT3 (and their accumulating counterparts, MAC3 and MACT3) are nondestructive, but they also saturate/truncate the result, so no 64-bit products are available. Conversely, MUL and MULU support 64-bit precision, but both are destructive, and MULI is both destructive and saturating.

For general control-oriented arithmetic, MUL and MULU will be used most; the saturating MUL3, MULT3, and MULI are clearly intended for pixel and sound processing, where their overflow handling is welcome.

On the whole, the V830's multiplying, accumulating, and saturating instructions are not much different from those of most SuperH chips or the media-processing extensions from ARM (Piccolo), MIPS (MDMX), or Intel (MMX). While the details might differ, every vendor seems to agree on the basic feature set required.

Unlike most low-cost chips—and even some expensive RISC processors—the V830 sports DIV and DIVU instructions for signed and unsigned 32-bit division. At 35–37 cycles each, and no pipelining, these instructions are not particularly speedy, but they're still much faster and more convenient than calculating quotients and remainders in software. SuperH chips use a software-controlled iterative process for division that delivers exactly the same 37-cycle latency as the V830. ColdFire, MIPS, and i960 have no divide support at all.

Odd Interlocks Stall Flag Operations

Apart from their tepid performance, MUL and MULU also need an extra clock cycle to update the CPU's status flags. If a subsequent instruction (usually a conditional branch) depends on the flags, the chip will stall while the flags are forwarded from the ALU. Worse, all of the logical operations (OR, ORI, AND, XOR, SHL, etc.) suffer from the same problem. This is particularly onerous because one of these instructions will often precede a conditional branch. None of the other arithmetic operations suffers from this interlock problem.

Another peculiar interlock exists for all the multiply-accumulate instructions: unless the destination register has been used recently (within the last three clock cycles), the MAC operation will stall for a cycle. This is because MAC operations, unlike others that use the multiplier, use the destination register as a source, and the V830 needs an extra clock to fetch its contents. If that register has been updated recently, however, its contents are already "in transit" (in the store queue) and can be routed to the multiplier one clock sooner.

MAC and On-Chip RAM Make DSP Loops Hum

Since the new MAC instructions are clearly intended for media processing, it's important that these pipeline well in tight pseudo-DSP loops, and the V830 does not disappoint. All multiply and multiply-accumulate instructions (except

MUL and MULU, as noted previously) have a one-cycle repeat rate. At 100 MHz, this allows the chips to run at 200 MOPS under ideal circumstances. With their on-chip data RAM, the V830 and V831 are equipped to keep conditions nearly ideal. Memory latency would normally kill a data-intensive loop of repetitive MAC instructions once it got under way. But with single-cycle access to a 4K block of on-chip RAM, the V830 and V831 should be able to maintain a steady flow of coefficients and scaling factors under most circumstances.

Here is where the BDLD and BILD instructions come into play. They initiate burst loads of the data or instruction RAM, respectively. Although the actual latency of these operations will depend on memory speed, and will certainly be several dozen cycles, they do not stall the processor. This allows the CPU to process data already in on-chip RAM while new data is loaded from external memory. Given a big enough instruction loop or quick enough external memory, a new BDLD can preload the data RAM on each iteration through the loop without stalling.

Lots of Saturating, Nondestructive Operations

A surprising number of instructions are nondestructive, given the V830's cramped opcode map. The MAX3 and MIN3 pair are nondestructive; the result of the comparison is stored in a third register.

The saturating arithmetic instructions SATADD3 and SATSUB3 are like MUL3: nondestructive, three-operand, and saturating at 32 bits. These two operations are vital for pixel manipulation, where adding colors improperly can result in "whiter than white" becoming black, and vice versa.

In addition to the usual 32-bit shift and rotate instructions, the V830 also has a pair of 64-bit shifts, SHLD3 and SHRD3. These work by concatenating any two registers (they need not be contiguous) and shifting out of one into the other. This is useful not for creating 64-bit patterns but for controlling which bits are shifted into the destination register. Normally executing in a single clock cycle, these instructions are susceptible to the same peculiar destination-register interlock as the MAC instructions.

The CAXI (compare and exchange, interlocked) instruction is the sole read-modify-write operation, used for semaphore synchronization in multiprocessor systems or (as is more likely in the V830's case) single-processor systems that run multitasking operating systems. CAXI loads a word from memory and compares it with a specified data register. If all 32 bits match, the chip stores r30 in its place; otherwise, memory is unchanged. When it's all over, CAXI has loaded and possibly updated a shared memory location, such as a counter or ownership flag. Such indivisible bus operations are vital in systems with shared resources (such as a frame buffer or DMA controller), but they are often not provided in newer processors, pushing the burden onto the hardware designer. Motorola's 68K processors have always had similar instructions, although this (like division) was cut when creating ColdFire.

For More Information

More information on NEC's V830 family can be obtained from NEC (Santa Clara, Calif) at 408.588.6000 or by visiting *www.nec.com/products/products.html*.

A Strong Competitor for Home Media

In all, the V830 is a worthwhile addition to NEC's V800 line, adding signal-processing capability while keeping the old values of the V800 family (tight code, regular register set).

Although the V830 core is well equipped for moderately taxing digital-signal processing tasks, it's not perfect. With just one addressing mode (register-indirect with 16-bit offset), the V830 lacks such niceties as postincremented or predecremented addressing. Nor can the V830 do circular addressing, a powerful DSP capability that the third-generation ColdFire (see MPR 9/16/96, p. 1) and most DSPs provide.

The V830 has no stack-management instructions or addressing modes, although r2 is nominally reserved as its stack pointer. Nesting subroutines and passing parameters among them is the programmer's problem. (In fact, there's no explicit return-from-subroutine instruction; code must jump indirectly through r31.) And the V830 has no memory-reference instructions; all data must pass through its registers.

The enhancements to the V830 are designed more for the signal-processing work of a network or modem connection, or for sound enhancement, than for pixel processing. For example, there are no SIMD instructions that operate on multiple pixels simultaneously, like those provided by MMX, VIS, or MDMX. Nor is there an FPU for high-precision work. In sum, the V830 is still a microcontroller at heart: a single-chip processor for low-cost, high-volume consumer applications with moderate processing demands, such as a set-top box.

For about $25, a 100-MHz V830 delivers a strong combination of integer performance and pseudo-DSP performance. ColdFire chips are cheaper, but also slower and tailored for 16-bit multiplication. Embedded MIPS parts are just as fast, but none so far incorporates the MDMX enhancements. Hitachi's upcoming SH-4 enjoys even better code density than the V830, but SH-4 is still several months away. Unlike the V830, SH-4's strengths are in 3D geometry setup, not in signal processing. In fact, the V830 and SH-4 are almost polar opposites in that regard. At the other end lies Hitachi's SH-DSP (see MPR 12/4/95, p. 10), which has true signal-processing capability at the expense of some code density and binary compatibility with other SuperH parts.

For network-enabled televisions or low-cost NCs, the V830 is an ideal choice: inexpensive, well equipped, and fast enough to handle HTML decoding and soft-modem functions with tolerable speed. In an industry marked by constant change, NEC has shown that even its nearly forgotten V800 family can change, becoming a strong competitor in an ever more crowded field. M