NEC Joins Embedded Processor Plethora

High-End V800 Architecture Takes Yet Another RISC-Like Approach

by Brian Case

*Earlier this year, NEC began shipping its V810 microprocessor, the first implementation of the new V800 architecture (see **070802.PDF**). The V800 architecture is aimed at consumer electronics applications, especially battery-operated devices ranging from video games to PDAs, as well as at the broader embedded control market. In this article, we review the architecture itself. See the sidebar "NEC's Highly Integrated V820," for details on the latest implementation.*

The 32-bit V800 shares many characteristics with conventional RISC architectures. It has a register-oriented instruction set and a load-store architecture, and most instructions facilitate a fast, simple pipeline.

The V800 also, however, deviates significantly from standard RISC design tenets. As with Hitachi's SH7000 architecture *(see **071103.PDF**)*, some of the non-RISC attributes stem from the desire to achieve high code density due to the needs of PDAs and other physically-compact, high-end embedded applications.

The V800 does not have a supervisor processor mode or instruction-set support for an MMU. NEC says the MMU was excluded to reduce power consumption. These omissions may also be reflections of most of the target market, where the need for them is not seen as compelling, but an MMU is needed for PDA operating systems such as PenPoint and Newton. Note that many of the more established V800 competitors provide protection and address mapping of some sort.

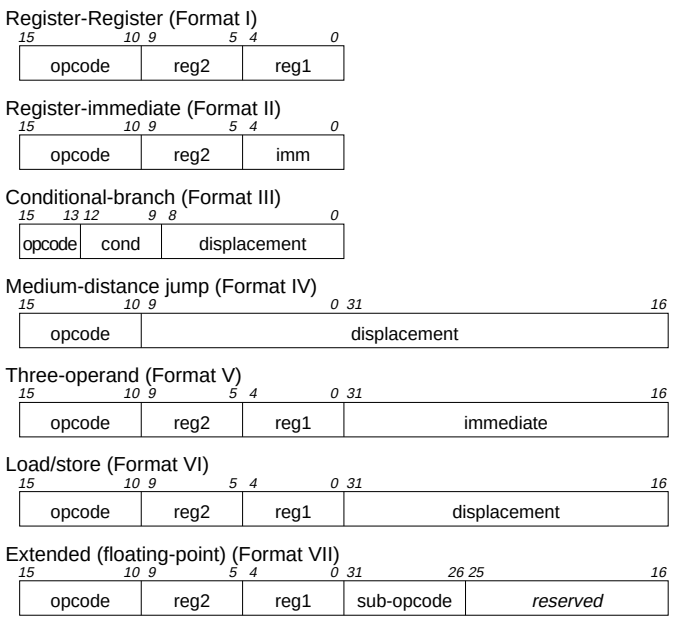


Figure 1. V800 instruction formats include both 16- and 32-bit versions, but most instructions use the 16-bit format.

Variable Instruction Length

One of the V800's non-RISC attributes is that it has two instruction lengths. Most of the V800 instruction set—59 out of 89 opcodes—is encoded in a 16-bit instruction format. A 32-bit format encodes the remaining one-third of the instruction set. Figure 1 shows the instruction formats, which are consistent and easy to decode. Most of the 32-bit formats are used for instructions requiring large constants, but the floating-point instructions use the opcode field in the extended format (format VII), which wastes 10 bits. This format may have been chosen to allow future expansion of the floating-point instruction set to support additional operand precisions.

Like most RISCs, the V800 architecture defines a register file with 32, 32-bit general-purpose registers with register number zero always reading as zero. Unlike most other RISCs that have floating-point instructions, there is no separate floating-point register file.

Some of the general registers have special purposes. As with many other architectures, a few are reserved by software convention. Unlike other RISCs, a few are implicit operands for the bit-string, multiply, and divide instructions. Even though these registers have special uses, they can be used for other purposes between execution of these instructions.

One curious aspect of the V800 architecture is the lack of three-operand register-to-register operations. Clearly, there is insufficient room to encode them in the 16-bit format, but given that the 32-bit format is available and that it has 10 unused bits, it would have been easy to add three-operand instructions. As with the MMU, NEC says power consumption motivated the exclusion of three-address instructions. Of course, a sequence of two, two-address instructions performs the same function and consumes the same number (four) of instruction bytes, but a single, 32-bit three-address instruction would be twice as fast (one cycle vs. two).

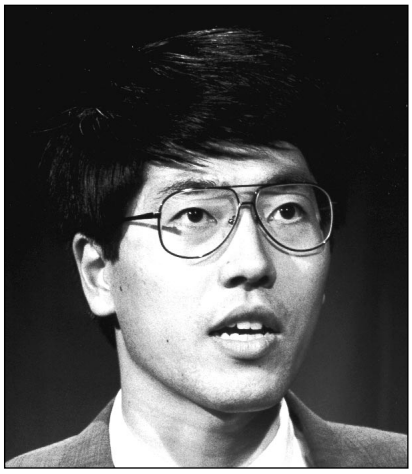
Loads and stores are available for bytes, halfwords, and words, and the memory addressing model is little endian. The single addressing mode adds a signed 16-bit offset to a register.

The conditional-branch architecture uses condition

codes, which are slightly advantageous in that they can save programs a few explicit compare instructions because the condition codes are set as a side-effect of many data manipulation instructions. The downside is they require more complex compiler algorithms and prevent early computation of branch conditions.

Register Usage Conventions

The V800 register set has some hardware and software usage conventions. Software conventions reserve R1 through R5. R1 is used as a temporary for address generation during the evaluation of complex addressing expressions. R2 is reserved for the interrupt stack pointer, while the program stack pointer is kept in R3. R4 points to global variables in memory. R5 is a pointer to static data. In most applications, the uses of R4 and R5 could be combined since neither is likely to be larger than 64K (the largest area directly addressable with a single 16-bit displacement load or store instruction).



Hitoshi Yamahata of NEC discusses the V800 architecture at the recent Microprocessor Forum.

At the other end of the register file, R26 through R31 are reserved by hardware conventions. R26 through R30 are used by the bit-string instructions, and their use is described below. The JAL instruction stores its return address in R31, the link pointer.

V800 Has RISC-Like Instruction Set

For the most part, the V800 instruction set is simple and meets the requirements of a pipelined, one-instruction-per-cycle implementation. The major exception is the group of bit-string instructions, which specify their register operands implicitly and do not fit the RISC model of single-cycle execution.

Integer arithmetic and logical operations use both the 16-bit and 32-bit instruction formats. Addition is available in register-to-register and register-immediate forms; register-immediate subtraction is performed by adding a negative constant. The two-operand nature of these instructions is a major limitation.

There are four add mnemonics but five add instructions: ADD can stand for either register-to-register (format I) or register-plus-5-bit-immediate (format II). The other instruction with add in its name, ADDI, adds a sign-extended sixteen-bit immediate to a register. These three instructions set the condition codes.

Two other instructions, MOVEA and MOVEHI, also perform addition but do not set the condition codes. MOVEA adds a sign-extended 16-bit immediate value to a register, while MOVEHI adds a 16-bit immediate value shifted left by sixteen bits to a register.

The V800 supplies basic logical instructions: AND, OR, XOR, and NOT. The first three can be either register-to-register (format I) or register-with-16-bit-immediate (format V). For these instructions, the 16-bit constant is zero-extended. The V800 also includes variable shift instructions to take advantage of a barrel shifter.

Unlike some other architectures aimed at embedded control, and even some workstation architectures, the V800 provides integer multiply and divide support with atomic instructions. These instructions can treat their operands as either signed or unsigned, and they both provide double-precision (64-bit) results.

One of the unusual aspects of the V800 architecture is the inclusion of floating-point instructions. Only single-precision operands are supported, but since the instruction mnemonics specify single-precision explicitly (".s" in Table 1), it seems that an upgrade path to higher precisions was contemplated. This is consistent with the use of format VII for floating-point instructions.

Floating-Point Supported

The floating-point instruction set is basic. A compact set of eight instructions is provided: add, subtract, multiply, divide, compare, truncate, and convert to/from integer. The compare instruction sets the carry, sign, and zero condition code bits, so normal integer conditional-branch instructions can be used.

Load and store instructions are straightforward. Byte, halfword, and word-length data can be loaded or stored, and loads always sign-extend bytes and halfwords. Loading unsigned data requires an extra instruction to mask off the sign extension. The only addressing mode is register-plus-displacement, using format VI, and the 16-bit displacement is sign extended. As with other RISC architectures, the most common addressing modes can be synthesized by using a zero displacement or by specifying R0 as the base.

A unique feature of the V800, compared with other 32-bit embedded architectures, is the group of input/output instructions for accessing I/O devices. These operate exactly like load and store instructions—same format, semantics, and addressing modes—but they activate a slightly different bus cycle. These instructions simply access a separate address space, and their function could as easily be implemented in system hardware by decoding a chunk of memory space as I/O addresses and using normal loads and stores. There might have been some advantage to these instructions if the V800 had a privileged supervisor processor mode and IN and OUT were legal only in supervisor mode, but this is not the case.

Program Control Has Strange Quirk

The V800 has non-delayed jumps and branches. An unconditional register-indirect jump is available using format I. Unconditional PC-relative jump (JR) and call (JAL) are provided with 26-bit, sign-extended displacements using format IV.

The conditional branches all use format III. There is no subroutine call with the short, 9-bit displacement. Since the V800 uses condition codes, the enumeration of possible branch conditions is in the branch instruction opcode, so there are sixteen different conditional branch instructions including NOP.

An explicit compare instruction performs subtraction and sets the condition codes without writing a result to the register file. Other RISCs with condition codes and three-address operations use a standard subtract instruction that specifies R0 as the result.

One problem with condition-code architectures is that creating a boolean (true or false value) in a register from the result of a comparison usually requires a tedious and pipeline-unfriendly sequence of conditional branches. The V800 solves the problem with the SETF instruction. This instruction uses format II to encode a destination register and a four-bit condition field, just like the one in the conditional branches. If the condition specified matches the condition codes, the destination register is set to one; otherwise, the register is cleared.

A small blunder in the architecture is the fact that the displacement fields in all the PC-relative branches are not shifted left one bit before being added to the PC. Since the instruction quantum of the V800 is 16 bits, the least-significant bit of the PC is always zero to force halfword alignment of instructions. When the V800 adds the PC and the displacement, it simply masks the low bit of the displacement to zero to avoid the possibility of jumping to an odd instruction address. The effect of this is to reduce the size of the offset by one bit. Other RISC architectures take care to squeeze as much range from displacements as possible by shifting the displacement left appropriately (one bit would be correct for the V800, two bits for most other RISCs).

Uncommon Bit-String Operations

The bit-string instructions form the most unusual group in the V800 instruction set. These instructions use several general registers as implicit operands:

- Bit-string source word address (R30)
- Bit-string destination word address (R29)
- Bit-string length (R28)
- Bit-string offset in source word (R27)
- Bit-string offset in destination word (R26)

Thus, these instructions can specify two bit strings starting anywhere in memory and of any length up to 4G bits.

The instruction set includes transferring bit strings without and with logical operations and searching for either the first zero or the first one moving either upwards or downwards. The bit-string transfer instructions all operate in the upward direction. Interestingly, a more complete set of logical operations is available for bit strings than for simple register operands.

It is difficult to imagine an application that would be possible with the bit-string instructions but impossi-

Bit String		
sch0bsu, sch0bsd	II	Find first 0 (up/down)
sch1bsu,sch1bsd	II	Find first 1 (up/down)
movbsu	II	Bit string transfer
notbsu	II	Bit string trans. w/ NOT
andbsu	II	Bit string trans. w/ AND
andnbsu	II	Bit string trans. w/ ANDNOT
orbsu	II	Bit string trans. w/ OR
ornbsu	II	Bit string trans. w/ ORNOT
xorbsu	II	Bit string trans. w/ XOR
xornbsu	II	Bit string trans. w/ XORNOT
Logical, Shift		
or, and, xor, not	I	OR, AND, XOR, NOT
ori, andi, xori	V	OR, AND, XOR immediate
shl, shr	I, II	Logical left/right shift
sar	I, II	Arithmetic right shfit
Program Control		
jmp	I	Register indirect jump
jr, jal	IV	PC-relative jump/call
bc, bl	III	Branch if carry/lower
be, bz	III	Branch if equal/zero
bge	III	Branch if greater or equal
bgt, bh	III	Branch if greater/higher
ble	III	Branch if less or equal
blt, bn	III	Branch if less/negative
bnc, bnl	III	Branch if no carry/not lower
bne, bnz	III	Branch if not equal/not zero
bnh, bnv	III	Branch if not higher/no overflow
bp, bv	III	Branch if positive/overflow
br, nop	III	Branch always/never
Load/Store		
ld.b, ld.h, ld.w	VI	Load byte/halfword/word
st.b, st.h, st.w	VI	Store byte/halfword/word
Input/Output		
in.b, in.h, in.w	VI	Port input byte/halfword/word
out.b, out.h, out.w	VI	Port output byte/halfword/word
Arithmetic		
mov	I, II	Data transfer
movhi	V	Add left-shifted-immediate
add, cmp	I, II	Add/Compare
addi	V	Add 16-bit immediate
movea	V	Add 16-bit imm., no cond. codes
sub	I	Subtract
mul, mulu	I	Signed/Unsigned multiply
div, divu	I	Signed/Unsigned divide
setf	II	Set register based on condition
Floating Point		
cmpf.s	VII	Compare
cvt.ws	VII	Convert from integer (toward 0)
cvt.sw, trunc.sw	VII	Convert/Truncate to integer
addf.s, subf.s	VII	Add/Subtract
mulf.s, divf.s	VII	Multiply/Divide
Special		
ldsr	II	Load into system register
stsr	II	Store system register
trap	II	Software trap
reti	II	Return from trap, interrupt
caxi	VI	Multiprocessor synchronization
halt	II	Halt

Table 1. V800 instruction set. The instruction formats, corresponding to the designations in Figure 1, are listed to the right of the mnemonic.

NEC's Highly Integrated V820

NEC has expanded its V800 family to three products with the announcement of the V820, the most highly integrated family member. The first two V800 chips are the 810—which has a CPU, FPU, and 1K of cache—and the 805, a derivative of the 810 with the system-bus width reduced from 32 to 16 bits *(see **070802.PDF**)*. The new chip uses the same core CPU as the other family members but adds a number of on-chip peripherals.

The V820 includes all of the features of the V810 and adds a DMA controller, memory controller, timer, PLL, and two serial ports. The DMA unit has four programmable channels and can handle the full 32-bit address range. The memory controller can be programmed to handle various types and speeds of DRAM. The 16-bit timer has three compare registers that can generate interrupts. The PLL generates a 6× clock signal, allowing the use of a 3–4 MHz clock input for typical operating frequencies. The two serial ports can handle synchronous or asynchronous protocols.

NEC rates the chip at 18 MIPS at its peak speed of 25 MHz, the same as the V810. Unlike the 810, however, the 820 is offered only at 5 V, although the company says it is "working to develop" a low-voltage version. While this performance is impressive compared with similar processors *(see **071403.PDF**)*, power consumption at 5 V is 750 mW; other processors achieve a far lower power rating by operating at 3.3 V. The V820 uses a 208-pin PQFP.

The V820 is manufactured in a 0.8-micron CMOS process similar to that used for the other family members, but the 820 requires a third metal layer not used by the other chips. The new peripherals increase the transistor count to 380,000, about 140,000 more than used by the V810. This, in turn, more than doubles the die area to 114 mm^2. The MPR Cost Model *(see **071004.PDF**)* estimates the manufacturing cost of the V820 to be about $35, three times that of the V810 and 60% more than the estimated cost of Hitachi's SH7032, which includes a similar set of on-chip peripherals.

While NEC is aggressively pricing the V805 at $28 in high volumes (100K or more), the V820 is priced at $80 in similar volumes. The price reflects the higher manufacturing cost, but it is more than twice the price of the SH7032. It is more expensive than even complete system solutions such as the Hobbit and Polar chip sets. The V820, while offering some peripheral integration, does not include interfaces, such as graphics and PCMCIA, that are important in portable system applications.

NEC says that it is aiming its V800 family at PDAs as well as embedded applications. The company has yet to deliver a complete PDA chip set such as Polar or Hobbit, nor has it identified any existing PDA operating system that can run on the V800 chips, none of which includes an MMU. The V800 has been more successful in the embedded market, with a key design win in Nintendo's CD-ROM player. While the V805 and V810 offer excellent price/performance for a variety of embedded applications, the V820 may struggle to find its market niche.

—LG

ble without them. The V800 architecture would be far better if the bit-string instructions were replaced with useful primitives that adhere to the RISC style of architecture design. The RISC primitives would fit into the natural pipeline flow, not require implicit register operands, and eliminate the need for microcode or hardware sequencing that the bit-string instructions need. A short routine using the correct primitives would probably operate just as fast as the bit-string instructions and would provide opportunities for customization and performance tuning as appropriate. The ARM architecture, with its powerful composite shift/arithmetic operations, comes close to supplying the suggested flexible primitives.

One possible benefit to the string instructions is lower power dissipation. If a program needs to perform many bit-string manipulations, it may save power to fetch only a single bit-string instruction instead of a loop full of RISC-adherent primitives. Given that the V800 has an instruction cache, the power savings is probably not large. The special group of instructions includes the usual operations for accessing system registers, causing a software trap, returning from a trap or interrupt, and halting the processor.

One instruction provided by the V800, CAXI, is unusual in an embedded control architecture. CAXI is an uninterruptible multiprocessor synchronization primitive. The range of current and future embedded control applications that require multiple processors seems limited at best; it is unlikely that multiple processors, which take more space and design time, would be better than a faster single processor. Perhaps the V800 designers had a captive, known application within NEC in mind, but without knowledge of this application, the inclusion of multiprocessor support without a supervisor mode and MMU does not make sense.

Special Registers Provide Unique Functions

The V800 special registers provide some typical and some not-so-typical functions. Typical functions include saving critical state on traps and interrupts, which is handled by the EIPC/EIPSW and FEPC/FEPSW register pairs. The two sets of registers allow the V800 to handle one level of nested interrupts. For certain classes of exceptions, the ECR contains information to help software determine what exception occurred.

The PSW contains routine information such as the condition codes, exception indicators, interrupt mask, interrupt disable bit, and address trap enable bit. The PIR (processor ID register) stores a fixed code for each implementation of the V800 family. The V810, for example, has the pattern 8-1-0 hexadecimal.

The TKCW (task control word) register should probably be renamed something like floating-point-exception control word. The bits in this register enable

the various floating-point exceptions. In the V810 implementation, these bits have fixed values.

The ADTRE special register holds a 32-bit address that is continuously compared to the PC. If the AE bit in the PSW is set and the PC matches the ADTRE, an exception is generated. This provides a hardware breakpoint that can greatly aid software debugging when the on-chip instruction cache is activated.

The most unusual special feature allows some or all of the cache to be cleared, loaded, or stored to memory simply by writing to the CHCW (cache control word). When clearing the cache, the number of entries is specified by CEC (clear-entry count), while the 12-bit CEN (clear-entry number) specifies the first entry to be cleared. The V810 cache uses eight-byte entries; thus, if the same cache structure is carried throughout the family, the CHCW appears to accommodate up to a 32K instruction cache.

When dumping or restoring the cache from memory, the SA field specifies the upper 24 bits of the full 32-bit memory address; the lower 8 bits are zero. Dump and restore were included to improve cache testability. In the V810 implementation, interrupts are held pending until the completion of an ongoing instruction cache clear, dump, or restore operation.

V800 Is A Worthy Competitor

Despite its various shortcomings, the V800 architecture is clean and simple enough to allow a small, fast, pipelined implementation. It makes some major departures from the RISC tenets, namely complex bit-string instructions, lack of three-address operations, and mixing 16- and 32-bit instructions, but the instruction set semantics are generally very RISC-like.

Architectural advantages include the simple addressing modes, large register file, floating-point instructions, good shift support, excellent multiply/divide support, and good code density due to 16-bit instructions. The bit-string instructions should probably be considered an advantage in the architectural sense, because they provide powerful functions, but the architectural benefit is small compared to the implementation complications both now and for future family members.

The V800 achieves high code density without sacrificing register file size by using 16- and 32-bit instruction lengths. This is in contrast to the SH7000, which limits itself to only 16 registers so that all operations can be encoded in its single 16-bit instruction format. The ARM achieves high code density by encoding an extremely rich set of operations in 32-bit instructions, but the richness restricts it to only 16 registers, as well. The V800's extra registers generally will result in fewer memory references. Specifically, they can be used to eliminate memory references entirely when calling typical leaf procedures.

Thus, the V800's two instruction lengths lead to an architectural advantage, but as with the bit-string instructions, a disadvantage is also created. The two lengths are probably of little consequence in current single-scalar implementations, but if the family is successful, superscalar implementations of the V800 could be more complex and costly to design than superscalar implementations of simpler architectures.

Shift support is on par with most other high-end embedded microprocessors and much better than the SH7000, which has only a few shift instructions and none that shift by a variable amount.

The floating-point instruction set could prove to be a competitive advantage for some applications. Some other embedded control architectures, such as the SH7000 and the ARM, do not have floating point instructions at all. On the other hand, the cost/performance of the floating point will be important. If performance is compromised to save die area, the presence of floating-point instructions may not prove compelling.

Since hardware support for single-precision floating-point is less expensive than that for double-precision, however, V800 implementations may not have to sacrifice too much performance. The simple fact that the first V800 microprocessor, the V810, has on-chip floating-point gives it an advantage over other microprocessors that require a separate chip or simply don't offer FP support.

PDAs using leading operating systems, as well as some applications in the high-end embedded control market, require memory management and a separate, protected supervisor mode. Even those applications that do not presently require these features may someday become sophisticated enough to benefit from them. Thus, the V800 suffers along with the SH7000 in comparison to nearly every other high-end embedded control architecture. As with the SH7000, the V800 could be easily modified to include these capabilities in the future.

The V800 is a worthy competitor in the embedded control market. As with all microprocessors, success will depend more on the volumes generated by design wins than on processor architecture. While it is true that a good architecture can help in securing customers, the design win in the HP LaserJet IV, which catapulted the 960 into the embedded RISC lead, was almost certainly not based on architectural superiority.

Still, it is worth noting that NEC, which already makes a wide range of MIPS-architecture chips, including the impressive R4200, was able to justify investing in the V800 family. While the MIPS architecture satisfies all functional needs—it has excellent memory management, protection, floating-point, etc.—it does not have great code density, a small processor core, or emphasize low power for battery-operated applications. Even with the clever combined integer/floating-point data path of the R4200, the V800 will have a smaller, more power-miserly implementation. ♦

