M•Core for the Portable Millennium

Motorola Fills Void in Low-Power 32-Bit Embedded Cores

by Peter Song

In a previous issue (see MPR 10/27/97, p. 12), we covered M•Core, Motorola's latest 32-bit architecture for embedded applications. This article examines the M•Core instruction set more closely.

M•Core joins the breed of 32-bit architectures that rely on 16-bit instructions to improve code density while compromising on features and programming flexibility. Like most others in this breed, it uses RISC principles—such as many registers and load/store instructions—to simplify design and improve performance. Similarly, it could not fit many of the programming-convenience features—such as three-operand instructions and large immediate fields—into 16 bits, requiring extra instructions to make up for these deficiencies. Unlike most others in this breed, however, it is optimized for microcontroller applications, offering more instructions and features while delivering better code density than the others.

Unlike Motorola's PowerPC and ColdFire, M•Core is designed from scratch, allowing complete freedom in every aspect of its architecture definition. It is a result of a customer-driven initiative and fills a void in Motorola's product line for a 32-bit CPU with very low power consumption. Although much of the power savings in a processor comes from a better process and a lower supply voltage, M•Core is simpler to design than Motorola's other embedded cores. It allows Motorola to deliver better MIPS/W earlier to the market than Motorola's other instruction sets could.

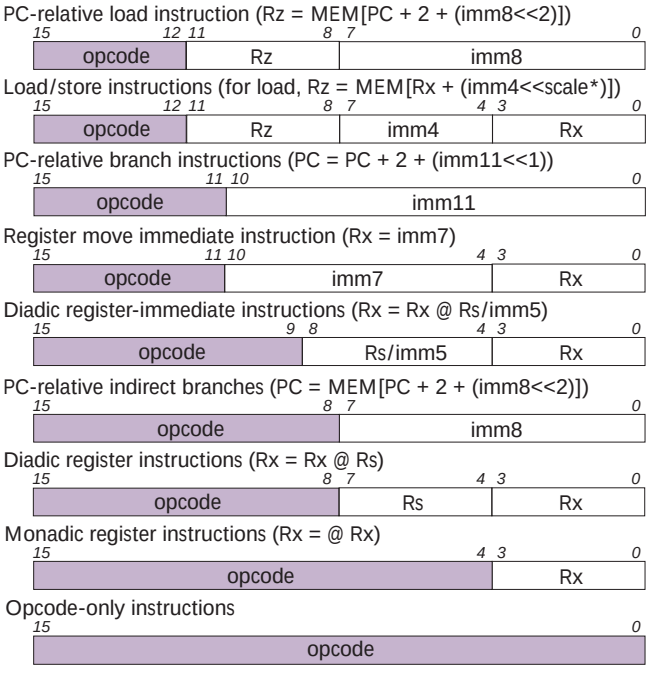

Figure 1. M•Core uses two-operand instruction formats for most instructions. *4-bit immediate value is scaled by 1, 2, or 4 to generate a byte, halfword, or word offset, respectively.

Sixteen Registers Force Two-Operand Formats

M•Core has 16 general-purpose registers, any one of which can be specified as an operand. Once the M•Core architects decided to use four bits for encoding a register operand, they had no choice but to adapt two-operand instruction formats, as Figure 1 shows, since three-operand formats would have left too few bits for the opcodes. ARM's Thumb (see MPR 3/27/95, p. 1) and MIPS-16 (see MPR 10/28/96, p. 40) offer three-operand instructions, but these instructions use only eight general-purpose registers. The M•Core architects decided in favor of offering more registers and a consistent, albeit limited, programming model.

Two-operand formats result in more instructions than three-operand formats, all other things being equal, because extra move instructions are needed to prevent source operands from being overwritten. Fortunately, many arithmetic and logical operations are commutative—the two source operands are interchangeable—allowing the to-be-saved operand to be used in the source-only position and thereby avoiding a move instruction. For noncommutative operations, such as subtract or divide, two flavors of instructions can be offered: one overwrites the first source operand and the other overwrites the second. M•Core offers reverse subtract instructions for this purpose, since the subtract operation is used frequently in a typical program.

M•Core's two-operand formats cause 2–7% code expansion in key applications, according to Motorola. This estimate is based on the assumption that 50% of all move instructions are to preserve the source operands. The other 50% are presumably used for passing arguments and results in subroutine call and return codes. For example, move instructions account for less than 5% of the total in fax-machine and engine-controller routines, indicating little need to copy source operands in these applications.

11-Bit Displacement Covers 98% of Branches

The 16-bit instruction format severely restricts the number of bits available for immediate operands. This restriction affects branch instructions the most, since as many as one in four instructions is a branch. M•Core provides 11-bit immediate operands for both conditional and unconditional branch instructions, covering the displacement needed by 98% of the branches in M•Core's key applications. The 11-bit

Compare Instruction	True Condition	False Condition
CMPHS (unsigned)	≥	<
CMPLS (unsigned)*	≤	>
CMPNE (signed, unsigned)	≠	=
CMPGT (signed)*	>	≤
CMPLT (signed)	<	≥

Table 1. Using branch-false instructions, M•Core provides only three compare instructions. *These instructions are not offered, since swapping the two operands produces the needed result.

immediate values are scaled by two to generate halfword offsets, allowing the branch instructions to jump as many as 2,047 instructions forward or backward. In comparison, MIPS-16 and Thumb provide an 8-bit displacement for conditional branches, although they provide 11 bits of displacement for unconditional branches.

For branches requiring more than 11 bits of displacement, M•Core offers PC-relative indirect-branch instructions, which load the program counter from memory. The instruction's 8-bit immediate value is scaled by four (to create a word offset) and added to the program counter. The result is then used as a pointer in memory, which contains the branch target address. In contrast, a PC-relative branch would use the result as the branch target address.

These instructions allow the program to branch to any word-aligned address in a 32-bit address space. They combine a pair of load and branch-indirect instructions, which would use eight bytes, into one instruction that uses six bytes. In ARM, which uses R15 as the program counter, a load to R15 naturally provides PC-relative indirect branching. But this feature is not available in Thumb mode, which uses only R0–R7.

Binaries containing these instructions are not relocatable, however, since the branch target addresses are encoded as absolute addresses, not as PC-relative offsets. Most of M•Core's target applications, which offer a single function or a set of dedicated functions, may not need relocatable binaries. The linker/loader can also adjust the binaries for changes in the branch target addresses.

Single Condition-Code Bit Saves Opcodes

Unlike most architectures that use multiple CC (condition-code) bits—typically, N, Z, V, and C—M•Core uses only one CC bit, denoting a true or false condition. It provides a set of compare-relationship instructions, which set or clear the CC bit, and a few branch-true and branch-false instructions. In contrast, most other architectures generally provide a single compare instruction, which sets and clears the four CC bits, and a plethora of branch instructions, which encode one of as many as 14 branch conditions, not counting the *always* and *never* conditions. A compare operation is rarely followed by multiple branch instructions, especially in code generated by C compilers, according to Motorola.

Using fewer branch and more compare instructions, M•Core uses fewer instruction bits overall than do most other architectures, since branch instructions require more bits—due to their displacement field—than do compare instructions. Using the saved bits, M•Core offers a larger branch displacement—11 versus 8 bits—for the conditional branches than do the other 16-bit instruction sets.

M•Core does not provide 14 different versions of compare instructions, nor are all 14 necessary. For unsigned comparisons, it provides only the CMPHS (compare high-or-same) and CMPNE (compare not-equal) instructions, as Table 1 shows. It offers branch-false instructions to handle the inverse of the CMPHS and CMPNE results. It does not provide an instruction for a lower-or-same comparison, since swapping the operands on CMPHS produces the same result: the R1 ≤ R2 comparison is the same as R2 ≥ R1. The two operands cannot be swapped, however, when one of them is an immediate operand. M•Core provides only one more instruction—CMPLT—for a signed compare, since CMPNE can be used for both signed and unsigned comparisons.

Most architectures update the CC bits as a side effect of executing arithmetic instructions, often generating branch conditions for free. In M•Core, the ADDC and SUBC instructions copy the carry output from the adder into the single CC bit, detecting 32-bit unsigned overflows for free. M•Core's other arithmetic instructions do not detect overflows, however, relying instead on software to detect them. For instance, the MULT instruction—which multiplies two 32-bit operands and returns the lower 32 bits of the result—provides no information on the upper 32 bits of the result. There are also no signed arithmetic instructions that report overflows.

The M•Core architects decided not to support signed overflows in the instruction set, mainly because the C programming language does not recognize overflows. In addition, most existing microcontroller applications use explicit range-checking instructions, since they use numbers that fit in far fewer than 32 bits.

The single CC bit allows M•Core to offer many predicated (conditionally executed) instructions, as Table 2 shows. The single CC code bit allows the opcodes to imply predication, but it also requires M•Core to offer both execute-if-true and execute-if-false versions of the instructions.

Bit-Twiddling Instructions Help Create Constants

Arithmetic and logical instructions occupy a large fraction of available opcodes in a typical instruction set, making it difficult to allocate enough bits for both the opcodes and the immediate operands using only 16 bits. Many microcontroller applications also make heavy use of logical and bit-manipulation operations, requiring bit patterns that are difficult to encode in a few bits of immediate operands. Fortunately, the arithmetic instructions tend to use small constants, especially in microcontroller applications, requiring only a few bits to encode most of them.

Motorola's analysis of its target applications shows that most constants are clustered near the low end of the number

spectrum and around the numbers that are powers of two. For small constants, M•Core supports 5-bit immediate operands in the diadic (two-operand) register-immediate format. This format is used to encode the unsigned ADDI, SUBI, and CMPLTI instructions, which map the immediate operand to a value from 1 to 32. Unlike MIPS-16, which provides a signed compare instruction using an 8-bit immediate, M•Core does not provide signed arithmetic instructions that use immediate operands. Not offering signed counterparts for these arithmetic instructions is a minor nuisance, especially in applications interfacing with physical world.

Bit setting, clearing, and testing operations—which are frequent in microcontroller applications—use constants that are powers of two. For these, M•Core provides the BSETI, BCLRI, and BTSTI instructions, which set, clear, or test, respectively, a bit in a register. It also provides the BGENI instruction, which sets one bit and clears all other bits in a register. Constants that are one smaller than a power-of-two number (e.g., 255) are used in bit-masking operations. M•Core provides the BMASKI instruction for these frequent operations. All of these instructions offer a 5-bit immediate operand, which is enough to identify any bit position within a 32-bit register.

M•Core also provides the MOVI instruction, which moves a 7-bit unsigned immediate value into a register. According to Motorola, the combination of the move and bit-manipulation instructions generates 90–95% of the constants used in M•Core's key applications. Increasing the immediate field to eight bits would raise this coverage by 1%, according to Motorola, but it would significantly reduce the number of available opcodes.

In comparison, Siemens's TriCore provides the IMASK instruction (see MPR 11/17/97, p. 13), which is a superset of M•Core's BMASKI but uses a 32-bit format. The other 16-bit instruction sets do not provide any bit-manipulation instructions, relying instead on existing means to construct arbitrary 32-bit patterns.

PC-Relative Loads Create Arbitrary Constants

For the remaining 5–10% of the constants, there is no shortcut way but to construct them as arbitrary 32-bit numbers. In a typical 32-bit instruction set, two move instructions are used to generate the upper and lower 16 bits in a register. This approach would not work well using 16-bit instructions, since so few bits—7 bits in M•Core—could be allocated for the immediate operand, requiring an unwieldy sequence of move and shift instructions to form a 32-bit constant. Instead, M•Core provides a PC-relative load instruction, which loads 32-bit data from memory.

This instruction differs from other load instructions in that it uses the program counter—not a general-purpose register—to generate the memory address. This avoids using

Mnemonic	Description	Mnemonic	Description	Mnemonic	Description
PC-relative load format		**PC-relative indirect format**		**Monadic register format**	
LRW	Load relative word	JMPI	Jump indirect	ABS	Absolute value
Load/store format		JSRI	Jump to subroutine indirect	ASRC	Arithmetic shift right, update C
LD.[W,H,B]	Load word, halfword, byte	**Diadic register format**		BMASKI #32	Bit mask immediate #32
ST.[W,H,B]	Store word, halfword, byte	ADDC	Add with C bit	BREV	Bit reverse
PC-relative branch format		ADDU	Add unsigned	CLRF*, CLRT*	Clear if false/true
BF, BT	Branch false, branch true	AND	Logical AND	DECF*, DECT*	Decrement if false/true
BR	Branch unconditional	ANDN	Logical AND NOT	DECGT	Decrement, set C if result > 0
BSR	Branch to subroutine	ASR	Arithmetic shift right	DECLT	Decrement, set C if result < 0
Register move immediate format		BGENR	Bit generate register	DECNE	Decrement, set C if result ≠ 0
MOVI	Move immediate	CMPHS	Compare high-or-same	DIVS	Divide signed
Diadic register-immediate format		CMPLT	Compare less than	DIVU	Divide unsigned
ADDI	Add immediate	CMPNE	Compare not equal	FF1	Find first 1
ANDI	Logical AND immediate	IXH	Index halfword	INCF*, INCT*	Increment if false/true
ASRI	Arithmetic shift right immediate	IXW	Index word	JMP	Jump
BCLRI, BSETI	Bit clear/set immediate	LOOPT	Decrement and loop if true	JSR	Jump to subroutine
BGENI	Bit generate immediate	LSL	Logical shift left	LDM, LDQ	Load multiple/quad registers
BMASKI	Bit mask immediate	LSR	Logical shift right	LSLC	Logical shift left, update C
BTSTI	Bit test immediate	MOV	Move	LSRC	Logical shift right, update C
CMPLTI	Compare less than immediate	MOVF*	Move if false	MVC	Move C bit to register
CMPNEI	Compare not equal immediate	MOVT*	Move if true	MVCV	Move inverted C bit to register
LSLI	Logical shift left immediate	MULT	Multiply	NOT	Logical NOT
LSRI	Logical shift right immediate	OR	Logical OR	SEXT[B,H]	Sign-extend byte/halfword
MFCR	Move from control register	RSUB	Reverse subtract	STM, STQ	Store multiple/quad registers
MTCR	Move to control register	SUBC	Subtract with carry	TSTNBZ	Test for no byte equal to zero
ROTLI	Rotate left immediate	SUBU	Subtract unsigned	XTRB[0,1,2,3]	Extract byte 0/1/2/3
RSUBI	Reverse subtract immediate	TST	Test	XSR	Extended shift right
SUBI	Subtract immediate	XOR	Logical Exclusive OR	ZEXT[B,H]	Zero-extend byte/halfword

Table 2. M•Core currently has 91 instructions and 13 spare opcodes. Coprocessor instructions and those in the opcode-only format are not shown. *predicated instructions

	M•Core	SH7000	Thumb (ARM)	MIPS-16 (-32)	TriCore	ColdFire
Instruction width	16 bits	16 bits	16 (32) bits	16 (32) bits	16, 32 bits	16–48 bits
GP (general-purpose) registers	2 x 16	16	32 (32)	32 (32)	16 adrs, 16 data	8 adrs, 8 data
ALU-accessible GPs	16	16	8 (32)	8 (32)	16	8
Three-operand instructions	none	none	7 (all ALU)	13 (all ALU)	all ALU	all ALU
Predicated instructions	8	none	none (all)	none (none)	6	none
Branch architecture	branch on CC	branch on CC	branch on CC	cmp-and-branch	cmp-and-branch	branch on CC
Condition code	C	C	N, Z, C, V	none	C, V	N, Z, C, V
Branch displacement (max)	11 bits	12 bits	11 (24) bits	11, 16 (26) bits	24 bits	16 bits
Indirect branch	reg, mem	reg	reg (reg, mem)	reg (reg)	reg	reg, mem
PC-relative load	yes	yes	yes (yes)	yes (no)	no	yes*
Multiply, divide instructions	both	mac, div step	mul (mul)	mul, div (both)	mul, div step	mul, div
Bit-, byte-manipulation instructions	many	none	none (none)	none (none)	few	few
Media instructions	none	in SH-DSP	in Piccolo	in MDMX	SIMD insts	none
Coprocessor instructions	yes	no	no (yes)	no (yes)	no	no

Table 3. M•Core offers more general-purpose registers, predicated instructions, and bit- and byte-manipulation instructions than other architectures offering 16-bit instructions. *ColdFire offers several PC-relative addressing modes for most ALU instructions. (Source: vendors)

precious registers to address a constant. It requires the constant to be stored within the range of its 8-bit unsigned displacement, which is scaled by four to yield a 10-bit range (1,024 bytes). It also requires a load access to the instruction memory, favoring a unified memory organization.

Both Thumb and the MIPS-16 instruction sets offer PC-relative load instructions, although the MIPS-32 instruction set does not. A typical RISC instruction set does not provide such instructions, since the program counter is a special-purpose register inaccessible to load/store instructions. A typical CISC instruction set, such as Motorola's 68K, offers a similar move instruction but as a 32-bit immediate operand, avoiding a load access to instruction memory.

M•Core Offers Plenty of Features in 16 Bits

As Table 3 shows, M•Core most resembles Hitachi's SH7000 (see MPR 8/23/93, p. 14), including the use of 16 bits to encode every instruction, two-operand formats, 16 general-purpose registers, and a single CC bit. M•Core offers more predicated instructions and more instructions for bit and byte manipulation than the SH7000. It also offers coprocessor instructions, which are not available in the SH7000.

Compared with Thumb and MIPS-16, which offer subsets of existing 32-bit instructions using 16-bit encodings, M•Core offers more instructions and registers and better code density, at least in many microcontroller applications. In Thumb and MIPS-16, only the move instructions can access all 32 registers, and all other instructions are limited to accessing only eight registers, requiring extra instructions to save and restore registers when eight is insufficient. In contrast, M•Core has two banks of 16 registers, although only one bank is accessible at a time and move instructions cannot transfer registers between the banks. MIPS-16 lacks load/store multiple instructions, which are used frequently in subroutine calls and returns as well as data-move operations.

A Thumb or MIPS-16 core must support both 16- and 32-bit instruction sets using two operating modes, since many operations are not available in the 16-bit instruction set. These include access to control registers and coprocessor instructions. In Thumb mode, the ARM's predicated instructions are no longer available.

NEC's V800 (see MPR 10/25/93, p. 25) and Siemens's TriCore use both 16- and 32-bit instructions, which can be freely intermixed in the code, offering a richer set of instructions than M•Core. Their 32-bit instruction formats provide more bits for a third operand, larger immediate values, and more opcodes, allowing them to support media and DSP operations not available in M•Core. Their rich set of instructions and features is likely to result in chips with a larger die and higher power consumption, however, making them less competitive in many of M•Core's target applications.

For low-cost, low-power applications, M•Core is definitely superior to Motorola's other instruction sets. Over the past two decades, the 68K architecture has grown to include too many bells and whistles, making it difficult to compete with other, simpler architectures—including M•Core—in embedded markets. The 68K's main strength is its established infrastructure, which is important in delivering low-cost solutions.

ColdFire is the 68K's illegitimate child, bearing a strong resemblance to but not binary compatibility with its parent. Because ColdFire is designed to be synthesizable, it cannot match the die size or power efficiency of the hand-tuned M•Core design. PowerPC is designed for speed but is far more complex. None of the three can match M•Core in simplicity, low cost, and low power consumption.

As M•Core's target applications grow, Motorola will undoubtedly extend M•Core, using more bits for the instructions. In extending M•Core, Motorola can learn from other architectures, which are already in the race to deliver more performance using less power, as it has done with M•Core's current instruction set. Motorola is not late in entering this race, a marathon that has just started. Using M•Core, Motorola is poised to meet the new millennium's insatiable demands for more performance and less power. M