M•Core Shrinks Code, Power Budgets

Motorola's Newest 32-Bit Family Aims to Outdo Low-Power Competitors

by Jim Turley

Processor powerhouse Motorola started up yet another generator at the recent Microprocessor Forum, taking the wraps off M•Core, the company's latest 32-bit embedded microprocessor family. A totally different instruction set from 68000, ColdFire, or PowerPC, M•Core promises to re-energize Motorola's efforts to power the next generation of wireless, portable consumer items.

The new architecture has been in development since 1993, but the first hints of its existence surfaced only last month (see MPR 9/15/97, p. 4). M•Core is intended to wrestle with ARM, promising better performance, better code density, and even lower power consumption than the best the English firm's many licensees can offer. Motorola even plans to license M•Core to outside companies. We expect Mitsubishi will be the most likely first candidate.

M•Core fills a void in Motorola's product line for a 32-bit CPU with very low power consumption. All of its current 32-bit products are either too old, too complex, or both. The 68K is far too complicated to ever reach the lowest power levels, and PowerPC, although generally characterized as a RISC processor, has one of the more complex internal architectures in existence. Even ColdFire, with its 68K legacy, is burdened with a core architecture that gives it a built-in handicap against its low-power competitors.

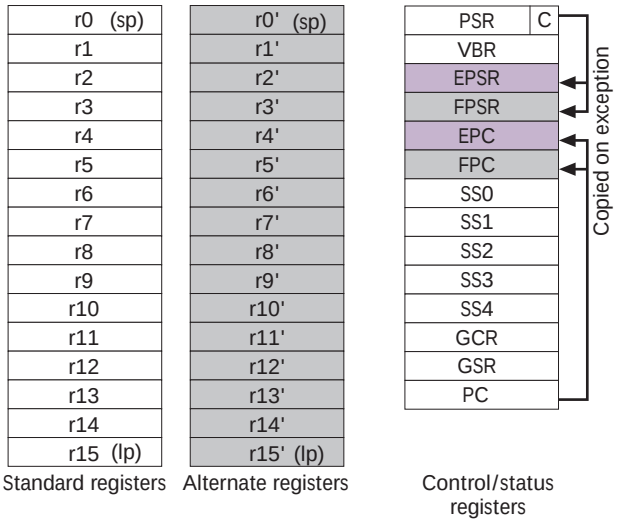


Figure 1. The M•Core architecture has two identical sets of general-purpose registers, either one of which is available to user-mode code. On exceptions, the PC and the PSR (processor status register) are copied to a pair of shadow registers.

To compete in the burgeoning market for portable handheld devices—like the ones Motorola itself makes—the company felt it needed something entirely new. M•Core is distinctly different from any of Motorola's other popular processors and gives the company its best chance yet of staking a sizable claim in this expanding territory.

Chip Shadows Registers Like a Microcontroller

The register file, as Figure 1 shows, is fairly straightforward. It consists of sixteen 32-bit registers, all of which are general-purpose. The only restrictions are that r0 and r15 are used as a stack pointer and link register by a few instructions.

In addition to the 16 main registers is another set of 16 identical registers. This "alternate" register set, R0'–R15', can, at the programmer's option, be used instead of the normal register file. Typically, this alternate register set is reserved for interrupt handlers or other time-critical routines. Such duplicate register files are common in 8- and 16-bit micros.

M•Core's control and status registers include a processor status register (PSR), two pairs of "shadow"registers, and five scratch registers as well as some global resources. The PSR contains various control and status flags, including the lone conditional bit, or C flag. The four shadow registers are used during interrupt handling and constitute M•Core's entire contribution to state preservation during exceptions. The five scratch registers, SS0–SS4, can be used by supervisory software for any purpose.

The M•Core architecture defines two privilege modes: user and supervisor. Supervisory code can access all the registers in Figure 1; user code has access to only the basic 16 registers (or their alternates) and the C flag. Except for a few operations that modify system registers, any instruction can be executed in either mode.

First-Level Interrupt Response Is Very Quick

Two input pins can initiate a hardware interrupt: the normal interrupt and the fast interrupt. Software faults and exceptions, can, of course, also initiate exception processing.

M•Core keeps first-order interrupt-response time extremely brief by avoiding a stack and using shadow registers. On an interrupt or exception, the chip copies only two system registers to their respective shadow registers. After that, exception handling is up to the programmer.

For a normal hardware interrupt or any software exception, the PC and PSR are copied into EPC and EPSR (or FPC and FSPR if the exception was caused by a fast interrupt). The chip then enters supervisor mode, disables tracing, and disables subsequent normal interrupts (or fast interrupts, depending on the cause of the exception).

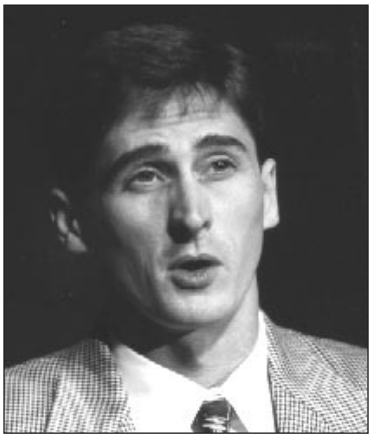
Finally, M•Core clears the exception-enable flag. Clearing EE doesn't hold off subsequent exceptions, nor does it prevent the two shadow registers from being overwritten if one does occur, but it does alert the chip that such an exception will be unrecoverable (i.e., that state information has been lost).

To allow nested exceptions or interrupts, programmers should copy the two shadow registers, re-enable them by setting EE, and set IE (or FE) to recognize new interrupts.

At the end of the exception handler, software uses either the RTE (for software exceptions and normal hardware interrupts) or the RFI (for fast interrupts) instruction, which restores the PC and PSR from the appropriate pair of shadow registers.

In truth, fast interrupts are no faster than other hardware interrupts; they merely use a different pair of shadow registers, which makes it possible to acknowledge a fast interrupt while still servicing a normal interrupt, or vice versa.

For applications in which interrupt latency is critical, programmers can set the IC (interrupt control) flag in the PSR. The IC bit determines whether interrupts and exceptions are recognized on instruction boundaries or on clock-cycle boundaries. The difference can be important when executing long-latency instructions such as multiplication or division (both of which M•Core supports in hardware). If a long-latency instruction is aborted because of an interrupt, M•Core will automatically restart it.



MICHAEL MUSTACCHI

Motorola's John Arends talks about M•Core's compact 16-bit instruction set at the Forum.

IRQ Vectors Harken Back to 68K

Each exception or hardware interrupt can be given its own interrupt handler via M•Core's vector table. The table consists of 128 four-byte entries, one for each of 128 possible exception vectors. The process is similar in concept to interrupt handling in the 68000 and ColdFire families, and the 6800 before them.

Each 32-bit vector is an address pointer to the appropriate exception handler. After any exception, M•Core loads the PC from the vector table and begins execution at the specified address. Programmers can use the least-significant bit of the vector to choose between the main and alternate register files for the exception handler.

Internal exceptions are assigned fixed vectors; hardware interrupts can take either the default vector (i.e., autovector) or a vector supplied by external hardware. Hardware vectoring allows the system designer to assign different exception handlers to different interrupt sources, such as timers, A/D converters, and the like.

M•Core differs from previous Motorola processors in one important aspect. With M•Core, the seven-bit interrupt vector is supplied simultaneously with the interrupt itself: the vector is latched on the falling edge of the interrupt-request pin. All 68K parts, on the other hand, recognize the interrupt first and then start a separate interrupt-acknowledge cycle to fetch the vector. The latter method can lead to lengthy—and, more important, uncontrolled—delays before the vector comes back and interrupt processing begins.

Instructions Are All 16 Bits

For a 32-bit microprocessor, it's unusual to have a 16-bit instruction word, but that is precisely what Motorola wanted. By keeping all instructions within a tight 16-bit encoding, Motorola wanted M•Core users to be able to get their best performance from a 16-bit external bus. In addition to reducing cost, a 16-bit bus should also reduce power consumption compared with a 32-bit bus, which was another of Motorola's goals.

Hitachi's SuperH family is the only other mainstream 32-bit architecture to use 16-bit instructions exclusively. Motorola's own 68K and ColdFire and NEC's V800 use a mixture of 16-bit, 32-bit, and longer instructions; so do members of the x86 family, with some 8-bit instructions thrown in.

M•Core also has similarities to ARM's Thumb (see MPR 3/27/95, p. 1) and the MIPS-16 (see MPR 10/28/96, p. 17) instruction sets. The latter two are "code compression" instruction sets that encode a useful subset of the full 32-bit instruction set and allow the chip to run in a code-compressed mode using just the subset. In both cases, however, ARM and MIPS chips still have access to their full 32-bit instruction sets, so these combination instruction sets are more like the 68K's mixed 16/32-bit design than a true 16-bit instruction set.

Addressing Modes Are Fixed

About 14% of M•Core's opcode space is unused, allowing room for promised future expansion. Motorola is already planning enhancements to the basic M•Core instruction set for special purposes. Thus, as M•Core parts proliferate in the coming years, they will all support a basic subset, with individual members executing a dozen or so extra instructions as necessary.

M•Core is subject to the usual limitations of a 16-bit instruction word. Virtually all register-to-register operations are destructive, with the result replacing one of the source operands. Immediate (literal) values are generally limited to 5–7 bits. Some specific options or encodings are illegal because they conflict with other instructions.

As a nouveau-RISC architecture, M•Core allows only loads and stores to memory; all ALU operations are to register contents. Four addressing modes are supported: register indirect, register indirect through R0 (the stack pointer),

register indirect with 4-bit offset, and PC-relative with 8-bit offset. Each of the four addressing modes is used by exactly two instructions: one load and one store.

M•Core Does a Lot With 16 Bits

It's tough to break into the market with an entirely new and incompatible instruction-set architecture unless the new offering is significantly better than any of its competitors. Even then, success isn't assured. In M•Core, though, Motorola has a strong design with a lot to offer.

The nearest—and most relevant—comparisons are to Thumb, MIPS-16, and SuperH. M•Core's designers were faced with the same problems as those of SuperH, inasmuch as there is no 32-bit instruction set to fall back on. Even little-used system-level operations have to be included in the 16-bit instruction map.

M•Core is unusual for its pair of signed and unsigned divide instructions. Most 32-bit CPUs (ARM and MIPS among them) don't have a divide operation at all. Although SuperH chips do, Hitachi makes integer division an awkward, iterative process. M•Core's throughput is no better than SuperH's, but at least it's a single instruction. Early-out hardware means that M•Core divides can be as short as 3 cycles or as long as 37; the same shortcut in SuperH code requires manually examining the operands.

Hitachi's SuperH chips share the same kind of 16×32-bit register file as M•Core, but SuperH chips use dedicated registers for multiplication and division. Separate instructions are needed to transfer data to/from these registers, which slows these operations somewhat compared with M•Core. (Most MIPS chips work the same way.) On the plus side, SuperH supports 64-bit multiplication (both signed and unsigned), which M•Core can't do. Extended-precision integer multiplication is useful in some DSP applications or for emulating floating-point arithmetic on the cheap.

The most significant difference between the instruction sets is Hitachi's inclusion of a multiply-accumulate instruction, which M•Core doesn't have. A simple MAC operation goes a long way toward making these chips useful in lightweight signal-processing applications. Given Motorola's focus on communications, it's remarkable that M•Core doesn't include this basic function. When Motorola adds extensions to the M•Core instruction set in 1999, a MAC will presumably be among them.

M•Core Holds Advantages Over Thumb

Compared with Thumb, M•Core has twice as many registers, since Thumb's 16-bit instructions can access only the first eight. M•Core also has the richer and more varied instruction set, which is all the more surprising given that M•Core doesn't have a 32-bit instruction set to fall back on. Thumb, which uses its 16-bit instruction set only for application code, actually has fewer useful operations and variations than M•Core does.

Both instruction sets include high-frequency instructions for signed and unsigned addition and subtraction, 32-bit multiplication, loading and storing multiple registers, and loading and storing bytes and halfwords (with zero-extension).

Only M•Core can calculate absolute value or add and subtract using immediate operands; it even has reverse subtraction (where the register is subtracted from a constant, instead of vice versa). M•Core also has instructions for signed and unsigned division, which ARM chips have never had. M•Core's FF1 (find first one) and BREV (bit reverse) are odd but crucial additions for those applications that deal with cryptography.

M•Core, ironically, also has much better support than Thumb for conditional execution, which has been a hallmark of ARM code. Conditional move, clear, shift, increment, decrement, and decrement-and-branch instructions can eliminate many short code hops. M•Core also allows unsigned comparisons (higher-or-same) and can extract any byte from a 32-bit word in a single operation.

In ARM's favor, Thumb can sign-extend bytes or halfwords loaded from memory, which M•Core does not support in one instruction, and Thumb has more conditions on which to control a branch, compared with M•Core's single condition flag.

First Chips Demonstrate Code-Density Goals

The first chip in the M•Core family will be a mixed DSP/CPU device similar to the company's 68356. The device mixes a 56600 16-bit DSP core and a separate M•Core processor with a host of peripherals; aimed at cellular baseband applications, it is expected to begin shipping in 1Q98.

The second M•Core device, boldly dubbed PowerStrike 1, will be a more conventional general-purpose processor suitable for evaluating M•Core's merits. It integrates a 33-MHz core with 32K of SRAM, a keypad input, six pulse-width modulation (PWM) channels, and two UARTs. Housed in an LQFP-144 package, the 3.3-V chip is expected to sell for $13 when it becomes available in 1Q98.

One of Motorola's overall goals in creating M•Core was to improve on the code density of any of the current crop of 32-bit embedded processors. The company set ARM as its target, an architecture with unusually good code density for a 32-bit chip and an obvious target for a company focused on telecommunications.

As Figure 2 shows, M•Core succeeds admirably, at least in Motorola's tests. On a number of C language code samples, M•Core object code is an average of 50% smaller than the same code compiled for an ARM7, or 11% smaller than Thumb code. The code fragments in these tests range from a few hundred bytes to about 6K in size. All 16 tests were compiled using the latest C compilers with all the size optimization switches turned on. Although the tests are small (and hand-picked by Motorola), the results are impressive.

Is M•Core Really Necessary?

In this industry, there are few undertakings more expensive than developing and maintaining a new microprocessor architecture. AMD, Intergraph, and others have learned this to their detriment when sales volumes could no longer cover the burdensome costs of development and tool support. With so many architectures already on the market, couldn't Motorola have found a cheaper route to lower power?

The company believes not. To achieve its stated goal of "best in class" power efficiency, it chose to optimize every aspect of the design to reduce power consumption. M•Core's 16-bit instruction word means narrower buses, fewer signal transitions, and fewer memory chips. Dense code requires less memory and fewer instruction fetches, lowering power and cost. Good signal-processing performance—which Motorola has promised but which is certainly not in evidence in the first M•Core disclosure—would help eliminate separate DSP chips. Motorola's other architectures could not have achieved the same level of code density and still retained their software compatibility.

A more aggressive stance on fabrication processes might have achieved similar goals with much less effort, however. Much of M•Core's expected benefit comes from the 0.35-micron process it's built on, compared with the antiquated 0.8-micron process used for most ColdFire chips today. All other things being equal, a 0.35-micron ColdFire device would probably also run from a 1.8–3.6-V supply and still reach 50 MHz—and, more important, drop power consumption by a factor of five or more. Motorola could have achieved perhaps 70–80% of the same benefit using the tools it already had, and avoided all the time, money, and effort expended to develop M•Core.

For comparison, IBM's 401GF (see MPR 6/17/96, p. 9) hits 50 MHz while drawing only 140 mW from its 3.3-V supply, proving that even a PowerPC instruction set needn't be a permanent handicap. Instruction sets and microarchitectures do make a difference, but fabrication process, gated clocks, and static cores are even more important, and can be applied to almost any architecture.

Motorola's Dilemma: M•Core or ARM?

ColdFire and M•Core were actually developed concurrently. Resource limitations within Motorola stalled M•Core's maturation, delaying its eventual rollout by exactly two years compared with ColdFire. Though they now seem unrelated, the two were meant to be fraternal twins.

Timing issues caused Motorola's Consumer Products Group to choose ARM (see MPR 3/31/97, p. 4); had M•Core development stayed on its original schedule, that embarrassment might have been avoided.

The decision to license the architecture of arch-rival ARM must have been a tough one. It was necessary, though, and probably helped push M•Core into the market that much sooner. Motorola is losing its edge in wireless communications to makers of digital gear (namely Nokia and Ericsson), which seem to have made ARM almost a standard in that market. For its products to be competitive, Motorola needed to use a low-power, flexible CPU, regardless of its provenance.

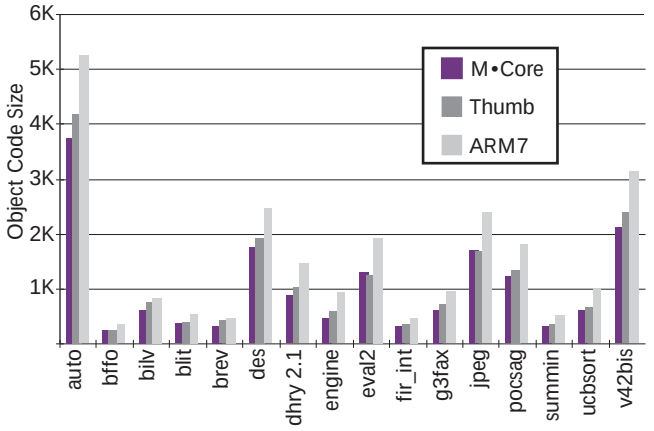


Figure 2. On a number of compiled C code fragments, M•Core consistently produced smaller object code (code and data) than ARM7 or Thumb. (Source: Motorola)

Now that ARM has its foot in the door, so to speak, it's there to stay. It will most likely be some years before the "borrowed" architecture is back out of Motorola, if ever. Naturally, Motorola's semiconductor group is now pushing M•Core aggressively to its product group, but ARM-based products are nearing production, and many will have long life cycles. Future generations of these products may stick with ARM to preserve an investment in software, tools, and interfaces. ARM was ready sooner, and the company can now thumb its nose at M•Core.

M•Core Spans MCU, MPU Gap

Looking forward, potential Motorola customers will face a bewildering array of choices for 32-bit embedded microprocessors. Those who want high performance will gravitate to PowerPC; those who want low power will choose M•Core; and those who want to split the difference and gain time to market will be pointed toward ColdFire. The 68K customers are self-selecting.

As these product lines progress, ColdFire will obviously take over from the 68K as the basis for a host of midrange application-specific devices. Motorola has set up ColdFire's design methods around synthesizable cores and peripherals in order to help make that happen.

M•Core, on the other hand, will become the company's entry-level 32-bit family: a move up from its many 16-bit microcontrollers for industrial, automotive, and consumer-electronics applications. General oversight for M•Core was given to Motorola's Transportation Systems Group, which is also responsible for the 68HC16 and 68300 chips used in those markets. As these applications call for more powerful controllers, M•Core will be waiting for them. M