VOLUME 10, NUMBER 7
MAY 27, 1996

MICROPROCESSOR REPORT

THE INSIDERS' GUIDE TO MICROPROCESSOR HARDWARE

68HC11 Grows Up to 16 Bits

Motorola's 68HC12 Line Boosts Performance up to 10 Times

by Jim Turley

Motorola's popular 8-bit microcontroller architecture, the 68HC11, now has a big brother. The new 68HC12 maintains all of its sibling's distinguishing features but doubles both clock rate and bus width, extends the address space, improves code density, and adds dozens of new instructions, including some for fuzzy logic.

Strategically, the 16-bit HC12 family—which initially includes two members—fills a small gap between Motorola's 8-bit 68HC11 family and the 16-bit 68HC16 line. The HC12 provides a source-level software upgrade for 8-bit designers who are outgrowing the HC11 but don't want to develop entirely new code and new hardware for the HC16.

Motorola believes the HC12 and HC16 can peacefully coexist in the company's 16-bit microcontroller lineup because the HC16 offers better performance and better signal-processing capabilities, while the HC12 offers compatibility with the HC11 and lower power dissipation. The first two chips in the new family are both priced below $25 and will begin sampling in June and October of this year.

Programming Model Identical to HC11

The HC12's register set and programming model are identical to those of the HC11, as Figure 1 shows. The two 8-bit accumulators, A and B, can be concatenated into a single 16-bit register, D. Index registers X and Y are used to reference memory-resident operands, while the stack pointer (SP) and program counter (PC) fulfill the obvious functions.

The HC12's exception stack and fault model are also identical to those of the HC11 chips, so users with existing code that examines or manipulates the stack will find that it works without modification. Source-code compatibility was a primary concern for the HC12's designers; given that Motorola already has a 16-bit product family, the HC12 had to offer something the HC16 didn't.

The familiar programming model belies major changes lurking beneath the surface. Motorola's customer surveys indicated that HC11 users were frustrated by the irregular and nonorthogonal treatment of the X and Y index registers. For example, most HC11 instructions can reference memory through either the X or Y index registers, but using the Y register adds a prefix byte to the object code, which requires an additional clock cycle to fetch over the HC11's 8-bit bus.

These concerns were addressed with the HC12, which encodes X- and Y-indexed instructions equally. At the same time, addressing modes were modified to support both SP and PC as index registers. Thus, compiler writers can more easily reference operands passed on the stack, and position-independent code can access data relative to the program counter.

Instruction Set Gets 65 New Mnemonics

The design goals for the HC12 stipulated total source-level (but not binary) compatibility with the HC11 so users could transfer existing assembly source without modification. This it does, duplicating every HC11 mnemonic and addressing mode, right down to some unintended quirks that users have learned to accept.

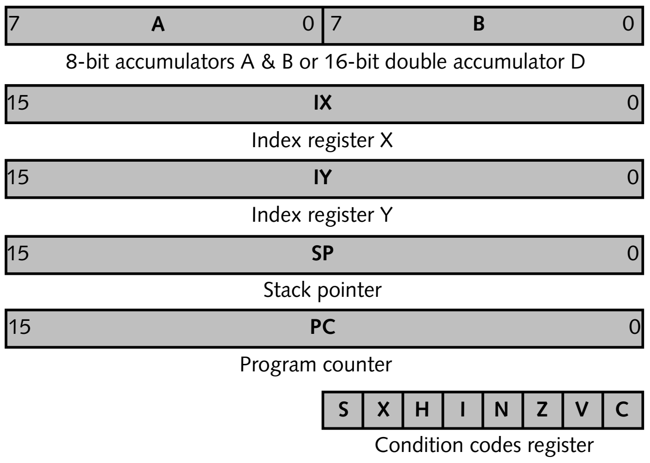

Figure 1. The register set of the 68HC12 is identical to that of its predecessor, the 8-bit 68HC11.

The HC12 truly implements its predecessor's instruction set one for one; unlike Philips' 8051XA tools *(see* 081304.PDF *)*, the HC12 assembler does not replace unimplemented legacy instructions with equivalent constructs. In addition to the HC11 instruction set, the HC12 has more than 65 new operations that should cheer compiler writers and relieve assembly programmers.

Table 1 lists the complete set of HC12 instructions with enhancements over the HC11 indicated. Some examples include the TFR (transfer) and EXG (exchange) instructions, which now handle mismatched register sizes. Exchanging an 8-bit with a 16-bit register zero-extends the 8-bit value; copying mismatched registers sign-extends the smaller register. This latter operation can be used by C compilers to cast a *char* to an *int*.

In cases where HC11 instructions sometimes produce unwanted side effects, as when the TAB and TBA (transfer A/B) instructions set condition codes even though most transfer instructions do not, the HC12 includes compatible instructions (complete with side effects) in addition to the preferred (and more orthogonal) versions. The generalized TFR instruction is now preferred over TAB and TBA, while ANDCC and ORCC replace SEI, CLI, SEC, and CLC.

No Binary Compatibility with HC11

The HC12's opcode map was completely rewritten, so binary

Data Transfer	
MOVB	Move byte
MOVW	Move word
LDAA/LDAB	Load A/B
LDD	Load double D
LDS	Load stack pointer
LDX/LDY	Load X/Y
LEAS	Load efective address, stack
LEAX/LEAY	Load effective address X/Y
STAA/STAB	Store A/B
STD	Store double D
STS	Store stack pointer
STX/STY	Store X/Y
PSHA/PSHB	Push A/B to stack
PSHC	Push condition codes to stack
PSHD	Push double D to stack
PSHX/PSHY	Push X/Y to stack
PULA/PULB	Pull A/B from stack
PULC	Pull condition codes from stack
PULD	Pull double D from stack
PULX/PULY	Pull X/Y from stack
TAB/TBA	Transfer A to B/B to A
TAP	Transfer A to condition codes
TFR	Transfer register to register
TPA	Transfer condition codes to A
TSX/TSY	Transfer stack pointer to X/Y
TXS/TXY	Transfer X/Y to stack pointer
SEX	Sign-extend 8 → 16
XGDX/XGDY	Exchange double D with X/Y
EXG	Exchange registers

Arithmetic	
ABA	Add B to A
ABX/ABY	Add B to X/Y
ADCA/ADCB	Add with carry to A/B
ADDA/ADDB	Add to A/B
ADDD	Add double D
SBA	Subract accumulators
SBCA/SBCB	Subtract with carry from A/B
SUBA/SUBB	Subract A/B
SUBD	Subract double D
DAA	Decimal adjust A
MUL	Multiply 8 × 8 → 16
EMUL	Multiply 16 × 16
EMULS	Multiply 16 × 16, signed
EMACS	Multiply-add 16 × 16 → 32
EDIV	Divide 32 ÷ 16
EDIVS	Divide 32 ÷ 16, signed
IDIV	Divide 16 ÷ 16
IDIVS	Divide 16 ÷ 16, signed
FDIV	Divide 16 ÷ 16, remainder 16

Flow Control	
BCS/BCC	Branch if carry set/clear
BEQ/BNE	Branch if equal/not equal
BGE/BLE	Branch if greater/less or equal
BGT/BLT	Branch if greater than/less than
BHI/BLO	Branch if higher/lower
BHS/BLS	Branch if higher/lower or same
BPL/BMI	Branch if plus/minus
BRCLR/BRSET	Branch if bits cleared/set
BRA/BRN	Branch always/never
BVS/BVC	Branch overflow set/clear
BSR	Branch to subroutine
CALL	Call subroutine
RTC	Return from CALL
DBEQ	Decrement, branch if zero
DBNE	Decrement, branch if not zero
IBEQ	Increment, branch if zero
IBNE	Increment, branch if not zero
JMP	Jump
JSR	Jump to subroutine
RTS	Return from subroutine
LBCS/LBCC	Long branch if carry set/clear
LBEQ/LBNE	Long branch if zero/not zero
LBGE/LBLE	Long branch if less/greater, eq.
LBGT/LBLT	Long branch if greater/less than
LBHI/LBLO	Long branch if high/low
LBHS/LBLS	Long branch if high/low, same
LBPL/LBMI	Long branch if plus/minus
LBRA/LBRN	Long branch always/never
LBVS/LBVC	Long branch if overflw set/clear
TBEQ/TBNE	Test and branch if zero/not zero
RTI	Return from interrupt

Arithmetic (con't)	
INC	Increment memory
INCA/INCB	Increment A/B
INS	Increment stack pointer
INX/INY	Increment X/Y
DEC	Decrement memory
DECA/DECB	Decrement A/B
DES	Decrement stack pointer
DEX/DEY	Decrement X/Y
MAXA	Maximum to A
MAXM	Maximum to memory
MINA	Minimum to A
MINM	Minimum to memory
EMAXD	Maximum to D, 16-bit
EMAXM	Maximum to memory, 16-bit
EMIND	Minimum to D, 16-bit
EMINM	Minimum to memory, 16-bit

Shift and Logical	
ANDA/ANDB	Logical AND A/B
ANDCC	Logical AND condition codes
ASL/ASR	Arithmetic shift left/right
ASLA/ASLB	Arithmetic shift left A/B
ASLD	Arithmetic shift left double D
ASRA/ASRB	Arithmetic shift right A/B
CMPA/CMPB	Compare A/B
COM	Complement memory
COMA/COMB	Complement A/B
CPD	Compare double D
CPS	Compare stack pointer
CPX/CPY	Compare X/Y
EORA/EORB	Exclusive-OR A/B
LSL/LSR	Logical shift left/right memory
LSLA/LSLB	Logical shift left A/B
LSLD	Logical shift left double D
LSRA/LSRB	Logical shift right A/B
LSRD	Logical shift right double D
NEG	Negate memory
NEGA/NEGB	Negate A/B
ORAA/ORAB	Logical-OR A/B
ORCC	Logical-OR condition codes
ROL	Rotate left memory
ROLA/ROLB	Rotate left A/B
ROR	Rotate right memory
RORA/RORB	Rotate right A/B
TST	Test memory
TSTA/TSTB	Test A/B

Miscellaneous	
BSET/BCLR	Set/clear bits in memory
SEC/CLC	Set/clear carry
SEI/CLI	Set/clear interrupt mask
SEV/CLV	Set/clear overflow
BITA/BITB	Bit test A/B
CLR	Clear memory
CLRA/CLRB	Clear A/B
TBL/ETBL	Table lookup and interpolate
SWI	Software interrupt
WAI	Wait for interrupt
BGND	Enter background debug
STOP	Stop processing
TRAP	Unimplemented opcode trap
NOP	No operation

Fuzzy Logic	
MEM	Determine fuzzy membership
REV	Rule evaluation
REVW	Rule evaluation, weighted
WAV	Weighted average

Table 1. The 68HC12 includes more than 50 new instructions compared with the 68HC11. New instructions, highlighted in purple, include new stack operations, extended-precision multiply and divide, minimum and maximum calculations, and new intersegment branches.

Price & Availability

The 68HC812A4 is sampling now for $19 in a 112-lead TQFP package; production quantities will be priced in the $10–$15 range. Production is scheduled for 1Q97.

The 68HC912B32 will begin sampling in 4Q96. Production pricing is expected to be $20–$25.

For more information, contact Motorola (Austin, Texas) at 800.765.7795; fax 512.891.4465; or browse to *freeware.aus.sps.mot.com/amcu/home.html.*

code from the HC11 will not run on the HC12, although about half the HC12's mnemonics produce the same encodings as on the HC11. The reworked encoding means some instructions will be shorter than their HC11 counterparts and some will be longer.

Motorola's tests indicate that simply reassembling HC11 source code for the HC12 has a negligible effect on object size. On the other hand, rewriting assembly code or passing C source through a new compiler can yield code-size reductions of as much as 30%, according to the company.

Users should also see a performance improvement along with tighter code. Several HC12 instructions execute in a single clock cycle, whereas the HC11 always takes at least two. Arithmetic instructions, in particular, have been improved, as Table 2 shows.

Fuzzy Logic Support a First

Four truly new instructions have been added to provide support for fuzzy-logic control applications. These complex instructions take from a few cycles to a few hundred cycles to execute and were made interruptible to avoid compromising worst-case latency.

The MEM instruction computes a *Y* value, given an *X* value as the index. Its operation goes beyond a simple linear table interpolation (which the HC12 can also perform) in that it operates on a trapezoidal membership function. That is, for values of *X* that are under the sloping side of a conceptual trapezoid, *Y* is interpolated from its slope; for values under the flat side of the trapezoid, *Y* is capped at 0xFF.

The rule-evaluation instructions REV and REVW process a list of pointers to values, alternately performing minimum and maximum operations to implement min-max fuzzy-rule evaluation. The REVW variation allows 8-bit weighting factors to be applied to each rule.

The WAV instruction is basically an 8-bit multiply-accumulate instruction, useful for low-precision pseudo-DSP operations but also as the basis for a weighted-average defuzzification. *(For background on fuzzy-logic programming, see* **061702.PDF** *)*

Mnemonic	Math Operations	68HC11		68HC12	
		cycles	time	cycles	time
MUL	8 × 8 → 16 (signed)	10	2.5 µs	3	0.375 µs
EMUL	16 × 16 → 32 (unsigned)†	20	5 µs	3	0.375 µs
EMULS	16 × 16 → 32 (signed)†	20	5 µs	3	0.375 µs
IDIV	16 ÷ 16 → 16 (unsigned)	41	10.25 µs	12	1.5 µs
IDIVS	16 ÷ 16 → 16 (signed)	—	—	12	1.5 µs
FDIV	16 ÷ 16 → 16 (fractional)	41	10.25 µs	12	1.5 µs
EDIV	32 ÷ 16 → 16 (unsigned)†	33	8.25 µs	11	1.375 µs
EDIVS	32 ÷ 16 → 16 (signed)†	37	9.25 µs	12	1.5 µs
EMACS	16 × 16 → 32 (signed)†	20*	5 µs*	12*	1.5 µs*

Table 2. Multiply and divide performance varies considerably between the HC11 and HC12 cores, in part because the HC12 is clocked twice as fast as its predecessor. †operations possible only on HC11 with math coprocessor. *per iteration.

The New Bus: Same as the Old Bus

The HC12 copies its predecessor's straightforward synchronous bus interface rather than following the asynchronous example of the HC16, 68300, and early 68000 processors. HC11 customers are used to simple peripheral logic with fixed and predictable (albeit sluggish) response times measured in multiples of a 4-MHz bus clock. No external logic is required (or able) to insert wait cycles or extend access times. Instead, chip-select logic in the HC12 can be programmed with the desired access time for each external device. We expect Motorola to expand the HC12 family in this way.

Internally, the HC12 uses the same intermodule bus (IMB) design philosophy that Motorola uses on its HC16 and most of the 68300-family devices. In fact, many HC16 and 68300 chips can and do use identical peripheral blocks, a fact that allows Motorola to spin application-specific variations of these two families on short notice.

The first two 68HC12 devices, the 68HC812A4 and 68HC912B32, both run at a comparatively speedy 8 MHz, twice as fast as the quickest HC11 part, boosting performance further. Their 16-bit buses also allow them to fetch code and access the stack in half the time.

With variable-length instructions, misalignment can be a problem on a 16-bit bus—something HC11 users never had to worry about. The HC12 design includes three 16-bit instruction buffers with byte-steering logic. The buffers allow the HC12 to fetch a 16-bit word on every cycle, ignoring instruction alignment. With the instruction in the buffer, decoding logic determines the instruction boundaries and aligns the instruction accordingly before execution. Although not as advanced as a cache, the pair of buffers helps keep the HC12's simple pipeline full.

Paging Expands Address Space

Expanding the tiny 64K address space of the HC11 proved to be tricky without compromising software compatibility or enlarging registers. The HC12's designers used a paging approach that stretches the chip's reach to 4M and is transparent to ported code unaware of the HC12's enhancements.

As Figure 2 shows, one-quarter of the HC12's address map, from 0x8000 through 0xBFFF, is a window onto additional external memory. Any access to this 16K space causes the contents of an 8-bit page register to be driven on the chip's high-order address lines. The page register selects one of 256 possible 16K pages. To software, the page register itself appears as another control register in the chip's memory-mapped on-chip peripheral space.

The page register is literally just gated onto the address bus; its contents are not available to the address-generation logic or to the ALU. A pair of new flow-control instructions, CALL and RTC, allow HC12 programs to jump between pages. The CALL instruction specifies a 24-bit target address, the upper eight bits of which are copied into the paging register after pushing the previous page value onto the stack. The RTC instruction restores the page address from the stack.

As simple as it seems, the HC12's method of code and data paging is more elegant than most. Other 8-bit and 16-bit controllers—such as the 8051, 68HC05, and 80251—require at least two instructions to change the page and jump to the target address, a construct that necessitates disabling interrupts, lest the chip get interrupted at an inopportune moment and vector to the wrong page.

An atomic page-switching instruction can switch pages while executing out of paged memory. In other systems, the paging code must be located in memory that cannot be paged. Otherwise, incautious programmers can wind up pulling the rug out from under themselves.

Two additional paging windows, located at 0x0000 and 0x7000, expand the data space even further. Like the 16K page at 0x8000, these 1K and 4K data pages each have their own paging register. Because they are not linked to flow-control instructions, they would normally be used only for accessing data.

Motorola Plays Low-Power Card

One distinction between the new HC12 parts and the faster HC16 devices is the former's lower power consumption. Whereas the original HC16 chips were 5-V–only parts, the A4 and B32 are designed to tolerate a wide supply range, from 3 to 5 V. A specified 10% tolerance brings the viable range from a low of only 2.7 V to a high of 5.5 V, covering the sweet spot for most two-cell battery technologies.

Typically, a broad supply tolerance forces significant speed derating at lower voltages, but both the A4 and the B32 run at 8 MHz across their entire supply range. This flat voltage rolloff suggests that these chips may have some substantial clock-speed headroom near 5 V. Although Motorola won't officially sanction operation beyond the published limit, the company hints that users might be able to discover a performance upside without too much effort.

The chips are built in a 0.65-micron two-layer-metal CMOS process, using Motorola's UDR (universal design rules) geometry, a process optimized for low power dissipation rather than for high speed. Still, the HC12's frequency characteristics and the fact that Motorola's PowerPC 602 runs at 66 MHz in a similar process make it appear that the performance of the HC12 is being artificially limited to avoid competing with the faster 16-bit and 32-bit devices.

The HC12 chips offer the familiar sleep and standby modes. On the A4, entering sleep mode cuts power consumption by 75% to 32 mW (typical) while the chip waits for an interrupt to resume operation. In standby mode, all operation ceases, and power drops to just 25 µW. Peripherals that aren't active have their clocks gated off automatically, and software can disable unused peripherals entirely.

Debug Functions Reduced to a Single Pin

Motorola continues to hone its background debug mode (BDM) feature, a set of on-chip debugging resources that first appeared with the debut of the 68300 family in 1989. As the name might imply, BDM works by placing the chip in a background mode, similar to low-power sleep or standby operation. While BDM is active, a developer can interrogate and modify system resources like CPU registers, memory-mapped I/O registers, internal EEPROM, or external memory via simple commands from an emulator or development system. On the HC12, the BDM interface has been reduced to a single pin.

Taking advantage of the BDM functions is simply a matter of serially transferring an 8-bit command word over the chip's BDM pin. Data may be clocked in asynchronously at conveniently arbitrary rates, up to a maximum of 500 kHz. The transfer protocol is simple enough that the emulator software can control it by toggling an external I/O line connected to the BDM pin.

The BDM features access memory-mapped resources while the processor is running at full speed. Both on-chip and off-chip memory can be examined and modified without altering the system's execution profile. The HC12 delays BDM memory accesses until the bus is unused, then steals a memory cycle. In the pathological case where the bus is always busy, the chip will wait for up to 128 cycles (16 microseconds) before it seizes the bus to access memory.

In addition to the basic BDM functions, the B32 version includes a set of internal breakpoint registers. Loading 16-bit address or data

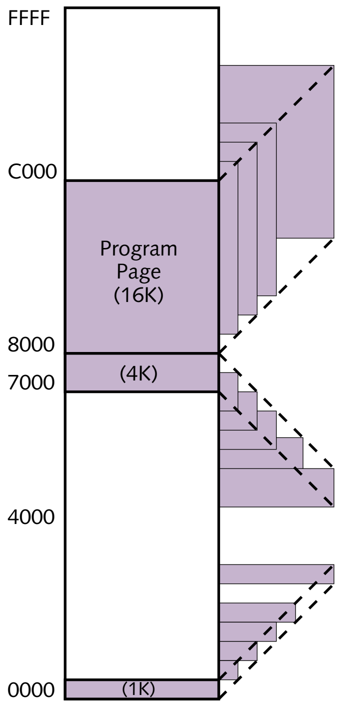

Figure 2. The HC12's memory map includes three segmented windows for data, one of which can also be used to expand the instruction space.

values into these registers enables the hardware breakpoints, which can force an exception on selective patterns of address, data, or cycle type.

New Options for Controller Users

Prices for the HC12 chips will fall below those of many members of the HC16 family. For 8-bit and 16-bit microcontrollers, price is determined largely by a chip's peripherals or on-chip memory, not its CPU core, which accounts for only a small portion of the die. High-end HC11 and low-end HC16 devices already overlap in price; placing the new HC12 parts in the middle will further muddy the price differential between these families.

The HC11 family is currently ranked third in worldwide unit shipments for 8-bit microcontrollers (behind the 68HC05 and 8051), according to published reports. With that kind of volume, it's logical for Motorola to do everything it can to protect is lucrative customer base.

Before the HC12, customers loyal to Motorola but looking for an upgrade for their 8-bit controller were steered toward the HC16, an architecture with a completely different programming model and hardware interface. Although most software tools for the HC16 accept HC11 assembly syntax, the translation is not straightforward, and the hardware differences are impossible to mask.

Both Intel and Philips lured their 8051 customers with easy 16-bit upgrades; Intel's is binary compatible, while Philips followed a looser definition and created a chip that maintained only source-level compatibility and overhauled the interior design. Both companies are now successfully introducing 8-bit customers to the joys of 16-bit computing.

In addition to HC11 upgrades, the prospects for the HC12 look good. Judging from the first two chips, Motorola is offering competitive performance for a reasonable price. The company's modular design philosophy, tried and proven with the HC16 and 68300, will allow it to duck and swerve as changes in the market drive new peripheral requirements. By sacrificing binary compatibility, the HC12 is able to achieve good code density even with an entirely new instruction set.

A few years ago, the conventional wisdom held that 8-bit users would skip over the 16-bit generation and move directly to 32-bit CPUs. The rumors of the demise of 16-bit microprocessors have been greatly exaggerated; in any field as cost-competitive as microcontrollers, every penny is worth saving. Thus, today's 16-bit controllers are going strong. When a design like the HC12 can offer users significant and tangible advantages in code compatibility, improved performance, better code density, and reduced power consumption, it should have no trouble making its way in the thick of the steadily growing 16-bit market. M