THE MC68020 32-BIT MICROPROCESSOR

BY PAUL F. GROEPLER AND JAMES KENNEDY

The latest member of Motorola's 68000 family includes on-board cache and virtual memory

THE MC68020, the newest addition to the Motorola M68000 family of microprocessors, is a full 32-bit processor with separate 32-bit data and address buses, an on-board instruction cache, dynamic bus sizing, and a coprocessor interface. It is object-code compatible with the earlier members of the M68000 family but has new addressing modes in support of high-level languages.

The MC68020 is an HCMOS (high-speed complementary metal-oxide semiconductor) microprocessor with some 200,000 individual transistors on a 375- by 350-millimeter die, operating at a 16.67-MHz clock frequency (60-nanosecond clock period) and dissipating less than 1.5 watts of power. It can process instructions at a sustained rate of 2 to 3 million instructions per second (MIPS) and at burst rates exceeding 8 MIPS.

MC68020 PARTS

Figure 1 is a block diagram of the MC68020 with the various internal sections labeled. We'll briefly describe their functions.

The sequencer and control unit are the chip managers. They control internal buses, registers, and the execution unit.

The execution unit contains the program counter (PC), the address, and the data. The PC section calculates instruction addresses and manages pointers. The address section calculates operand addresses and stores the registers available to the user. The data section performs all data operations, such as immediate data value moves. It also contains the barrel shifter, which performs one-cycle shifts of any amount on data.

The bus controller manages cache and external memory accesses. It also provides control for the various parts of the 68020 microprocessor and interprets the nanorom information. This information is combined with decoding the instruction pipe to generate control for the micromachine.

The instruction prefetch and decode unit fetches and decodes an instruction for execution by the execution unit. The prefetch is a three-word-deep on-chip instruction store. It eliminates the need for the processor to sequentially fetch an instruction from external memory, decode and execute it, and fetch another.

Instead, because of the sequential nature of instruction accesses, the prefetch can anticipate the next access and make it before it is needed. Thus, external memory fetches are anticipated and overlapped with current processor execution.

The instruction addresses for the prefetch are calculated independently of data addresses, allowing for parallel accesses of instruction and data addresses. This simultaneous access will occur if the data access is from external memory and the instruction access is from the instruction cache. When this happens, a simultaneous instruction and data access occurs.

The 256-byte instruction cache increases performance by reducing the

(continued)

Paul F. Groepler is a systems applications designer in the High-end Applications Engineering Department at Motorola. James Kennedy is a software engineer in the MPU Design Department at Motorola. You can reach them at Motorola Inc., POB 6000, Austin, TX 78762.

number of instruction fetches, or bus cycles, from main memory. This lets system performance increase because the processor's bus use is decreased, freeing the bus for other system bus masters. Only instructions are stored in the cache. Data accesses must still be read from main memory.

Hardware and software may control the cache. Hardware control is in the form of the cache disable (CDIS pin), which can disable the cache. The CDIS pin has priority and overrides any software setting.

Software control is in the form of two control registers: the cache control register (CACR) and the cache address register (CAAR). The CACR and CAAR are organized as shown in figures 2 and 3. Bits 4 to 31 are unused and always read as zeros. The CACR lets the systems programmer enable or freeze the cache, clear an entry, or clear the entire cache. The CAAR is a 32-bit register that provides an address for cache control functions. This register is used only for the clear entry (CE) function in conjunction with the CACR.

Dynamic Bus Sizing

A nice feature of the MC68020 for the designer as well as the programmer is the dynamic bus. Now a designer need not worry about excessive hardware "glue" to interface to 16- or 8-bit peripherals and a programmer need not worry about word or long-word aligned data in data space. The MC68020 allows transfers of 8-, 16-, and 32-bit data between 8-, 16-, and 32-bit ports. The only requirement for data alignment is that it occur on a byte boundary. Instructions and any associated extension words must still fall on word address boundaries, but word/long-word alignment is no longer required for program space operands.

The processor lets misaligned transfers occur by determining the data

(continued)

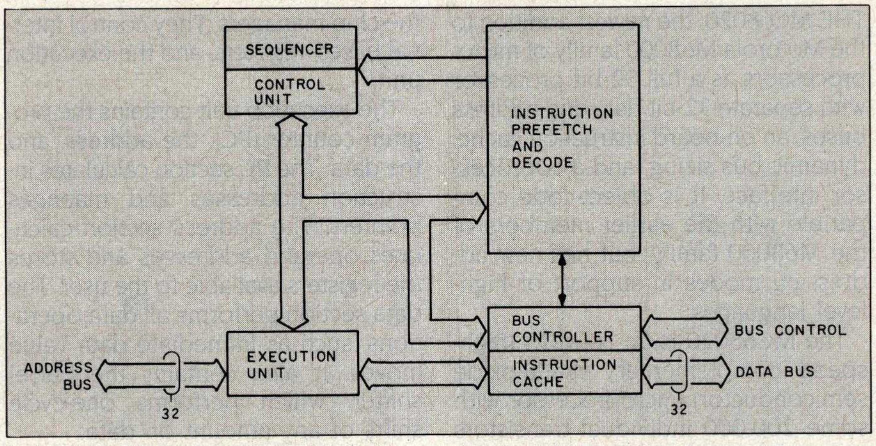

Figure 1: MC68020 *block diagram.*

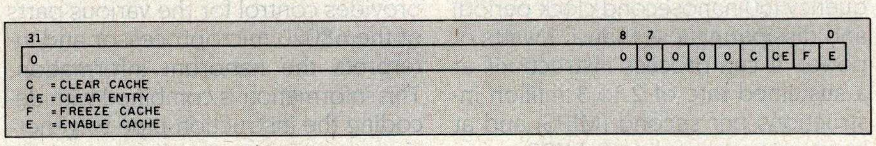

Figure 2: *The cache control register.*

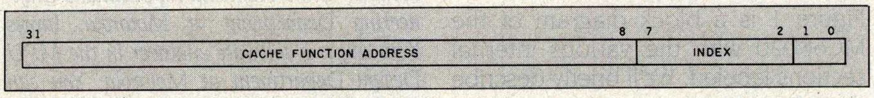

Figure 3: *The cache address register.*

port size during each bus cycle. By handshaking lines between the processor and external memory or peripherals, the MC68020 can transfer this mismatched or mis-sized data. Figure 4 illustrates the workings of the internal hardware that makes this possible and the alignment of operand bytes for an 8-, 16-, and 32-bit bus interfacing with the MC68020.

Misaligned operand transfers can lead to an increased number of bus cycles because the processor might not be able to successfully transfer the misaligned data across the port within one bus cycle. A normal transfer occurring from an aligned 32-bit operand address across a 32-bit bus to a 32-bit peripheral would take only one transfer cycle.

In a mis-sized transfer, as well as a normal transfer, the peripheral uses the DSACK0 and DSACK1 pins to signal to the processor that it has a bus width of 8, 16, or 32 bits. The processor outputs the operand transfer size using the SIZ0 and SIZ1 pins.

Notice that with an 8-bit peripheral, only data bits D31 to D24 need to be connected to the peripheral. Four bus cycles are necessary to complete this transfer with only 1 byte being moved across the bus per cycle.

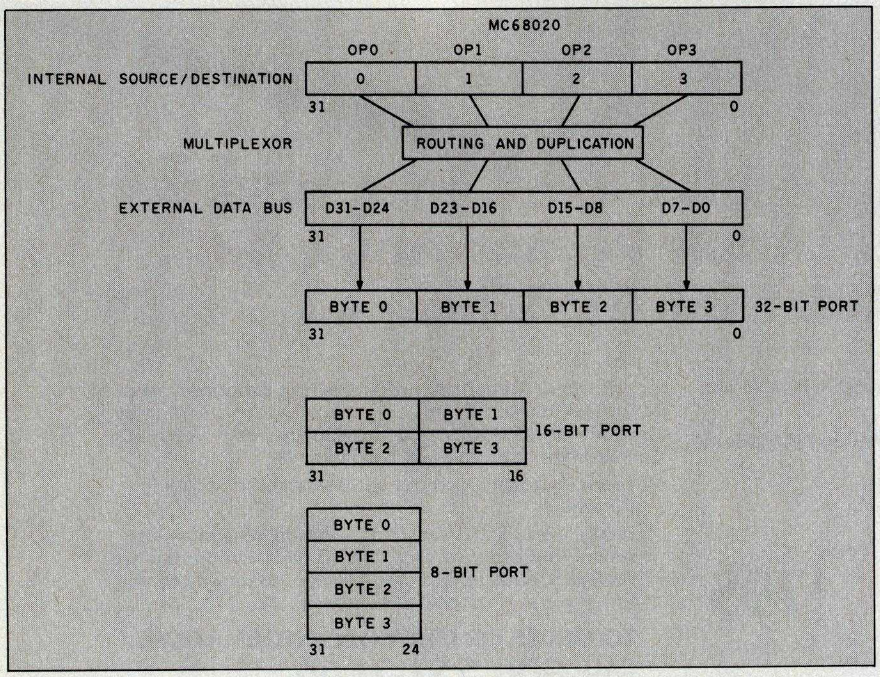

Figure 4: *The MC68020 interface to various port sizes.*

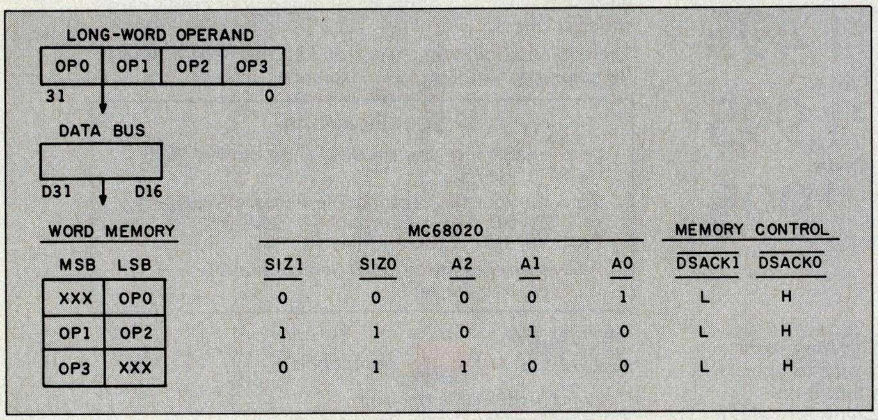

Figure 5: *Example of a misaligned long-word transfer to a word-wide bus.*

The MC68020 relaxes word and long-word alignment restrictions for data. It is now possible to execute an operand transfer across a memory boundary that only needs to be byte aligned. Even and odd word restrictions are gone. Some performance degradation can occur, due to extra bus cycles needed to transfer misaligned long-word or word data across boundaries.

Figure 5 shows a misaligned long-word transfer across a word-wide bus. In the example, a byte box with "xxx" denotes that the location in memory is not overwritten and remains unchanged. As you can see from this example, it is important for the system designer to control the enabling/disabling of the appropriate data buffers to avoid overwriting or misreading nonpertinent data during a misaligned cycle.

For clarity's sake, figure 5 also shows line A2 in offset of the transferred data into word memory. The first cycle runs with A2/A1/A0 being 001, showing an offset of 1 byte in memory. SIZ1/SIZ0 is 00, indicating that the processor has a long word left to transfer. The second cycle shows A2/A1/A0 with a 2-byte displacement, and SIZ1/SIZ0 showing the processor with 3 bytes left to transfer. The third cycle has no offset on the address pins and the SIZE pins indicating 1 byte left to transfer. There is a two-bus-cycle degradation here, but it is transparent to the programmer, letting him ignore the restrictions of data alignment.

Coprocessor Interface

Though the MC68020 is powerful, it might not have all the special commands or capabilities that a designer requires. For this reason, the designers of the MC68020 incorporated a general coprocessor interface and instructions. The coprocessor interface provides a means by which Motorola

(continued)

and other coprocessors (floating point, fast Fourier transform, or graphics processors) can extend the MC68020.

The coprocessor interface is designed to support synchronous operation between the MC68020 and up to eight coprocessors. With this interface, downward compatibility is possible because a coprocessor can be coupled with a main processor other than the MC68020 (e.g., 68008, 68000, 68010, 68012). All the main processor must do is provide instruction sequences that emulate the protocol of the coprocessor interface.

The coprocessor operates based on an F-line operation code, essentially the first word of a coprocessor instruction. It is so named because the hexadecimal F in the upper nybble of the instruction word causes the processor to flag the instruction during decode. The F-line indicates to the main processor that it must call the coprocessor for proper execution of the instruction. See figure 6 for the format of the F-line word.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-Id			TYPE			TYPE DEPENDENT					

Figure 6: *F-line format.*

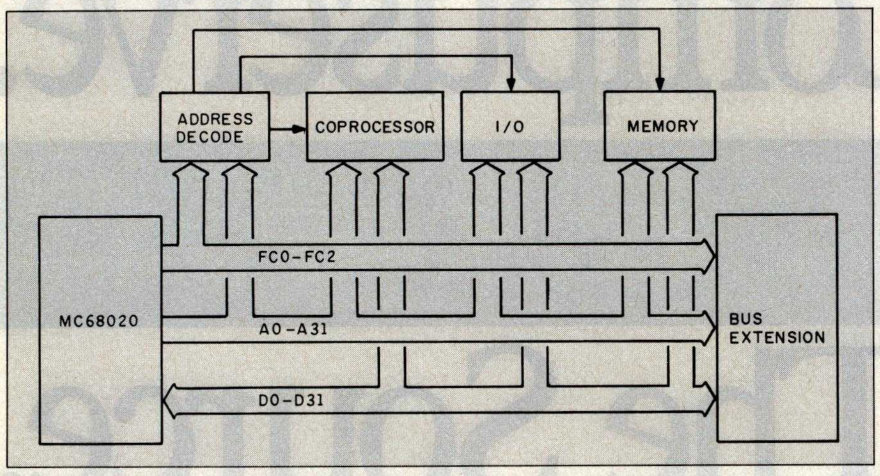

Figure 7: *Coprocessor system configuration.*

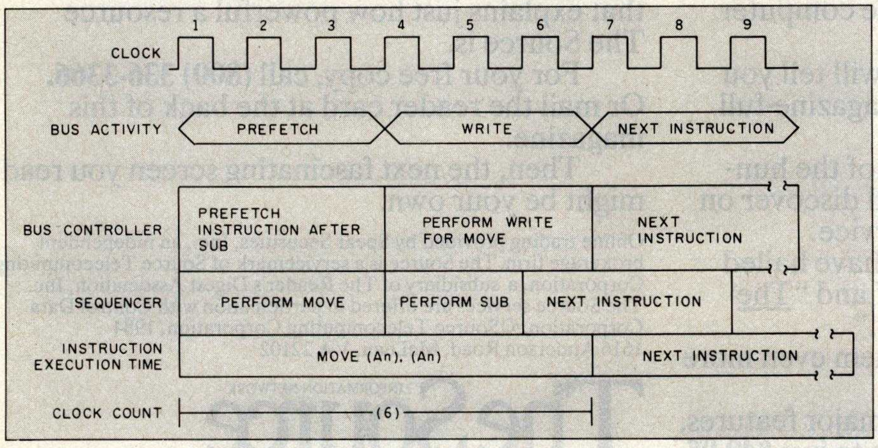

Figure 8: *Overlap example.*

The coprocessor identifier (Cp-Id) field identifies which coprocessor is to be selected. The Type field identifies which type of coprocessor operation is to be performed (branch, general, save, etc.).

Communication between the main processor and coprocessors is synchronous, but the main processor might not need to wait for the coprocessor to complete an instruction before it begins execution of its next instruction.

Hardware connection is a simple extension of the MC68000 bus interface and is shown in figure 7. The coprocessor is connected as a peripheral to the main processor and is selected based on combinations of function codes (FC2–FC0 are 111) and address bits (A19–A16 are 0010) as well as bits A15–13, described in figure 6.

Overlap

Overlap occurs when the sequencer and bus controller are operating on different instructions simultaneously. For example, in figure 8 a MOVE (An),(An) instruction and a SUB Dn,Dn instruction can operate concurrently for some of the total execution time. The overlap takes place during the external bus activity associated with the MOVE. Since for a certain clock time the bus controller is busy performing the write to external memory associated with the MOVE, the sequencer can continue with the next instruction, subtract (SUB). The SUB instruction does not require any external bus activity, so the sequencer alone can operate on it. This overlap time is shown from clocks 4 through 6.

Also note that part of the instruction following the SUB might have some of its execution time overlapped under the MOVE instruction. This occurs if calculations, such as effective address calculations, are needed to perform the instruction. An example of this would be if another MOVE instruction followed the SUB instruction.

Because the bus controller was performing an external bus cycle associated with the MOVE during the time the SUB was taking place internally,

(continued)

the execution time is attributed to the MOVE instruction alone. If the pipe had been depleted and the SUB instruction had not been inside, the described overlap would not have taken place.

This example illustrates an important point about this concurrent machine. With a sequential microprocessor (no prefetch and no concurrency of operation), instruction timing is easy to calculate. It is virtually impossible with a concurrent microprocessor such as the MC68020. Each individual instruction is dependent on the instruction previous to it and is subject to the rules built into the prefetch mechanism. The best timings that can be given are in terms of best, average, and worst-case boundaries. Performance ratios and benchmarks become requisite in measuring performance.

Programming

As shown in the programming model (figure 9), the MC68020 has eight 32-bit multifunction data registers, seven 32-bit general addressing registers, three 32-bit stack pointers (user, master, and interrupt), a 32-bit program counter, a 16-bit status register, a 32-bit vector base register, two 3-bit alternate function code registers, a 32-bit cache address register, and a 32-bit cache control register. The MC68020 is object-code compatible with the M68000 family but has several new addressing mode capabilities (table 1) and several new and enhanced instructions (table 2).

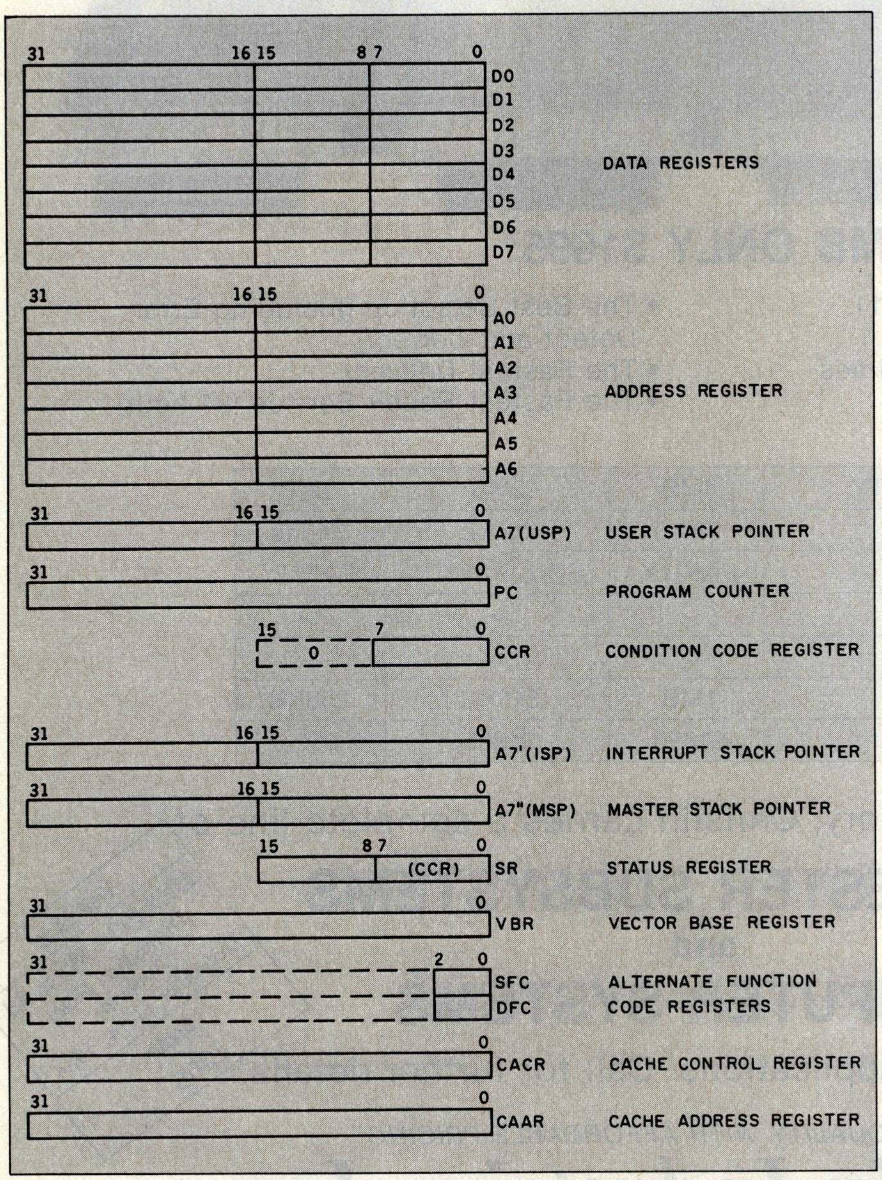

Figure 9: MC68020 *programming model.*

A principle in the MC68020 design is support for high-level language and system software implementation. This support is provided by the inclusion of special instructions that allow array bounds checking with a single instruction, safe manipulation of system queues, support for linked lists, expansion of system trap capabilities, and module support.

The MC68020 has three new 32-bit registers: the master stack pointer (MSP), cache control register (CACR), and cache address register (CAAR) (figure 8). The interrupt stack pointer is virtually the same as the M68000 family's supervisor stack pointer and therefore is not really a new register. The terminology has been changed to reflect a multiprocessing environment. The MSP was created to facilitate multiprocessing by letting each process have a small master stack area where process-specific exception data is stored, while maintaining a common large interrupt stack area among all the processes. The CACR clears the entire cache, clears a single cache entry, freezes the cache, and enables the cache. Each of these functions is controlled by simply setting a bit in the CACR. The CAAR is used with the cache clear entry function to clear a single entry in the cache.

Table 1 shows the MC68020 addressing modes. Of particular interest are the memory indirect and program counter memory indirect addressing modes.

There are two forms of memory indirect addressing and program counter memory indirect addressing: indirect pre-indexed and indirect post-

(continued)

indexed. Program counter memory indirect is similar to memory indirect addressing; however, where the address register is normally added into the calculation, the current program counter is used. The result is position-independent code where the memory pointer is accessed relative to the current program counter. In indirect pre-indexed and indirect post-indexed addressing modes, the base displacement (bd) and outer displacement (od) can be null (zero), word (16-bit), or long-word (32-bit) sizes.

Indirect pre-indexed addressing adds the index register into the address calculation before the memory indirection is performed. Indirect pre-indexed can be used to access operands through an array of pointers or through a pointer located in a record item or an array of records. The address register or program counter, index register, and base displacement are added together and used as the address of the memory pointer. The 32-bit memory pointer is fetched and the outer displacement is added to form the effective address.

Indirect post-indexed addressing performs the memory indirection and then adds the index register to calculate the effective address. Indirect post-indexed can be used to access an element of an array that is pointer addressed. The base displacement and address register or program counter are added to form the memory pointer address. The 32-bit quantity at the memory pointer address is fetched and added to the index register and outer displacement to form the operand's effective address.

Table 1: MC68020 *addressing modes.*

Addressing Modes	Syntax
Register direct	
Data register direct	Dn
Address register direct	An
Register indirect	
Address register indirect	(An)
Address register indirect with postincrement	(An)+
Address register indirect with predecrement	–(An)
Address register indirect with displacement	(d_{16},An)
Register indirect with index	
Address register indirect with index (8-bit displacement)	(d_8, An, Xn)
Address register indirect with index (base displacement)	(bd,An,Xn)
Memory indirect	
Memory indirect post-indexed	([bd,An],Xn,od)
Memory indirect pre-indexed	([bd,An,Xn],od)
Program counter indirect with displacement	(d_{16},PC)
Program counter indirect with index	
PC indirect with index (8-bit displacement)	(d_8,PC,Xn)
PC indirect with index (base displacement)	(bd,PC,Xn)
Program counter memory indirect	
PC memory indirect post-indexed	([bd,PC],Xn,od)
PC memory indirect pre-indexed	([bd,PC,Xn],od)
Absolute	
Absolute short	xxx.W
Absolute long	xxx.L
Immediate	$<data>

Notes:
Dn = Data Register, D0–D7.
An = Address Register, A0–A7.
d_8,d_{16} = A two's-complement or sign-extended displacement; added as part of the effective address calculation; size is 8 or 16 bits (d_{16} and d_8 are 16- and 8-bit displacements); when omitted, assemblers use a value of zero.
Xn = Address or data register used as an index register; form is Xn.SIZE*SCALE, where SIZE is .W or .L (indicates index register size) and SCALE is 1, 2, 4, or 8 (index register is multiplied by SCALE); use of SIZE and/or SCALE is optional.
bd = A two's-complement base displacement; when present, size can be 16 or 32 bits.
od = Outer displacement, added as part of effective address calculation after any memory indirection; use is optional with a size of 16 or 32 bits.
PC = Program Counter.
<data> = Immediate value of 8, 16, or 32 bits.
( ) = Effective address.
[ ] = Use as indirect address to long-word address.

Scaling and Suppression

An index register can be scaled—the value in the register is read and then logically shifted (zero fill) zero, one, two, or four bit positions to the left before it is used. This has the effect of multiplying the value in the register by 1, 2, 4, or 8. The original value in the register is not affected by this operation. Using scaling, the same index value can be used to point to individual bytes, words, long words, and quad words, without disrupting the value.

Let address register A0 be used as an index and contain the value 3. The A0*1 (assuming 0 to be the first element in the array) will point to the fourth element in a byte-wide array, A0*2 will point to the fourth element in a word-wide array, A0*4 will point to the fourth element in a long-word-wide array, and A0*8 will point to the fourth element in a quad-word-wide array. This scaling takes no overhead on the MC68020.

The suppression of the base address register or program counter allows the use of any index register in place of the base register. Since data

(continued)

registers can be used as index registers, this gives the MC68020 the ability to have addresses in data registers. Also, with suppression of the program counter the user has access to program space.

BIT-FIELD INSTRUCTIONS

The MC68020 has eight new bit-field manipulation instructions over the previous M68000 instruction set. These instructions can be used to manipulate individual bits in registers or memory. The bit-field instruction mnemonics are described in table 2.

A bit field is simply an array of bits. It can be small enough to be contained in a register or large enough to require millions of bytes of memory. Some examples of bit-field applications are bit-mapped graphics, communications with packed data, and assembler op-code construction.

In each bit-field instruction, the field selection is specified by a field offset and field width. The field offset denotes the starting bit of the field in bits from the base address, and the field width determines the number of bits to be included in the field. The base address is the effective address and can be in memory or a data register.

In a data register, the offset starts with the leftmost bit, bit 31, and the width determines the amount of bits to the right of the offset. Register wraparound is allowed; that is, if the combination of offset and width extend the bit field past bit 0 in the register, the field wraps back around

(continued)

Table 2: MC68020 *instruction set summary.*

Mnemonic	Description
ABCD	Add decimal with extend
ADD	Add
ADDA	Add address
ADDI	Add immediate
ADDQ	Add quick
ADDX	Add with extend
AND	Logical AND
ANDI	Logical AND immediate
ASL,ASR	Arithmetic shift left, right
Bcc	Branch conditionally
BCHG	Test bit and change
BCLR	Test bit and clear
BFCHG	Test bit field and change
BFCLR	Test bit field and clear
BFEXTS	Signed bit field extract
BFEXTU	Unsigned bit field extract
BFFFO	Bit field find first one
BFINS	Bit field insert
BFSET	Test bit field and set
BFTST	Test bit field
BRA	Branch
BSET	Test bit and set
BSR	Branch to subroutine
BTST	Test bit
CALLM	Call module
CAS	Compare and swap operands
CAS2	Compare and swap dual operands
CHK	Check register against bound
CHK2	Check register against upper and lower bounds
CLR	Clear
CMP	Compare
CMPA	Compare address
CMPI	Compare immediate
CMPM	Compare memory to memory
CMP2	Compare register against upper and lower bounds

Mnemonic	Description
DBcc	Test condition, decrement, and branch
DIVS,DIVSL	Signed divide
DIVU,DIVUL	Unsigned divide
EOR	Logical exclusive OR
EORI	Logical exclusive OR immediate
EXG	Exchange registers
EXT	Sign extend
JMP	Jump
JSR	Jump to subroutine
LEA	Load effective address
LINK	Link and allocate
LSL,LSR	Logical shift left, right
MOVE	Move
MOVEA	Move address
MOVE CCR	Move condition code register
MOVE SR	Move status register
MOVE USP	Move user stack pointer
MOVEC	Move control register
MOVEM	Move multiple registers
MOVEP	Move peripheral
MOVEQ	Move quick
MOVES	Move alternate address space
MULS	Signed multiply
MULU	Unsigned multiply
NBCD	Negate decimal with extend
NEG	Negate
NEGX	Negate with extend
NOP	No operation
NOT	Logical complement
OR	Logical inclusive OR
ORI	Logical OR immediate
PACK	Pack BCD
PEA	Push effective address

Mnemonic	Description
RESET	Reset external devices
ROL,ROR	Rotate left, right
ROXL,ROXR	Rotate with extend left, right
RTD	Return and deallocate
RTE	Return from exception
RTM	Return from module
RTR	Return and restore condition codes
RTS	Return from subroutine
SBCD	Subtract decimal with extend
Scc	Set conditionally
STOP	Stop
SUB	Subtract
SUBA	Subtract address
SUBI	Subtract immediate
SUBQ	Subtract quick
SUBX	Subtract with extend
SWAP	Swap register words
TAS	Test operand and set
TRAP	Trap
TRAPcc	Trap conditionally
TRAPV	Trap on overflow
TST	Test operand
UNLK	Unlink
UNPK	Unpack BCD
Coprocessor Instructions	
cpBcc	Branch conditionally
cpDBcc	Test coprocessor condition, decrement, and branch
cpGEN	Coprocessor general instruction
cpRESTORE	Restore internal state of coprocessor
cpSAVE	Save internal state of coprocessor
cpScc	Set conditionally
cpTRAPcc	Trap conditionally

The MC68020 is an improvement on the M68000 family.

and continues with bits 31, 30, etc. Table 3 shows the Motorola assembler syntax and examples of the bit-field instructions.

Division and Multiplication

The MC68020 has several enhancements to the original M68000 multiply and divide instructions. The most important of these enhancements is the ability to have 32-bit operands for multiplication with a 64-bit result, and a 64-bit dividend with a 32-bit divisor and quotient for division.

The MC68020 also has special multiply and divide instructions for high-level languages where the result is the same size as the operands. That is, a 32-bit operand times a 32-bit operand yields a 32-bit result. This is equivalent to multiplication in Pascal or C, where an integer times an integer results in an integer of the same size data type. If overflow occurs, it will be reflected in the setting of the condition codes after the operation. For division there is provision for a 32-bit dividend divided by a 32-bit divisor to yield a 32-bit quotient with no re-

Table 3: *Bit-field instructions, syntax, and examples.*

Instruction	Motorola Assembler Syntax	Assembler Example
BFCHG	BFCHG <EA>{offset:width}	BFCHG (A0){D0:7}
BFCLR	BFCLR <EA>{offset:width}	BFCLR D1{25:10}
BFEXTS	BFEXTS <EA>{offset:width},Dn	BFEXTS (A3){D2:5},D7
BFEXTU	BFEXTU <EA>{offset:width},Dn	BFEXTU D5{2:5},D1
BFFFO	BFFFO <EA>{offset:width},Dn	BFFFO (A6){D0:32},D7
BFINS	BFINS <EA>{offset:width},Dn	BFINS D4{6:9},D2
BFSET	BFSET <EA>{offset:width}	BFSET D3{30:9}
BFTST	BFTST <EA>{offset:width}	BFTST D1{0:32}

<EA> = effective address of the base of the bit field
offset = bit offset from base address of bit field to start to bit field
width = bit width of bit field from 1 to 32 bits
Dn = data register

Table 4: *Division and multiplication syntax and operation.*

Instruction	Motorola Assembler Syntax	Operation
DIVS.W	DIVS.W <EA>,Dn	32/16 --> 16r:16q
DIVS.L	DIVS.L <EA>,Dq	32/32 --> 32q
DIVS.L	DIVS.L <EA>,Dr:Dq	64/32 --> 32r:32q
DIVSL.L	DIVSL.L <EA>,Dr:Dq	32/32 --> 32r:32q
DIVU.W	DIVU.W <EA>,Dn	32/16 --> 16r:16q
DIVU.L	DIVU.L <EA>,Dq	32/32 --> 32q
DIVU.L	DIVU.L <EA>,Dr:Dq	64/32 --> 32r:32q
DIVUL.L	DIVUL.L <EA>,Dr:Dq	32/32 --> 32r:32q
MULS.W	MULS.W <EA>,Dn	16 x 16 --> 32
MULS.L	MULS.L <EA>,Dl	32 x 32 --> 32
MULS.L	MULS.L <EA>,Dh:Dl	32 x 32 --> 64
MULU.W	MULU.W <EA>,Dn	16 x 16 --> 32
MULU.L	MULU.L <EA>,Dl	32 x 32 --> 32
MULU.L	MULU.L <EA>,Dh:Dl	32 x 32 --> 64

<EA> = effective address of source operand
Dn = data register
Dq = quotient in data register
Dr = remainder in data register
Dh = high 32 bits of product in data register
Dl = low 32 bits of product in data register

mainder. Several variations of syntax and operations exist for divide and multiply instructions. For more information, see the Motorola assembler syntax and operation examples shown in table 4.

BINARY-CODED DECIMAL

Two MC68020 instructions, PACK and UNPACK, can store BCD (binary-coded decimal) data in packed form (two digits per byte) and then be expanded after calculations. The PACK instruction reduces 2 bytes of numeric data into a single byte, while the UNPACK instruction reverses this operation. In both cases, a user-defined constant is added to the original value to allow conversion from or to ASCII, EBCDIC (extended binary-coded-decimal interchange code), or any other data format.

HIGH-LEVEL LANGUAGES AND SYSTEM SOFTWARE

The MC68020 has extended the bounds-checking capability of the M68000 family with the introduction of two new instructions, CHK2 and CMP2 (check 2 and compare 2). CHK2 and CMP2 perform comparisons on the upper and lower bounds and can be signed or unsigned. The CMP2 instruction sets the condition codes according to the result of the operation. The CHK2 instruction sets the condition codes and causes a system trap if either boundary condition fails.

The MC68020 also offers other new security and system-level instructions. The CAS and CAS2 instructions use the same read-modify-write cycle as the M68000's TAS (test and set). These operations are indivisible and noninterruptible, which ensures data security in single and multiprocessor systems.

The CAS (compare and swap) instruction compares the contents of a data register (the compare register) to the operand at the effective address. If the operand and the contents of the data register are equal, the contents of a second data register (the update register) are used to update the operand at the effective address. If the

(continued)

compare register and the operand are not equal, the operand is unchanged, but the value in the compare register is updated with the operand at the effective address. The CAS2 instruction is basically the same as CAS, but there are two compare registers (upper and lower bound), two update registers, and two operands at two different effective addresses. The CAS and CAS2 instructions are useful for updating system counters and for insertion and deletion from linked lists.

The MC68020 also has expanded system trap capabilities in the form of the TRAPcc instruction, where any condition code is allowed to be the trapping condition. The TRAPcc instruction can be followed by a word or long-word quantity that can be used to convey information to the trap handler, such as a high-level language statement number or other debugging information.

The MC68020 introduces module support to the M68000 family. Modules are high-level subroutines that can have different levels of protection or access. Two new instructions, CALLM and RTM, support this module implementation.

The CALLM instruction initiates the module call by referencing a module descriptor. The module descriptor contains access information, control information, and the entry point for the called module. If the module access is valid, the CALLM instruction creates a module stack frame, stores the current module state in that frame, and loads a new module state from the module descriptor.

The RTM instruction removes the module state that was stored on the module stack frame and returns to the calling module. The MC68020 module support is broken into two types: type 0 where there is no access level change and type 1 where the access level can be changed. No external hardware is necessary for type 0, but for type 1 CALLM, the MC68020 relies on external hardware (a memory management unit) to verify that calling modules possess the proper access level for the called module.

VIRTUAL MEMORY

The MC68020 supports virtual memory, the ability to make a small amount of main memory look like a large or infinite amount of memory by using secondary storage devices to swap currently executing code segments into the main memory. In a virtual memory system, the processor has access to a limited amount of fast main memory (the physical memory of the system), while the user writes programs that might require millions of bytes of memory (the virtual memory of the system).

If the processor attempts to access a memory location not currently residing in physical memory, a page fault occurs. The processor suspends the current instruction until the required memory is moved into physical memory from slower but larger secondary storage. When the required program segment is in physical memory, the instruction is allowed to complete execution. All this activity is transparent to the user, so physical memory appears to be the same size as virtual memory. Virtual memory size has been increased from a 16-megabyte direct addressing range in the MC68010 to 4 gigabytes in the MC68020.

CONCLUSION

The MC68020 is a fully compatible member of, and an improvement on, the M68000 family of processors. It is backed by the same powerful software and hardware design-support tools that back other members of the M68000 family. For more information, see the *MC68020 32-Bit Microprocessor User's Manual* (Prentice-Hall, 1984) and the three-part article by Thomas W. Starnes, "Design Philosophy Behind Motorola's MC68000" (April–June 1983 BYTE). ■