The MC68020 and System V/68

Bob Beims
M68000 Applications Engineer
Motorola Microprocessor Products Division
William Cannon at US 290 West
Austin, Texas 78735

As microcomputer systems have evolved over the past few years, most of the "new" features that have appeared are not really new at all; rather, they were originally implemented in main frame and mini-computers. Even the architectures of the microprocessors themselves have their roots in the architectures of larger machines, such as the DEC PDP-11 and VAX-11 mini-computers. In addition to the hardware features that have migrated from the mini-computer world into microcomputers, software features are also being implemented on micro-computers that were previously found in minis. Probably the most prominent example of this is the emergence of multi-user, multi-tasking operating systems for microcomputers, with the UNIX operating system coming to the front as the standard OS for most supermicros (UNIX is a trademark of AT&T Bell Laboratories). In order to implement UNIX on any computer system, the main CPU of that system must have a fairly high performance capacity, with an advanced instruction set, to support the demands of this operating system. In the past three years, one microprocessor has emerged as the de facto standard for implementing UNIX based micro-computers, the MC68000 and its related family of processors and peripherals. This paper will explore the relationship between the newest member of the M68000 Family, the MC68020 32-bit microprocessor, and Motorola's version of the UNIX operating system, System V/68.

The development of a solid operating system obviously does not happen over night, even though it would seem that UNIX is a new kid on the block, judging from the burst of new product introductions that use it. However, UNIX is now almost 14 years old, and it has been present in microcomputers for over 3 years. In fact, two years ago Mini-Micro Systems (Nov. '82) published an article on supermicro computers, and listed twenty commercially available machines that used UNIX (or a UNIX derivative or look-alike) as their operating system. This indicates that this operating system is quite mature, in its capabilities, in the amount of support available for it, and in the number of application programs written to run under it. The main reason for all of the activity in the last few months is, of course, due to the new posture that AT&T has taken with respect to UNIX. Motorola has taken advantage of its strong position in the UNIX market place (of the twenty machines listed by Mini-Micro two years ago, 17 were MC68000 based) and has entered into an agreement with AT&T to support UNIX for the M68000 Family with a licensed System V version of UNIX, System V/68. In fact, Motorola is the first company to have its UNIX System V port verified by AT&T. The implication of this agreement between Motorola and AT&T, and the existing hardware and software base that has used the M68000 Family for UNIX in the past, is that the M68000 processors will continue to be the standard for UNIX machines.

There are several key questions that this paper will attempt to answer: Why is the M68000 Family so commonly used for UNIX machines? What unique features are implemented on the MC68020 that help it to be a better UNIX machine? What is Motorola's System V/68, and what is its relationship to AT&T System V? And finally, what kind of system level support is being offered by Motorola to help system integrators develop MC68020-System V/68 machines?

The M68000 Architecture - What Makes It Good For UNIX.

As was mentioned above, in 1982 there were at least 17 companies that had developed microcomputer hardware to support UNIX, and all of them used the MC68000 microprocessor. Now in 1984, there are 70 vendors that have supermicros on the market (according to Systems & Software, Sept. '84); and most of those run UNIX (or a look-alike) and use the MC68000 or MC68010 as the main processor. The main reasons for the popularity of the M68000 Family are 1) a clean, 32-bit architecture with a linear address space, 2) high performance, with processors readily available that run at clock

speeds of up to 12.5 MHz, 3) flexibility: these processors have features such as multiprocessor support, memory management and virtual memory support that are a must for the design of a high-performance UNIX machine, and 4) availability, the MC68000 was the first widely available machine with a 32-bit architecture, which is very much like the architecture of the DEC PDP-11 family (which is where UNIX has been used the most). The MC68020 is poised to take advantage of the M68000 Family popularity in the UNIX market place, and will reinforce all of the above reasons for being the choice for the microprocessor at the heart of UNIX based supermicros. The MC68020 uses the exact same architecture as is implemented in the MC68000 and the MC68010, so that porting UNIX (and applications that run under it) to the new processor will be straight forward. This new processor is at least four times faster than its predecessors, and UNIX needs all of that performance to meet the needs of the emerging office automation environment. The MC68020 has several advanced features that make it even more flexible in multi-processor environments, such as the ones that are used in supermicro systems. And finally, it is available now for system prototyping, making it one of the earliest 32-bit microprocessors on the market.

The M68000 Architecture

Before detailing some of the new features of the MC68020 that help it to improve the performance of UNIX, a short review of the basic architecture of the M68000 Family of processors will point out the reasons for their popularity. The main precept that underlies all of the architectural features of the processors is that they are designed to be general purpose machines. Therefore, the programmer's model, instruction set, and hardware interface are all designed to be non-dedicated, flexible, and regular. The programmer's model, shown in figure 1, illustrates these concepts. The register set consists of two groups of eight 32-bit registers, the data registers D0-D7 and the address registers A0-A7. Of these 16 registers, only one has a function that is dictated by the processor itself; A7 is used as the stack pointer when it is needed for interrupts or other exceptions, or when implicitly used by subroutine calls and returns. All of the remaining 15 registers are completely general purpose; they can be used for any function defined by the programmer. This large register space, and the non-dedicated manner in which they are used, results in highly efficient compiler code, which is particularly important to UNIX, which is written mostly in the C programming language.

The instruction set of the M68000 Family is not particularly large (the MC68000 only implements 56 instruction types), but it is quite efficient in terms of memory usage and

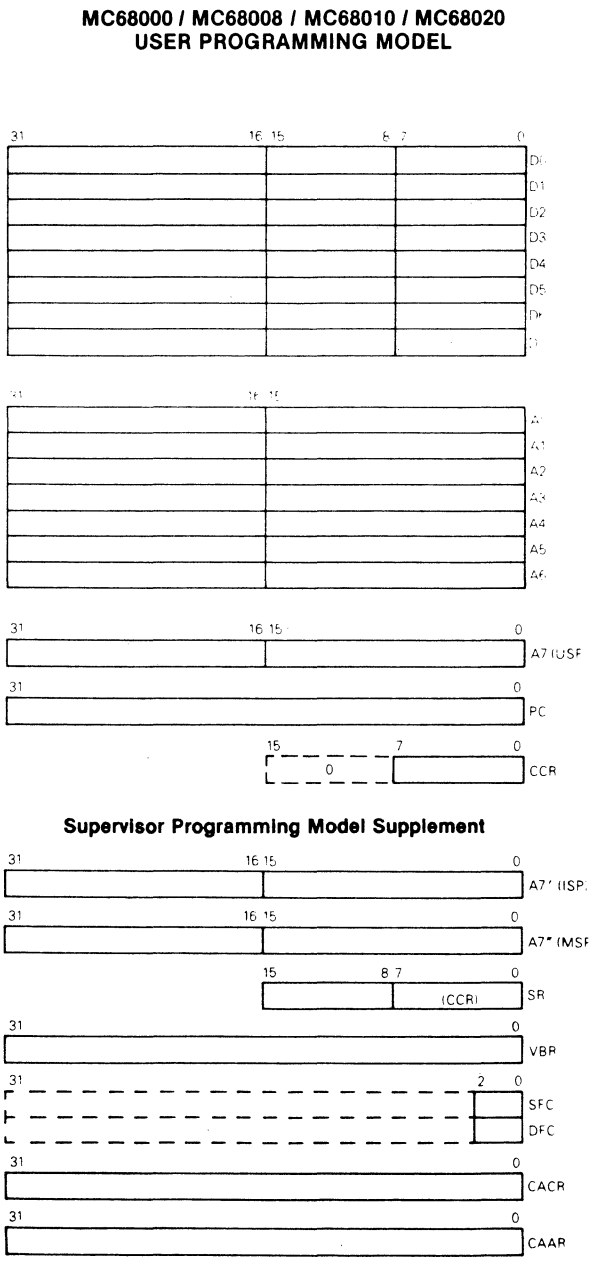


Figure 1. MC68020 Programmer's Model.

execution speed. Most of this efficiency is the result of the powerful addressing modes (14 of them on the MC68000) that can be used by any instruction. Figure 2 shows a summary of the MC68020 instruction set (which is a superset of the MC68000 and MC68010 instruction sets) and addressing modes. Note that the instruction set supports a 1 & 1/2 address architecture, where one operand for an arithmetic or logical operation is always a data register and the

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 and 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
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 and 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
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and 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

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 twos-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 twos 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

Figure 2. M68000 Family Instruction Set and Addressing Mode Summaries.

other operand may be another register or a memory location that is determined by the addressing mode that is used. The M68000 architecture does support one variation on the classical 1 & 1/2 address machine: the register operand can be either the destination or the source, where it is always the destination in the classical machine. Also, there is one very important exception to this 1 & 1/2 address limitation: the MOVE instruction, which is used for general data movement where no ALU operation is performed on the data, is a true 2 address instruction so that both the source and the destination addressing modes can specify a memory location. The move instruction illustrates a very important concept that the M68000 architects had in mind when they were defining the instruction set; that a small number of the instructions implemented by any machine will be used most of the time, and thus they should be very fast and very flexible. The MC68000 designers performed extensive studies on instruction usage and optimized the most commonly used ones for higher performance and flexibility. Furthermore, these few instructions are most useful when coupled with a robust set of addressing modes. For example, the MOVE instruction can take on over 1500 different forms when coupled with all of the combinations of addressing modes that are available.

As was mentioned earlier, the bulk of the UNIX operating system is written in C; thus, any microprocessor that can efficiently support the C language will implicitly support the UNIX operating system efficiently. The instruction set and register set of the M68000 Family processors are well suited to the C language, mainly because of their general purpose nature and the linear address space that is supported by the addressing modes. One of the other features that make the processors good for UNIX is the extensive exception handling mechanism that is supported. Any time that a user program attempts to do something that it shouldn't, either by mistake or on purpose, the processor will detect it, prevent the action, and let the operating system know about it in a graceful manner. This protection mechanism extends off of the chip as well, since the processor presents information to the outside world that identifies the purpose and privilege of every bus cycle. By using this information, an external memory management unit can provide protection against unauthorized use of certain memory areas, and can assign attributes such as read/write or execute only to segments of memory.

What the MC68020 Does to Improve UNIX Performance

When the MC68020 designers began working on that processor, an obvious question that was

asked was: since the MC68000 and MC68010 are already good, complete, high-performance machines, how do we improve upon them? The most obvious answer was to simply speed the processors up, while maintaining the exact same programmer's model and instruction set. This was the primary goal of the chip designers; to achieve the highest possible performance, given the constraints of the current technology, while maintaining absolute user code compatibility with the earlier machines. The MC68020 does this quite admirably due to several factors:

- The processor runs at a clock frequency of 16.67 MHz in contrast to the 8 to 12.5 MHz clock limit of the earlier processors.
- The MC68020 uses 32-bit data paths, both internally and externally, in contrast to the 16-bit paths used by its predecessors.
- A pipelined internal architecture is used, with three independent 32-bit ALUs for data manipulation and address calculations, while the previous M68000 processors use a single 16-bit ALU.
- An on-chip instruction cache (256 bytes deep, organized as 64 32-bit entries) is included to help bridge the gap between processor speed and memory speed in order to keep the pipeline full as much as possible.
- A 32-bit barrel shifter is included that allows very fast shift operations, regardless of the shift count.

These features, which are purely performance oriented, give the MC68020 at least four times the processing speed of the MC68000, and up to six or more times the speed of the MC68000 for certain applications. Since the MC68000 has been widely benchmarked by the computer industry and is generally considered to be a 0.6 MIPS machine (@ 8 MHz, no wait-states), this indicates that the same piece of code that runs on the MC68000 at that speed will execute on the MC68020 at above 2 MIPS, and possibly as fast as 3.5 MIPS. Indeed, early benchmarks, performed both inside Motorola and by outside firms, show that the MC68020 easily reaches performance levels of 2.5 MIPS for general programs, and up to 4 MIPS for optimized programs (that take advantage of the cache and the 32-bit ALUs).

The next question that the chip designers asked themselves, after considering performance and compatibility, was: what can be done to improve upon the MC68000 architecture? After several studies and input from existing M68000 users, the following additions were made to the programmer's model and instruction set:

- Several instructions were enhanced to support 32-bit operands.
- Bit-field instructions were added to better support compilers and certain hardware applications, such as graphics.
- Two of the existing addressing modes were extended significantly to allow scaled indexing, 32-bit displacements, and memory indirection.
- A new stack pointer (in addition to the two already present in the MC68000) was added to simplify task switching for operating systems.
- A general purpose, flexible coprocessor interface was developed to allow easy expansion of the instruction set and hardware features. The MC68881 Floating Point Coprocessor is the first device from Motorola to take advantage of this interface.
- Several miscellaneous instructions were added to support EBCDIC and ASCII strings, modular programming, multi-processor communications, and compiler run-time exception protection.
- Exception handling is extended for better debug support, with a new program trace facility and a program breakpoint facility.

Figure 3 gives a summary of the new instructions of the MC68020. The two features that are of most interest to UNIX programmers (or more specifically, C programmers) are the bit field instructions and the new addressing modes. The other features will also improve the overall performance of the MC68020 in a UNIX system, but not as significantly as these two. The following paragraphs discuss the bit field instructions and new addressing modes, while more details about the other additions to the instruction set can be obtained from the MC68020 User's Manual from Motorola.

MC68020 NEW INSTRUCTIONS

BFEXTx	**CHK2.s**	**CALLM**
BFINS	**CMP2.s**	**RTM**
BFFFO	**Tcc**	**CAS**
BFTST	**TPcc.s #xxx**	
BFCLR	**PACK**	
BFSET	**UNPK**	
BFCHG		

Figure 3.

Bit Field Instructions

In many advanced programs written in C, variables that will not assume very large values are often packed in memory through the use of bit fields. For example, one 6-bit, one 4-bit and two 3-bit variables can be packed into a 16-bit integer rather than using an entire integer for each variable, thereby conserving memory. However, most microprocessors do not have a specific mechanism designed to manipulate such data structures. For example, if one of the 3-bit variables were contained in bits 7-9 of a word, then the following code segments might be used to fetch or store that variable:

```
FETCH   MOVE.W   #$0380,D0      Load Mask
        AND.W    [ea],D0        Get field
        LSR.W    #7,D0          Justify

STORE   LSL.W    #7,D0          Shift field
        MOVE.W   #$FC7F,D1      Load mask
        AND.W    [ea],D1        Get word
        OR.W     D0,D1          Add field
        MOVE.W   D1,[ea]        Store it
```

This code is fairly straight forward and it can be generated by a compiler, however it does require a significant amount of memory and does not make good use of the register set, since an extra working register is needed for intermediate calculations. The bit field instructions on the MC68020 support operations such as the ones coded above with single instructions that are very flexible and fast. The general format of the bit field instructions is:

label BFopc <ea>{offset,width},Dn

where "opc" is one of the bit field operation specifiers (EXTract Signed or Unsigned, INSert, Find First One, TeST, SET, CLeaR and CHanGe), [ea] is any of the general addressing modes, "offset" is the number of bits from the effective address where the field starts, and width is how wide the bit field is. Both the offset and the width can be constant values, specified in the instruction stream, or variable values contained in any data registers. The offset can take on any value in the range of -2^{31} to $+2^{31}-1$ and the width of the field can be from 1 to 32 bits. The BFTST, BFSET, BFCLR and BFCHG instructions do not use a destination data register, since they simply read the specified bit field, set the condition codes according to the source value, and write the modified bit field back to memory (the modified value is all 1's, all 0's or the compliment of the field for the BFSET, BFCLR and BFCHG instructions, respectively; the BFTST operation does not modify the field). The Find First One operation (the mnemonic for which is BFFFO) scans the specified bit field, starting with the most significant bit, and stops when the first 1 is found. The offset value of that bit is then returned in the destination data register for use as a pointer. Finally, the BFEXTS , BFEXTU and BFINS instructions are used to move bit fields in and out of data registers. For example, the previous code segments could each be replaced by a single instruction as follows:

```
FETCH   BFEXTU   [ea]{#6,#3},D0

STORE   BFINS    D0,[ea]{#6,#3}
```

Note that the offset calculation for a bit field is from the most significant bit of the byte specified by the effective address to the most significant bit of the bit field, and that bit fields may span any byte, word or long-word boundary in an arbitrary fashion. An example will be given later to show the power of this addressing scheme when coupled with the new addressing mode extensions of the MC68020.

In addition to the use of bit fields to reduce the amount of storage required by a program, the MC68020 bit field instructions are also designed to be used for hardware control functions and complex data manipulations. For example, a UNIX programmer might use bit field data types to map variables onto the hardware control registers of a device that he is writing a driver for; then the bit field manipulation instructions such as BFTST, BFSET and BFCLR can be used to toggle control registers in the device or check status fields easily. The compiler is better able to generate high-performance code for such applications since it does not have to use intermediate registers, and the hardware features of the MC68020, particularly the barrel shifter, can be fully utilized to get the highest possible performance. Also, the ability to manipulate arbitrary sized fields that can be addressed in a fully general, linear manner, is of particular interest in bit mapped graphics applications. For example, the bit field instructions are quite useful in developing raster op routines.

New Addressing Modes

As mentioned earlier, the 14 addressing modes implemented by the MC68000, coupled with the 56 powerful instructions of that machine provide a very high performance architecture that can support compilers very well. However, there are some routine address calculations, that are not directly supported by the MC68000, that are performed by almost all high level languages. For example, consider the following code segment that is used to locate a long word value in an array that is located at some

constant offset from a data area pointer. In this example, D0 contains the offset, in long words, of the desired value from the start of the array, which is located more than -128 or +127, but less than -32k or +32k-1 (ie. within the reach of a 16-bit displacement), from the data area pointer address, which is contained in A0.

```
MOVE.W    D0,D1          Get offset
LSL.W     #2,D1          x 4
LEA       off(A0),A1     Calc base
MOVE.L    0(A1,D1.W),Dn
```

This simple address calculation requires two intermediate registers, and even though this code can be easily generated by a compiler and will execute quite fast, it could be eliminated by the addition of two capabilities to the addressing mechanism of the MC68000: index scaling and displacements larger than 8 bits (note that in the above example, if "off" where larger than 15 bits, then another LEA instruction would be required to add that offset to the base in two calculations).

The MC68020 architects had two methods with which they could extend the addressing modes of the machine, either they could use one or more of the four remaining addressing mode encodings in the opcode word (the addressing modes are encoded in two three bit fields, all but four of the encodings are currently defined), but that approach would have restricted the potential for future enhancements. So, rather than developing completely new addressing modes, two of the existing addressing modes were extended. These two addressing modes are the address register indirect with index and displacement, and Program Counter relative with index and displacement. Both of these addressing modes use bits in extension words to the op code word to fully specify the addressing mode, and some of the bits in those extension words were not used by the MC68000 addressing modes; this allowed for a clean addition to these mechanisms. In review, the format of the address register indirect with index and displacement is shown below:

disp(An,Xn.s)

where "disp" is an 8-bit displacement, An is any address register that is used as the base, and Xn is any address or data register that is used as an index from the base plus the displacement. The "s" specifies how much of the index register is to be used in the address calculation, the lower word or the entire long word. The additional capabilities that have been added by the MC68020 are:

- Displacements may be up to 32 bits, in two's compliment form, thus allowing a range of +/- 2 gigabytes.
- The index register may be multiplied by 1, 2, 4 or 8 for use in the address calculation, and the original index value is left unchanged in the register. This allows an index to specify an offset in bytes, words, long words or quad words.
- One level of memory indirection may be performed, where an address is calculated, a long word value is fetched from that address, that value is then used as the base address in further calculations and the operand is then fetched at the resultant address. The indirect reference may occur before or after the index calculation is performed, and two displacements may be specified, one for use before the indirect reference and one for use after it.

Figure 4 gives a summary of the two enhanced addressing modes. When all of the permutations of displacement, index, base register and indirect options are combined, these new addressing capabilities increase the original two addressing modes to eighty distinct modes. Obviously, no programmer is going to remember eighty different modes, but since the rules for usage of the new addressing mechanisms are simple and regular (there are no exceptions to the above rules), the programmer simply builds the addressing mode that he wants by using those rules and not by remembering specific cases. This not only makes these addressing modes easy to use for the assembly language programmer, but also make them quite useful to compiler writers since they can be generated easily by simple algorithms.

MC68020 ADDITIONAL ADDRESSING MODES SYNTAX SUMMARY

DIRECT:

<ea> = (bd.size, An, Xn.size*scale)	Data Space
<ea> = (bd.size, PC, Xn.size*scale)	Program Space

INDIRECT, PRE-INDEX:

<ea> = ([bd.size, An, Xn.size*scale], od.size)	Data Space
<ea> = ([bd.size, PC, Xn.size*scale], od.size)	Program Space

INDIRECT, POST-INDEX:

<ea> = ([bd.size, An], od.size, Xn.size*scale)	Data Space
<ea> = ([bd.size, PC], od.size, Xn.size*scale)	Program Space

bd, od = Base and Outer Displacements, may be 0, 8, 16 or 32 bits

An, PC = Base register, which may be taken as zero (useful to force Program Space reference without PC displacement).

Xn = Index Register, may be None or a data/address register, either 16 or 32 bits

size = 8, 16, or 32 bit identifier (B, W, or L)

scale = Scaling factor of 1, 2, 4, or 8

Figure 4.

As an example of how the new addressing modes will help the performance of a UNIX system, recall the address calculation example given earlier and compare it to the following instruction that performs the same function.

```
MOVE.L    (off,A0,D0.W*4),Dn
```

Compared to the previous example, this code is not only more understandable, but it requires fewer bytes of code and executes much faster in terms of absolute clock cycles and real time (performance depends on cache hits and instruction order). The table below shows a comparison of the two examples for an 8 MHz MC68000 and a 16.67 MHz MC68020.

	Bytes	Clocks	Nsec
MC68000	12	26	3250
MC68020, Best case	6	4	240
Worst case	6	12	720

As a last example of the increased capabilities of the MC68020 as a result of the new bit field instructions and the new addressing modes, consider the example shown in figure 5. In this example, a base pointer that locates the beginning of a record file in memory is passed to a routine on the stack, and FILEPTR is a constant value equal to the offset from the stack pointer to the file pointer. The record that is needed by this routine is specified by the offset REC5. Within each record is a string of long words that are used to hold arbitrary sized variables (bit fields), and D3 contains an offset from the start of the record to a long word that contains the most significant bit of an array of bits. D4 contains the offset from the start of the bit array to the most significant bit of the desired bit field, which is 24 bits wide (in this case it happens to be constant, although the width of the field could also be contained in a register). Even though this is a fairly complex description of the location of a data item, the MC68020 addressing mechanisms allow the programmer to code the address of the operand in a single instruction as shown. It is important, also, to consider the flexibility that is not shown explicitly by this example, such as the fact that the constant offsets FILEPTR and REC5 can have any value between -2^{31} and $+2^{31}-1$, the offset in D4 can also assume any signed value between -2^{31} and $+2^{31}-1$, the bit field may traverse any byte boundary, and D3 may be scaled by 1, 2 or 8 instead of 4. Finally, remember that the bit field that is extracted is automatically right justified in D0, with the register zero filled beyond the bit field (it could be sign filled with the BFEXTS instruction).

```
BFEXTU ([FILEPTR,A7],REC5,D3.W*4){D4,#24},D0
```

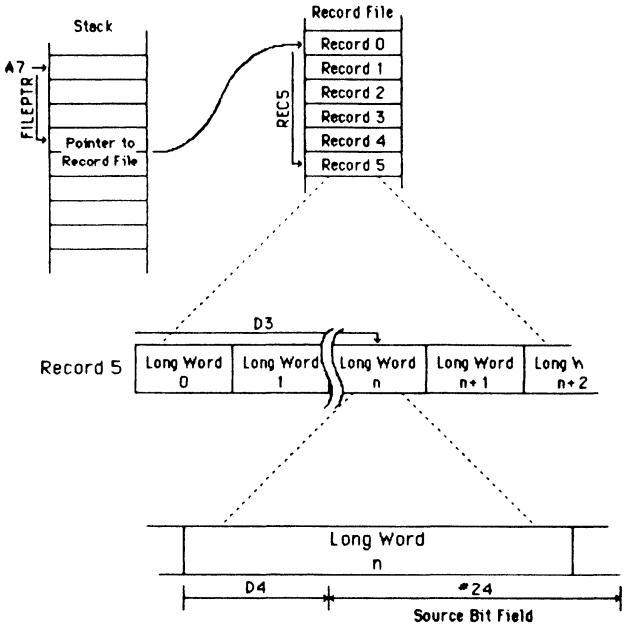


Figure 5. Bit Field Addressing Example.

Miscellaneous Enhancements

Some of the other enhancements to the M68000 architecture that were mentioned above are of general use to any computer system, such as the extensions of several instructions to 32-bit operands. For example, the multiply and divide instructions support 32x32 bit multiply and 64/32 bit divide operations, in addition to the 16-bit operations previously supported by the MC68000. Other instructions that have been enhanced are shown in figure 6, and include better bounds checking mechanisms for compiler

MC68020 32-BIT INSTRUCTION EXTENSIONS

CHK.L	**(NOW ALLOWS WORD OR LONG OPERANDS)**
Bcc.L	**(NOW ALLOWS 8, 16, OR 32-BIT DISPLACEMENTS)**
DIVx.L	**(64 ÷ 32 BIT WITH 32-BIT QUOTIENT AND REMAINDER)**
TDIV.x	**(32 ÷ 32 BIT WITH 32-BIT QUOTIENT AND REMAINDER)**
MULx.L	**(32 × 32 BIT WITH 64-BIT RESULT)**
TMULx	**(32 × 32 BIT WITH 32-BIT RESULT)**
EXTB.L	**(NOW ALLOWS BYTE→WORD, BYTE→LONG, OR WORD→LONG)**
LINK.L	**(NOW ALLOWS 16 OR 32-BIT DISPLACEMENT)**

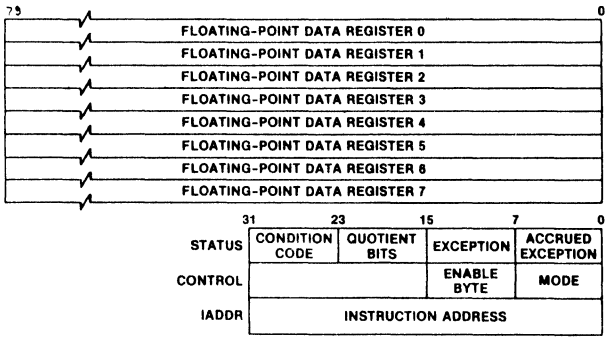
Figure 6.

run-time checks, a translation mechanism that allows packed BCD data to be converted to EBCDIC or ASCII and vice versa, and a new module call and return for better support of modular programming. This last enhancement will increase the performance of any operating system, including UNIX; however, the call/return mechanism used by the entire operating system will have to be re-worked, so this feature may not be used in the initial UNIX release from most vendors, but rather it will be utilized more in proprietary operating systems or within application program environments.

Another feature that was added to the MC68020, specifically to support operating systems, is a new stack pointer that allows for more efficient context switch control. The new stack pointer operates in a manner that is transparent to applications programs and existing operating systems, thus old operating systems are not disturbed by its presence. The main purpose for the new stack pointer, called the Master Stack Pointer (MSP) is to allow the operating system to maintain separate stacks for the currently executing user task and all interrupt tasks. In this manner, when an interrupt is processed by the MC68020, the stack space associated with the user task that was executing is not disturbed by the interrupt stack operations. If a task switch is required due to the occurence of the interrupt, the operating system can simply load the MSP with a pointer to the context block for the new task and activate it; while it has a valid stack pointer available at all times for interrupts. This gives the operating system a more secure environment, while enhancing the performance of the task scheduler by eliminating the need to copy large amounts of information from one stack space to another. While this is another very useful feature of the MC68020, UNIX handles interrupts in such a way that the use of the MSP is not critical to system performance, therefore it is beyond the scope of this paper to describe it fully.

The Coprocessor Interface

Another subject that is beyond the scope of this paper, but does have some bearing on the performance of UNIX on the MC68020, is the coprocessor interface that is implemented by the chip. The general concept behind this interface is to allow the architecture and capabilities of the main processor to be expanded with the addition of external hardware. The first coprocessor that is available from Motorola is the MC68881 Floating Point Coprocessor, which is an implementation of the IEEE draft 10 floating point standard, with several extensions. With the addition of the FPCP to the MC68020, floating point performance reaches the level of many minicomputers that have floating point accelerators. Furthermore, the range of functions available in hardware on the FPCP are well beyond the simple add, subtract, multiply and divide functions that are found on most floating point processors. Also included is a full range of transcendental and trigonometric functions that make the MC68020/MC68881 pair function very well in applications that require real time calculations such as CAD/CAE, 3-D graphics processors, and signal processors. The programmer's model and a summary of the MC68881 instruction set is shown in figure 7.



FADD	Add
FCMP	Compare
FDIV	Divide
FINT	Integer Part
FMUL	Multiply
FREM	IEEE Remainder
FSQRT	Square Root
FSUB	Subtract
FABS	Absolute Value
FACOS	Arc Cosine
FASIN	Arc Sine
FATAN	Arc Tangent
FATANH	Hyperbolic Arc Tangent
FCOS	Cosine
FCOSH	Hyperbolic Cosine
FETOX	e To The x Power
FETOXM1	e To The x Power - 1
FGETEXP	Get Exponent
FGETMAN	Get Mantissa
FLOG10	Log Base 10
FLOG2	Log Base 2
FLOGN	Log Base e
FLOGNP1	Log Base e of x + 1
FMOD	Modulo Remainder
FNEG	Negate
FSCALE	Scale Exponent
FSGLDIV	Single Precision Divide
FSGLMUL	Single Precision Multiply
FSIN	Sine
FSINCOS	Simultaneous Sine and Cosine
FSINH	Hyperbolic Sine
FTAN	Tangent
FTANH	Hyperbolic Sine
FTENTOX	10 To The x Power
FTST	Test
FTWOTOX	2 To The x Power

Figure 7. MC68881 Instruction Set Summary and Programmer's Model.

Memory Management

An important consideration in choosing any computer for UNIX or any other multi-user operating system is the method that it uses for memory access control. While the MC68020 does not have an integral memory management unit, it does have the capabilities to support virtual memory, due to its ability to continue any instruction that causes a page fault. Furthermore, the asynchronous bus interface that is used by all M68000 processors lends itself easily to memory management schemes, since external translators can dynamically control the speed of bus accesses and can terminate any illegal bus cycle and cause the processor to take an exception to repair the cause of the fault, or abort a task that was attempting to access an illegal area of the memory map. Due to the flexible nature of the bus interface and the fact that the processor supports any memory management scheme that a designer wishes to implement, the M68000 family has enjoyed widespread acceptance in virtual memory systems. While Motorola does not wish to dictate the memory management scheme to all system designers, it does offer a VLSI solution to the designer that does not wish to design his own MMU.

The MMU that will be available for the MC68020 in 1985 is the second coprocessor to be built for the processor and is called the MC68851 Paged Memory Management Unit (PMMU). It supports a demand paged virtual memory environment by automatically searching translation tables, set up in physical memory by the processor, to translate 32-bit logical addresses to 32-bit physical addresses, while providing protection against unauthorized accesses. To speed up the translation process, an on-chip translation cache, organized as a 64-entry content addressable array, is used to translate the addresses in 45 nsec (this allows the MC68020 to run 4 clock cycle bus cycles at 16.67 MHz). The translation mechanism is quite flexible, with user defined translation tables and page sizes from 256 bytes up to 32 KBytes. Coprocessor instructions are also implemented to support operating system functions such as paging algorithms and access protection checking.

In addition to the VLSI PMMU, Motorola is also supporting a second memory management scheme, intended for prototyping of systems that will later be based on the MC68020/MC68851 combination: the MC68461 Memory Management Controller (MMC). This device is implemented in Motorola's MCA2800ALS gate array and provides the control functions for a discreet translation cache external to the MMC. When combined with an external cache, the MMC supports a memory management architecture that is a subset of the possible architectures supported the the MC68851, thus allowing for upward compatibility. The MMC is an integral part of the first system level product from Motorola that is based on the MC68020, the VM04 VERSAmodule CPU board. Also, the mezzanine board memory management unit for the VM04, which uses the MMC and a 512 entry cache, is called the Memory Management Board (MMB) and is available for system prototyping.

System V/68

Now that the features of the MC68020 that make it a good machine for UNIX have been covered, the next question to answer is: what UNIX products will be offered by Motorola in support of the MC68020? The answer to that is quite simple: Motorola is the first third party vendor to have a port of UNIX System V qualified by AT&T, and that product is in the process of being upgraded to support the MC68020 in the first half of 1985. The name of the Motorola UNIX port is System V/68 and it is currently supported by Motorola an two systems, the EXORmacs multi-user development system and the VME/10 single-user development system. It is also supported by AT&T for DEC machines, such that a systems integrator can use his present VAX system to cross compile applications to run under System V/68 on an M68000 based system, or use a DEC computer to generate the operating system kernel, file system and utilities for porting to an end system that utilizes the MC68000, MC68008 or MC68010 (with the MC68451 MMU).

Because of the fact that the Motorola UNIX product is a direct port of the AT&T Bell Laboratories' UNIX, much of the definition of what System V/68 is is apparent. System V/68 includes all of the file handling capabilities of UNIX, including pipes and filters, I/O redirection, multi-tasking with foreground and background tasks controlled by a single user, a configurable environment and a flexible command language. On top of this, the standard UNIX shell presents a user and applications interface to the file system; and on top of the shell, System V/68 includes all of the standard UNIX utility programs, the programmer's workbench, the C and FORTRAN 77 languages and networking capabilities. In addition to the standard UNIX package, Motorola also offers an optional PASCAL compiler (revision 2.1), a macro assembler, linker and loader, and support for Motorola hardware development stations such as the HDS-400 emulator with symbolic debugging capabilities. In order to support customers that are currently using the VERSAdos operating system in their development environment, Motorola also provides a tool kit that runs under System V/68 contains file conversion utilities and allows VERSAdos libraries to be maintained on a System V/68 host.

While the details of what UNIX System V is are beyond the scope of this paper, it is interesting to look at the core of UNIX to see what is required to port System V/68 to an end system. The UNIX kernel is the heart of the operating system and, like all of the utilities, it is written mostly in C (about 95%). The host processor and system hardware portions of the kernel are written in assembly language for the highest performance. The entire kernel is about 70-80 KBytes of code, and accounts for less than 10% of the code for the entire operating system. Since almost all of the operating system is written in C, the size and speed of System V/68 is directly related to the efficiency of the M68000 code generator provided in the C compiler. The only change required to take full advantage of the new features of the MC68020 is to replace this code generator and recompile the system. Preliminary results of doing this show the MC68020 kernel and operating system to be approximately 20% smaller than the MC68010 version of same system, due mostly to the more efficient addressing modes. Of course, performance is much greater than that of a comparable MC68010 system; however, enough studies have not been performed at this time for a definite ratio to be published.

The First MC68020 - System V/68 Machine

The first port of System V/68 to a native MC68020 machine by Motorola is now in progress, although that machine will not be available commercially until the first half of 1985. That system will be built around the VM04, which was introduced in June of this year when the MC68020 was introduced. The VM04 is a high performance VERSAmodule computer that includes memory management support with the MMC (the PMMU will be included when it is available) and has a socket for an optional MC68881. Also included on the board is a 4K x 32 direct mapped cache and a local bus extension to a high speed private bus called the RAMbus. Connected to the RAMbus is a second board, the VM13 1MByte memory module that supports all of the features of the MC68020 bus interface at full speed. Figure 8 shows a block diagram of these two boards, in a typical system configuration. This two board set, called the Benchmark 20, is available now as an evaluation system environment for the MC68020, MC68881 and the MC68851 architecture that is supported by the MMB.

Of course, UNIX does not perform a very useful function without some kind of I/O; and indeed, the I/O portion of a UNIX system is critically related to the performance of the system. In recognition of this, two new I/O controller boards are being developed specifically to support the VM04/VM13 - System V/68 port. The first controller board is the VM32

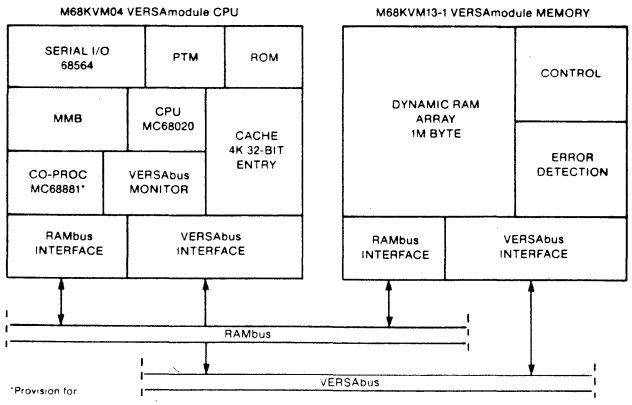


Figure 8. VM04/VM13 System Block Diagram.

asynchronous communications controller, that supports 8 RS-232C full duplex comm ports, and has a local M68000 processor as a dedicated controller. The second is a high performance Winchester disk controller, called the VM23. This board supports up to four ST506 disk drives, four floppy disk drives, a streaming tape backup, and a parallel printer port. Local intelligence is provided by a 10 MHz MC68010 that has the following local resources: up to 128KBytes of local RAM, 64KBytes of ROM, an MC68440 DMA controller and a special purpose VERSAbus controller built around an AMD 2940 address sequencer for high-speed 32-bit transfers from the local memory to the VERSAbus. The use of powerful processors for local intelligence is a common design practice for UNIX systems, due to the heavy I/O demands of the operating system.

Summary

When the MC68000 microprocessor was introduced five years ago, it was heralded as a mini-computer on a chip, due to its high performance and its advanced architecture. The MC68020 continues that tradition, by extending the architecture beyond the previous limits of VLSI, and by achieving performance levels that were only available to mini-computer users a short time ago. At the same time that Motorola was enhancing the M68000 Family, the user community was beginning to embrace more and more large computer architectural ideas and place them into microcomputers. The standard supermicro configuration that has been emerging and maturing over the last two to three years is an MC68000 or MC68010 based machine, with intelligent I/O controllers, that runs a version of UNIX from AT&T Bell Laboratories. Now these two forces are coming together to create very high performance, full featured desktop computers that rival the performance of machines that cost hundreds of thousands of dollars - the MC68020 32-bit microprocessor and System V/68 - an unbeatable combination.

A18652 PRINTED IN USA 4-85 IMPERIAL LITHO C30050 10,000 AR219