MC68000 MICROPROGRAMMED ARCHITECTURE

Motorola Semiconductor Group
Austin, Texas

This paper considers microprogramming as a tool for implementing large scale integration, single-chip microprocessors. Design trade-offs for microprogrammed control are discussed in the context of semiconductor design contraints which limit the size, speed, complexity and pin-out of circuits. Aspects of the control unit of a new generation microprocessor, which has a two level microprogrammed structure, are presented.

INTRODUCTION

The field of single-chip, large scale integration (LSI) microprocessors is advancing at an incredible rate. Progress in the underlying semiconductor technology, MOS, is driving the advance. Every two years, circuit densities are improving by a factor of two, circuit speeds are increasing by a factor of two, and at the same time speed-power products are decreasing by a factor of four. Finally, yield enhancement techniques are driving down production costs and hence product prices, thereby increasing demand and opening up new applications and markets.

One effect of this progress in semiconductor technology is advances in LSI microprocessors. The latest generation, currently being introduced by several companies, is an order of magnitude more powerful than the previous generation, the 8-bit microprocessors of three or four years ago. The new microprocessors have 16-bit data paths and arithmetic capability. They directly address multiple-megabyte memories. In terms of functional capability and speed they will out-perform all but the high end models of current 16-bit minicomputers.

As LSI microprocessor technology matures it becomes feasible to apply traditional implementation techniques, that have been proven in large computers, to the design of microcomputers. LSI microprocessor design is now at the stage where better implementation techniques are required in order to control complexity and meet tight design schedules. One such technique, microprogramming, is the subject of this paper. Most of the traditionally claimed benefits of microprogramming, e.g. regularity (to decrease complexity), flexibility (to ease design changes) and reduced design cost, apply to the implementation problems facing today's LSI microprocessor designer.

This paper describes the control structure of one of the new generation, single-chip microprocessors, the MC68000 processor from Motorola, with special attention to the constraints which LSI technology imposes on processor implementation. There are four such constraints: circuit size, circuit speed, interconnection complexity and package pin count. The implications of these constraints on the structure of a microprocessor control unit and its microcode are explored.

LSI SEMICONDUCTOR TECHNOLOGY

Though progressing quickly, LSI technology still imposes strict constraints on the microprocessor designer.

Circuit Size and Density

There is a fairly constant bound on the size of LSI chip that can be economically produced. Even though circuit densities are improving, at any given time there is a limit on the number of gates that can be put on a chip. The major constraint on an LSI designer is to fit his design into a fixed maximum number of gates.

Circuit Speed

As with circuit density, the LSI designer has a fixed maximum circuit speed with which to work. Speed is limited primarily by the power dissipation limits of the semiconductor package. The problem is compounded by the fact that the processor technology and main memory technology in microprocessor applications are the same. The speed gap between ECL logic and core memory enjoyed by the large computer designer is not available to single-chip microprocessor designers.

Interconnect Complexity

Internal interconnections on an LSI circuit often take as much chip area as the gates they connect. Furthermore, the designer does not have the option of running jumper wires across his circuit when he runs out of surface area. In some cases, it is cheaper to duplicate functions on various sections of the chip than to provide connection to a single centralized function. Another implication of the interconnect problem is that regular structures, such as ROM arrays, can be packed much more tightly than random logic.

Pin-Out

Semiconductor packaging technology limits the number of connections an LSI chip may have to the outside world. Common packages today have 24 or 40 pins; 48 and 64 pin packages are considered large. The pin-out limitations can be overcome by time multiplexing pin use,

but the resulting slowdown is usually not acceptable.

Another constraint on LSI designers, not inherent in the technology, is the intensely competitive climate of the semiconductor industry. During the development of any new product it is likely that other companies are working on comparable products. Furthermore, the first product available, of a given type, usually gets the largest share of the market. This situation places LSI designers under tight schedules. Any techniques for reducing product design times can affect product success.

CONTROL UNIT DESIGN TRADEOFFS

Combinatorial Logic versus Microprogramming

Although previous LSI microprocessor implementations at Motorola have not been microprogrammed, a microprogrammed implementation for MC68000 was considered early in the project. The benefits of microprogramming were convincing enough that once the feasibility of a microcoded implementation was established, the alternative of combinatorial logic implementation was not seriously considered. This is in spite of the fact that the implementers' proven expertise was in combinatorial implementations and they had no experience with microprogramming.

Besides several non-technical reasons for microprogramming (very tight design schedule, limited staff, etc.) there are compelling technical advantages to microprogramming, especially regularity and flexibility.

The design time constraint appears to be eased by microcoding. The regular structure of control store, in contrast with arbitrary control logic, decreases the complexity of the control unit. This in turn decreases the design time. A more complex controller can be implemented at a given design cost. Regularity of the structure simplifies the layout of the chip. Considerable time savings (possibly months) can be realized in the layout step and errors are less likely. Microprogramming allows the processor architects to delay binding some decisions. Once the basic control structure is determined, the circuit designers can go to work, even though the actual microcode may not be written. This reduces the inherent sequentiality of the design process by allowing more overlap of the design efforts of microcoders and circuit designers, and therefore shortens design time.

The regular structure of control store in a microcoded implementation has several other benefits. The regularity decreases the interconnection complexity and therefore the size of the control circuitry. In other words, an array of read only memory cells may take less chip area than the equivalent combinatorial logic. Also, the regularity of structure facilitates detailed simulation and testing.

The MC68000 processor is designed to be enhanced with new instructions in future versions. Microprogramming makes it more likely that such expansion will not involve a major redesign of the chip. The flexibility of microprogramming can also ease the problems of design changes and correction of design errors. Some such changes can be made merely by changing the control store contents with no redesign of the logical circuitry.

Besides regularity and flexibility, microprogramming provides another benefit. The clocking functions in microprogrammed control are much cleaner than those randomly distributed throughout a combinatorial maze with its associated delay, distribution and regeneration requirements. For instance, accurate delay elements are difficult to construct on an integrated circuit, which causes increased tolerances in control signals and slower clocks for combinatorial circuits.

On-Chip versus Off-Chip Control Store

Given the size constraint for LSI chips it would be very attractive to consider off-chip control store for a microprogrammed LSI processor. Other constraints make this impractical, however. The pin-out limitation severely limits the width of the control word from off-chip control store. This implies that the control unit microcode must be vertical. This in turn limits the overall speed of the processor since many micro cycles are required in vertical microcode to implement a single macro instruction. The technology speed constraint does not allow brute force solution to this problem by speeding up the internal cycle time. Time multiplexing pins to sequentially access horizontal micro instructions would also slow down the processor.

The LSI-11 from Digital Equipment Corporation uses off-chip control store. The result is a fairly narrow (22 bits) micro instruction. This structure causes a fundamental limitation to the potential speed of the LSI-11, as discussed by Snow and Siewiorek.[7] Because of the above considerations an on-chip control store implementation is preferable.

Horizontal versus Vertical Microcode

The decision to use horizontal or vertical microcode involves conflicting sets of constraints. Horizontal microcode is indicated for several reasons. Vertical microcode is highly encoded and requires a significant amount of combinatorial logic to decode the micro instructions. Horizontal microcode provides fully decoded (or nearly so) fields which can directly drive the execution unit with little intervening logic. Vertical microcode also typically requires more micro cycles to emulate a given macro instruction. Because of the LSI technology circuit speed constraint these extra micro cycles cannot be hidden in each macro cycle by speeding up the internal circuitry. Thus both the interconnect constraint (elimination of random logic) and the speed constraint argue for horizontal microcode.

On the other hand, vertical microcode has advantages, the major one being reduction of the size of the control store. Horizontal micro instructions tend to be very wide and are often duplicated several times in control store.

One solution is to use a two level control store, or hybrid vertical/horizontal structure. This is similar to the two level control proposed by Grasselli.[3,6] In the two level control structure each macro instruction is emulated by a sequence of micro instructions. The micro instructions are narrow, consisting primarily of pointers to nano instructions. (Micro instructions also contain information about branching in the micro sequence.) The nano instructions are wide, providing fairly direct, decoded control of the execution unit. Nano instructions can be placed randomly in the nano store since no sequential accesses to nano instructions are required. Also only one copy of

each unique nano instruction need be stored, no matter how many times it is referred to by micro instructions.

The Appendix contains an analytic treatment of two level control. A derivation of the potential savings in control store space, with examples, is given.

An extention of the two level concept is made in the Nanodata QM-1.[5,6] In that machine a mircro instruction can specify a *sequence* of nano instructions. This approach was not taken in the MC68000 for two reasons. First, the initial microprograms showed that micro sequences tend to be very short (one, two, or three micro instructions), so sequential nano instructions cannot be used to advantage. Secondly, unless some facility for nano branches is implemented multiple copies of some nano instructions must be kept in nano store.

THE MC68000 CONTROL UNIT IMPLEMENTATION

Actual implementation of the general structure derived above involves many other design problems. The remaining sections of the paper discuss the actual implementation chosen and the design considerations involved. Major problems to be solved were minimization of the size of control store, speed-up of the control unit, and reduction of interconnect between the control and execution units. Control store size was minimized by providing a suitable micro instruction branching capability to facilitate sharing subsequences. The control unit speed requirements made micro instruction prefetch necessary so that each nano store access and the subsequent micro store access are overlapped as much as possible.[2] Execution Unit interconnect is minimized by placing the nano store directly above the Execution Unit (with space for some decoding). Fields in the nano store are allocated such that control store output lines are close to the corresponding Execution Unit control points.

The MC68000 control unit supports an instruction set which consists of general single and dual operand instructions involving byte, word (16 bits), or double word operands. Operations are generally memory-to-register, register-to-memory, or register-to-register with some notable exceptions such as the general memory-to-memory move. In addition to standard instructions such as add, compare, and shift; the MC68000 processor is designed to support such instructions as load and store multiple registers, pack (ASCII), multiply and divide, and various forms of bit manipulation.

The MC68000 processor provides eight 32-bit address manipulation registers and eight 32-bit data manipulation registers. Address registers allow 16- and 32-bit operations and data registers allow 8-, 16-, and 32-bit operations. All address and data registers are accessible to the programmer. In addition, there is a program counter with limited user accessibility and there are several registers not available to the user which are used for temporary storage during instruction execution.

The register file is divided into three sections as shown in Figure 1. Two buses connect all of the words in the register file. The register file sections are either isolated or concatenated using the bi-directional bus switches. This permits general register transfer operations across register sections. A limited arithmetic unit is located in each segment containing address register words and a general capability arithmetic and logical unit is located in the data low word section. This allows address and data calculations to occur simultaneously. For example, it is possible to do a register-to-register word addition concurrently with a program counter increment (the program counter is colocated with the address register words and carry out from the arithmetic unit low is provided as carry in to the arithmetic unit high). Special functional units for bit manipulation, packing and unpacking data are located in the data section.

Two factors combined to suggest the desirability of the configuration shown in Figure 1. The first was a very dense two-port static RAM cell which conveniently supported a two-bus structure. The second was the 16-bit data width which made 16-bit segmentation of the registers desirable.

In addition to the configuration of the Execution Unit, other factors contributed to the design of the control unit. The instruction set was specified and considered frozen. The first version assumed that op codes and instruction formats would remain static as defined, though holes were left in the original op code space to allow planned orderly expansion of the available instruction set.

Restriction to fixed instruction formats has several important consequences:

1. Certain fields, such as register designators, can be extracted directly from known positions in the instruction. This tends to reduce control store size and simplify instruction decoding.

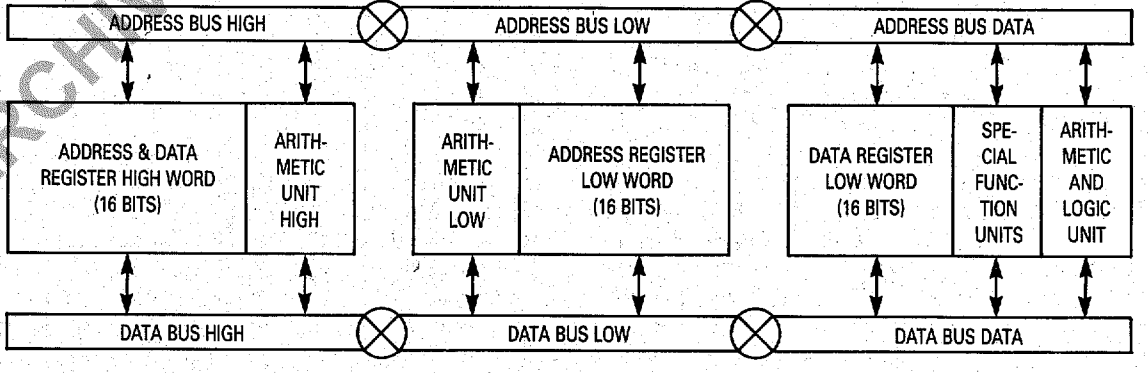

FIGURE 1. MC68000 EXECUTION UNIT GENERAL CONFIGURATION

2. Register selection and ALU functions tend to remain unchanged for the duration of a single instruction execution. These register designators and ALU functions can be extracted from the instruction (decoded, if necessary) and routed directly to the execution unit, bypassing the control store.
3. The control store need only contain information about when a register is read or written or when the ALU should operate.

Taking advantage of these observations can lead to simplification of the control and reduction of the required micro control store size.

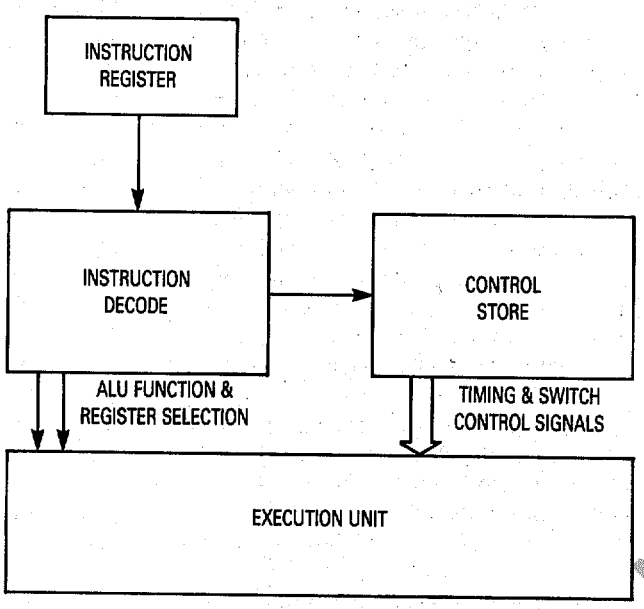

FIGURE 2. SIMPLIFIED BLOCK DIAGRAM OF THE MC68000 CONTROL STRUCTURE

A simplified block diagram of the MC68000 control structure, shown in Figure 2, illustrates an application of the above observations to the controller design. The basic idea is to extract from the macro instruction word all information which is macro instruction static; that is, information which does not depend on timing during the instruction execution for its usefulness. Signals which are not timing dependent bypass the control store and act directly on the Execution Unit.

In a typical microcontrol implementation, the Instruction Decode provides a starting address to the Control Store. The Control Store generates a sequence of control signals, for the Execution Unit, and its own next state information. Branching is accomplished using feedback from the Execution Unit to alter the next state information into the Control Store. At the end of execution of the macro instruction, the Control Store causes loading of the next macro instruction into the Instruction Register and transfers next state control to the Instruction Decode unit.

Traditional implementaion of the MC68000 control section using a single Control Store with internal state sequencing information was investigated. It was found to be impractical because the control store was too large for a single chip implementation. Methods for reduction of the total control store area required were considered. It was determined that necessary control store area could be substantially reduced through the use of a two-level control store structure. The structure selected for the MC68000 control unit is shown in Figure 3.

In a two-level structure, the first level (micro control store) contains sequences of control word addresses for the lower level (nano control store). Dynamic operation is illustrated in Figure 4 (bus activity for an indexed address, register to memory add). The Instruction Decode provides the starting address for a single macro instruction routine. The micro control store provides a sequence

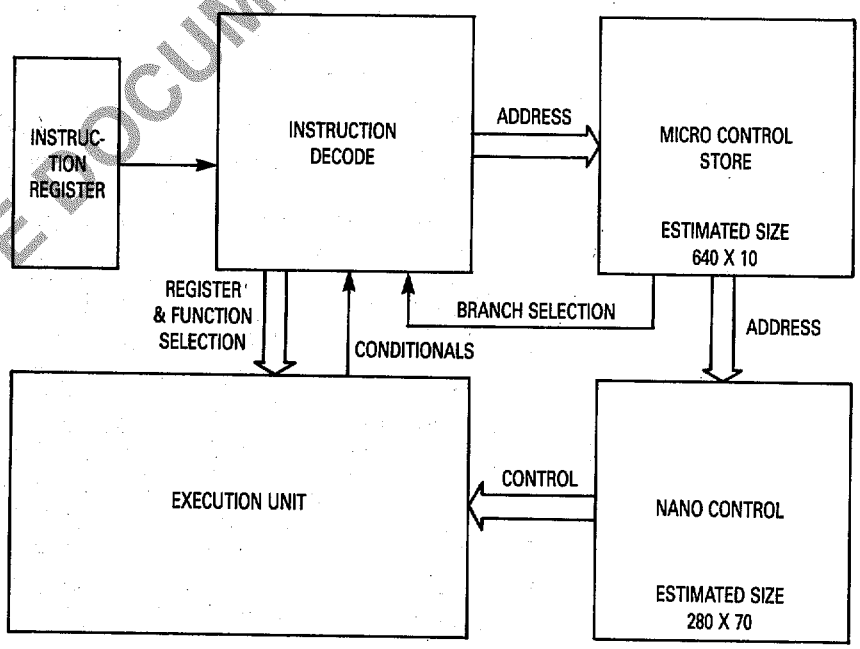

FIGURE 3. MC68000 CONTROL STRUCTURE

of addresses into the nano control store. The nano control store contains an arbitrarily ordered set of unduplicated machine state control words. The practicality of this structure for space reduction rests on two mutually dependent assumptions.

First, the number of different control states actually implemented is a small fraction of the number of possible control states. For example, a reasonably horizontal control word for the MC68000 Execution Unit contains about 70 bits, implying a possible 2^{70} different control words. Most of the possible control states are not meaningful for macro instruction execution. The implementation of the complete set of macro instruction sequences for the MC68000 processor requires only about 200 to 300 ($< 2^9$) unique nano words. This set of nano words is a very small fraction of the set of possible states. Nano words are uniquely specified by no more than nine bits of address. As a result, words in the micro control store address sequences need only allocate nine bits for each nano control store address.

Second, there must be some redundant use of the necessary control words to realize a reduction in control store area. If there were a one-to-one correspondence between nano control store addresses in the micro control store and control words in the nano control store, then the nano address in the micro instruction could be replaced by the contents of the addressed nano instruction and the address bits eliminated. If, however, there are more addresses in the instruction sequences than there are unique control words, a reduction in total control store size may be possible. (For a heuristic derivation of these dependencies and possible advantages, see the appendix.) In the MC68000 control unit, for example, each different control word is used an average of between two and three times. There are about 650 nano addresses in a complete implementation of micro control store address sequences for the instruction set; yet there are only about 280 different control words used. A single level implementation would have required about 45K control store bits, while the two level structure uses only a little more than half as much.

Another parameter which can have a significant effect on control store size is the extent to which the control word is encoded. In a two level control structure, each control word in the nano control store is uniquely represented by an address in the micro control store. The address in the micro control store could be considered to be a maximal encoding of the control word because there is a one-to-one correspondence between unique control words and unique addresses in the micro control store address sequences. (The nano control store could be viewed as merely an orderly method for translating maximally encoded state information to a significantly more horizontal format.) At the other extreme, one bit could be allocated in the control word for each switch, or control point, that must be driven in the Execution Unit. If the control word is encoded, the decoding logic necessary between the control store output and the Execution Unit must be considered with respect to both timing and space constraints.

In the MC68000 control unit, bits in the nano control store are assigned generally on a functional basis with individual subfields assigned to specific control subfunctions. For example, separate short control fields are assigned to program counter control and arithmetic and logic unit output control.

Within a specific subfield the control bits are encoded into the minimum bits necessary to provide the required subfunction states subject to the constraint that the decode to individual control lines involve no more than approximately two logic levels. Some space is necessary between the nano control store output and the Execution Unit control points for alignment of the control store outputs with their respective control points and to combine

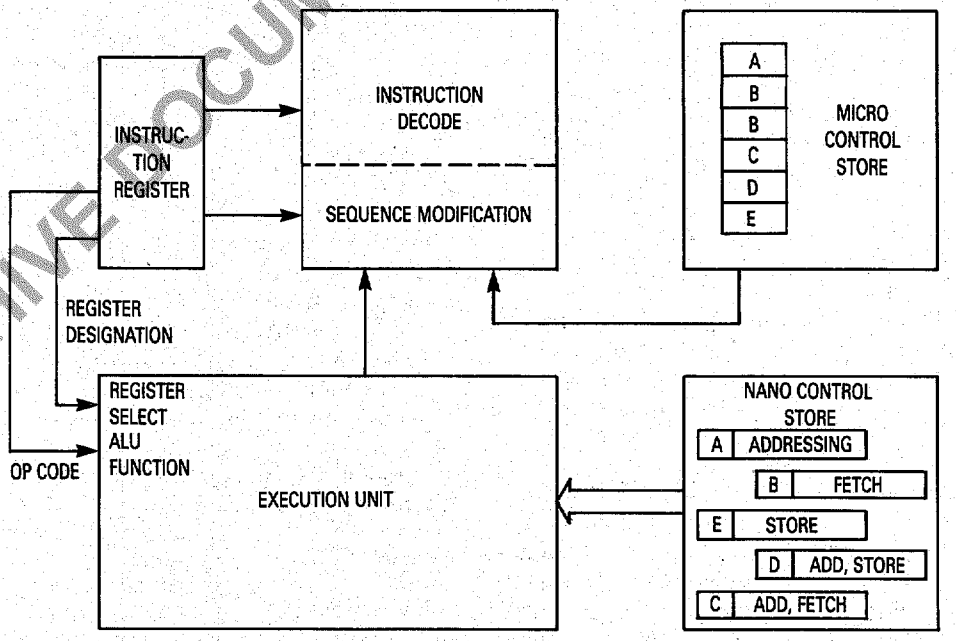

FIGURE 4. MC68000 CONTROL UNIT DYNAMIC OPERATION

certain timing information with appropriate control point variables. Within this space it is possible to provide minimal decoding at very little cost in additional area while the signal encoding in the nano control store saves considerable nano word width and, hence, total storage space required. In the MC68000 control unit the nano word width is approximately 70 bits, while the Execution Unit contains about 180 control points.

The decision to implement the MC68000 control unit using a two level storage structure was based on minimizing control store area. Although necessary control store area was significantly reduced, introduction of the two level concept created several problems.

One problem with a two level control store is that access to a memory is not instantaneous and in a two level structure the accesses must be sequential. The alternative, combinatorial decoding, does not proceed in zero time either, which partially compensates for the extra memory access. Further compensation for the extra access time requires complex control timing techniques such as instruction prefetch, access overlap, and multiple word accesses.

Another problem associated with a two level structure is the delay associated with conditional branching. Viewed in a strictly sequential fashion, a condition set in the Execution Unit must affect the selected micro control store address sequence which, in turn, affects the nano word selected. The nano word selected ultimately causes actions which can be dependent upon the value of the tested combination. Techniques used to minimize the sequential nature of this type of delay include physical organization of words within both control store levels and simultaneous access to more than a single control store word. For example, an access to a row of nano store (containing multiple nano words) can be initiated early in a cycle, with subsequent single-word selection based on conditional branch information. Also, in many cases, probable outcome of a conditional branch favors one branch more than another. In such an instance, it may be possible or even desirable to prefetch instruction words associated with the most likely branch condition. The best example of the usefulness of this idea is its application to a decrement and branch-not-zero type instruction. Branching is heavily favored and a prefetch at the destination location can greatly minimize execution delays associated with looping.

If common micro instruction routines, such as the address calculation routines, are to be shared among several macro instructions, then mechanisms must be provided which facilitate functional branches for both entering and leaving the common routines. Care must be taken to avoid delays associated with functional branches in a two level store, especially when the common routines are short (making it more difficult to overlap accesses to different routines in different control stores).

The capability to perform direct branches in the micro control store allows various macro instruction sequences to share common ending routines. It also permits more flexibility in organizing the micro routine sequences within the control store address space.

Branching mechanisms are mentioned here because they occur very commonly. In the MC68000 micro control store, the average micro instruction sequence encounters some type of branch at about one out of every two nano addresses. Implementation of efficient branching mechanisms is critical for providing fast execution times with a microprogrammed structure. The details of the branching mechanisms are, however, very specific to a particular implementation and are, therefore, not relevant beyond the general considerations already presented.

OTHER ISSUES

Minimization

Special care has been taken during the microprogramming to detect duplicate nano instructions. Beyond that, in some cases it has been possible to rewrite micro sequences, delaying or anticipating some actions, so as to use previously created nano instructions. For instance, operands are moved into temporary registers early in instruction execution. This tends to make subsequent execution cycles more independent of operand sources and, hence, improves the chance for common nano instructions. Each unique nano instruction that can be eliminated reduces nano store size.

Micro store size is minimized by careful detection of subsequences of micro instructions; for instance, those for address calculations, or storing results.

Simplicity of Structure

Frieder and Miller[2] argue that simplicity of structure makes microprogramming easier. The MC68000 arithmetic and register unit (the Execution Unit) is quite simple: all resources (registers, temporaries and functional units) are tied to each of the two internal buses. This structure simplifies microprogramming, since all transfers of information use the same mechanism.

Generality of Structure

Frieder and Miller[2] also call for generality of structure. This is probably important for general purpose emulation. In the MC68000 processor the architecture is known and fixed. Various non-general assumptions were made, for instance, about macro instruction decoding and the location of register fields in macro instructions. Because of the strict technology constraints, the control structure is optimized for emulation of a single architecture.

User Microprogramming

Current technology requires that large on-chip control store be implemented in ROM, which is much denser than alterable memory. Technology advances will certainly ease this requirement in the near future. Single-chip computers have already been built with small on-chip alterable memories. Clearly, custom microprogramming, user microprogramming and dynamically altered microprograms will be feasible on LSI microprocessors in the future.

SUMMARY

Microprogramming is a viable tool for implementation of LSI microprocessors. The current state of the art in semiconductor technology places certain constraints on

the size, speed, interconnect complexity and pin-out of today's integrated circuits. These constraints affect the form of microprogrammed control that can be used. The structure described here: two level, overlapped, hybrid vertical and horizontal microcontrol, implements a new generation microprocessor within these constraints.

ACKNOWLEDGEMENTS

This paper and the MC68000 microprocessor would have been impossible without the creative guidance of T. Gunter, the administrative guidance of G. Daniels, and the inspirational guidance of C. Crook.

REFERENCES

1. Dollhoff, T. L., "The Negative Aspects of Microprogramming," Datamation (July, 1964), p. 64–66.
2. Frieder, G. and J. Miller, "An Analysis of Code Density for the Two Level Programmable Control of the Nanodata QM-1," Tenth Annual Workshop on Microprogramming (October, 1975) p. 26–32.
3. Grasselli, A., "The Design of Program-Modifiable Micro-Programmed Control Units," IRE Transactions on Electronic Computers EC-11 no. 6 (June, 1962).
4. Rosin, R. F., "Contemporary Concepts of Microprogramming and Emulation," Computing Surveys 1 no. 4 (December, 1969) p. 197–212.
5. Rosin, R. F., G. Frieder and R. Eckhouse, "An Environment for Research in Microprogramming and Emulation," Comm. ACM 15 no. 8 (August, 1972) p. 748–760.
6. Salisbury, A., *Microprogrammable Computer Architectures,* American Elsevier, 1976.
7. Snow, E. A. and D. Siewiorek, "Impact of Implementation Design Tradeoffs on Performance: The PDP-11, a Case Study," Dept. of EE and Comp. Sci., Carnegie-Mellon University (July, 1977).

Appendix: Control store size reduction with a two-level control store.

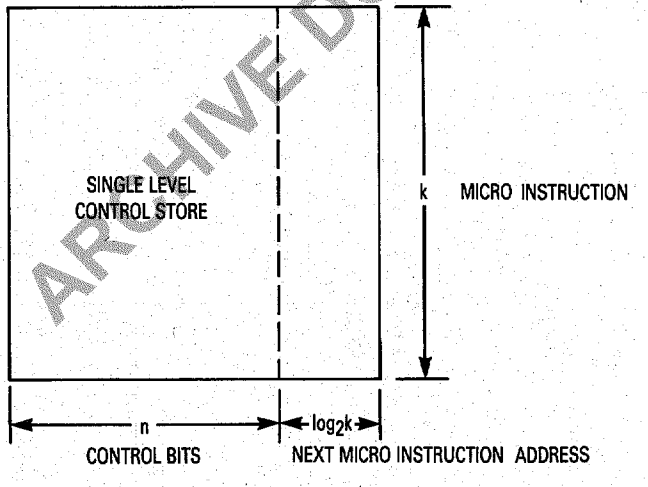

FIGURE A1. MODEL OF A SINGLE LEVEL CONTROL STORE

Assume:

- n = number of individually-controlled switches in an execution unit (width of the horizontal control word)
- k = total number of control states required to implement all instructions
- ρ = proportion of unique control states to total number of control states

Single-Level Control Store

In a simplified model of a single-level control store there are k micro instructions, each containing a control state (n bits) and a next micro instruction address ($[\log_2 k]$ bits). See Figure A1.

Total size of single-level control store:

$$S_1 = k\,(n + [\log_2 k])$$

Two-Level Control Store

A simplified model of a two-level control store has a micro control store of k micro instructions with a nano address ($[\log_2 v]$bits) and a next micro instruction address ($[\log_2 k]$ bits). The nano control store has v ($=\rho k$) nano instructions, each containing a control state (n bits).

Total size of two-level control store:

$$S_2 = k\,([\log_2 v] + [\log_2 k]) + nv \qquad (2)$$

where $v = \rho k$ (3)

Control Store Size Comparison

Two-level store requires less control store bits than single control store when:

$S_2 < S_1$

using

(1), (2) and (3) gives:

$$k([\log_2 \rho k] + [\log_2 k]) + n\rho k < k(n + [\log_2 k]) \qquad (4)$$

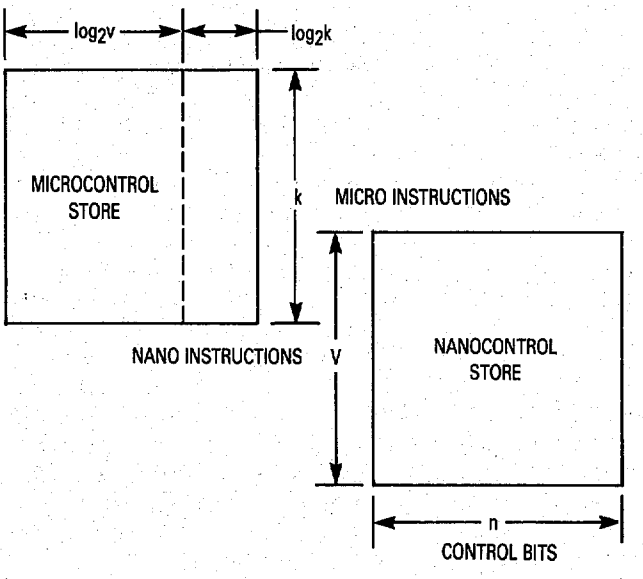

FIGURE A2. MODEL OF A TWO LEVEL CONTROL STORE

Simplifying (4) gives:

$$[\log_2 k] + [\log_2 \rho] + n\rho < n \quad (5)$$

Solving for n and k in (5) gives the result that two-level store is smaller than single-level control store if

$$n > \frac{[\log_2 k] + [\log_2 \rho]}{1-\rho} \quad (6)$$

or

$$k < \frac{1}{\rho}\, 2^{n(1-\rho)} \quad (7)$$

Example

In typical microprogrammed machines, n (the width of the horizontal control word) varies from 20 to 360. k varies from 50 to 4000. Typical values for ρ are not known.

In the MC68000 microprocessor

$$\begin{aligned} n &= 70 \\ k &= 650 \\ \rho &= .4 \\ S_1 &= k\ (n + [\log_2 k]) \\ &= 52400 \\ S_2 &= k\ ([\log_2 v] + [\log_2 k]) + nv \\ &= 30550 \\ \frac{S_2}{S_1} &= .58 \\ \Delta S &= S_1 - S_2 = 21850 \text{ bits} \end{aligned}$$

Design and Implementation of System Features for the MC68000

MOTOROLA Semiconductor
Austin, Texas

ABSTRACT

The MC68000 combines state-of-the-art technology, advanced circuit design techniques, and computer science to achieve an architecturally advanced 16-bit microprocessor. The processor is implemented by a microprogrammed control of an execution unit. The processor incorporates advanced system features, including multilevel vectored interrupts, privilege states, illegal instruction policing, and bus cycle abort. This paper discusses the implementation of the system features and the influence of the implementation method on the processor design.

1. MC68000 Overview

1.1. Resources

Figure 1 shows the register resources of the MC68000 microprocessor. The first eight registers (DO–D7) are used as data registers for byte (8-bit), word (16-bit), and long (32-bit) data operations. The second set of nine registers (A0–A7, A7′) are used as address registers, supporting both software stack operations and base addressing. There is a separate 32-bit program counter and a 16-bit status register.

Figure 2 shows the format of the status register. The trace control (T), supervisor/user (S), and Interrupt Mask (I0–I2) appear in the upper system byte. The condition codes appear in the lower user byte: extend (X), negative (N), zero (Z), overflow (V), and carry (C).

MC68000 memory is organized as 16-bit words, addressable to 8-bit bytes. All address computations are done to 32-bit resolution, but only the low order 24 bits are brought out due to pin count limitations.

1.2. Instructions

The MC68000 supports five basic data types with six basic types of addressing and 56 instruction types. The supported data types are bits, BCD digits, bytes, words, and long words. The basic address types include register direct, register indirect, absolute, immediate, program counter relative, and implied. The register indirect addressing modes include the capability to do post-increment, predecrement, displacement, and indexed addressing. Instruction categories include data movement, arithmetic operations (add, sub, multiply, divide), logical operations (and, or, exclusive-or, not), shift and rotate operations, bit manipulation instructions, program control, and system control instructions.

1.3. Structure

To convey an understanding of the relationship between system features as desired (originally specified) and as ultimately supported, it is necessary to first describe the philosophy of the control structure which will provide the background for implementation tradeoffs. The MC68000 uses a microprogrammed control unit which is tightly coupled to the execution unit and the bus interface. (The control structure and execution unit are described in greater detail elsewhere [2].) Tight coupling permits full overlap of fetch, decode, and execute cycles. Overlap of these processing phases has impact on implementation of system features in the MC68000.

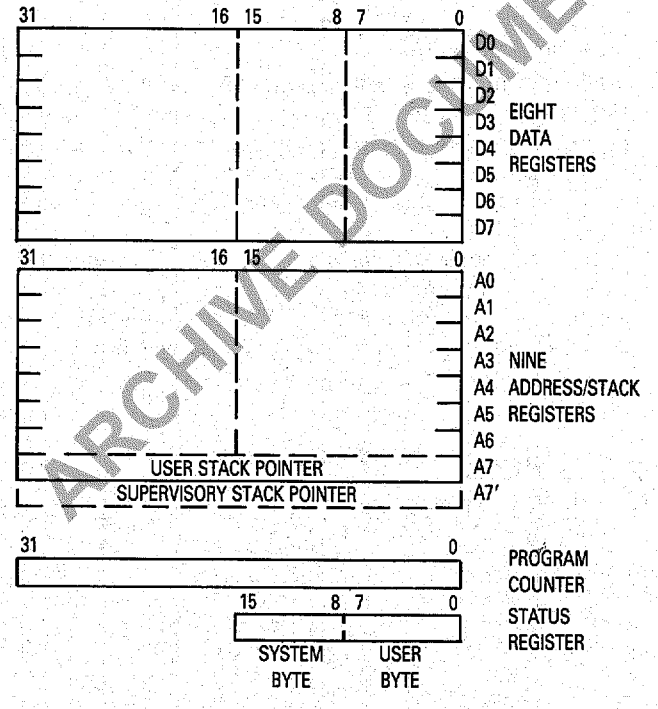

FIGURE 1. MC68000 REGISTERS

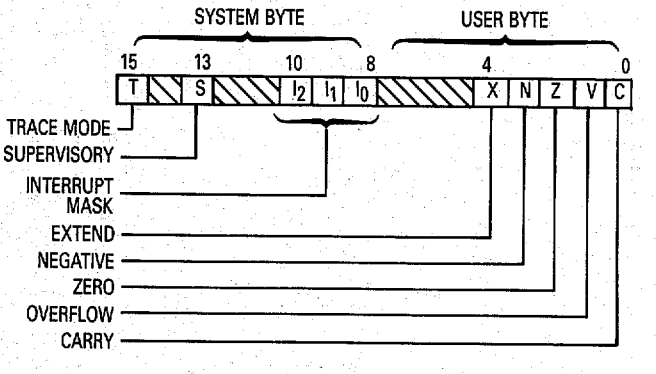

FIGURE 2. MC68000 STATUS REGISTER FORMAT

A basic block diagram of the two level control structure used by the MC68000 is shown in Figure 3. The micro control store contains a set of routines. Each routine is a sequence of micro orders which implements a macro instruction or a portion of a macro instruction (such as an addressing mode). The macro instruction register decode (Instruction Decode) provides a starting address to the micro control store which subsequently provides its own next addresses for a sequence of micro orders which performs operations required by a particular macro instruction. Each micro word contains an address which is used to reference a word in the nano control store. The nano control store contains the set of unique control words which is required to support the entire instruction set. Words in the nano control store are field-encoded such that with two to three levels of decoding they will directly drive control points in the execution unit. To aid in reducing the size of the control unit the MC68000 employs a residual control technique [1]. Information which remains static for the duration of a macro instruction is held in a register (not translated through the control stores) so that space in the control stores is reserved for information which changes from micro cycle to micro cycle.

A simplified view, as illustrated by Figure 4, assumes that instructions exhibit only fetch, decode, and execute cycles. The boundary between macro instructions is controlled by the execute cycle (which may require several machine cycles to complete). The basic philosophy of the control structure is that fetch, decode, and execute cycles will be overlapped across every macro instruction boundary. This implies that the micro routine for each macro instruction must insure that:

1) The next macro instruction word is accessed with sufficient time to be fully decoded by the end of the current macro instruction.
2) The word following the next macro instruction is fetched by the end of the current macro instruction.

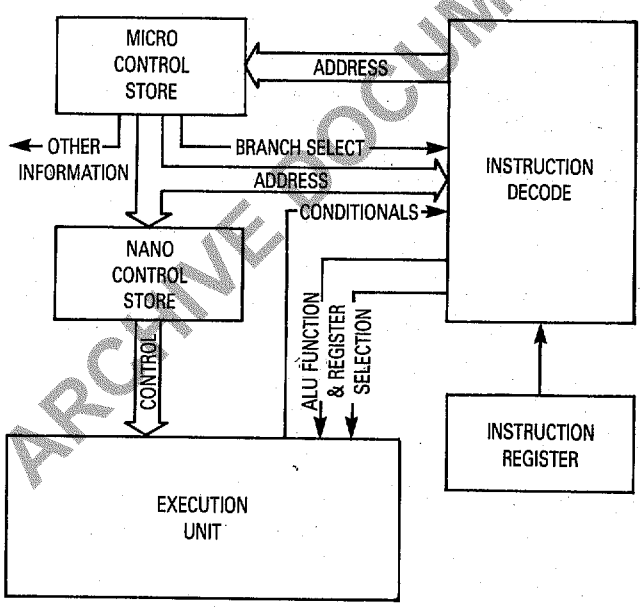

FIGURE 3. BLOCK DIAGRAM OF THE MC68000 CONTROL UNIT

F3	F4	F5			F6	F7
D2	D3	D4			D5	D6
E1	E2	E3	E4	E4	E4	E5

FIGURE 4. SIMPLIFIED INSTRUCTION EXECUTION SEQUENCE

As an example, assume a simple sequence of single word instructions as shown in Figure 5. It is the responsibility of the micro routine for the subtract instruction to ensure that the add instruction is placed into IR with sufficient time to decode and that a fetch is made to the word following the add instruction. Even if the subtract instruction consists of multiple words (additional words might contain an immediate value, displacement, or an address) the above stated constraints still apply. The micro routines will make as many accesses to the instruction stream as there are words in the definition of the associated macro instruction. The accesses will, however, be skewed forward by two words to provide the necessary overlap.

SUB
ADD
CMP

FIGURE 5. SIMPLE SEQUENCE OF INSTRUCTIONS

2. System Features

The MC68000 includes features beyond efficient instruction execution. It also has system features for easier program and memory management, and for handling of exceptional conditions.

2.1. Privilege States

The MC68000 processor operates in one of two states of privilege: the "user" state or the "supervisor" state. The privilege state determines which operations are legal. It is used to choose between the supervisor stack pointer and the user stack pointer in instruction references.

When the processor starts a bus cycle, it classifies the reference via an encoding on the three Function Code pins. This allows external translation of addresses, control of access, and differentiation of special processor states, such as interrupt acknowledge.

Table 1. Classification of References

Function Code	Reference Class
0 0 0	(Reserved)
0 0 1	User Data
0 1 0	User Program
0 1 1	(Reserved)
1 0 0	(Reserved)
1 0 1	Supervisor Data
1 1 0	Supervisor Program
1 1 1	Interrupt Acknowledge

2.1.1. Supervisor/User State

For instruction execution, the supervisor state is determined by the S-bit of the status register; if the S-bit is on, the processor is in supervisor state, otherwise, it is in the user state. The supervisor state is the higher state of privilege. All instructions can be executed in supervisor state. The bus cycles generated by instructions executed in supervisor state are classified as supervisor references. While the processor is in the supervisor privilege state, those instructions which use either the system stack pointer implicitly or address register seven (A7) explicitly access the supervisor stack pointer.

The user state is the lower state of privilege. Most instructions execute the same in user state as in supervisor state. However, some instructions which have important system effects are made "illegal." For example, to insure that a user program cannot enter the privileged state except in a controlled manner, the instructions which modify the entire status register are privileged. The bus cycles generated by an instruction executed in user state are classified as user state references. While the processor is in the user privilege state, those instructions which use either the system stack pointer implicitly, or address register seven (A7) explicitly access the user stack pointer.

2.1.2. Change of Privilege State

Once the processor is in the user state executing instructions, only exception processing (described below) can change the privilege state. During exception processing the current setting of the S-bit of the status register is saved, and the S-bit is forced on, putting the processor in supervisor state. Thus when instruction execution resumes at the address specified to process the exception, the processor is in the supervisor privilege state.

The transition from supervisor to user state can be accomplished by any of four instructions: RTE, MOVE to Status Register, ANDI to Staus Register, and EORI to Status Register. The RTE instruction fetches the new status register and program counter from the supervisor stack, loads each into its respective register, and then begins the instruction fetch at the new program counter address in the privilege state determined by the S-bit of the new status register. The MOVE, ANDI, and EORI to Status Register instructions each fetch all operands in the supervisor state, perform the appropriate update to the status register, and then fetch the next instruction at the next sequential program counter address in the privilege state determined by the new S-bit.

2.1.3. Implementation

The creation of privilege states and the subsequent system features affected implementation cost of the processor. Inclusion of separate implicit stack pointers for the user and supervisor states caused increased complexity in the register decoders which had to distinguish between a user and supervisor register for all implicit and explicit references to the system stack pointer (A7). Addition of one 32-bit register to the execution unit was not a major cost factor, but did increase the width of the execution unit. Since the supervisor mode is used to create user environments, instructions were required which allowed manipulation of the user stack pointer while in supervisor mode. This further complicated the register decoders and forced introduction of an additional specialized decoder for explicit control of the user stack pointer while in supervisor mode.

The function code pins are employed to classify processor bus cycles. Two bits in the micro control store classify an access as data space, program space, interrupt acknowledge, or unknown. The unknown state is combined with a value from a special decoder to determine whether the associated access is to data or program space. Unknown states occur in micro routines which may be shared by data space access macro instructions and instruction space access macro instructions. For example, the base plus displacement addressing mode (data space access) and the program counter plus displacement addressing mode (program space access) share the same micro routine.

The user/supervisor function code information is obtained from the status register. User/supervisor information is not provided by the micro control store due to micro control store word width constraints. Its exclusion from the micro control store implies that there must exist some means in the nano control store for direct manipulation of the user/supervisor bit in the status register during exception processing. In addition, there are several privileged instructions which can change the user/supervisor bit. Any prefetches done prior to manipulation of the status register must be discarded. Since the micro routines for manipulation of the user byte of the status register are shared with routines for manipulation of the entire status register, the delay associated with ignoring the prefetches is suffered by both instruction types.

2.2. Exception Processing

2.2.1. Processing States

The processor is always in one of three processing states: normal, exception, or halted. The normal processing state is that associated with instruction execution; the bus cycles are to fetch instructions and operands, and to store results. The STOP instruction is a special case of the normal state in which no further bus cycles are started.

The exception processing state is associated with interrupts, trap instructions, tracing and other exceptional conditions. The exception may be internally generated, by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, by a bus error, or by a reset. Exception processing is designed to provide an efficient context switch so that the processor may handle unusual conditions.

Exceptions can be grouped according to their generation. The Group 0 exceptions are Reset, Bus Error, and Address Error. These exceptions cause the instruction currently being executed to be aborted, and the exception processing to commence at the next minor cycle of the processor. The Group 1 exceptions are trace and interrupt, as well as the privilege violations and illegal instructions. These exceptions allow the current instruction to execute to completion, but preempt the execution of the next instruction by forcing exception processing to occur.

The Group 2 exceptions occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, and Zero Divide exceptions are in this group.

Table 2. Exception Groups

Group 0	Reset Bus Error Addr Error	Exception processing begins at the next minor cycle
Group 1	Trace Interrupt Illegal Privilege	Exception processing begins before the next instruction
Group 2	TRAP, TRAPV, CHK, Zero Divide	Exception processing is started by normal instruction execution

The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a Bus Error another Bus Error occurs, the processor assumes that the system is unusable and halts.

2.2.2. Exception Processing Initiation

The processor hardware recognizes three distinct types of exception conditions: internal exceptions (Group 2), non-catastrophic exceptions (Group 1), and catastrophic exceptions (Group 0). Exception processing for Group 2 exceptions is initiated through normal instruction execution. Group 2 exceptions are detected and processed via micro routines without the aid of specialized additional hardware.

When a Group 1 exception arises, execution of the current macro instruction continues unaffected (including prefetch and decode of the next macro instruction). At the end of the current macro instruction, the micro routine specifies that the next micro control store address is to come from the macro instruction register decoder. However, the existence of a Group 1 exception condition will force substitution of a micro control store address for the appropriate exception processing micro routine.

Occurrence of any Group 0 exception implies that the currently executing micro routine cannot continue; the exception micro routine address preempts the current micro routine at the next minor cycle.

2.2.2.1. Exception Vectors

Exception vectors are memory locations from which the processor fetches the address of a routine which will handle that exception. All exception vectors are two words in length, except for the reset vector, which is four words in length. A vector number is an eight-bit number, which when multiplied by four gives the address of an exception vector. Vector numbers are generated internally or externally, depending on the cause of the exception. The exception vectors are assigned to low addresses in the supervisor data space.

2.2.2.2. Exception Processing Sequence

All exception processing is done in supervisor state, regardless of the setting of the S-bit in the status register. The bus cycles generated during exception processing are classified as supervisor references. All stacking operations during exception processing use the supervisor stack pointer.

Exception processing occurs in four identifiable steps. During the first step, an internal copy is made of the status register. After the copy is made, the special processor state bits in the status register are changed. The S-bit is forced on (1), putting the processor into supervisor privilege state. Also, the T-bit is forced to 0 (off), which will allow the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated.

In the second step, the vector number of the exception is determined. For interrupts, the vector number is obtained by a processor fetch, classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the address of the exception vector.

The third step is to save the current processor status. Only for the Reset exception is this not done. The current program counter value and the saved copy of the status register are stacked using the supervisor stack pointer. The program counter value stacked usually points to the next unexecuted instruction. Additional information defining the current context is stacked for the Bus Error and Address Error exceptions.

The last step is the same for all exceptions. The new program counter value is fetched from the exception vector. The processor then resumes instruction execution. The instruction at the address given in the exception vector is fetched, and normal instruction decoding and execution is started.

2.2.2.3. Reset

2.2.2.3.1. Description

The Reset pin provides the highest level exception. The processing of the Reset signal is designed for system initiation, and recovery from catastrophic failure. Whatever processing was in progress at the time of the reset is aborted. The processor interrupt priority mask is set at level seven. The vector number is internally generated to reference the reset exception vector at location 0 in the supervisor program space. Because no assumptions can be made about the validity of register contents, in particular the supervisor stack pointer, neither the program counter nor the status register is saved. The address contained in the first two words of the reset exception vector is used to initialize the supervisor stack pointer, and the address in the next two words is used to initialize the program counter. Finally instruction execution is started at the address in the program counter.

2.2.2.3.2. Hardware Support

Hardware support for reset permeates the machine because reset must provide machine initialization from any internal state. Activation of the reset pin preempts all other pending conditions and current activities. Normal operation of the control unit is suspended and the control unit is forced to a state from which it begins executing the reset micro routine. Bits in the nano control store are provided to allow the micro routine to obtain the reset

vector address, force the machine into supervisor mode, and set the priority mask (to the level specified by a decoder — in this case, level seven). Additionally, since the register designators for the preempted instruction are unknown, the nano control store must provide bits which can directly specify selection of the implicit stack pointer for its initialization.

2.2.2.4. Interrupts

2.2.2.4.1. Description

The MC68000 provides seven levels of interrupt priorities. Devices may be chained externally within interrupt priorities, allowing an unlimited number of peripheral devices to interrupt the processor. Interrupt priority levels are numbered from one to seven, level seven being the highest priority. The status register contains a three-bit mask which indicates the current processor priority, and interrupts are inhibited for all priority levels less than or equal to the current processor priority.

An interrupt request is made to the processor by encoding the interrupt request level on the interrupt request pins, a zero indicates no interrupt request. Interrupt requests arriving at the processor do not force immediate exception processing, but are made pending. Pending interrupts are detected between instruction executions. If the priority of the pending interrupt is lower than or equal to the current processor priority, execution continues with the next instruction and the interrupt exception processing is postponed. (The recognition of level seven is slightly different, as explained below.)

If the priority of the pending interrupt is greater than the current processor priority, the exception processing sequence is started. First a copy of the status register is saved, and the privilege state is set to supervisor, tracing is suppressed, and the processor priority level is set to the level of the interrupt being acknowledged. The processor fetches the vector number from the interrupting device, classifying the access as an interrupt acknowledge and displaying the level number of the interrupt being acknowledged on the address bus. External logic can respond to the interrupt acknowledge read in one of three ways: put a vector number on the data bus, request automatic vectoring, or indicate that no device is responding (Bus Error). If external logic requests an automatic vectoring, the processor internally generates a vector number which is determined by the interrupt level number. If external logic indicates a Bus Error, the interrupt is taken to be spurious, and the generated vector number references the spurious interrupt vector. The processor then proceeds with the usual exception processing. Normal instruction execution commences in the interrupt handling routine.

Priority level seven is a special case. Level seven interrupts cannot be inhibited by the interrupt priority mask, thus providing a "non-maskable interrupt" capability. An interrupt is generated each time the interrupt request level changes from some lower level to level seven.

2.2.2.4.2. Hardware Support

On-chip logic provides detection and comparison of interrupt requests. Arrival of interrupt requests does not affect execution of the current instruction. If an interrupt of sufficient priority arrives, a pointer to the interrupt micro routine will be substituted for the micro routine pointer from IR decode at the next macro instruction boundary. An interrupt acknowledge is accomplished by the interrupt micro routine via an internal path involving no less than six separate registers. Support for translation and extension of interrupt vector addresses and creation of interrupt auto vector addresses is the responsibility of the field translate hardware in the MC68000. The micro routine uses the address from this special function unit as a pointer to the location of the program counter for the particular interrupt. Vectored, auto vectored, and spurious interrupts are all handled by the same micro routine; the differences occur in vector generation by the field translate unit.

2.2.2.5. Internally Generated Exceptions

2.2.2.5.1. Description

Traps are exceptions caused by instructions. They arise either from processor recognition of abnormal conditions during instruction execution, or from use of instructions whose normal behavior is trapping.

Some instructions are used specifically to generate traps. The TRAP instruction always forces an exception, and is useful for implementing system calls for user programs. The TRAPV and CHK instructions force an exception if the user program detects a runtime error, which may be an arithmetic overflow or a subscript out of bounds. The divide instructions will force an exception if a division operation is attempted with a divisor of zero.

Illegal instruction is the term used to refer to any of the word bit patterns which is not the bit pattern of the first word of a legal MC68000 instruction. Those word patterns with bits [15 12]=1010 or 1111 are distinguished as unimplemented instructions, and separate exception vectors are given to these patterns to permit efficient emulation. If during instruction execution such an illegal instruction is fetched, an illegal instruction exception occurs. This facility allows the operating system to detect program errors, or to emulate unimplemented instructions in software.

In order to provide system security, various instructions are privileged. An attempt to execute one of the privileged instructions while in the user privilege state will cause an exception.

To aid in program development, the MC68000 includes a facility to allow instruction by instruction tracing. In trace state, after each instruction is executed an exception is forced, allowing a debugging program to monitor the execution of the program under test.

The trace facility uses the T-bit in the supervisor portion of the status register. If the T-bit is off (0), tracing is disabled, and instruction execution proceeds from instruction to instruction as normal. If the T-bit is on (1), at the beginning of the execution of an instruction, a trace exception will be generated after the execution of that instruction is completed. If the instruction is not executed, either because an interrupt is taken, or the instruction is illegal or privileged, the trace exception does not occur. If the instruction is executed and an interrupt is pending on completion, the trace exception is processed before

the interrupt exception. If the instruction generates an exception, the generated exception is processed before the trace exception.

As an extreme illustration of the above rules, consider the arrival of an interrupt during the execution of a TRAP instruction while tracing is enabled. First the trap exception is processed, then the trace exception, and finally the interrupt exception. Instruction execution resumes in the interrupt handler routine.

2.2.2.5.2. Hardware Support

Trace, privilege violation, illegal instruction, and all instructions which cause a trap are handled in much the same fashion as an auto vectored interrupt by the hardware. They all share (except for some small initial differences) a single micro routine. Again, as with interrupts, the field translate unit provides the vector address for the program counter. The decode of an illegal instruction, or a privileged instruction in user mode causes the macro instruction decode logic to generate a pointer to a special micro routine which returns the machine to supervisor mode and effects a trap. Considerable additional hardware is required to detect these errors and to create the address of the exception vector; and increased control store space is necessary for the special micro routine.

2.2.2.6. Bus Error/Address Error

2.2.2.6.1. Description

Bus Error exceptions occur when external logic requests that a Bus Error be processed by an exception. The current bus cycle which the processor is making is aborted. Whatever processing the processor was doing, instruction or exception, is terminated, and the processor immediately begins exception processing.

Exception processing for Bus Error follows the usual sequence of steps. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is generated to refer to the Bus Error vector. Since the processor was not between instructions when the Bus Error exception request was made, the context of the processor is more detailed. To save more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are of course saved. Besides the usual information, the processor saves the internal copy of the first word of the instruction being processed, and the address which was being accessed by the aborted bus cycle. Also saved is specific information about the access: whether it was a read or a write, whether the processor was processing an instruction or not, and the classification displayed on the function code pins when the Bus Error occurred. Although this information is not sufficient in general to effect full recovery from the Bus Error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address contained in the Bus Error exception vector.

If a Bus Error occurs during the exception processing for a Bus Error, Address Error, or Reset, the processor is halted, and all processing ceases. This simplifies the detection of catastrophic system failure, since the processor removes itself from the system rather than destroying all memory contents. Only the RESET pin can restart a halted processor.

Address Error exceptions occur when the processor attempts to access a word or a long word operand at an odd address. The effect is much like an internally generated Bus Error, so that the bus cycle is aborted, and the processor begins exception processing. After exception processing commences, the sequence is the same as that for Bus Error, except that the vector number refers to the Address Error vector. Likewise, if an Address Error occurs during the exception processing for a Bus Error, Address Error, or Reset, the processor is halted.

2.2.2.6.2. Hardware Support

During execution of a micro routine the program counter value is often moved to a temporary register where it is manipulated. An updated value of the program counter is returned to the program counter register prior to the end of the micro routine. Macro instructions contain from one to five words and it is not convenient or efficient to attempt to maintain an updated value in the program counter throughout all micro routines. Since occurrence of a Group 0 exception truncates execution of the current micro routine the internal state of the execution unit and, most importantly, the program counter (which is stacked during exception processing) are not well defined for the current implementation. Conditions for processing a Group 1 or a Group 2 exception are such that the program counter is always well-defined.

2.2.3. Multiple Exceptions

2.2.3.1. Description

This section describes the processing which occurs when multiple exceptions arise simultaneously. Group 0 exceptions have highest priority; Group 2 exceptions have lowest priority. Within Group 0, Reset has highest priority, followed by Bus Error and Address Error. Within Group 1, trace has priority over external interrupts, which in turn takes priority over illegal instruction and privilege violation. Since only one instruction can be executed at once, there is no priority relation within Group 2.

The priority relation between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. The description above of the tracing a TRAP instruction when a interrupt arrives is an example of the application of the priority relation. In another example, if a Bus Error occurs during a TRAP instruction, the Bus Error takes precedence, and the TRAP instruction processing is aborted.

2.2.3.2. Hardware Support

It is possible for several exception conditions to be present at once. An exception priority network is used to provide hardware arbitration among multiple exception conditions which can occur. The network keeps track of the arrival and status of exception conditions, forms the micro control store starting address for the highest priority exception condition, and generates the address substitution signal at the appropriate time.

3. Summary and Conclusions

The MC68000 is a register oriented architecture with system features provided by carefully defined privilege states and exception processing. The privilege states divide processing into user and supervisor modes, with additional protection provided by functional separation of program and data space. Exception processing is defined to divide exception conditions into three logical priority groupings according to the manner in which they are handled by the hardware. In addition, a complete hierarchy is specified for hardware action in processing of multiple exceptions.

The two level microprogrammed control unit which implements the MC68000 architecture accommodates the priority groupings for exception conditions fairly easily. Additional hardware is required to provide support mechanisms associated with privilege states and exception vector generation. A priority encoder and extra logic are required to implement the hierarchical treatment of multiple exception conditions. Additional width in the nano control store words supports the implicit stack pointer references, privilege state changes, and vector address manipulation required for exception processing. Processing for the various exception conditions tends to be fairly homogeneous, so different exception conditions share complete or partial micro routine sequences (via join micro program flow). In the MC68000, clearly defined, well specified system features and a clean, regular control structure minimize the classical conflict between the architecture and the implementation.

References

[1] Alan B. Salisbury, *Microprogrammable Computer Architectures,* American Elsevier Publishing Company, Inc. (1976), pp 47–48.

[2] E. P. Stritter and H. L. Tredennick, "Microprogrammed Implementation of a Single Chip Microprocessor," *Proceedings of the 11th Annual Microprogramming Workshop,* Nov. 1978, pp 8–16.

Instruction Prefetch on the MC68000

The MC68000 uses a two-word tightly coupled prefetch mechanism to enhance performance. This mechanism is described in terms of the microcode operations involved.

DEFINITION: The execution of an instruction begins when the microroutine for that instruction is entered.

Using this definition, some features of the prefetch mechanism can be described.

1) When execution of an instruction begins, the operation word and the word following have already been fetched. The operation word is in the instruction decoder.
2) In the case of multiword instructions, as each additional word of the instruction is used internally, a fetch is made to the instruction stream to replace it.
3) The last fetch from the instruction stream is made when the operation word is discarded and decoding is started on the next instruction.
4) If the instruction is a single word instruction causing a branch, the second word is not used. But because this word is fetched by the preceding instruction, it is impossible to avoid this superfluous fetch. In the case of an interrupt or trace exception, both words are not used.
5) The program counter points to the last word fetched from the instruction stream.

```
            ORG     0;
            DATA.L  INITIAL_SSP;
            DATA.L  BEGIN;
            ORG     INTVECTOR;
            DATA.L  INTHANDLR;
            ORG     PROGRAM;
BEGIN:      NOP     ;
            BRA     LABEL;
            ADD     D0 TO D0;
LABEL:      SUB     DISP (A0) FROM A1;
            CMP     D2 TO D3;
            SGE     D7;
            ...
INTHANDLR:  MOVE.W  xxx.L TO yyy.L;
            NOP     ;
            SWAP;
            ...
```

FIGURE 1. CONTENTS OF MEMORY

The following example illustrates many of the features of prefetch. The contents of memory are assumed to be as illustrated in Figure 1.

The sequence we shall illustrate consists of the power-up reset, the execution of NOP, BRA, SUB, the taking of an interrupt, and the execution of the MOVE.W xxx.L to yyy.L. The order of operations described within each microroutine is not exact, but is intended for illustrative purposes only.

Microroutine	Operation	Location	Operand
Reset	Read	0	SSP High
	Read	2	SSP Low
	Read	4	PC High
	Read	6	PC Low
	Read	(PC)	NOP
	Read	+(PC)	BRA
	<begin NOP>		
NOP	Read	+(PC)	ADD
	<begin BRA>		
BRA	PC=PC+d		
	Read	(PC)	SUB
	Read	+(PC)	DISP
	<begin SUB>		
SUB	Read	+(PC)	CMP
	Read	DISP (A0)	<src>
	Read	+(PC)	SGE
	<begin CMP>	<take INT>	
INTERRUPT	Write	−(SSP)	PC Low
	Write	−(SSP)	PC High
	Read	<INT ACK>	Vector #
	Write	−(SSP)	SR
	Read	(VR)	PC High
	Read	+(VR)	PC Low
	Read	(PC)	MOVE
	Read	+(PC)	xxx High
	<begin MOVE>		
MOVE	Read	+(PC)	xxx Low
	Read	+(PC)	yyy High
	Read	xxx	<src>
	Read	+(PC)	yyy Low
	Read	+(PC)	NOP
	Write	yyy	<dest>
	Read	+(PC)	SWAP
	<begin NOP>		

FIGURE 2. INSTRUCTION OPERATION SEQUENCE

Literature Distribution Centers:
USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Center; 88 Tanners Drive, Blakelands Milton Keynes, MK145BP, England.
ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.; P.O. Box 80300; Cheung Sha Wan Post Office; Kowloon Hong Kong.

AI9566 PRINTED IN USA 5-86 IMPERIAL LITHO C39930 40,000