MOTOROLA
SEMICONDUCTOR
APPLICATION NOTE

Order this document
by AN947/D

AN947

MC68881 Floating-Point Coprocessor as a Peripheral in an M68000 System

ARCHIVE DOCUMENT NOT FOR NEW DESIGN

INTRODUCTION

The MC68881 floating-point coprocessor is a complete implementation of the proposed IEEE Standard for Binary Floating-Point Arithmetic (Task P754, Draft 10.0). All data formats, data types, operations, modes, conversions, and exception handling required in a conforming implementation of the proposed standard are supported entirely in hardware. Additionally, a full library of fast elementary transcendental functions is implemented in the hardware.

The MC68881 is a high performance floating-point unit designed to interface with the 32-bit MC68020 as a coprocessor; it provides a logical extension of the MC68020 instruction set and register set in a manner which is completely transparent to the programmer. The MC68020 microprocessor implements the M68000 coprocessor interface entirely in hardware. All interprocessor transfers are initiated by the MC68020. During the processing of an MC68881 instruction, the MC68020 transfers instruction information and data to the coprocessor via standard M68000 write bus cycles and receives data, requests for service, and status information from the coprocessor via standard M68000 read bus cycles. The MC68881 contains a number of coprocessor interface registers which are addressed like memory by the MC68020 micromachine. These registers, which are not part of the MC68881 programmer-visible register set, are mapped into CPU address space in an MC68020 system. The MC68020 distinguishes CPU address space accesses from program and data accesses by a unique function code encoding. (The MC68881 registers that are used by the floating-point instructions and hence visible to the programmer are the floating-point registers, FP0-FP7, and the control, status, and instruction address registers.)

The MC68881 can also be used as a peripheral processor in systems where the main processor does not implement the M68000 coprocessor interface on-chip (e.g. MC68000, MC68008, and MC68010 systems). Since the coprocessor interface is based solely on standard M68000 bus cycles, it is easily emulated by software in these systems. The MC68881 is considered to be a peripheral processor in these systems because its coprocessor interface registers are memory-mapped in data address space. Two methods of software emulation of the coprocessor interface are possible: 1) M68000 F-line instruction trap (traps are exceptions caused by instructions), or 2) in-line code implemented as either subroutine calls or macros.

When assembled for execution in an MC68020 system, the first word of an MC68881 instruction always has a hexadecimal F (binary 1111) in the most-significant nibble (Figure 1). When MC68000, MC68008, or MC68010 processors encounter an F-line instruction, the current processor status is saved, the F-line emulation trap vector is fetched, and instruction execution resumes in the trap handler. When this trap handler is a software emulation of the coprocessor interface, object code containing MC68881 instructions is upward compatible to an MC68020 system without recompiling, reassembling, or relinking. MC68881 instruction performance will significantly increase when such code is ported to a MC68020 system where the coprocessor interface is implemented by on-chip hardware. However, a performance penalty is paid for the upward compatibility provided by the F-line emulation trap method.

The current processor status (either three or four words depending upon the M68000 processor being used) must be pushed onto the stack when the exception is taken, and popped off of the stack when the RTE (return from exception) instruction is executed. More significantly, the trap handler must decode the MC68881 instruction to determine the proper protocol for a given instruction. The performance penalty can become intolerable if the coprocessor emulation trap handler must support all M68000 effective addressing modes. This can be relieved by utilizing only specific addressing modes.

Upward compatibility of object code is not relevant to every M68000 system, especially when it incurs a significant performance penalty. Such systems can emulate the coprocessor interface using in-line code, macros, or subroutine calls. Macros are particularly attractive since they provide the performance of in-line code while allowing the source code to be upward compatible to an MC68020 system via re-compilation or reassembly. This application note provides both a discussion of the coprocessor interface protocol followed by detailed information on both a macro technique and an example of an F-line instruction trap technique for software emulation of the coprocessor interface.

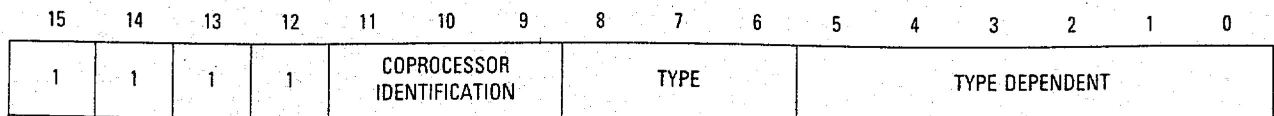
MC68881 AS AN MC68020 SYSTEM COPROCESSOR

COPROCESSOR CONCEPT

The M68000 coprocessor interface is the first general purpose coprocessor interface. The main processor instruction set and internal details are unknown to the coprocessor, and the coprocessor instruction set and internal details are unknown to the main processor. The burden

MOTOROLA

© MOTOROLA INC., 1987



Coprocessor ID
  Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the MC68881.
Type
  Specifies the type of coprocessor instruction:
    000 — General Instruction (Arithmetics, FMOVE, FMOVEM)
    001 — FDBcc, FScc, FTRAPcc
    010 — FBcc.W
    011 — FBcc.L
    100 — FSAVE
    101 — FRESTORE
    110 — (Undefined, Reserved)
    111 — (Undefined, Reserved)
Type Dependent
  Normally specifies the effective address or conditional predicate, but usage depends on the Type field.

Figure 1. MC68881 Coprocessor Operation Word Format

ARCHIVE DOCUMENT NOT FOR NEW DESIGN

of decoding the coprocessor instruction is not placed on the main processor, and the coprocessor does not monitor the main processor instruction stream.

The MC68881 instruction set and register set are logical extensions of the MC68020 sets. The MC68020/MC68881 execution model appears to the programmer as if implemented on one chip. All data transfers are performed by the main processor at the request of the MC68881; thus memory management, bus errors, address errors, and bus arbitration all function as if MC68881 instructions were executed by the main processor. The main processor also performs all effective address calculations and processes all coprocessor-detected exceptions at the request of the MC68881.

The interface is designed to provide the programmer with a sequential instruction execution model even though main processor and coprocessor instruction execution overlap occurs. For some instructions this overlap to enhances throughput.

MC68020 INTERPROCESSOR BUS TRANSFERS

The coprocessor interface is based upon standard M68000 asynchronous read and write bus cycles. No new bus signals are required by the MC68020 to initiate a floating-point instruction, and the MC68020 and MC68881 may run at different clock rates. The MC68881 is connected like a peripheral to the main processor and requires one extra pin, chip select (CS), in order to be accessed. Chip select is generated from the MC68020 function codes and address lines, similar to the way peripheral chip selects are generated. All other MC68881 pins connect to signals present on the M68000 Family processors.

The MC68881 has a set of coprocessor interface registers by which the main processor and coprocessor communicate. When performing a coprocessor access, the MC68020 outputs a 111 on its function code lines thus specifying CPU address space. Therefore, the MC68881 coprocessor interface registers are memory-mapped into CPU address space and do not infringe upon data or program address space. A portion of this CPU address space is allocated for coprocessor communication. The MC68020 outputs 0010 on bits 19-16 of the address bus for coprocessor accesses as shown in Figure 2. The MC68020 also outputs the Cp-ID field (bits 11-9 from the first word of the coprocessor instruction of Figure 1) on bits 15-13 of the address bus. The main processor selects the appropriate coprocessor interface register within the selected coprocessor via bits 4-0 of the address bus as shown in Figure 3. The MC68881 chip select is therefore based upon three elements: the MC68020 three function code outputs, the Cp-ID field (address bits 15-13 of the address bus), and the CPU space type field (bits 19-16 of the address bus).

Notice, the MC68020 handles only four CPU address space cycles:

CPU Space Type Field (A19-A16)	CPU Space Transaction
0000	Breakpoint Acknowledge
0001	Access Level Control
0010	Coprocessor Communications
1111	Interrupt Acknowledge

Therefore, when decoding the chip select for the MC68881, only two bits are needed (A18 and A17) to distinguish a coprocessor operation from the other CPU address space operations. A suggested method for connecting the MC68881 to the MC68020 is illustrated in Figure 4.

Figure 5 illustrates the connection of an MC68881 to an MC68000 or MC68010 as a peripheral processor over a 16-bit data bus. The MC68881 is configured to operate over a 16-bit data bus when the $\overline{SIZE}$ pin is connected to V_{CC}, and the A0 pin is connected to ground (GND). The sixteen least significant data pins (D0-D15) must be connected to the sixteen most-significant data pins (D16-D31) when the MC68881 is configured to operate over a 16-bit bus (i.e., connect D0 to D16, D1 to D17,...and D15 to D31). The $\overline{DSACK1}$ pin of the MC68881 is connected to the $\overline{DTACK}$ pin of the main processor, and the $\overline{DSACK0}$ pin is not used.

ADDRESS BUS

31–20	19–16	15–13	12–5	4–0
0 0 0 0 0 0 0 0 0 0 0 0	0 0 1 0	x x x	0 0 0 0 0 0 0 0	x x x x x
	COPROCESSOR COMMUNICATIONS	COPROCESSOR IDENTIFICATION		COPROCESSOR INTERFACE REGISTER SELECT

Figure 2. Coprocessor CPU Space Decoding (Function Code = 111)

ADDRESS BITS 0-4 IN COROCESSOR CPU SPACE (REGISTER OFFSET)

Offset	31–16	15–0
$00	RESPONSE (R)*	CONTROL (W)*
$04	SAVE (R)*	RESTORE (R/W)*
$08	OPWORD (R/W)	COMMAND (W)*
$0C	(RESERVED)	CONDITION (W)*
$10	OPERAND (R/W)*	
$14	REGISTER SELECTOR (R)**	(RESERVED)
$18	INSTRUCTION ADDRESS (W)*	
	OPERAND ADDRESS (R/W)	

*Denotes required registers for minimum support of all coprocessor instruction types. The MC68881 implements these registers.
**Denotes additional registers implemented by the MC68881.

Figure 3. Coprocessor Interface Registers

MC68020: FC0-FC2, A20-A31, A16-A19, A13-A15, A5-A12, A1-A4, A0, AS, DS, R/W, D24-D31, D16-D23, D8-D15, D0-D7, DSACK0, DSACK1
CHIP SELECT DECODE
MC68881: CS, SIZE, A1-A4, A0, AS, DS, R/W, D24-D31, D16-D23, D8-D15, D0-D7, DSACK0, DSACK1
V_{CC} → SIZE; V_{CC} → A0
MAIN PROCESSOR CLOCK
COPROCESSOR CLOCK
32-BIT DATA BUS CONNECTION

MC68020: FC0-FC2, A20-A31, A16-A19, A13-A15, A5-A12, A1-A4, A0, AS, DS, R/W, D24-D31, D16-D23, D8-D15, D0-D7, DSACK0, DSACK1
CHIP SELECT DECODE
MC68881: CS, SIZE, A1-A4, A0, AS, DS, R/W, D24-D31, D16-D23, D8-D15, D0-D7, DSACK0, DSACK1
V_{CC} → SIZE; GND → A0
MAIN PROCESSOR CLOCK
COPROCESSOR CLOCK
16-BIT DATA BUS CONNECTION

Figure 4. Suggested MC68020 to MC68881 Connections

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

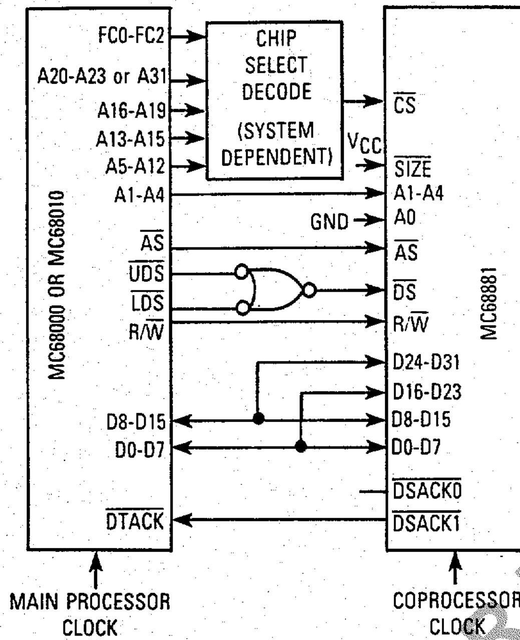


Figure 5. 16-Bit Data Bus Peripheral Processor Connection

When connected as a peripheral processor, the MC68881 chip select (CS) decode is system dependent. If the MC68000 is used as the main processor, the MC68881 CS must be decoded in the supervisor or user data spaces. However, if the MC68010 is used for the main processor, the MOVES instruction may be used to emulate any CPU space access that the MC68020 generates for coprocessor communications. Thus, the CS decode logic for such systems may be the same as in an MC68020 system, such that the MC68881 will not use any part of the data address spaces.

Figure 6 illustrates the connection of an MC68881 to an MC68008 as a peripheral processor over an 8-bit data bus. The MC68881 is configured to operate over an 8-bit data bus when the SIZE pin is connected to ground (GND). The eight least-significant data pins (D0-D7) must be connected to the twenty-four most-significant data pins (D8-D31) when the MC68881 is configured to operate over an 8-bit data bus (i.e., connect D0 to D8, D16, and D24; D1 to D9, D17, and D25;...and D7 to D15, D23, and D31). The DSACK0 pin of the MC68881 is connected to the DTACK pin of the MC68008, and the DSACK1 pin is not used.

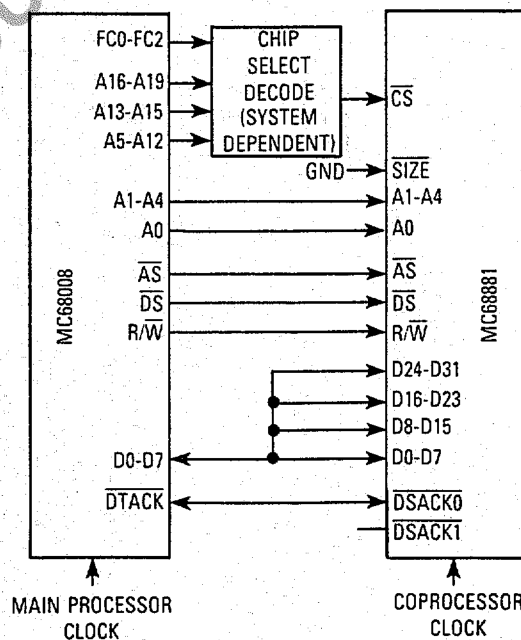


Figure 6. 8-Bit Data Bus Peripheral Processor Condition

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

When connected as a peripheral processor, the MC68881 chip select ($\overline{CS}$) decode is system dependent, and the $\overline{CS}$ must be decoded in the supervisor or user data spaces.

MC68881 INSTRUCTION DEFINITION

The MC68881 instructions can be subdivided according to the type of coprocessor operation performed: general, branch, save, restore, or conditional. Each instruction, when assembled, consists of from one to eight words (Figure 7). The first word (operation word) always has a hexadecimal F (1111) in the high-order nibble as seen in Figure 1. The type field (bits 8-6) of the operation word indicates the coprocessor instruction type. For instruction types which require an effective address (general, save, restore, and conditional), the type-dependent field of the operation word specifies the effective addressing mode. For the conditional instruction type, this field specifies the condition to be evaluated by the coprocessor: the conditional predicate (CPRED).

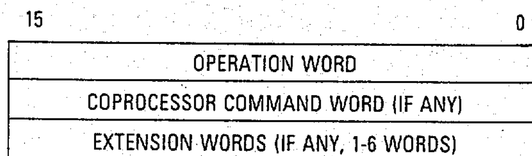


Figure 7. General Format of Coprocessor Instruction

The format of the second word of the coprocessor instruction varies with different instruction types. For the general instructions, the second word is the coprocessor command word specifying the function to be performed by the coprocessor. The MC68881 has been designed such that all general type operations are specified by a single command word. In order to process a conditional instruction type, the main processor must deliver the conditional predicate (CPRED) to the MC68881 for evaluation. Since the type-dependent field of the operation word may specify the effective addressing mode, the CPRED is found in the low-order six bits of the second word. The additional word(s) following the second word in Figure 8 is the extension to the effective addressing mode or the immediate operands present in the instruction. In the branch, save, and restore instructions, all information needed to initiate processing in the coprocessor is found in the operation word. Thus, the extension word(s) (if any) directly follows the operation word (no coprocessor command word).

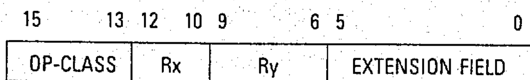


Figure 8. MC68881 General Instruction Command Word

All of the MC68881 arithmetic, move, move multiple, move constant, and transcendental instructions are of the general type. All general type instructions are initiated by the main processor writing the coprocessor command word (the second word of the instruction) to the MC68881 coprocessor interface command register. The format of the MC68881 command word is shown in Figure 8. General type instructions are broken down into groups, called op-classes, based on the function of the instruction and argument location (external or internal to the coprocessor). The values Rx, Ry, and the extension field depend on the specific op-class. For instance, the values required for a floating-point register to floating-point register operation are as follows: Rx is the source floating-point register, Ry is the destination floating-point register, and the extension field is the operation to be performed (add, move, sin, etc.). Table 1 lists the op-classes, their definitions, and their respective Rx, Ry, and extension fields.

For the branch and the conditional type instructions, the main processor initiates processing by writing the conditional predicate (CPRED-from the six low-order bits of either the first or second word of the instruction, respectively) to the coprocessor for evaluation. These conditional predicates are found on lines 62-97 of the EQUATE table listed in **APPENDIX A MACROS** at the back of this document.

In the case of an operating system context switch, the coprocessor internal state can be saved by the FSAVE instruction. This MC68881 instruction only saves the invisible state of the machine (that which is not normally available to the user). Thus all control, status, instruction address, and floating-point data registers (the user-visible registers) must be saved by the user. Only the registers beneficial (those being used) to the programmer need to be stored. To initiate the FSAVE instruction sequence, the main processor reads a word (the save format word) from the save interface register location of the coprocessor. The save format word provides the status of the coprocessor state (the null state, the idle state, or the busy state) and also the size (0 bytes, 24 bytes, or 180 bytes respectively) of the internal state of the machine to be saved. The save format word is written to the effective address by the main processor at the end of the instruction execution no matter which state the coprocessor is in.

The restore type instruction which restores a previously saved state is initiated by the main processor writing a save format word to the coprocessor interface restore register. This informs the coprocessor which coprocessor internal state is to be restored. If visible registers are saved after the execution of the FSAVE command, they must be restored before the execution of the FRESTORE instruction.

When executing any MC68881 instruction, the MC68020 follows a basic protocol. First, the coprocessor information (command word, conditional predicate, or format word) is written to the appropriate coprocessor interface register by the main processor (the FSAVE instruction is initiated by a read). The main processor then reads the appropriate coprocessor interface register to acquire the coprocessor status and any service requests. The coprocessor may indicate that it is busy processing a previous instruction and ask the main processor to query again. (This is the mechanism by which the sequential instruction execution is maintained because the coprocessor

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

Table 1. Command Word Fields of General-Type Instructions

Op-Class Field	Rx Field	Ry Field	Extension Field	Instruction Class
000	Source FP Data Register	Destination FP Data Register	Operation to Perform (MOVE, ADD, etc.)	FP Data Register to FP Data Register Instructions
001	—	—	—	Unused, Reserved
010	Source Data Format (see Note 2)	Destination FP Data Register	Operation to Perform (MOVE, ADD, etc.)	External Operand to FP Register
010	111	Destination FP Data Register	Constant ROM Offset	Move Constant to FP Data Register
011	Destination Data Format (see Note 2)	Source FP Data Register	0000000 (Unless Packed BCD, see Note 2)	Move out FP to External Destination
100	FPcr Select (see Note 1)	000	0000000	Move/Move Multiple to Control Register
101	FPcr Select (see Note 1)	000	0000000	Move/Move Multiple from Control Register
110	A/D S/D 0 (see Note 3)	00m (see Note 4)	Contains Register List	Move Multiple FP Data Register to MC68881
111	A/D S/D 0 (see Note 3)	00m (see Note 4)	Contains Register List	Move Multiple FP Data Register from MC68881

NOTES:

1. **FPcr** — **Floating-Point Control Register**
   - 000 Reserved
   - 001 FPIAR (Instruction Address Register)
   - 010 FPSR (Status Register)
   - 011 FPSR and FPIAR
   - 100 FPCR (Control Register)
   - 101 FPCR and FPIAR
   - 110 FPCR and FPSR
   - 111 FPCR, FPSR, and FPAIR

2. **Value** — **Data Format**
   - 000 Long Word Integer
   - 001 Single Precision Real
   - 010 Extended Precision Real
   - 011 Packed Decimal Real
   - 100 Word Integer
   - 101 Double Precision Real
   - 110 Byte Integer
   - 111 For Op-Class=011, Packed Decimal Real with Dynamic k Factor specified as CPU Data Register by Extension Field Encoding, rrr0000

3. A/D=0 Most-Significant Bit of Register List Selects FP7
   A/D=1 Most-Significant Bit of Register List Selects FP0
   S/D=0 Bit Mask in Extension Field
   S/D=1 Bit Mask in CPU Data Register Selected by Extension Field, 0rrr0000

4. When Bit Mask is Transferred to Command Word: "m" is the Most-Significant Bit of the Register List

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

will not execute a new instruction until finished processing the previous one). The coprocessor may indicate an exceptional condition and request the main processor to begin exception processing by providing the proper exception vector. The coprocessor may request additional service of the main processor, for example, evaluating effective address and transfering data through the coprocessor interface operand register. Finally, the coprocessor may indicate to the main processor that no further servicing is required.

INTERFACE REGISTERS

The MC68881 contains a number of interface registers which are memory-mapped within the MC68020 CPU address space. These are the registers identified by "*" and "**" in Figure 8. The coprocessor registers are memory-mapped into the main processor's system like any other peripheral, although they are accessed in a different address space.

The main processor initiates communication with the MC68881 by writing to (or reading from) a specific 16-bit

register, which is memory-mapped into the system. The specific register chosen depends on the instruction type. A general instruction coprocessor command word is written to the command register, and a branch or conditional instruction CPRED is written to the condition register. The main processor must read the save interface register to initiate the sequence of saving the internal state. To restore this state, the main processor writes to the restore interface register. The save and restore functions support virtual memory, demand paging, and task time-slicing.

After the initial write to the register in the general, conditional, and branch instructions, the response register is read by the processor to determine its next action (e.g., come-again or evaluate effective address and transfer data to/from the coprocessor).

An example of the communication sequence may be demonstrated with the memory to floating-point register add instruction. The main processor first writes the coprocessor command to the command interface register and queries the response register until requested to pass data. At which time, the host reads the data from memory and writes it to the operand register, four bytes at a time. The response register is re-read until the coprocessor signals the main processor to stop. The MC68020 is then free to process the next main processor instruction, while the MC68881 performs the floating-point add on the data. In this case, as in all normal processor/coprocessor communications, the host processor processses only the requests specified in the coprocessor primitives until released by the MC68881.

The MC68881 uses only three registers other than those previously mentioned. These are: the register select register used in the move multiple instructions, the instruction address register used only when exceptions are enabled, and the control register used by the main processor either to handshake the processing of the coprocessor exception or to abort invalid coprocessor service requests.

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

RESPONSE PRIMITIVES

Response primitives are service requests from the coprocessor to the main processor. These primitives have capabilities which allow for the synchronization of the main processor/coprocessor general, conditional, and branch instruction executions. Also the coprocessor may request services such as external memory accesses, transferal of data, and exception processing by the main processor. Figure 9 is a list of all possible coprocessor response primitives recognized by the MC68020. If the come-again (CA) bit is set, the main processor processes the primitive and then reads the response register again to seek further service requests. If the CA bit is not set, the main processor is released from further services (except when the MC68020 is in trace mode). If the PC bit is set, the main processor writes the program counter position of the first word of the coprocessor instruction to the instruction address interface register prior to performing the requested service. In the event that the MC68881 generated a trap exception, this PC value is required by the exception trap handler to determine which instruction caused the exception.

The MC68881 utilizes only six of the possible responses. These are the primitives noted by an "*" in Figure 9. The transfer multiple coprocessor registers primitive allows the transfer of multiple coprocessor registers to or from memory. The dr bit specifies in which direction the transfer is to be made: from the coprocessor to memory if the bit is set, or from the memory into the coprocessor if the bit is cleared. The transfer single main processor register requests the main processor to transfer the contents of one of its registers to or from the coprocessor. If D/A equals one, the transferred register is an address register. If D/A equals zero, a data register is transferred. The register number is located in the register field.

The evaluate effective address and transfer data primitive requests the main processor to evaluate the effective address specified by the floating-point instruction and to transfer data to or from that address (from or to the coprocessor). The valid EA type field indicates which addressing modes are valid for the transfer, while the length field gives the number of bytes to be transferred.

The Null primitive alerts the main processor to the coprocessor status after all other service requests (excluding exception requests) have been granted by the main processor. If the CA bit is set, the main processor queries the response register until the bit is cleared, at which time the main processor is released by the coprocessor. Even though the main processor may be signaled for release (when CA equals zero), it can still pass the program counter (PC equals one) and/or accept pending interrupts (IA equals one and CA equals one). The processing finished (PF) bit is a status bit which indicates whether or not the coprocessor has finished its instruction. The MC68020 tests the bit only while in trace mode to ensure that the instruction processing is complete. In the case of a conditional instruction the null primitive also contains a T/F bit (bit 0). This bit is tested by the main processor to determine whether or not the conditional predicate is true (one) or false (zero).

For all the MC68881 primitive responses, the CA bit is always set (CA equals one) with the exception of the null and exception request primitives. Both the take pre-instruction exception and the take mid-instruction exception primitives contain the exception vector number. Pre-instruction exceptions occur under two conditions: 1) after no further information is needed from the main processor in a previous floating-point instruction, and 2) before the coprocessor begins processing the present instruction. These pre-existing exceptions represent either an illegal command word for the present instruction, or the termination of previous instruction with an exception. This delayed reporting allows for synchronization between the host and the coprocessor in the event of any pre-existing exceptions. A floating-point register to memory move (op-class 001) operation is the only instruction capable of generating a mid-exception primitive. It is detected in the last read of the response register during instruction execution because the MC68881 performs the floating-point calculation and releases the main processor only after the data transfer to memory. With the memory to floating-point register (op-class 010) or the floating-point register to floating-point register (op-class 000), the main

BUSY

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0

*TRANSFER MULTIPLE COPROCESSOR REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	0	0	0	1	LENGTH (BYTES)							

TRANSFER STATUS AND SCANPC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	0	0	1	SP	0	0	0	0	0	0	0	0

SUPERVISOR CHECK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	0	1	0	0	0	0	0	0	0	0	0	0

TAKE ADDRESS AND TRANSFER DATA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	0	1	0	1	LENGTH (BYTES)							

*TRANSFER MULTIPLE MAIN PROCESSOR REGISTERS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	0	1	1	0	0	0	0	0	0	0	0	0

TRANSFER OPERATION WORD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	0	1	1	1	0	0	0	0	0	0	0	0

*NULL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	0	IA	0	0	0	0	0	0	PF	T/F

EVALUATE EFFECTIVE ADDRESS AND TRANSFER ADDRESS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	1	0	LENGTH (BYTES)							

*TRANSFER SINGLE MAIN PROCESSOR REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	1	1	0	0	0	0	0	0	D/A	REGISTER		

TRANSFER MAIN PROCESSOR CONTROL REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	1	1	0	1	0	0	0	0	0	0	0	0

Figure 9. Coprocessor Response Primitives (Sheet 1 of 2)

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

TRANSFER TOP-OF-STACK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	0	1	1	1	0	0	0	0	0	0	1	0	0

TRANSFER FROM INSTRUCTION STREAM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	1	1	1	LENGTH (BYTES)							

*EVALUATE EFFECTIVE ADDRESS AND TRANSFER DATA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	dr	1	0	VALID EA TYPE			LENGTH (BYTES)							

*TAKE PRE-INSTRUCTION EXCEPTION

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	0	0	VECTOR NUMBER							

*TAKE MID-INSTRUCTION EXCEPTION

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	0	1	VECTOR NUMBER							

TAKE POST-INSTRUCTION EXCEPTION

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	1	0	VECTOR NUMBER							

INVALID (RESERVED)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	1	1	1	1	1	x	x	x	x	x	x	x	x

WRITE TO PREVIOUSLY EVALUATED EFFECTIVE ADDRESS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	1	0	0	0	0	0	LENGTH (BYTES)							

INVALID (RESERVED)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	1	1	1	0	0	0	x	x	x	x	x	x	x	x

NOTES:

dr: 0 = Into Coprocessor
1 = Out of Coprocessor

D/A: 0 = Data
1 = Address

* Supported by MC68881

Valid EA Types: 000 = Control Alterable
001 = Data Alterable
010 = Memory Alterable
011 = Alterable
100 = Control
101 = Data
110 = Memory
111 = Any Mode Allowed (No Restrictions)

Figure 9. Coprocessor Response Primitives (Sheet 2 of 2)

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

processor is freed to execute the next instruction as the coprocessor performs the requested operation.

The save, restore, and move multiple instructions do not generate exceptions.

SOFTWARE TO EMULATE THE COPROCESSOR INTERFACE

DEFINITIONS AND ASSUMPTIONS

In order to utilize the floating-point coprocessor in a MC68000/MC68008/MC68010 system, a software emulation of the coprocessor interface must be developed. There are two possible methods of software emulation: 1) in-line code such as macros or subroutine calls, or 2) the M68000 F-line emulation trap.

The coprocessor must reside in a different address space than in a MC68020 system. When the MC68020 accesses the coprocessor, it does so in CPU space by outputting a 111 on the function code lines. The equivalent function codes generated on either MC68000, MC68008, or MC68010 signify an interrupt acknowledge bus cycle, i.e. these processors do not implement CPU space. Thus the MC68881 must be accessed as a peripheral with the coprocessor interface registers memory-mapped in data address space in these systems, not in CPU space.

To accommodate the practical use of this application, the demonstration software will perform a floating-point instruction in the fastest way possible while not violating the safety provided in the IEEE standard and the MC68881. To do this, a number of factors are first considered. The interprocessor protocol used with all instruction type classes may be minimized to include only those checking mechanisms necessary to perform the basic function. The most significant simplification is to restrict the use of the ENABLE byte of the floating-point control register. Each bit of this byte represents a type of coprocessor detected exception. Figure 10 identifies all coprocessor detected exceptions, their corresponding vector number (passed to the MC68020 in an exception primitive), and their position in the ENABLE byte. Two exceptions are not represented in this byte: the protocol violation and the illegal coprocessor command. This demonstration software ignores all take exception responses from the MC68881 (generated by/to the conditions specified in the ENABLE byte) which reduces the overhead required for their recognition in the software. An exception primitive, due to a floating-point operation, will be generated only if the MC68881 records an exception and the corresponding bit in the ENABLE byte is also set. Thus, an exception primitive cannot be generated if the default condition ($00) in the ENABLE byte is never altered by the user. This eliminates the need for monitoring the response register for pre-instruction or mid-instruction exceptions. If a coprocessor-detected exception occurs, it will never be detected by the main processor.

Note that two coprocessor detected exceptions, the protocol violation and the illegal coprocessor command, are not represented by a bit in the ENABLE byte. A protocol violation occurs anytime communication between the main processor and coprocessor is improper and is reported by the MC68881 as a mid-instruction exception. An illegal coprocessor command is a coprocessor command not implemented by the MC68881 and is reported as a pre-instruction exception. Therefore, by not checking for exceptions, a protocol violation or and illegal coprocessor command may occur without being detected.

Another consequence of the previously mentioned simplification is that the program counter will never be requested of the main processor if exceptions are disabled. The PC bit of the responses will never be set if the bits in the ENABLE byte are not set. Therefore, the overhead for testing of this bit can be saved.

	15	14	13	12	11	10	9	8
ENABLE BYTE	BSUN	SNAN	OPERR	OVFL	UNFL	DZ	INEX2	INEX1

ENABLE EXCEPTION BIT	VECTOR DEC	NUMBER HEX	VECTOR DEC	OFFSET HEX	ASSIGNMENT
BSUN	48	$30	192	$0C0	BRANCH OR SET ON UNORDERED CONDITION
INEX2/INEX1	49	$31	196	$0C4	INEXACT RESULT
DZ	50	$32	200	$0C8	FLOATING-POINT DIVIDE BY ZERO
UNFL	51	$33	204	$0CC	UNDERFLOW
OPERR	52	$34	208	$0D0	OPERAND ERROR
OVFL	53	$35	212	$0D4	OVERFLOW
SNAN	54	$36	216	$0D8	SIGNALING NAN
NONE	11	$0B	44	$02C	F-LINE EMULATOR
NONE	13	$0D	52	$034	COPROCESSOR PROTOCOL VIOLATION

Figure 10. MC68881 ENABLE Byte and Coprocessor Exceptions

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

IN-LINE CODE, MACROS, OR SUBROUTINES

The first approach discussed is the one to directly drive the MC68881 as a peripheral from the user program in user data space. This approach is used when speed of the MC68881 instruction execution is more important than upward compatibility of the object code. Two methods are available to drive the peripheral in user data space: in-line code (macros) or run-time libraries (subroutines). The trade-off between the two approaches concerns time versus space. Each time a macro is used the length of the source code increases by the size of the macro. When a subroutine is called, the overhead of the subroutine call and the execution of the RTS instruction must be incurred. No F-line trap is taken in either method, which saves the time to perform the corresponding stacking and instruction decoding.

Macros allow the coding of a repeated pattern of instructions which may contain variable entries at each iteration of the pattern. By incorporating the use of macros with conditional assembly instructions, some of the necessary floating-point instruction decode can be done during assembly time, reducing the run-time overhead. Assuming that the assembler used has the ability to pass parameters and manipulate them within the macro during assembly, the programmer need not generate the code to parse the floating-point instruction to detect the addressing mode used — it is passed directly by the macro into the assembled code. Subroutines can also act as an extension to the in-line routine to perform functions common to each instruction type (exception handling routines, error checking routines, etc.). The major advantages of using these methods over the F-line trap approach are: 1) the time saved by not taking the F-line trap, and 2) the time saved by assembly time instruction decode.

This application note includes software for the macro approach to drive the code in user data space. The same code would be pertinent if implementing a run-time library approach.

There are different ways to define the macros to drive the MC68881 depending on the particular application. The ideal method would be to write a macro supporting each floating-point instruction employed to achieve source code compatibility with the MC68020. In order to support the complete MC68881 instruction set, the library of macro definitions would be quite large. For conciseness, multiple coprocessor instructions are consolidated into single macro instructions, collected by the method of operand transfer required of the instruction (located in **APPENDIX A MACROS**). As an example, all MOVE-INS, MOVE-OUTS, etc. have their own macros corresponding to their respective directions and precisions. Most MC68881 instructions are supported, but source code compatibility with the standard MC68881 instruction set is lost due to consolidation of specific instructions into single macros. The conditional trap and the move multiple coprocessor system register instructions are not included in this example set.

Every macro requires at least the same amount of information supplied by the programmer as the represented MC68881 floating-point instruction and, in some cases, more information. This data is passed to the macro by parameters. An example of a general instruction class macro call is:

MACRONAME FUNCTION,SOURCE,DESTINATION

MACRONAME specifies the method of operand transfer, the FUNCTION is the general operation to be performed, the SOURCE is the location of the source operand, and the DESTINATION is the location of the destination operand. Different macros request different information of the user. For instance, the FSAVE instruction only requires one parameter to be passed to the macro. Basically, each macro follows a similar format and is described in **Functional Description.** The methodology of these macro definitions can be employed by the programmer who wishes to develop a separate macro for each instruction.

Functional Description

The following paragraphs provide the programmer with information on how to use the macros. All of the macros for the general instruction which transfer a source operand to the coprocessor (move-in) are listed below:

```
MEMREGB    Function,SourceEA,FPn
MEMREGW    Function,SourceEA,FPn
MEMREGL    Function,SourceEA,FPn
MEMREGS    Function,SourceEA,FPn
MEMREGD    Function,(An),FPn
MEMREGX    Function,(An),FPn
MEMREGP    Function,(An),FPn
```

They only differ by the precision of the data transferred. Each macro of this class transfers a source operand (specified by SourceEA or (An)) of a specific precision to the MC68881 and performs the operation specified by function. A list of the functions describing the supported MC68881 instructions and their functions can be found on lines 25-61 of the EQUATE table of the demonstration software found in **APPENDIX A MACROS.**

The macros which transfer data from a single floating-point register to an effective address (move out) are:

```
REGMEMB    FMOVE,FPm,DestinationEA
REGMEMW    FMOVE,FPm,DestinationEA
REGMEML    FMOVE,FPm,DestinationEA
REGMEMS    FMOVE,FPm,DestinationEA
REGMEMD    FMOVE,FPm,(An)
REGMEMX    FMOVE,FPm,(An)
REGMEMP    FMOVE,FPm,(An),[k-factor]
```

The MC68881 FMOVE command is the only MC68881 instruction supporting this direction of transfer. These macros request the passing of the instruction as a parameter to be consistent with the other macros. The k-factor, requested by REGMEMP, may be passed in data register D0 or as immediate data. Thus:

[k-factor] = D0

or

[k-factor] = #xxxx

A k-factor in the range +1 to +17 indicates the desired number of significant digits in the decimal mantissa after conversion to packed decimal format. A k-factor in the range -64 to 0 indicates the desired number of significant digits to the right of the decimal point in a fixed-point format.

ARCHIVE DOCUMENT: NOT FOR NEW DESIGN

The general instruction macro which performs "floating-point register to floating-point register" operations is:

REGREG Function,FPn,FPm,[FPq]

Since the MC68881 performs all data manipulations in extended precision (no user-specified precisions), only one macro is needed to support these general instructions. The new parameter introduced, [FPq], supports the one special case general instruction"FSINCOS" which generates the sine placing it in FPm and generates the cosine and placing it in FPq.

The constants supported on-chip by the MC68881 are available to the programmer from the coprocessor ROM with the macro:

FMOVEROM #CC,FPn

CC is the hex number representing the constant to be accessed. A list of the constants and their corresponding identification numbers is found on lines 660-681 of the EQUATE table in the demonstration software found in **APPENDIX A MACROS.**

The two macros which move data into or out of the coprocessor control, status, and instruction address registers are:

MOVINCSI SourceEA,Register
MOVOUCSI Register,DestinationEA

MOVINCSI moves data into the control, status, or instruction address registers, and MOVOUCSI moves data out. The register field is specified by CONTROL, STATUS, or IADDRESS corresponding to the register to be transferred.

As the M68000 microprocessors use the MOVEM instruction to move multiple registers into and out of memory, the MC68881 also supports moving multiple floating-point registers. The macros which support the movement of multiple floating-point data registers are:

FMOVEMMR SourceEA,fp0,fp1,fp2,fp3,fp4,fp5, fp6,fp7,Postincrement
FMOVEMRM fp0,fp1,fp2,fp3,fp4,fp5, fp6,fp7,DestinationEA,Predecrement

Each parameter of the fp0, fp1,...fp7 string represents the selection bit for that floating-point register. If a register is to be moved, then the corresponding parameter in the macro call must be set to a one (otherwise, the bit must be set to a zero). If the programmer decides to use the indirect addressing with postincrement mode in the FMOVEMMR macro, then the postincrement field must be set to a Y; otherwise, this parameter must be set to a N. The same system applies to the FMOVEMRM macro with respect to the predecrement field (Y if addressing mode is used, N if not).

The no operation or synchronizing instruction is supported by the macro:

FNOPP

No parameters are necessary to perform this function.

Only one macro is needed to support the branch instruction class, the conditional branch:

FBCC.[Size] Condition,Label

The size specification allows the macro to distinguish between a long branch and a short branch. If the size is not specified, the default is long. The user must specify the condition to be tested in the condition field. A list of the conditions and their corresponding label is found on lines 67-98 of the EQUATE table found in **APPENDIX A MACROS.** If after execution, the condition is satisfied, the macro will cause a branch to label.

The two macros which support the MC68881 decrement and branch and the conditional set instructions are:

FDBCC Condition,Dn,Label
FSCC Condition,Label

In both macros, the condition and label parameters serve the same purpose as those of the FBCC macro. The conditions are listed on lines 67-98 of the EQUATE table found in **APPENDIX A MACROS.** Note that Dn of FDBCC cannot be equal to D0 because it is used as a work register by the macro. Any other data register may be used.

The save or restore of the internal or invisible state of the coprocessor is executed by the macros:

FSAVEST −(An)
FRESTRST (An)+

To ensure proper restoration of the MC68881 state, the FRESTRST command should be used only on data stacked by the FSAVEST command. One exception to this rule is for a software reset. A software reset of the MC68881 occurs if a null save format word is stored on the stack and then restored into the MC68881. This can be used by the operating system to initialize the MC68881 when starting a new task.

Theory of Operation

The following paragraphs provide information on how to develop macros for the user who will either create his own, or need to modify the demonstration software to suit a particular application. The coprocessor recognizes each coprocessor instruction by the specific bit pattern written to the various coprocessor registers. The save instruction is an exception because it is initiated with a read from the save interface register. For each coprocessor instruction type class, a unique format for the bit pattern exists which is the basis for instruction grouping into macros. A detailed description of the macro development as well as a general discussion of each type instruction class follows.

All macros are developed following a simple protocol: 1) know the bit pattern of the information to be written to the coprocessor (the instruction to be performed) and write it to the appropriate interface register, 2) test for the known possible responses from the coprocessor, 3) perform requested operation (if any), and 4) test for the release of the main processor. If it is necessary to add exception detection, the programmer must add software to compare the response to the appropriate pre-exception or post-exception bit pattern and call a user-specified exception processing macro or subroutine if the result is positive.

Each MC68881 instruction follows a specific protocol starting from the write of the operation until the receipt of the null primitive. MEMREGn (n = B, W, L, S, D, X, P) and MOVINCSI follow the sequence shown in Figure 11. REGMEMn (n = B, W, L, S, D, X, P) and MOVOUCSI follow the sequence shown in Figure 12. Instructions represented by these REGMEMn macros generate at least one null come-again response after the initial write to the

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

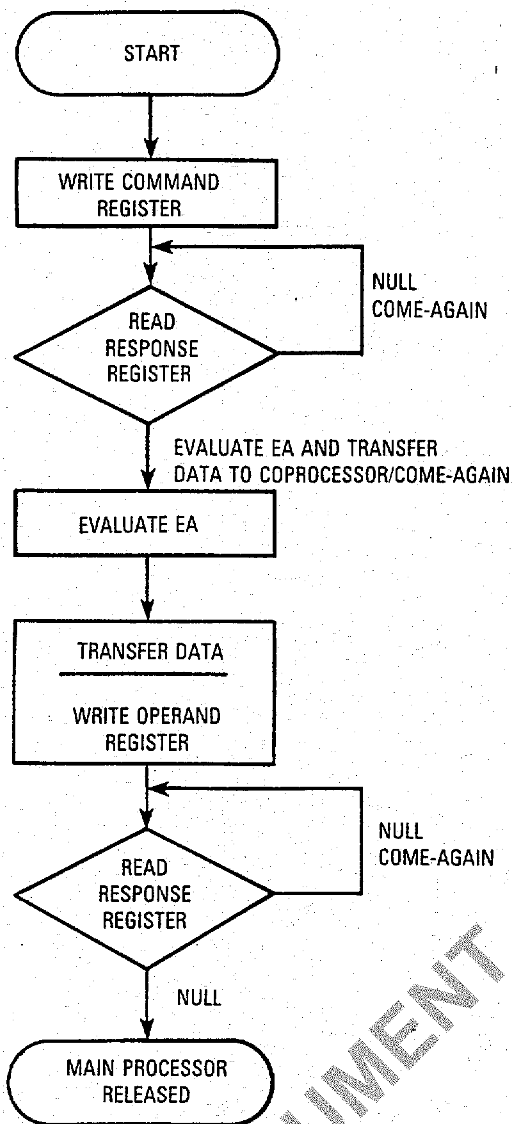


Figure 11. Move-In Sequence

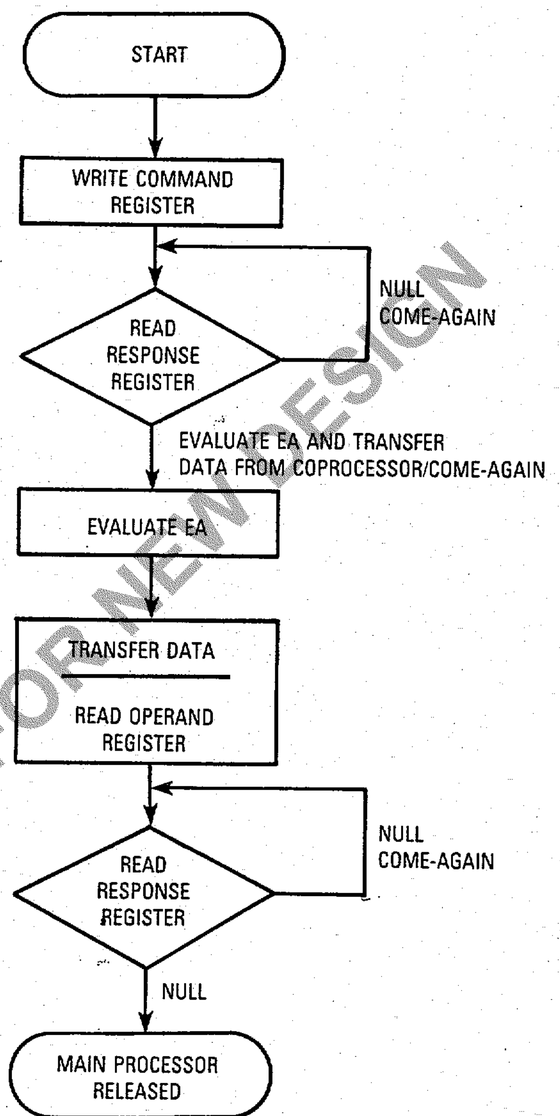


Figure 12. Move-Out Sequence

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

command interface register to allow the coprocessor to perform the specified operation before the transfer of data to memory. FMOVEMMR and FMOVEMRM follow the protocols seen in Figures 13 and 14, respectively. The demonstration software only supports the static forms of the instruction, i.e. the bit mask is transferred in the instruction rather than in a main processor data register (as in the dynamic form). Thus, Figures 13 and 14 represent only the static forms of the move multiple floating-point register instructions. The REGREG macro follows the sequence of Figure 15. The FDBCC, FBCC, and FSCC macros basically execute the same code except for the function to be performed after evaluating of the result of the conditional test as seen in Figure 16. Figures 17 and 18 represent the protocol followed by the FSAVE and FRESTORE instruction, respectively.

The macro detail will be explained by discussing an example of a general coprocessor instruction, the MEMREGW (lines 435-470) macro. This macro performs a floating-point operation on a word datum at an effective address pointed to by source EA address, and leaves the result in a specified floating-point register. The macro call takes the form:

MEMREGW Function,SourceEA,FPn

The function, SourceEA, and FPn are all parameters passed to the macro. Function is the operation of the MC68881 instruction, and FPn (n equals 0-7) is the specific floating-point register used. Both of these parameters represent a binary bit pattern. Thus, the need for an EQUATE table arises. An "EQU" statement (refer to the M68000 Assembler Manual) defines a symbol as a binary value when referenced anywhere in the source code. In the EQUATE table (lines 25-61) found in **APPENDIX A MACROS**, all general floating-point instructions are assigned the appropriate bit pattern (e.g. FMOVE EQU $00) to represent the extension field of the command word in Figure 8. Also, as seen on lines 119-126, each floating-point register (FPn) is equated to its corresponding numerical value (e.g. FP6 EQU $06). The other parameter passed to the MEMREGW macro is the effective address (SourceEA).

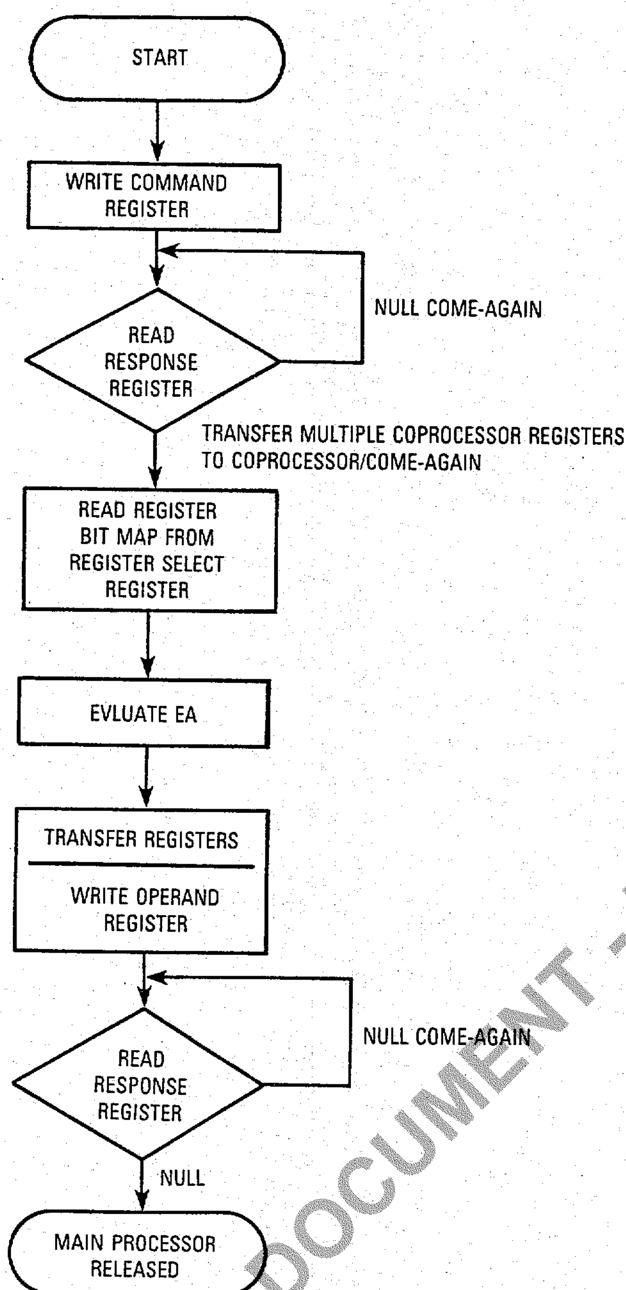


Figure 13. Move-Multiple-In Sequence

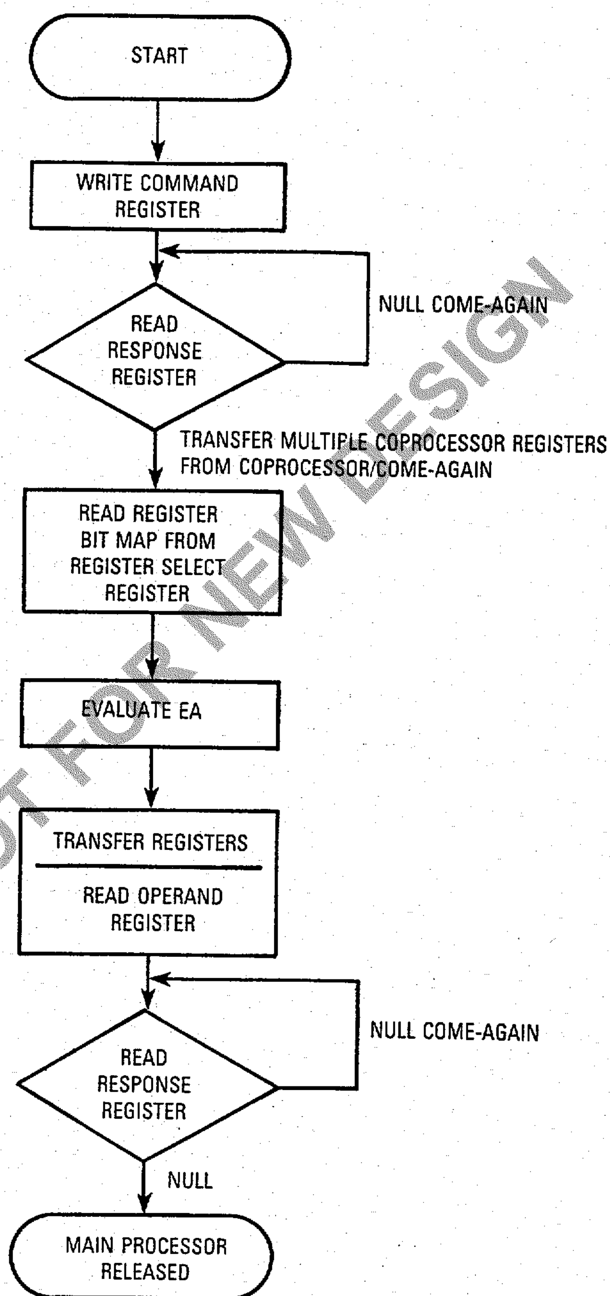


Figure 14. Move-Multiple-Out Sequence

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

Since parameters are separated by commas in the format of these macro calls, the indexed register indirect with offset addressing mode, (d(An,Dn)), cannot be passed as a single parameter. The comma between An and Dn causes the assembler to see this effective address as two parameters; Therefore, the macro will be passed one additional parameter for this case. Anticipating this case, when four parameters are passed in, the macro simply recombines the appropriate two parameters and reconstructs the effective address as a single parameter. The first line of this macro (line 452) is a conditional assembly command where the assembler tests for the occurrence of a fourth parameter, signifying the use of the indexed address mode. This test is done by comparing this parameter ('˜4') against a null character string (''). If a fourth parameter is present, a separate routine (lines 461-469) will be used to combine parameters ˜2 and ˜3 to reconstruct the effective address parameter.

Once the assembler has chosen which routine to assemble, the next task entails developing the command word shown on line 453 of the listing:

```
MOVEM.W
    #$5000+(˜3<<7)+▲1,MC68881+COMMAND
```

This task demonstrates how the command word is formed for all the effective addressing modes except the

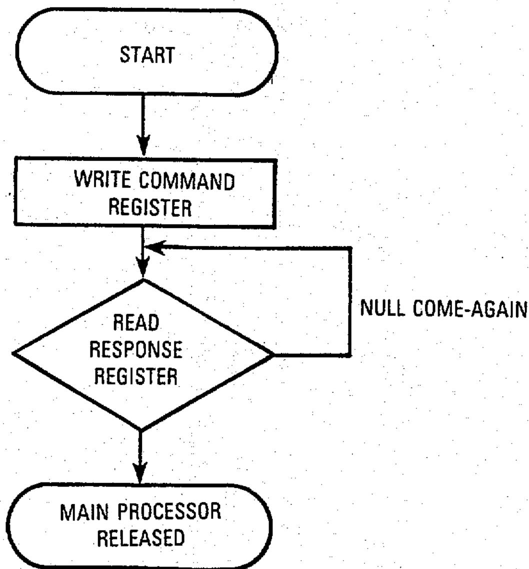


Figure 15. Register/Register Sequence

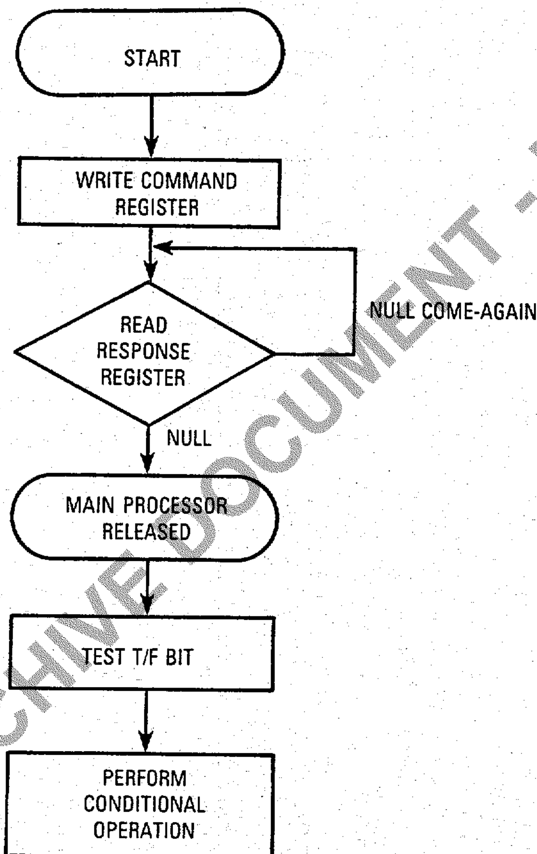


Figure 16. Branch and Conditional Sequence

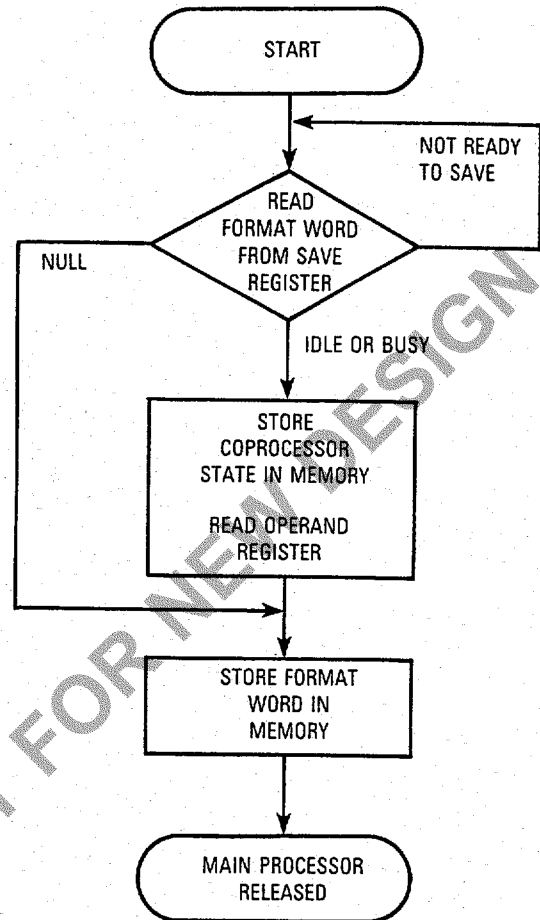


Figure 17. Save Sequence

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

indexed modes. The assembler, instructed by the arithmetic operator +, adds the three fields to generate the proper command word. The immediate data is the command word developed by the addition of the isolated fields seen in Figure 8. The command word base, 5000, represents the op-class 2 and data format for a word operand (RX). Each macro for a general instruction will have a unique command word base (shown in Table 1) specifying the op-class and data precision. A good understanding of the command word structure in Table 1 is helpful in developing general instruction macros.

The second field, Ry in Figure 8, is added to the command word by the assembler and represents the number of the floating-point register used in the transfer. This parameter is passed to the macro by the programmer as the third parameter, FPn. The symbol, <<, causes the assembler to shift the value of the third parameter to the left seven bits, placing it in its proper position in the command word.

The third field of the summation is the extension field which specifies the binary representation of the instruction to be performed. These representations are shown in the demonstration software in lines 25-61.

Note, when the addressing mode is indexed, the extension field remains the first parameter passed to the macro, but the Ry field becomes the fourth parameter

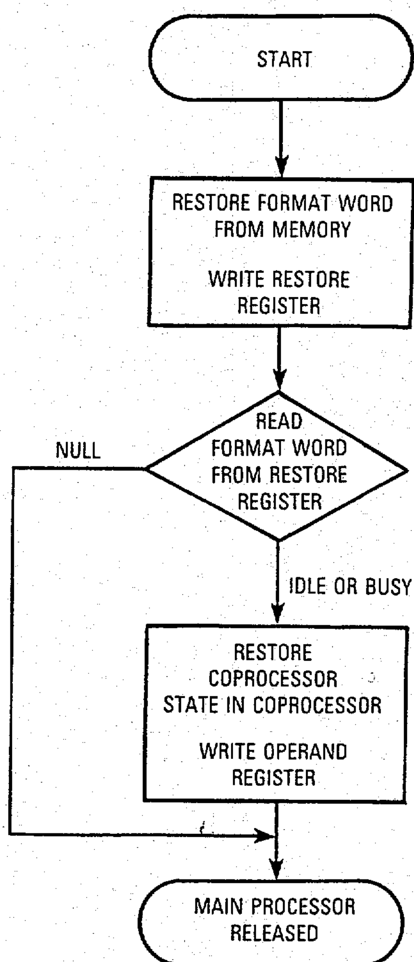


Figure 18. Restore Sequence

and is accommodated by line 462. When using the indexed addressing mode, the assembler needs to recreate a comma (in the indexed format) which was not passed with the parameters. In line 466, a comma is placed between the second and third parameters passed to the macro recreating the proper EA.

The command word is always written to the absolute address of the coprocessor command interface register. The demonstration software uses MC68881 to represent the base address of the coprocessor interface registers in data space. Specific interface registers are referenced by adding the displacement of that particular register to the base address. Consequently, each register has a symbol equated to the appropriate displacement in the EQUATE table (e.g., COMMAND EQU $0A) on lines 104-114.

Note that if the MC68881 is mapped into the highest page or the lowest page in the address map, the macros can use short absolute addressing mode instead of long absolute addressing mode. This will allow the macros to assemble into smaller object codes and may execute faster since the processor spends less time fetching extension words.

Once the command word is written to the command register, only two responses can be read from the response register: 1) null come-again and evaluate effective address and 2) transfer data. Only two responses will occur because exceptions are not allowed. By only testing the response register (line 454) for the null come-again ($8900), the main processor will pass the data when it reads any response other than the null come-again response.

This macro, as well as all other macros except FSAVEST and FRESTRST, must test the response register for the coprocessor release of the main processor. This service protects against spurious protocol violations. Protocol violations are unexpected accesses to the MC68881 interface registers. For example, the coprocessor may be expecting data to be written to the operand register but instead receives a write to the command register. A spurious violation occurs when an expected register access occurs sooner than expected in systems where the processor and coprocessors are running at different clock speeds. Since exceptions are assumed to be disabled by the macros, the CA bit is monitored to determine the coprocessor state. When CA is set to zero, the main processor is released. The following instructions perform this function throughout the macro definitions:

```
@NULREL  TST.B    MC68881+RESPONSE
         BMI.S    @NULREL
```

In summary, this example defines the sequence to be executed in all macros of the general instructions op-class 010. Each macro causes the main processor to write the move-in operation to the command register and to read the response register until asked to pass the data. After evaluating the effective address of the data and writing it to the operand register, the main processor rereads the response register until released by the coprocessor.

The packed BCD, double precisions, and extended precisions operations would require the use of several other conditional assembly instructions to support all the addressing modes that the byte, word, long word, and single precision macros allow. These instructions are necessary due to the fact that multiple accesses from memory are required to transfer data through the 32-bit wide operand register. To simplify this application, these three precisions, MEMREGD, MEMREGX, and MEMREGP, are only supported by the address register indirect addressing mode. The other addressing modes can be implemented by following the demonstration software as an example.

The move-out macros (REGMEMn, n = R, W, L, S, D, X, P) of op-class 011 of the general instruction class are structured in the same manner as MEMREGn (n = R, W, L, S, D, X, P). Coprocessor distinction between the move-in and the move-out operations result from the different op-class specifications within the command word.

The one difference between the two op-classes is in the packed BCD macros. This difference is due to the nature of the MC68881 FMOVE out packed BCD from the coprocessor instruction which requires the user to submit additional information to the coprocessor: the k-factor. The k-factor is passed to the operand register from either a data register or as immediate data in the command word. To be able to handle all data registers, the packed BCD macro would be extensive using elaborate conditional statements. Therefore, the programmer is only allowed to use data register D0, which fixes that part of

ARCHIVED DOCUMENT - NOT FOR NEW DESIGN

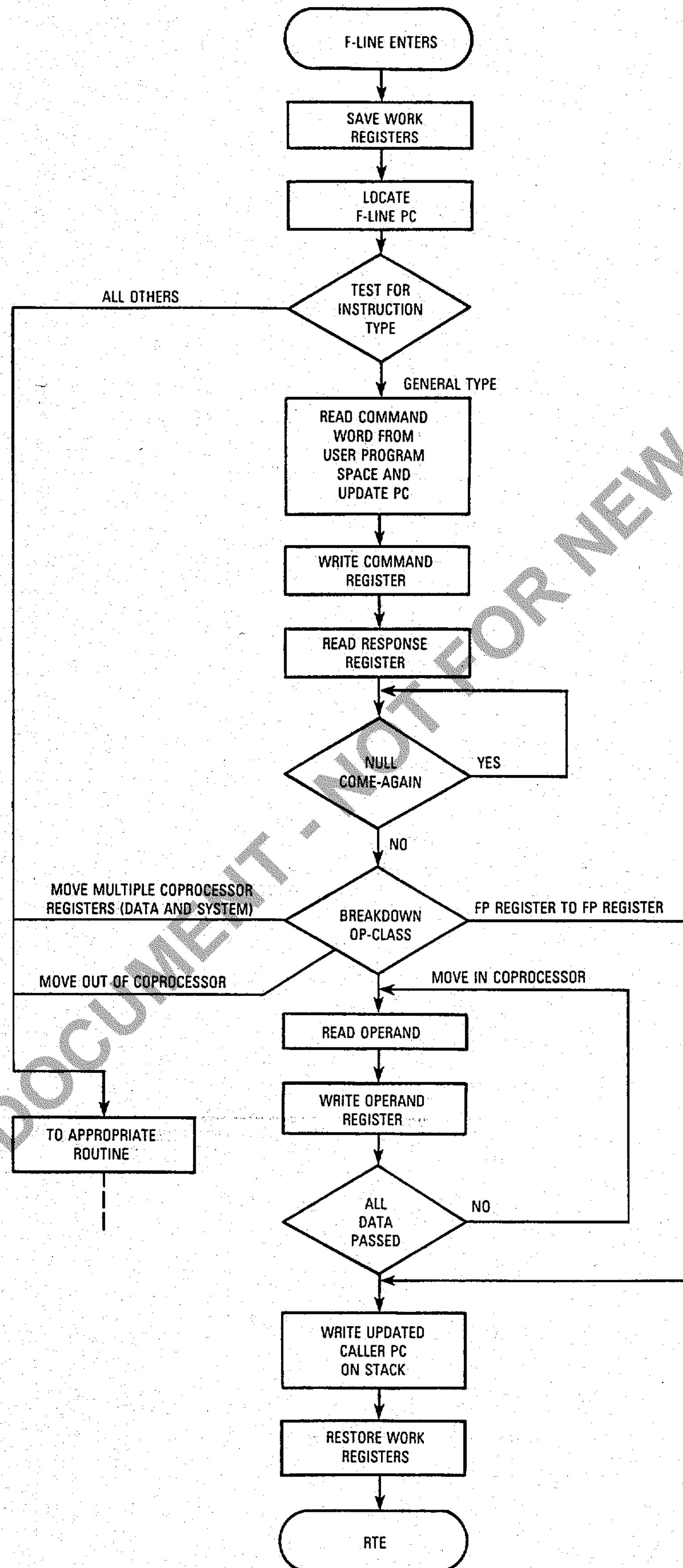


Figure 19. F-Line Emulation Sequence

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

the extension field representing the data register as a constant. Thus, only one pair of conditional instructions is needed.

The REGREG macro supports op-class 000 which performs a coprocessor register-to-register operation. No services are needed of the main processor other than to submit the coprocessor instruction. Therefore, after writing the command word to the command register, the response register is queried until the null primitive is granted. The several other conditional assembly statements in REGREG support the unique general arithmetic instruction, FSINCOS. Since this instruction requires two destination floating-point registers for the results of the operation (FPm and FPq), another parameter must be passed to the coprocessor. The conditional assembly statement tests for the existence of a fourth parameter. To be able to support this instruction in the REGMEMn and MEMREGn macros, similar procedures should be followed. An example is implemented in the MEMREGB macro (line 372-434).

Op-class 010 with Rx equal to 111 represents the operation performing the access of the coprocessor constants (FMOVEROM). A command word specifying the constant to be retrieved is written to the command register. Since no further services of the main processor are needed, the remaining function is to test the response register for the release signal (CA equals zero).

Both macros, MOVINCSI (op-class 100) and MOVOUCSI (op-class 101) move the coprocessor system registers. Each perform the same instruction sequence as MEMREGn and REGMEMn, respectively, with the only difference being the value of the command word. The move multiple coprocessor system register instructions are not supported by the macros.

The final op-classes of the general instructions to be discussed are those corresponding to the movement of multiple floating-point data registers (FMOVEMRM and FMOVEMMR). Due to the nature of the M68000 Family memory organization, the macros are constructed differently. The user specifies which registers are moved by selecting the corresponding parameter in the macro call, and the coprocessor detects which registers are affected by the bit mask specified in the command word. The binary bit mask is formed by the parameter list. The list is treated as a string of concatenated bits which is required by the MC68881 to represent the register select mask.

Since the floating-point data registers are 96 bits wide, three consecutive accesses of 32 bits each must be made to acquire the data. FMOVEMRM (lines 761-834) organizes this data so that the high-order bit is situated in low-order memory. The coprocessor delivers FP0 first (if selected) and FP7 last, except when the indirect addressing with predecrement mode is being used. In which case, the coprocessor sends FP7 first and FP0 last so that FP0 is always placed in low memory. Therefore, the conditional test for the predecrement mode is required to reverse the order of the bit mask sent to the coprocessor. The FMOVEMMR (lines 835-881) macro moves data into the registers by moving FP0 first as the coprocessor always expects FP0 first. The FMOVEMMR macro does not allow the predecrement addressing mode.

The second type class to be discussed is the branch instruction class which is supported by the FBCC macro. The main processor writes the conditional predicate (CPRED) to the condition register and reads the response register until signaled to be released. Then the T/F bit of the response primitive is examined, and if status indicates, the branch is taken.

The FDBCC and FSCC macros support the conditional type instructions. Both macros follow the same protocol as FBCC (i.e., write CPRED to the condition register, read the response register, and after being released, perform requested function if condition satisfied). All of the branch and conditional macros must modify a data register which serves as a temporary variable. When the coprocessor grants the null release primitive, the T/F bit is also passed in the response. As the MC68881 does not expect another response register access, the response is saved in D0 so the CA bit can be tested. When CA equals zero, the T/F bit is already available in D0.

The no-operation macro exists for the no-operation or synchronizing FNOP instruction. It is a branch never instruction. The main processor writes the second word of the coprocessor instruction to the condition register and queries the response register until released by the coprocessor.

One MC68881 conditional instruction not implemented is the conditional trap (FTRAPcc) instruction because the MC68020 has these coprocessor traps. The trap instruction is not available on the MC68000, the MC68008, or the MC68010. To cause a trap from the users space in a MC68000/MC68008/MC68010 system, the overflow bit in the control register can be set, and the TRAPV instruction executed. However, the trap handler can not distinguish between the simulated coprocessor condition and the overflow condition that would normally use this trap vector.

The final two coprocessor instruction types to be discussed are the save and restore performed by macros FSAVEST and FRESTRST, respectively. Only one addressing mode is supported in the macros. Several other conditional assembly instructions, similar to those in the FMOVEMMR and FMOVEMRM macro, can be implemented to utilize more addressing modes. To initiate the save sequence, the main processor reads the format word from the save register. This 16-bit register is reread until the high-order byte no longer contains a 01 (coprocessor busy). At this point, the length (in bytes) of the coprocessor data to be transferred resides in the low-order byte of the format word. The main processor isolates this length and begins to transfer the data from the operand register (making long word accesses) to memory via the indirect addressing with predecrement addressing mode. After saving the invisible portion of the coprocessor state, the main processor stores the format word at the top of the stack, in low-order memory. This assures proper restoration of the MC68881 state when the FRESTRST macro is executed. In FRESTRST, the main processor writes the previously saved format word from memory to the restore register, reads the restore register, and begins writing the stored data to the operand register until the proper number of bytes has been transferred. Indirect addressing with postincrement addressing mode is used.

ARCHIVE DOCUMENT NOT FOR NEW DESIGN

In summary, the performance of the MC68881, driven as a peripheral in a MC68000/MC68008/MC68010 system, is enhanced by using the macro approach. This is primarily due to the fact that most of the instruction decode is done at assembly. This in-line code is upwardly source code compatible to a MC68020 system via re-compilation or reassembly. For instance, the following code provides an example of how to alter a macro (for reassembly) in order to acquire floating-point source code compatibility when porting the user software to an MC68020 system (equate table must be deleted):

```
MEMREGB    MACRO
           ˘1.B     ˘2,˘3
           ENDM
```

The macro call will remain the same. For example, this macro call:

```
MEMREGB    FADD,D0,FP0
```

expands to create the following MC68881 floating-point ADD source code when used in conjunction with the previous macro definition:

```
FADD    D0,FP0
```

A few consequences of this technique exist: 1) the object code is not MC68881 replaceable because if the code were moved up to a MC68020/MC68881 system, the MC68881 would still be a peripheral processor in user data space (to benefit from the MC68020 coprocessor interface, the macros would have to be changed and the user program reassembled), 2) a macro library and/or other routines are required to contain the macro software, 3) the full environment is not presented to the user as not all addressing modes nor the FTRAPcc instruction are supported, all checking done by the MC68020 is not implemented (e.g., illegal format errors), and exceptions are not enabled, 4) the MC68881 is not an independent operating hardware device because peripheral I/O access is used, and 5) the demonstration software does not support the M68000 immediate addressing mode.

F-LINE TRAP SOFTWARE EMULATION

As an alternative to using macros or in-line code, an F-line trap emulation could be implemented in an MC68000/MC68008/MC68010 system when the user requires the user program object code containing MC68881 instructions to be upward compatible to the MC68020 without recompiling, reassembling, or relinking. By using this approach, the coprocessor will be driven as a peripheral from supervisory space by supervisor software. Complete source and object code compatibility with the MC68881 instruction set can be maintained.

Because some M68000 systems separate user and supervisor space, different types of emulations must be developed. This application note includes two examples of an F-line emulation of the general instruction operation performed on data moved into the coprocessor: the protected and unprotected versions. The software for other types such as move outs can be inferred from the examples given. The protected version (**APPENDIX B PROTECTED F-LINE EMULATION SOFTWARE**) is used on systems which segregate user and system address spaces. The unprotected version (**APPENDIX C UNPROTECTED F-LINE EMULATION SOFTWARE**) can be used on any M68000 system which allows direct access to user spaces from the supervisor state.

Functional Description

If the coprocessor instruction were decoded by the trap routine to determine the addressing modes used to access the instruction operands, then a significant overhead would be incurred with a commensurate loss of performance. Hence, the demonstration software presumes a single addressing mode will always be used. This is register indirect, (A0). If the programmer desires to use other addressing modes, this can be accomplished by simply performing:

```
LEA    EA,A0
```

before executing the floating-point instruction. Note, the LEA instruction will not work when using PC relative addressing mode in a system that splits program and data spaces (although this is rarely encountered). Also, as implemented in the macro approach, no error or exception checking is performed by the F-line emulator approach.

In the examples, all memory to floating-point register operations are supported including FMOVECR and all FPn to FPm operations.

Theory of Operation

The following paragraphs provide information for users creating their own F-line trap emulations. The read-write protocol of the move-in macros (Figure 11) is implemented in both the protected and unprotected forms of the emulation. The F-line trap emulation differs from the protocol of the macro approach. In the emulation, after the main processor has transferred the data to the coprocessor, the final read of the response register is no longer needed. Sufficient time will expire between any two consecutive floating-point instructions due to the overhead of the F-line trap which ensures that no spurious protocol violations will occur.

A flowchart of the unprotected emulation version is seen in Figure 19. In the protected version, the same sequence of events occurs with the exception that the floating-point instruction and source operands are accessed in user memory from supervisor space utilizing the MC68010 MOVES and MOVEC instructions. Only the unprotected version (**APPENDIX C UNPROTECTED F-LINE EMULATION SOFTWRAE**) is referenced.

When a coprocessor instruction is encountered, the F-line trap is taken. The location of the coprocessor instruction (program counter) and other information (depending on the main processor executing the instruction) is placed on the stack. The data at the program counter location (the operation word of the coprocessor instruction) is examined to determine whether the instruction is a general type (line 25). If so, the second word (the command word) is written to the command register (line 30). Then, the main processor queries the response register until the coprocessor no longer processes the previous instruction (no null come-again).

ARCHIVED DOCUMENT - NOT FOR NEW DESIGN

Next, the op-class specified in the command word is examined to determine the main processor's next action. First, the main processor tests for the move multiple coprocessor registers (data or system) into or out of the MC68881 (op-classes with high-order bits set, e.g. 1xx) in lines 33-34. If found, the main processor would jump to a routine to handle this special function. This function is not implemented in this application but is a straightforward routine.

Subsequently, testing for a floating-point register-to-register operation occurs in line 35. In this case, no further services are needed of the main processor, and a jump to the RTE instruction is taken.

Finally, a distinction between the move-in and move-out operations is made (line 37-38). (An additional routine can be developed to support the move-out sequence.) When a move-in operation has been identified, the main processor then extracts the precision of the external operand from the command word. If the instruction is found to be a FMOVECR (precision 111), the main processor immediately branches to the RTE instruction. Otherwise, the main processor branches to the small routines for handling the respective data transfers. Since long words, packed BCD, single, double, and extended precision data transfers all require at least one 32-bit data transfer, one routine handles all five data types (lines 54-56). Two other routines (lines 59-60 and lines 63-64) support the byte and word transfers. After the data has been delivered to the coprocessor, the main processor returns from exception via the RTE instruction (the instruction which completes the F-line trap and re-enters the user program).

In both versions of the F-line trap emulation, the work registers were stored at the beginning of the routine and then restored prior to the exit.

In summary, the F-line emulation trap is initiated when the main processor identifies a coprocessor instruction by a hexadecimal F in the most-significant nibble of the first word of the instruction (and takes the F-line trap). The object code containing the MC68881 instruction is upward compatible to an MC68020 system without recompiling, reassembling, or relinking.

The major consequence in implementing an F-line emulation instead of in-line code is the time factor incurred by both the overhead of the F-line trap and the instruction decode in the trap routine. Listed in Table 2 are the clock cycles required to perform the various operations using the two F-line emulations and the macro approaches. Even with the overhead associated with the F-line emulation this approach offers a speed advantage over floating-point software packages and at the same time maintains MC68020/MC68881 upward compatibility. Timings are based on a no-wait state system (four clock cycle bus cycle). The MC68881 overhead is not taken into account because the main processor is released to perform the next instruction of the user program while the coprocessor executes its instructions. FMOVE is an example instruction that could be replaced by any of the general instructions (excluding the move multiples).

CONCLUSION

The MC68881 floating-point coprocessor can be utilized as a peripheral in a MC68000/MC68008/MC68010 system by either directly driving the device as a peripheral or by simulating the complete coprocessor instruction set. Either method, depending on the application, is sufficient to utilize the high performance of the MC68881 and offers superior speed and versatility over floating-point software packages. The macros or in-line code of this application provide a faster way to access the device for the users interested in achieving the highest performance of the MC68881. Alternately, for applications that can trade-off performance to achieve object code upward compatibility with MC68020 systems, an example of an F-line emulation trap has also been included.

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

Table 2. Operation Execution Time (Clock Cycle) for MC68010

Operation (MC68881)	Macro	F-Line (Protected)	F-Line (Unprotected)
F(op).B EA,FPn	88	410	342
F(op).W EA,FPn	88	410	342
F(op).L EA,FPn	96	462	398
F(op).S EA,FPn	96	462	398
F(op).D EA,FPn	124	516	436
F(op).X EA,FPn	156	570	474
F(op).P EA,FPn	156	570	474
F(op).X FPm,FPn	40	296	236
FMOVECR #ccc,FPn	40	370	316

(op) = MOVE, ADD, SUB

APPENDIX A
MACROS

```
 1
 2  ********************************************************************************
 3  *        SOURCE CODE TO DRIVE THE MC68881 AS A PERIPHERAL                        *
 4  *                                                                                *
 5  *        TO NOT SHOW MACRO EXPANSION IN THE LIST FILE DELETE 'OPT MEX' BEFORE    *
 6  *        ASSEMBLY.                                                               *
 7  *                                                                                *
 8  *        TO SHOW THE CONDITIONAL ASSEMBLY INSTRUCTIONS IN THE MACRO EXPANSION    *
 9  *        DELETE THE 'OPT NOCL'(LOCATED AFTER THIS BOX) BEFORE ASSEMBLY.          *
10  *                                                                                *
11  ********************************************************************************
12          OPT NOCL
13          OPT MEX
14  ********************************************************************************
15  *                                                                                *
16  *           THIS IS THE EQUATE FILE TO SUPPORT THE MACROS USED                  *
17  *                  TO DRIVE THE MC68881 AS A PERIPHERAL                          *
18  *                      WITH THE M68000 FAMILY                                    *
19  *                                                                                *
20  ********************************************************************************
21
22  ********************************************************************************
23  *       THESE ARE THE INSTRUCTION BIT PATTERN EQUATES
24  ********************************************************************************
25  FMOVE    EQU      $00        MOVE
26  FINT     EQU      $01        INTERGER PART
27  FSINH    EQU      $02        SINH
28  FSQRT    EQU      $04        SQUARE ROOT
29  FLOGNP1  EQU      $06        LOGN (1+X)
30  FETOXM1  EQU      $08        [(E**X)-1)]
31  FTANH    EQU      $09        TANH
32  FATAN    EQU      $0A        ARCTAN
33  FASIN    EQU      $0C        ARCSIN
34  FATANH   EQU      $0D        ARCTANH
35  FSIN     EQU      $0E        SINE
36  FTAN     EQU      $0F        TANGENT
37  FETOX    EQU      $10        E**X
38  FTWOTOX  EQU      $11        2**X
39  FTENTOX  EQU      $12        10**X
40  FLOGN    EQU      $14        LOGN
41  FLOG10   EQU      $15        LOG10
42  FLOG2    EQU      $16        LOG2
43  FABS     EQU      $18        ABSOLUTE VALUE
44  FCOSH    EQU      $19        COSH
45  FNEG     EQU      $1A        NEGATE
46  FACOS    EQU      $1C        ARCCOS
47  FCOS     EQU      $1D        COSINET
48  FGETEXP  EQU      $1E        GET EXPONENT
49  FGETMAN  EQU      $1F        GET MANTISSA
50  FDIV     EQU      $20        DIVIDE
51  FMOD     EQU      $21        MODULO REMAINDER
52  FADD     EQU      $22        ADD
53  FMUL     EQU      $23        MULTIPLY
54  FSGLDIV  EQU      $24        SINGLE DIVIDE
55  FREM     EQU      $25        IEEE REMAINDER
56  FSCALE   EQU      $26        SCALE EXPONENT
57  FSGLMUL  EQU      $27        SINGLE MULTIPLY
58  FSUB     EQU      $28        SUBTRACT
59  FCMP     EQU      $38        COMPARE
60  FTST     EQU      $3A        TEST
61  FSINCOS  EQU      $30        SIMULTANEOUS FP SINE AND COSINE
62
```

ARCHIVED DOCUMENT NOT FOR NEW DESIGN

```
63 ******************************************************************************
 64 *     THESE ARE THE NUEMONICS USED AS THE CONDITION CODES FOR THE            *
 65 *         BRANCH INSTRUCTIONS                                                *
 66 ******************************************************************************
 67 EQ       EQU       $01       EQUAL
 68 NEQ      EQU       $0E       NOT EQUAL
 69 GT       EQU       $12       GREATER THAN
 70 NGT      EQU       $1D       NOT GREATER THAN
 71 GE       EQU       $13       GREATER THAN OR EQUAL
 72 NGE      EQU       $1C       NOT (GREATER THAN OR EQUAL)
 73 LT       EQU       $14       LESS THAN
 74 NLT      EQU       $1B       NOT LES THAN
 75 LE       EQU       $15       LESS THAN OR EQUAL
 76 NLE      EQU       $1A       NOT (LESS THAN OR EQUAL)
 77 GL       EQU       $16       GREATER OR LESS THAN
 78 NGL      EQU       $19       NOT (GREATER OR LESS THAN)
 79 GLE      EQU       $17       GREATER OR LESS OR EQUAL
 80 NGLE     EQU       $18       NOT (GREATER OR LESS OR EQUAL)
 81 OGT      EQU       $02       ORDERED GREATER THAN
 82 ULE      EQU       $0D       UNORDERED OR LESS OR EQUAL
 83 OGE      EQU       $03       ORDERED GREATER THAN OR EQUAL
 84 ULT      EQU       $0C       UNORDERED OR LESS THAN
 85 OLT      EQU       $04       ORDERED LESS THAN
 86 UGE      EQU       $0B       UNORDERED OR GREATER OR EQUAL
 87 OLE      EQU       $05       ORDERED LESS THAN OR EQUAL
 88 UGT      EQU       $0A       UNORDERED OR GREATER
 89 OGL      EQU       $06       ORDERED GREATER OR LESS THAN
 90 UEQ      EQU       $09       UNORDERED OR EQUAL
 91 OR       EQU       $07       ORDERED
 92 UN       EQU       $08       UNORDERED
 93 F        EQU       $00       FALSE (NEVER)
 94 T        EQU       $0F       TRUE (ALWAYS)
 95 SF       EQU       $10       SIGNALING FALSE (NEVER)
 96 ST       EQU       $1F       SIGNALING TRUE (ALWAYS)
 97 SEQ      EQU       $11       SIGNALING EQUAL
 98 SNEQ     EQU       $1E       SIGNALING NOT EQUAL
 99
100 ******************************************************************************
101 *         THESE EQUATES REPRESENT THE OFFSETS FOR THE BASE ADDRESS OF        *
102 *           THE MC68881 INTERFACE REGISTERS!                                 *
103 ******************************************************************************
104 COMMAND  EQU       $0A       COMMAND REGISTER
105 RESPONSE EQU       $00       RESPONSE REGISTER
106 OPER     EQU       $10       OPERAND REGISTER
107 COND     EQU       $0E       CONDITION REGISTER
108 SAVE     EQU       $04       SAVE REGISTER
109 RESTORE  EQU       $06       RESTORE REGISTER
110 REGSEL   EQU       $14       REGISTER SELECT
111 CONTROL  EQU       $9000     MC68881 CONTROL REGISTER
112 STATUS   EQU       $8800     MC68881 STATUS REGISTER
113 IADDRESS EQU       $8400     MC68881 INSTRUCTION ADDRESS REGISTER
114 TFBIT    EQU       $0        TRUE/FALSE BIT OF THE RESPONSE REGISTER
115
116 ******************************************************************************
117 *         THESE EQUATES REPRESENT THE FLOATING POINT REGISTERS               *
118 ******************************************************************************
119 FP0      EQU       $00       FLOATING POINT REGISTER #0
120 FP1      EQU       $01          "       "        "   #1
121 FP2      EQU       $02          "       "        "   #2
122 FP3      EQU       $03          "       "        "   #3
123 FP4      EQU       $04       FLOATING POINT REGISTER #4
124 FP5      EQU       $05          "       "        "   #5
125 FP6      EQU       $06          "       "        "   #6
126 FP7      EQU       $07          "       "        "   #7
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
127 ****************************************************************************
128  *                                                                          *
129  *        MC68881 SINGLE PRECISION FP-REG. VALUE TO MEMORY OPERATION        *
130  *                                                                          *
131  *        REGMEMS   INSTRUCTION,FPM,<EA>                                    *
132  *                                                                          *
133  *        WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FMOVE)         *
134  *                FPM= SOURCE FP REGISTER                                   *
135  *                <EA>= DESTINATION ADDRESSING MODE                         *
136  *                                                                          *
137  *        NO REGISTERS MODIFYED OR DESTROYED!                               *
138  *                                                                          *
139  *        VALID ADDRESSING MODES:  DN, (AN)+, -(AN), D(AN), D(AN,IX)        *
140  *                                 XXX.W, XXX.L, (D,PC), D(PC,IX)           *
141  *                                                                          *
142 ****************************************************************************
143 REGMEMS  MACRO
144          IFC '\4',''                         IS <EA>= INDIRECT WITH INDEXING
145          MOVE.W #$6400+(\2<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
146 \@NULCA  CMPI #$8900,MC68881+RESPONSE       READ RESPONSE REGISTER
147          BEQ.S    \@NULCA                   REREAD UNTIL EVALUATE EA AND TRANSFER
148 *                                           DATA
149          MOVE.L MC68881+OPER,\3             LOW ORDER WORD
150 \@NULREL TST.B MC68881+RESPONSE             IS RESPONSE NULL RELEASE?
151          BMI.S \@NULREL                     BRANCH UNTIL NULL RELEASE
152          ENDC
153          IFNC '\4',''                       IS <EA> NOT = INDIRECT WITH INDEXING
154          MOVE.W #$6400+(\2<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
155 \@NULCA  CMPI #$8900,MC68881+RESPONSE       READ RESPONSE REGISTER
156          BEQ.S    \@NULCA                   REREAD UNTIL EVALUATE EA AND TRANSFER
157 *                                           DATA
158          MOVE.L MC68881+OPER,\3,\4          SINGLE PRECISION DATA TRANSFER
159 \@NULREL TST.B MC68881+RESPONSE             IS RESPONSE NULL RELEASE?
160          BMI.S \@NULREL                     BRANCH UNTIL NULL RELEASE
161          ENDC
162          ENDM
163 ****************************************************************************
164 *                                                                          *
165 *        MC68881 LONG WORD LENGTH FP-REG. VALUE TO MEMORY OPERATION        *
166 *                                                                          *
167 *        REGMEMW   INSTRUCTION,FPM,<EA>                                    *
168 *                                                                          *
169 *   INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FMOVE)                      *
170 *                FPM= SOURCE FP REGISTER                                   *
171 *                <EA>= DESTINATION ADDRESSING MODE                         *
172 *                                                                          *
173 *        NO REGISTERS MODIFYED OR DESTROYED!                               *
174 *                                                                          *
175 *        VALID ADDRESSING MODES:  DN, (AN)+, -(AN), D(AN), D(AN,IX)        *
176 *                                 XXX.W, XXX.L, (D,PC), D(PC,IX)           *
177 *                                                                          *
178 ****************************************************************************
179 REGMEML  MACRO
180          IFC '\4',''                  IS <EA>=INDIRECT WITH INDEXING
181          MOVE.W #$6000+(\2<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
182 \@NULCA  CMPI #$8900,MC68881+RESPONSE  READ RESPONSE REGISTER
183          BEQ.S    \@NULCA              REREAD UNTIL EVALUATE EA AND TRANSFER
184 *                                      DATA
185          MOVE.L MC68881+OPER,\3        LONG WORD TRANSFER
186 \@NULREL TST.B MC68881+RESPONSE        IS RESPONSE NULL RELEASE?
187          BMI.S \@NULREL                BRANCH UNTIL NULL RELEASE
188          ENDC
189          IFNC '\4',''                  IS <EA> NOT = INDIRECT WITH INDEXING
190          MOVE.W #$6000+(\2<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
191 \@NULCA  CMPI #$8900,MC68881+RESPONSE  READ RESPONSE REGISTER
192          BEQ.S    \@NULCA              REREAD UNTIL EVALUATE EA AND TRANSFER
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
193  *                                        DATA
194            MOVE.L MC68881+OPER,\3,\4      LONG WORD TRANSFER
195  \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
196            BMI.S \@NULREL                 BRANCH UNTIL NULL RELEASE
197            ENDC
198            ENDM
199  ***************************************************************************
200  *                                                                         *
201  *         MC68881 WORD LENGTH FP-REG. VALUE TO MEMORY OPERATION           *
202  *                                                                         *
203  *         REGMEMW   INSTRUCTION,FPM,<EA>                                  *
204  *                                                                         *
205  *         WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FMOVE)       *
206  *                 FPM= SOURCE FP REGISTER                                 *
207  *                 <EA>= DESTINATION ADDRESSING MODE                       *
208  *                                                                         *
209  *         NO REGISTERS MODIFYED OR DESTROYED!                             *
210  *                                                                         *
211  *         VALID ADDRESSING MODES:  DN, (AN)+, -(AN), D(AN), D(AN,IX)      *
212  *                                  XXX.W, XXX.L, (D,PC), D(PC,IX)         *
213  *                                                                         *
214  ***************************************************************************
215  REGMEMW MACRO
216           IFC '\4',''                       IS <EA>=INDIRECT WITH INDEXING
217           MOVE.W #$7000+(\2<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
218  \@NULCA  CMPI #$8900,MC68881+RESPONSE      READ RESPONSE REGISTER
219           BEQ.S    \@NULCA                  REREAD UNTIL EVALUATE EA AND TRANSFER
220  *                                          DATA
221           MOVE.W MC68881+OPER,\3            WORD DATA TRANSFER
222  \@NULREL TST.B MC68881+RESPONSE            IS RESPONSE NULL RELEASE?
223           BMI.S \@NULREL                    BRANCH UNTIL NULL RELEASE
224           ENDC
225           IFNC '\4',''                      IS <EA> NOT = INDIRECT WITH INDEXING
226           MOVE.W #$7000+(\2<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
227  \@NULCA  CMPI #$8900,MC68881+RESPONSE      READ RESPONSE REGISTER
228           BEQ.S    \@NULCA                  REREAD UNTIL EVALUATE EA AND TRANSFER
229  *                                         DATA
230           MOVE.W MC68881+OPER,\3,\4        WORD DATA TRANSFER
231  \@NULREL TST.B MC68881+RESPONSE           IS RESPONSE NULL RELEASE?
232           BMI.S \@NULREL                   BRANCH UNTIL NULL RELEASE
233           ENDC
234  *
235  *
236  *
237           ENDM
238  ***************************************************************************
239  *                                                                         *
240  *        MC68881 BYTE LENGTH FP-REG. VALUE TO MEMORY OPERATION            *
241  *                                                                         *
242  *        REGMEMB   INSTRUCTION,FPM,<EA>                                   *
243  *                                                                         *
244  *        WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FMOVE)        *
245  *                FPM= SOURCE FP REGISTER                                  *
246  *                <EA>= DESTINATION ADDRESSING MODE                        *
247  *                                                                         *
248  *        NO REGISTERS MODIFYED OR DESTROYED!                              *
249  *                                                                         *
250  *        VALID ADDRESSING MODES:  DN, (AN)+, -(AN), D(AN), D(AN,IX)       *
251  *                                 XXX.W, XXX.L, (D,PC), D(PC,IX)          *
252  *                                                                         *
253  ***************************************************************************
254  REGMEMB  MACRO
255           IFC '\4',''                      IS <EA>=INDIRECT WITH INDEXING
256           MOVE.W #$7800+(\2<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
257  \@NULCA  CMPI #$8900,MC68881+RESPONSE     READ RESPONSE REGISTER
258           BEQ.S    \@NULCA                 REREAD UNTIL EVALUATE EA AND TRANSFER
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
259  *                                            DATA
260           MOVE.B MC68881+OPER,\3              BYTE DATA TRANSFER
261  \@NULREL TST.B MC68881+RESPONSE              IS RESPONSE NULL RELEASE?
262           BMI.S \@NULREL                      BRANCH UNTIL NULL RELEASE
263           ENDC
264           IFNC '\4',''                        IS <EA> NOT = INDIRECT WITH INDEXING
265           MOVE.W #$7800+(\2<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
266  \@NULCA  CMPI #$8900,MC68881+RESPONSE        READ RESPONSE REGISTER
267           BEQ.S    \@NULCA                    REREAD UNTIL EVALUATE EA AND TRANSFER
268  *                                            DATA
269           MOVE.B MC68881+OPER,\3,\4           BYTE DATA TRANSFER
270  \@NULREL TST.B MC68881+RESPONSE              IS RESPONSE NULL RELEASE?
271           BMI.S \@NULREL                      BRANCH UNTIL NULL RELEASE
272           ENDC
273           ENDM
274  ************************************************************************
275  *                                                                      *
276  *        MC68881 DOUBLE PRECISION FP-REG. VALUE TO MEMORY OPERATION    *
277  *                                                                      *
278  *        REGMEMD   INSTRUCTION,FPM,<EA>                                *
279  *                                                                      *
280  *        WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FMOVE)     *
281  *                FPM= SOURCE FP REGISTER                               *
282  *                <EA>= AN ADDRESS REGISTER, SURROUNDED BY PARENTHESIS, *
283  *                      CONTAINING THE PREVIOUSLY LOADED EFFECTIVE ADDRESS *
284  *                      (I.E. (A0)).                                    *
285  *                                                                      *
286  *        NO REGISTERS MODIFYED OR DESTROYED!                           *
287  *                                                                      *
288  *        VALID ADDRESSING MODES:  (AN)                                 *
289  *                                                                      *
290  ************************************************************************
291  REGMEMD  MACRO
292           MOVE.W #$7400+(\2<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
293  \@NULCA  CMPI #$8900,MC68881+RESPONSE        READ RESPONSE REGISTER
294           BEQ.S    \@NULCA                    REREAD UNTIL EVALUATE EA AND TRANSFER
295  *                                            DATA
296           MOVE.L MC68881+OPER,\3              HIGH ORDER LONG WORD
297           MOVE.L MC68881+OPER,4\3             LOW ORDER LONG WORD
298  \@NULREL TST.B MC68881+RESPONSE              IS RESPONSE NULL RELEASE?
299           BMI.S \@NULREL                      BRANCH UNTIL NULL RELEASE
300           ENDM
301  ************************************************************************
302  *                                                                      *
303  *        MC68881 EXTENDED PRECISION FP-REG. VALUE TO MEMORY OPERATION  *
304  *                                                                      *
305  *        REGMEMX   INSTRUCTION,FPM,<EA>                                *
306  *                                                                      *
307  *        WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FMOVE)     *
308  *                FPM= SOURCE FP REGISTER                               *
309  *                <EA>= AN ADDRESS REGISTER, SURROUNDED BY PARENTHESIS, *
310  *                      CONTAINING THE PREVIOUSLY LOADED EFFECTIVE ADDRESS *
311  *                      (I.E. (A0)).                                    *
312  *                                                                      *
313  *        NO REGISTERS MODIFYED OR DESTROYED!                           *
314  *                                                                      *
315  *        VALID ADDRESSING MODES:  (AN)                                 *
316  *                                                                      *
317  ************************************************************************
318  REGMEMX  MACRO
319           MOVE.W #$6800+(\2<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
320  \@NULCA  CMPI #$8900,MC68881+RESPONSE        READ RESPONSE REGISTER
321           BEQ.S    \@NULCA                    REREAD UNTIL EVALUATE EA AND TRANSFER
322  *                                            DATA
323           MOVE.L MC68881+OPER,\3              HIGH ORDER LONG WORD
324           MOVE.L MC68881+OPER,4\3             MID-ORDER
```

ARCHIVE DOCUMENT, NOT FOR NEW DESIGN

```
325          MOVE.L MC68881+OPER,8\3         LOW ORDER WORD
326  \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
327          BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
328          ENDM
329  ***********************************************************************
330  *                                                                     *
331  *       MC68881 PACKED BCD FP-REG. VALUE TO MEMORY OPERATION          *
332  *                                                                     *
333  *       REGMEMP   INSTRUCTION,FPM,<EA>,[K-FACTOR]                     *
334  *                                                                     *
335  *       WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FADD)      *
336  *               FPM= SOURCE FP REGISTER                               *
337  *               <EA>= DESTINATION ADDRESSING MODE                     *
338  *               [K-FACTOR]= OPTIONAL IMMEDIATE K-FACTOR               *
339  *                                                                     *
340  *    ***IF [K-FACTOR] OPTION NOT TAKEN, THE K-FACTOR MUST BE PLACED IN D0!  *
341  *                                                                     *
342  *       VALID ADDRESSING MODES:   (AN)                                *
343  *                                                                     *
344  ***********************************************************************
345  REGMEMP  MACRO
346          IFC '\4',''                     IS K-FACTOR IN REGISTER?
347          MOVE.W #$7C00+(\2<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
348  \@NULCA  CMPI #$8900,MC68881+RESPONSE    READ RESPONSE REGISTER
349          BEQ.S    \@NULCA                REREAD UNTIL TRANSFER MAIN PROCESSOR REG
350          MOVE.L D0,MC68881+OPER          PASS K-FACTOR FROM D0
351  \@AGAIN  CMPI #$8900,MC68881+RESPONSE    READ RESPONSE REGISTER
352          BEQ.S    \@AGAIN                REREAD UNTIL EVALUATE EFFECTIVE ADDRESS
353  *                                       AND TRANSFER DATA
354          MOVE.L MC68881+OPER,\3          LOW ORDER LONG WORD
355          MOVE.L MC68881+OPER,4\3         MID-ORDER LONG WORD
356          MOVE.L MC68881+OPER,8\3         HIGH ORDER LONG WORD
357  \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
358          BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
359          ENDC
360          IFNC '\4',''                    IS K-FACTOR IN INSTRUCTION?
361          MOVE.W #$6C00+(\2<<7)+\4,MC68881+COMMAND  MEM. TO REG. OPERATION
362  \@NULCA  CMPI #$8900,MC68881+RESPONSE    READ RESPONSE REGISTER
363          BEQ.S    \@NULCA                REREAD UNTIL EVALUATE EFFECTIVE ADDRESS
364  *                                       AND TRANSFER DATA
365          MOVE.L MC68881+OPER,\3          LOW ORDER WORD
366          MOVE.L MC68881+OPER,4\3         MID-ORDER WORD
367          MOVE.L MC68881+OPER,8\3         HIGH ORDER WORD
368  \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
369          BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
370          ENDC
371          ENDM
372  ***********************************************************************
373  *                                                                     *
374  *       MC68881 BYTE IN MEMORY OR IN Dn TO FP-REG. OPERATION          *
375  *                                                                     *
376  *       MEMREGB   INSTRUCTION,<EA>,FPN                                *
377  *                                                                     *
378  *       WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FADD)      *
379  *               <EA>= SOURCE ADDRESSING MODE                          *
380  *               FPN= DESTINATION REGISTER                             *
381  *                                                                     *
382  *       NO REGISTERS MODIFYED OR DESTROYED!                           *
383  *                                                                     *
384  *       VALID ADDRESSING MODES:  DN, (AN)+, -(AN), D(AN), D(AN,IX)    *
385  *                                XXX.W, XXX.L, (D,PC), D(PC,IX)       *
386  *                                                                     *
387  *       THE COMMENTED OUT CODE SHOWS HOW A USER MAY IMPLEMENT FSINCOS  *
388  *       IN A MEM. TO REG. TRANSFER USING THE FOLLOWING INSTRUCTION FORMAT:  *
389  *                                                                     *
390  *       MEMREGB INSTRUCTION,<EA>,FPN,FPQ        (FPQ= 2ND DESTINATION REG.)  *
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
391  *                                                                              *
392  ********************************************************************************
393  MEMREGB  MACRO
394           IFC '\1','FSINCOS'               IS INSTRUCTION FSINCOS
395           IFC '\5',''                      IS INDEXING PART OF THE ADDR.MODE
396           MOVE.W #$5800+(\4<<7)+\3+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
397   \@NULCA CMPI #$8900,MC68881+RESPONSE     READ RESPONSE REGISTER
398           BEQ.S    \@NULCA                 REREAD UNTIL EVALUATE EA AND TRANSFER
399                                            DATA
400           MOVE.B \2,MC68881+OPER           BYTE DATA TRANSFER
401   \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
402           BMI.S \@NULREL                   BRANCH UNTIL NULL RELEASE
403           ENDC
404           IFNC '\5',''                     IS <EA> NOT = INDIRECT WITH INDEXING
405           MOVE.W #$5800+(\5<<7)+\4+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
406   \@NULCA CMPI #$8900,MC68881+RESPONSE     READ RESPONSE REGISTER
407           BEQ.S    \@NULCA                 REREAD UNTIL EVALUATE EA AND TRANSFER
408                                            DATA
409           MOVE.B \2,\3,MC68881+OPER        BYTE DATA TRANSFER
410   \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
411           BMI.S \@NULREL                   BRANCH UNTIL NULL RELEASE
412           ENDC
413           ENDC
414           IFNC '\1','FSINCOS'              IS INSTRUCTION NOT FSINCOS
415           IFC '\4',''                      IS <EA>=INDIRECT WITH INDEXING
416           MOVE.W #$5800+(\3<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
417  \@NULCA  CMPI #$8900,MC68881+RESPONSE     READ RESPONSE REGISTER
418           BEQ.S \@NULCA                    REREAD UNTIL EVALUATE EA AND TRANSFER
419  *                                         DATA
420           MOVE.B \2,MC68881+OPER           MOVE DATA INTO OPERAND REGISTER
421  \@NULREL TST.B MC68881+RESPONSE           IS RESPONSE NULL RELEASE?
422           BMI.S \@NULREL                   BRANCH UNTIL NULL RELEASE
423           ENDC
424           IFNC '\4',''                     IS <EA>=NOT INDIRECT WITH INDEXING
425           MOVE.W #$5800+(\4<<7)+\1,MC68881+COMMAND *MEM. TO REG. OPERATION
426  \@NULCA  CMPI #$8900,MC68881+RESPONSE     READ RESPONSE REGISTER
427           BEQ.S \@NULCA                    REREAD UNTIL EVALUATE EA AND TRANSFER
428  *                                         DATA
429           MOVE.B \2,\3,MC68881+OPER        BYTE DATA TRANSFER
430  \@NULREL TST.B MC68881+RESPONSE           IS RESPONSE NULL RELEASE?
431           BMI.S \@NULREL                   BRANCH UNTIL NULL RELEASE
432           ENDC
433           ENDC
434           ENDM
435  ********************************************************************************
436  *                                                                              *
437  *        MC68881 WORD IN MEMORY OR IN Dn TO FP-REG. OPERATION                  *
438  *                                                                              *
439  *        MEMREGW   INSTRUCTION,<EA>,FPN                                        *
440  *                                                                              *
441  *        WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FADD)              *
442  *                <EA>= SOURCE ADDRESSING MODE                                  *
443  *                FPN= DESTINATION REGISTER                                     *
444  *                                                                              *
445  *        NO REGISTERS MODIFYED OR DESTROYED!                                   *
446  *                                                                              *
447  *        VALID ADDRESSING MODES:  DN, (AN)+, -(AN), D(AN), D(AN,IX)            *
448  *                                 XXX.W, XXX.L, (D,PC), D(PC,IX)               *
449  *                                                                              *
450  ********************************************************************************
451  MEMREGW  MACRO
452           IFC '\4',''                      IS <EA>=INDIRECT WITH INDEXING
453           MOVE.W #$5000+(\3<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
454  \@NULCA  CMPI #$8900,MC68881+RESPONSE     READ RESPONSE REGISTER
455           BEQ.S    \@NULCA                 REREAD UNTIL EVALUATE EA AND TRANSFER
456  *                                         DATA
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
457          MOVE.W \2,MC68881+OPER          WORD DATA TO FP-REG.
458 \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
459          BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
460          ENDC
461          IFNC '\4',''                    IS <EA> NOT = INDIRECT WITH INDEXING
462          MOVE.W #$5000+(\4<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
463 \@NULCA  CMPI #$8900,MC68881+RESPONSE    READ RESPONSE REGISTER
464          BEQ.S    \@NULCA                REREAD UNTIL EVALUATE EA AND TRANSFER
465 *                                        DATA
466          MOVE.W \2,\3,MC68881+OPER       WORD DATA TO FP REG.
467 \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
468          BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
469          ENDC
470          ENDM
471 ****************************************************************************
472 *                                                                          *
473 *        MC68881 LONG WORD IN MEMORY OR IN Dn TO FP-REG. OPERATION         *
474 *                                                                          *
475 *        MEMREGL   INSTRUCTION,<EA>,FPN                                    *
476 *                                                                          *
477 *        WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FADD)          *
478 *                <EA>= SOURCE ADDRESSING MODE                              *
479 *                FPN= DESTINATION REGISTER                                 *
480 *                                                                          *
481 *        NO REGISTERS MODIFYED OR DESTROYED!                               *
482 *                                                                          *
483 *        VALID ADDRESSING MODES:  DN, (AN)+, -(AN), D(AN), D(AN,IX)        *
484 *                                 XXX.W, XXX.L, (D,PC), D(PC,IX)           *
485 *                                                                          *
486 ****************************************************************************
487 MEMREGL  MACRO
488          IFC '\4',''                     IS <EA>=INDIRECT WITH INDEXING
489          MOVE.W #$4000+(\3<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
490 \@NULCA  CMPI #$8900,MC68881+RESPONSE    READ RESPONSE REGISTER
491          BEQ.S    \@NULCA                REREAD UNTIL EVALUATE EA AND TRANSFER
492 *                                        DATA
493          MOVE.L \2,MC68881+OPER          LONG WORD DATA TO FP REG.
494 \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
495          BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
496          ENDC
497          IFNC '\4',''                    IS <EA> NOT = INDIRECT WITH INDEXING
498          MOVE.W #$4000+(\4<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
499 \@NULCA  CMPI #$8900,MC68881+RESPONSE    READ RESPONSE REGISTER
500          BEQ.S \@NULCA                   REREAD UNTIL EVALUATE EA AND TRANSFER
501 *                                        DATA
502          MOVE.L \2,\3,MC68881+OPER       LONG WORD DATA TO FP REG.
503 \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
504          BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
505          ENDC
506          ENDM
507 ****************************************************************************
508 *                                                                          *
509 *        MC68881 SINGLE PRECISION VALUE MEMORY TO FP-REG. OPERATION        *
510 *                                                                          *
511 *        MEMREGS   INSTRUCTION,<EA>,FPN                                    *
512 *                                                                          *
513 *        WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FADD)          *
514 *                <EA>= SOURCE ADDRESSING MODE                              *
515 *                FPN= DESTINATION REGISTER                                 *
516 *                                                                          *
517 *        NO REGISTERS MODIFYED OR DESTROYED!                               *
518 *                                                                          *
519 *        VALID ADDRESSING MODES:  DN, (AN)+, -(AN), D(AN), D(AN,IX)        *
520 *                                 XXX.W, XXX.L, (D,PC), D(PC,IX)           *
521 *                                                                          *
522 ****************************************************************************
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
523 MEMREGS  MACRO
524          IFC '\4',''                          IS <EA>=INDIRECT WITH INDEXING
525          MOVE.W #$4400+(\3<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
526 \@NULCA  CMPI #$8900,MC68881+RESPONSE   READ RESPONSE REGISTER
527          BEQ.S \@NULCA                   REREAD UNTIL EVALUATE EA AND TRANSFER
528 *                                        DATA
529          MOVE.L \2,MC68881+OPER          SINGLE PRECISION DATA TO FP REG.
530 \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
531          BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
532          ENDC
533          IFNC '\4',''                    IS <EA> NOT = INDIRECT WITH INDEXING
534          MOVE.W #$4400+(\4<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
535 \@NULCA  CMPI #$8900,MC68881+RESPONSE   READ RESPONSE REGISTER
536          BEQ.S    \@NULCA                REREAD UNTIL EVALUATE EA AND TRANSFER
537 *                                        DATA
538          MOVE.L \2,\3,MC68881+OPER       SINGLE PRECISION DATA TO FP REG.
539 \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
540          BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
541          ENDC
542          ENDM
543 ******************************************************************************
544 *                                                                            *
545 *        MC68881 DOUBLE PRECISION VALUE MEMORY TO FP-REG. OPERATION          *
546 *                                                                            *
547 *        MEMREGD   INSTRUCTION,<EA>,FPN                                      *
548 *                                                                            *
549 *        WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FADD)            *
550 *                <EA>= SOURCE ADDRESS REGISTER,SURROUNDED BY PARENTHEIS,     *
551 *                      CONTAINING THE PREVIOUSLY ENTERED ADDRESSING MODE     *
552 *                      (I.E. (AN)).                                          *
553 *                FPN= DESTINATION REGISTER                                   *
554 *                                                                            *
555 *        NO REGISTERS MODIFYED OR DESTROYED!                                 *
556 *                                                                            *
557 *        VALID ADDRESSING MODES:   (AN)                                      *
558 *                                                                            *
559 ******************************************************************************
560 MEMREGD  MACRO
561          MOVE.W #$5400+(\3<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
562 \@NULCA  CMPI #$8900,MC68881+RESPONSE   READ RESPONSE REGISTER
563          BEQ.S    \@NULCA                REREAD UNTIL EVALUATE EA AND TRANSFER
564 *                                        DATA
565          MOVE.L \2,MC68881+OPER          HIGH ORDER LONG WORD
566          MOVE.L 4\2,MC68881+OPER         LOW ORDER LONG WORD
567 \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
568          BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
569          ENDM
570 ******************************************************************************
571 *                                                                            *
572 *        MC68881 EXTENDED PRECISION VALUE MEMORY TO FP-REG. OPERATION        *
573 *                                                                            *
574 *        MEMREGX   INSTRUCTION,<EA>,FPN                                      *
575 *                                                                            *
576 *        WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FADD)            *
577 *                <EA>= SOURCE ADDRESS REGISTER,SURROUNDED BY PARENTHESIS,    *
578 *                      CONTAINING THE PREVIOUSLY ENTERED ADDRESSING MODE     *
579 *                      (I.E. (AN)).                                          *
580 *                FPN= DESTINATION REGISTER                                   *
581 *                                                                            *
582 *        NO REGISTERS MODIFYED OR DESTROYED!                                 *
583 *                                                                            *
584 *        VALID ADDRESSING MODES:   (AN)                                      *
585 *                                                                            *
586 ******************************************************************************
587 MEMREGX  MACRO
588          MOVE.W #$4800+(\3<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
589  \@NULCA  CMPI #$8900,MC68881+RESPONSE   READ RESPONSE REGISTER
590           BEQ.S     \@NULCA              REREAD UNTIL EVALUATE EA AND TRANSFER
591  *                                       DATA
592           MOVE.L \2,MC68881+OPER         HIGH ORDER LONG WORD
593
594           MOVE.L 4\2,MC68881+OPER        MID-ORDER LONG WORD
595           MOVE.L 8\2,MC68881+OPER        LOW ORDER LONG WORD
596  \@NULREL TST.B MC68881+RESPONSE         IS RESPONSE NULL RELEASE?
597           BMI.S \@NULREL                 BRANCH UNTIL NULL RELEASE
598           ENDM
599  ***************************************************************************
600  *                                                                         *
601  *        MC68881 PACKED BCD VALUE MEMORY TO FP-REG. OPERATION             *
602  *                                                                         *
603  *        MEMREGP   INSTRUCTION,<EA>,FPN                                   *
604  *                                                                         *
605  *        WHERE:  INSTRUCTION= FP INSTRUCTION NUEMONIC (I.E. FADD)         *
606  *                <EA>= SOURCE ADDRESS REGISTER,SURROUNDED BY PARENTHESIS, *
607  *                      CONTAINING THE PREVIOUSLY ENTERED ADDRESSING MODE  *
608  *                      (I.E. (AN)).                                       *
609  *                FPN = DESTINATION REGISTER                               *
610  *                                                                         *
611  *        NO REGISTERS MODIFYED OR DESTROYED!                              *
612  *                                                                         *
613  *        VALID ADDRESSING MODES:  (AN)                                    *
614  *                                                                         *
615  ***************************************************************************
616  MEMREGP  MACRO
617           MOVE.W #$4C00+(\3<<7)+\1,MC68881+COMMAND  MEM. TO REG. OPERATION
618  \@NULCA  CMPI #$8900,MC68881+RESPONSE   READ RESPONSE REGISTER
619           BEQ.S     \@NULCA              REREAD UNTIL EVALUATE EA AND TRANSFER
620  *                                       DATA
621           MOVE.L \2,MC68881+OPER         HIGH ORDER LONG WORD
622           MOVE.L 4\2,MC68881+OPER        MID-ORDER LONG WORD
623           MOVE.L 8\2,MC68881+OPER        LOW ORDER LONG WORD
624  \@NULREL TST.B MC68881+RESPONSE         IS RESPONSE NULL RELEASE?
625           BMI.S \@NULREL                 BRANCH UNTIL NULL RELEASE
626           ENDM
627  ***************************************************************************
628  *                                                                         *
629  *        MC68881 FP-REG. TO FP-REG. OPERATION                             *
630  *                                                                         *
631  *        REGREG   INSTRUCTION,FPM,FPN,FNQ                                 *
632  *                                                                         *
633  *        WHERE:  INSTRUCTION= NUEMONIC FOR THE FP INSTRUCTION (I.E. FADD) *
634  *                FPM= FP SOURCE REGISTER                                  *
635  *                FPN= FP DESTINATION REGISTER                             *
636  *                FNQ= SECOND FP DESTINATION REGISTER FOR FSINCOS          *
637  *                                                                         *
638  *        NO REGISTERS MODIFIED OR DESTROYED!                              *
639  *                                                                         *
640  ***************************************************************************
641  REGREG   MACRO
642           IFC '\1','FSINCOS'               IF INSTR. IS FSINCOS DO THIS ROUTINE
643           MOVE.W #(\2<<10)+(\4<<7)+\3+\1,MC68881+COMMAND  REG. TO REG. FSINCOS
644  \@NULCA  TST.B MC68881+RESPONSE           READ RESPONSE REGISTER
645           BMI.S     \@NULCA                REREAD UNTIL NULL RELEASE (CA=0)
646           ENDC
647           IFNC '\1','FSINCOS'              ROUTINE FOR ALL OTHER ARITHMETIC INSTRS.
648           MOVE.W #(\2<<10)+(\3<<7)+\1,MC68881+COMMAND   REG. TO REG. OPERATION
649  \@NULCA  TST.B MC68881+RESPONSE           READ RESPONSE REGISTER
650           BMI.S     \@NULCA                REREAD UNTIL NULL RELEASE (CA=0)
651           ENDC
652           ENDM
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
653  ************************************************************************
654  *                                                                      *
655  *        MC68881 CONSTANT IN ROM TO FP-REG. OPERATION                  *
656  *                                                                      *
657  *        FMOVEROM  CC,FPN                                              *
658  *                                                                      *
659  *        WHERE:     CC = MC68881 CONSTANT                              *
660  *                  $00       PI                                        *
661  *                  $0B       LOG10(2)                                  *
662  *                  $0C       E                                         *
663  *                  $0D       LOG2(E)                                   *
664  *                  $0E       LOG10(E)                                  *
665  *                  $0F       0.0                                       *
666  *                  $30       LOGN(2)                                   *
667  *                  $31       LOGN(10)                                  *
668  *                  $32       10^0                                      *
669  *                  $33       10^1                                      *
670  *                  $34       10^2                                      *
671  *                  $35       10^4                                      *
672  *                  $36       10^8                                      *
673  *                  $37       10^16                                     *
674  *                  $38       10^32                                     *
675  *                  $39       10^64                                     *
676  *                  $3A       10^128                                    *
677  *                  $3B       10^256                                    *
678  *                  $3C       10^512                                    *
679  *                  $3D       10^1024                                   *
680  *                  $3E       10^2048                                   *
681  *                  $3F       10^4096                                   *
682  *               FPN= FP DESTINATION REGISTER                           *
683  *                                                                      *
684  *        NO REGISTERS MODIFIED OR DESTROYED!                           *
685  *                                                                      *
686  ************************************************************************
687  FMOVEROM MACRO
688           MOVE.W #$5C00+(\2<<7)+\1,MC68881+COMMAND    REG. TO REG. OPERATION
689  \@NULCA  TST.B MC68881+RESPONSE              READ RESPONSE REGISTER
690           BMI.S     \@NULCA                 REREAD UNTIL NULL RELEASE (CA=0)
691           ENDM
692  ************************************************************************
693  *                                                                      *
694  *        MC68881 CONDITIONAL BRANCH                                    *
695  *                                                                      *
696  *        FBCC.<SIZE> CONDITION,ADDRESS                                 *
697  *                                                                      *
698  *        WHERE:   <SIZE>= ALLOWABLE BRANCH SIZES                       *
699  *                 CONDITION= CC, THE FLOATING POINT CONDITION (I.E. GT) *
700  *                 ADDRESS= BRANCH ADDRESS                              *
701  *                                                                      *
702  *                                                                      *
703  *        REGISTERS MODIFIED OR DESTROYED: 0 1 2 3 4 5 6 7              *
704  *                                         D X                          *
705  *                                         A                            *
706  *                                                                      *
707  ************************************************************************
708  FBCC     MACRO
709           MOVE.W #\1,MC68881+COND         BEGIN COPROCESSOR COMMUNICATION
710  \@NOPASS MOVE.W MC68881+RESPONSE,D0      IS CA-BIT SET
711           BMI.S    \@NOPASS               REREAD UNTIL NULL RELEASE (CA=0)
712           BTST #TFBIT,D0                  IS CONDITION TRUE
713           BNE.\0 \2                       BRANCH IF CONDITION TRUE!
714           ENDM
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
715 ***************************************************************************
716 *                                                                         *
717 *        MC68881 TEST FP CONDITION, DECREMENT, AND BRANCH                 *
718 *                                                                         *
719 *        FDBCC   CONDITION,DN,ADDRESS                                     *
720 *                                                                         *
721 *        WHERE:  CONDITION= CC, FLOATING POINT CONDITION                  *
722 *                DN= MAIN PROCESSOR DATA REGISTER TO BE DECREMENTED       *
723 *                ADDRESS= BRANCH ADDRESS                                  *
724 *                                                                         *
725 *        REGISTERS MODIFIED OR DESTROYED: 0 1 2 3 4 5 6 7                 *
726 *                                         D X                             *
727 *                                         A                               *
728 *                                                                         *
729 ***************************************************************************
730 FDBCC    MACRO
731          MOVE.W #\1,MC68881+COND          BEGIN COPROCESSOR COMMUNICATION
732 \@NOPASS MOVE.W MC68881+RESPONSE,D0       IS CA-BIT SET
733          BMI.S    \@NOPASS                REREAD UNTIL NULL RELEASE (CA=0)
734          BTST #TFBIT,D0                   IS CONDITION TRUE
735          DBNE \2,\3                       SUBTRACT 1 FROM COUNTER UNTIL COUNTER
736 *                                         EQUALS -1
737          ENDM
738 ***************************************************************************
739 *                                                                         *
740 *        MC68881 CONDITIONAL SET                                          *
741 *                                                                         *
742 *        FSCC CONDITION,ADDRESS                                           *
743 *                                                                         *
744 *        WHERE:  CONDITION= CC, FLOATING POINT CONDITION                  *
745 *                ADDRESS= BRANCH ADDRESS                                  *
746 *                                                                         *
747 *        REGISTERS MODIFIED OR DESTROYED: 0 1 2 3 4 5 6 7                 *
748 *                                         D X                             *
749 *                                         A                               *
750 *                                                                         *
751 ***************************************************************************
752 FSCC     MACRO
753          MOVE.W #\1,MC68881+COND          BEGIN COPROCESSOR COMMUNICATION
754 \@NOPASS MOVE.W MC68881+RESPONSE,D0       IS CA-BIT SET
755          BMI.S    \@NOPASS                REREAD UNTIL NULL RELEASE (CA=0)
756          BTST #TFBIT,D0                   IS CONDITION TRUE
757          SNE \2                           SET BYTE AT POINTER(\2) TO 1'S IF
758 *                                         CONDITION TRUE, IF CONDITION FALSE
759 *                                         SET BYTE TO 0'S
760          ENDM
761 ***************************************************************************
762 *                                                                         *
763 *        MC68881 FP MOVE MULTIPLE COPROCESSOR REGISTERS TO MEMORY         *
764 *                                                                         *
765 *        FMOVEMRM FPR0,FPR1,FPR2,FPR3,FPR4,FPR5,FPR6,FPR7,<EA>,PREDECREMENT *
766 *                                                                         *
767 *        WHERE:  FPR0=(FP REG.#0)     1 IF SELECTED, 0 IF NOT             *
768 *                FPR1=(   "    #1)    "              "                    *
769 *                FPR2=(   "    #2)    "              "                    *
770 *                FPR3=(   "    #3)    "              "                    *
771 *                FPR4=(   "    #4)    "              "                    *
772 *                FPR5=(   "    #5)    "              "                    *
773 *                FPR6=(   "    #6)    "              "                    *
774 *                FPR7=(   "    #7)    "              "                    *
775 *                <EA>= DESTINATION ADDRESSING MODE                        *
776 *                PREDECREMENT=  Y (IF PREDECREMENT MODE IS BEING USED), OR *
777 *                               N (IF OTHER MODE IS BEING USED).          *
778 *                                                                         *
779 *        REGISTERS MODIFIED OR DESTROYED: 0 1 2 3 4 5 6 7                 *
780 *                                         A X                             *
```

ARCHIVE DOCUMENT NOT FOR NEW DESIGN

```
781  *                                          D XXXX                      *
782  *                                                                      *
783  *        VALID ADDRESSING MODES:  AN, -(AN), D(AN), D(AN,IX)           *
784  *                                 XXX.W, XXX.L                         *
785  *                                                                      *
786  ************************************************************************
787  FMOVEMRM MACRO
788           IFC '\A','Y'                IS THE ADDRESSING MODE PREDECREMEN
789  *
790  *  THIS CODE IS FOR PREDECREMENT ADDRESSING MODE
791  *
792           MOVE.W #$E000+%\8\7\6\5\4\3\2\1,MC68881+COMMAND    FP REGISTER BIT
793  *                                        MASK INTO COMMAND REGISTER
794  \@NULCA  CMPI #$8900,MC68881+RESPONSE    READ RESPONSE REGISTER
795           BEQ.S    \@NULCA                REREAD UNTIL TRANSFER MULTIPLE REGS.
796  *
797  *   THIS CODE CALCULATES THE TOTAL # OF REGISTERS TO BE TRANSFERRED
798  *
799           MOVEQ #\1+\2+\3+\4+\5+\6\7+\8-1,D3
800  *
801           TST.W MC68881+REGSEL             READ REGISTER RESPONSE REGISTER
802           MOVE.L MC68881+OPER,A0           A0=ADDRESS OF THE OPERAND REG.
803  \@AGAIN  MOVE.L (A0),D0                  LOAD HIGH ORDER WORDS
804           MOVE.L (A0),D1                  LOAD MID ORDER WORDS
805           MOVE.L (A0),D2                  LOAD LOW ORDER WORDS
806           MOVEM.L D0-D2,\9                STACK HIGH ORDER WORD IN LOW ORDER
807  *                                        MEMORY AND LOW ORDER WORD IN HIGH
808  *                                        ORDER MEMORY
809           DBRA D3,\@AGAIN                 HAVE ALL REGISTERS BEEN TRANSFERRED
810  *
811  *
812  \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
813           BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
814           ENDC
815           IFC '\A','N'                    IS ADDRESSING MODE NOT PREDECREMENT
816  ************************************************************************
817  *
818  *        THIS CODE IS FOR ALL VALID ADDRESSING MODES OTHER
819  *        THAN PREDECREMENT
820  *
821  ************************************************************************
822           MOVE.W #$F000+%\1\2\3\4\5\6\7\8,MC68881+COMMAND    CP REGISTER BIT
823  *                                        MASK AND START CP COMM NICATION
824  \@NULCA  CMPI #$8900,MC68881+RESPONSE    READ RESPONSE REGISTER
825           BEQ.S    \@NULCA                REREAD UNTIL TRANSFER MULTIPLE REGS.
826           MOVEQ.L #(\8+\7+\6+\5+\4+\3+\2+\1)*3-1,D0  COUNT REG. FOR DBRA STMT.
827           TST.W MC68881+REGSEL             READ REGISTER RESPONSE REGISTER
828           LEA \9,A0                        SET UP A MEMORY POINTER
829  \@AGAIN  MOVE.L MC68881+OPER,(A0)+        LOAD DATA ON TO THE STACK
830           DBRA D0,\@AGAIN                  LOOP UNTIL ALL DATA IS LOADED
831  \@NULREL TST.B MC68881+RESPONSE          IS RESPONSE NULL RELEASE?
832           BMI.S \@NULREL                  BRANCH UNTIL NULL RELEASE
833           ENDC
834           ENDM
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
835 ******************************************************************************
836 *                                                                            *
837 *        MC68881 FP MOVE TO MULTIPLE COPROCESSOR REGISTERS FROM MEMORY       *
838 *                                                                            *
839 *        FMOVEMMR <EA>,FPR0,FPR1,FPR2,FPR3,FPR4,FPR5,FPR6,FPR7,POSTINCREMENT *
840 *                                                                            *
841 *        WHERE:  <EA>= DESTINATION ADDRESSING MODE                           *
842 *                FPR0=(FP REG.#0)     1 IF SELECTED, 0 IF NOT                *
843 *                FPR1=(   "   #1)     "              "                       *
844 *                FPR2=(   "   #2)     "              "                       *
845 *                FPR3=(   "   #3)     "              "                       *
846 *                FPR4=(   "   #4)     "              "                       *
847 *                FPR5=(   "   #5)     "              "                       *
848 *                FPR6=(   "   #6)     "              "                       *
849 *                FPR7=(   "   #7)     "              "                       *
850 *                POSTINCREMENT= Y (IF POST-INCREMENT MODE IS BEING USED)     *
851 *                               N (IF OTHER VALID MODE IS BEING USED).       *
852 *                                                                            *
853 *         REGISTERS MODIFIED OR DESTROYED: 0 1 2 3 4 5 6 7                   *
854 *                                          A X                               *
855 *                                          D X                               *
856 *                                                                            *
857 *        VALID ADDRESSING MODES:  AN,  (AN)+,  D(AN),  D(AN,IX)              *
858 *                                 XXX.W, XXX.L,  (D,PC),  D(PC,IX)           *
859 *                                                                            *
860 ******************************************************************************
861 FMOVEMMR MACRO
862          MOVE.W #$D000+%\2\3\4\5\6\7\8\9,MC68881+COMMAND  CP REGISTER BIT
863 *                                              MASK AND START CP COMMUNICATION
864 \@NULCA  CMPI #$8900,MC68881+RESPONSE          READ RESPONSE REGISTER
865          BEQ.S    \@NULCA                      REREAD UNTIL TRANSFER MULTIPLE REGS.
866          MOVEQ.L #(\9+\8+\7+\6+\5+\4+\3+\2)*3-1,D0  DECREMENT REG. FOR DBRA
867          TST.W MC68881+REGSEL                  READ REGISTER RESPONSE REGISTER
868          IFC '\A','N'                          IS ADDRESSING MODE NOT POSTINCREMENT
869          LEA \1,A0                             SET UP A MEMORY POINTER
870 \@AGAIN  MOVE.L (A0)+,MC68881+OPER             LOAD DATA ON TO THE STACK
871          DBRA D0,\@AGAIN                       LOOP UNTIL ALL DATA IS LOADED
872 \@NULREL TST.B MC68881+RESPONSE                IS RESPONSE NULL RELEASE?
873          BMI.S \@NULREL                        BRANCH UNTIL NULL RELEASE
874          ENDC
875          IFC '\A','Y'                          IS ADDRESSING MODE POSTINCREMENT
876 \@AGAIN  MOVE.L \1,MC68881+OPER                LOAD DATA ON TO THE STACK
877          DBRA D0,\@AGAIN                       LOOP UNTIL ALL DATA IS LOADED
878 \@NULREL TST.B MC68881+RESPONSE                IS RESPONSE NULL RELEASE?
879          BMI.S \@NULREL                        BRANCH UNTIL NULL RELEASE
880          ENDC
881          ENDM
882 ******************************************************************************
883 *                                                                            *
884 *        MC68881 FP MOVE TO CONTROL, STATUS,OR INSTRUCTION ADDRESS REGISTER  *
885 *                                                                            *
886 *        MOVINCSI <EA>,REGISTER                                              *
887 *                                                                            *
888 *        WHERE:   <EA>= VALID SOURCE ADDRESSING MODE                         *
889 *                 REGISTER= CONTROL,STATUS, OR IADDRESS                      *
890 *                                                                            *
891 *        NO REGISTERS MODIFIED OR DESTROYED!                                 *
892 *                                                                            *
893 *        VALID ADDRESSING MODES:  DN, AN,  (AN)+,  -(AN),  D(AN),  D(AN,IX)  *
894 *                                 XXX.W, XXX.L,  (D,PC),  D(PC,IX)           *
895 *                                                                            *
896 ******************************************************************************
897 MOVINCSI MACRO
898          IFC '\3',''                           IS ADDR.MODE INDEXED?
899          MOVE.W #\2,MC68881+COMMAND            MOVE BIT PATTERN IN COMMAND REG.
900 \@NULCA  CMPI.W #$8900,MC68881+RESPONSE        IS RESPONSE NULL COME AGAIN?
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
901          BEQ.S \@NULCA                        COME AGAIN UNTIL NEW RESPONSE
902          MOVE.L \1,MC68881+OPER               PASS DATA TO REGISTER
903 \@NULREL TST.B MC68881+RESPONSE               IS RESPONSE NULL RELEASE?
904          BMI.S \@NULREL                       BRANCH UNTIL NULL RELEASE
905          ENDC
906          IFNC '\3',''                         IS ADDRESS MODE INDEXED?
907          MOVE.W #\3,MC68881+COMMAND           MOVE BIT PATTERN IN COMMAND REG.
908 \@NULCA  CMPI.W #$8900,MC68881+RESPONSE       IS RESPONS NULL COME AGAIN?
909          BEQ.S \@NULCA                        COME AGAIN UNTIL NEW RESPONSE
910          MOVE.L \1,\2,MC68881+OPER            PASS DATA TO REGISTER FROM INDEXED ADDR.
911 *                                             MODE
912 \@NULREL TST.B MC68881+RESPONSE               IS RESPONSE NULL RELEASE?
913          BMI.S \@NULREL                       BRANCH UNTIL NULL RELEASE
914          ENDC
915          ENDM
916 ***************************************************************************
917 *                                                                         *
918 *        MC68881 FP MOVE FROM CONTROL/STATUS/INSTRUCTION ADDRESS REGISTER *
919 *                                                                         *
920 *        MOVOUCSI REGISTER,<EA>                                           *
921 *                                                                         *
922 *        WHERE:   REGISTER= CONTROL,STATUS, OR IADDRESS                   *
923 *                 <EA>= VALID SOURCE ADDRESSING MODE                      *
924 *                                                                         *
925 *        NO REGISTERS MODIFIED OR DESTROYED!                              *
926 *                                                                         *
927 *        VALID ADDRESSING MODES:  DN, AN, (AN)+, -(AN), D(AN), D(AN,IX)   *
928 *                                 XXX.W, XXX.L, (D,PC), D(PC,IX)          *
929 *                                                                         *
930 ***************************************************************************
931 MOVOUCSI MACRO
932          IFC '\3',''                          IS ADDR.MODE INDEXED?
933          MOVE.W #\1+$2000,MC68881+COMMAND     MOVE BIT PATTERN TO COMMAND REG.
934 \@NULCA  CMPI.W #$8900,MC68881+RESPONSE       IS RESPONSE NULL COME AGAIN?
935          BEQ.S \@NULCA                        COME AGAIN UNTIL NEW RESPONSE
936          MOVE.L MC68881+OPER,\2               PASS DATA TO REGISTER
937 \@NULREL TST.B MC68881+RESPONSE               IS RESPONSE NULL RELEASE?
938          BMI.S \@NULREL                       BRANCH UNTIL NULL RELEASE
939          ENDC
940          IFNC '\3',''                         IS ADDR.MODE INDEXED?
941          MOVE.W #\1+$2000,MC68881+COMMAND     MOVE BIT PATTERN TO COMMAND REG.
942 \@NULCA  CMPI.W #$8900,MC68881+RESPONSE       IS RESPONSE NULL COME AGAIN?
943          BEQ.S \@NULCA                        COME AGAIN UNTIL NEW RESPONSE
944          MOVE.L MC68881+OPER,\2,\3            PASS DATA TO REGISTER FROM INDEXED A
945 *                                             MODE
946 \@NULREL TST.B MC68881+RESPONSE               IS RESPONSE NULL RELEASE?
947          BMI.S \@NULREL                       BRANCH UNTIL NULL RELEASE
948          ENDC
949          ENDM
950 ***************************************************************************
951 *                                                                         *
952 *        MC68881 FSAVE THE INTERNAL OF THE MACHINE                        *
953 *                                                                         *
954 *        THIS IS A PRIVILEDGED INSTRUCTION WHICH IS GENERALLY ONLY USED   *
955 *        IN THE OPERATING SYSTEM FOR CONTEXT SWITCHING!                   *
956 *                                                                         *
957 *        FSAVEST <EA>                                                     *
958 *                                                                         *
959 *        WHERE:  <EA>= PREDECREMENT MODE   -(AN)                          *
960 *                                                                         *
961 *        REGISTERS MODIFIED OR DESTROYED:  0 1 2 3 4 5 6 7                *
962 *                                        A X                              *
963 *                                        D X X                            *
964 *                                                                         *
965 *        VALID ADDRESSING MODES:  -(AN)                                   *
966 *                                                                         *
```

ARCHIVED DOCUMENT - NOT FOR NEW DESIGN

```
967 *************************************************************************
968 FSAVEST  MACRO
969 \@START  MOVE.W MC68881+SAVE,D0        READ THE SAVE REGISTER
970          MOVE.W D0,D1                  MAKE A COPY OF THE FORMAT WORD
971          ANDI.W #$FF00,D1              ISOLATE THE FORMAT WORD
972          BEQ.S \@NULL                  IF NULL IDLE, NO STATE SAVE
973          CMPI.W #$0100,D1              IS THE COPROCESSOR BUSY
974          BEQ.S \@START                 KEEP CHECKING UNTIL CP IS FINISHED
975 *                                      PROCESSING
976          LEA MC68881+OPER,A0           LOAD OPERAND REGISTER TO A0
977 *
978          MOVE.B D0,D1                  THE LENGTH OF THE DATA TO BE TRANSFERED
979          LSR.B #2,D1                   DEVIDE BY 2 TO ADJUST FOR WORD TRANSFER
980          EXT.W D1                      ESTABLISH COUNT AS A WORD FOR DBRA
981          SUBQ.W #1,D1                  D1= COUNTER FOR DBRA
982 \@LOAD   MOVE.L (A0),\1                STORE THE INVISBLE STATE
983          DBRA D1,\@LOAD                REPEAT UNTIL ALL DATA IS TRANSFERRED
984 \@NULL   SWAP D0                       PLACE FORMAT WORD IN UPPER 16 BITS OF D0
985          MOVE.L D0,\1                  STORE FORMAT WORD ON THE STACK
986          ENDM
987 *************************************************************************
988 *                                                                       *
989 *        MC68881 FRESTORE OF THE INTERNAL OF THE MACHINE                *
990 *                                                                       *
991 *        THIS IS A PRIVILEDGED INSTRUCTION WHICH IS GENERALLY ONLY USED *
992 *        IN THE OPERATING SYSTEM FOR CONTEXT SWITCHING!                 *
993 *                                                                       *
994 *        FRESTRST <EA>                                                  *
995 *                                                                       *
996 *        WHERE:  <EA>= POSTINCREMENT MODE    (AN)+                      *
997 *                                                                       *
998 *        REGISTERS MODIFIED OR DESTROYED:  0 1 2 3 4 5 6 7              *
999 *                                        A X                            *
1000 *                                       D X X                          *
1001 *                                                                      *
1002 *       VALID ADDRESSING MODES:  (AN)+                                 *
1003 *                                                                      *
1004 ************************************************************************
1005 FRESTRST MACRO
1006         MOVE.L \1,D0                   MOVE FORMAT WORD AND RESERVED WORD TO D0
1007         SWAP D0                        PLACE FORMAT WORD AS THE LOW ORDER
1008         MOVE.W D0,MC68881+RESTORE      STORE FORMAT WORD IN RESTORE REG.
1009         MOVE.W MC68881+RESTORE,D0      READ THE RESTORE REGISTER
1010         MOVE.W D0,D1                   MAKE A COPY OF THE RESPONSE FORMAT WORD
1011         ANDI.W #$FF00,D1               ISOLATE THE FORMAT WORD
1012         BEQ.S \@NULREL                 IF NULL IDLE RESPONSE, NO STATE RESTORED
1013         LEA MC68881+OPER,A0            LOAD OPERAND REGISTER TO A0
1014 *
1015         MOVE.B D0,D1                   THE LENGTH OF THE DATA TO BE TRANSFERED
1016         LSR.B #2,D1                    DEVIDE BY 2 TO ADJUST FOR WORD TRANSFER
1017         EXT.W D1                       ESTABLISH COUNT AS A WORD FOR DBRA
1018         SUBQ.W #1,D1                   D1= COUNTER FOR DBRA
1019 \@LOAD  MOVE.L \1, (A0)                STORE THE INVISBLE STATE
1020         DBRA D1 \@LOAD                 REPEAT UNTIL ALL DATA IS TRANSFERRED
1021 \@NULREL EQU *
1022         ENDM
1023 ************************************************************************
1024 *                                                                      *
1025 *       MC68881 FNOPP COMMAND                                          *
1026 *                                                                      *
1027 *       FNOP                                                           *
1028 *                                                                      *
1029 *       NO REGISTERS MODIFIED OR DESTROYED!                            *
1030 *                                                                      *
1031 ************************************************************************
1032 FNOPP   MACRO
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN

```
1033            MOVE.W #$0000,MC68881+COND         FNOP COMMAND TO FP REG.
1034  \@NOPAS   TST.B MC68881+RESPONSE             TEST RESPONSE
1035            BMI.S \@NOPAS
1036            ENDM
```

ARCHIVE DOCUMENT - NOT FOR NEW DESIGN