TOSHIBA

16-BIT MICROPROCESSOR

TLCS-68000

USERS MANUAL

AUGUST 1988

TOSHIBA CORPORATION

TOSHIBA TLCS-68000 USERS MANUAL 1988

4416

The information contained herein is presented only as a guide for the applications of our products. No responsibility is assumed by TOSHIBA for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of TOSHIBA or others.

The products described in this document are strategic products subject to COCOM regulations. They should not be exported without authorization from the appropriate governmental authorities.

The products described in this document contain components made in the United States and subject to export control of the U.S. authorities. Diversion contrary to the U.S. law is prohibited.

June 1988

Preface

Thank you very much for making use of TOSHIBA microcomputer LSIs and development systems.

The TLCS-68000 family, including the TMP68000, is the general-purpose 16/32-bit microprocessor family which is developed by technical cooperation with Motorola Inc., and is compatible with the Motorola M68000 family. The TMP68000/10/08 have various features such as the general purpose 32-bit register set, the large linear address space, the powerful instruction set, and flexible addressing modes. The common 32-bit internal architecture is upward compatible with all the family MPUs. This manual describes overview of the architecture and function of each instruction set, which are requires for software development for each MPU (TMP68000/10/08) of TLCS-68000 family. Toshiba provides various microcomputer LSIs and its development system for wide range of application.

TOSHIBA

TLCS-68000

16-Bit Microprocessors

User's Manual

TOSHIBA CORPORATION

CONTENTS

1. ARCHITECTURAL DESCRIPTION

1.1 INTRODUCTION

The TMP68000, with a 16-bit data bus and 24-bit address bus, was only the first in a family of processors which implement a comprehensive, extensible computer architecture. It was soon followed by the TMP68008, with an 8-bit data bus and 20-bit address bus, by the TMP68010, which introduced the virtual machine aspects of the TLCS-68000 architecture.

This manual is intended to serve as a programmer's reference for both systems and applications programmers for four of the current implementations of the TLCS-68000 - the TMP68000, the TMP68008, the TMP68010. The hardware system design aspects of these processors, such as bus structure and control, are presented in the respective advance information data sheets for each device.

The TMP68000 and the TMP68008 are identical from the view of the programmer, with the exception that the TMP68000 can directly access 16 megabytes (24 bits of address) and the TMP68008 can directly access 1 megabyte (20 bits of address). The TMP68010 have much in common with the first two devices but also possess some additional instructions and registers as well as full virtual machine/memory capability. Since the processors are so similar to the programmer, only the differences are highlighted. When the TLCS-68000 is referenced, the feature described is common to all. If a particular feature is applicable only to one processor, the TMP part number will be referenced.

1.2 PROGRAMMER'S MODEL

The TLCS-68000 executes instructions in one of two modes - user mode or supervisor mode. The user mode is intended to provide the execution environment for the majority of application programs. The supervisor mode allows some additional instructions and privileges and is intended for use by the operating system and other system software. See "4. EXCEPTION PROCESING" for further details.

To provide for the upward compatibility of code written for a specific implementation of the TLCS-68000, the user programmer's model is common to all implementations. The user programmer's model is shown in Figure 1.1.

As shown in the user programmer's model, the TLCS-68000 offers 16 32-bit general purpose registers (D0~D7, A0~A7), a 32-bit program counter, and an 8-bit condition code register. The first eight registers (D0~D7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) operations. The second set of seven registers (A0~A6) and the stack pointer (USP) may be used as software stack pointers and base address registers. In addition, the address registers may be used for word and long word operations. All of the 16 registers may be used as index registers.

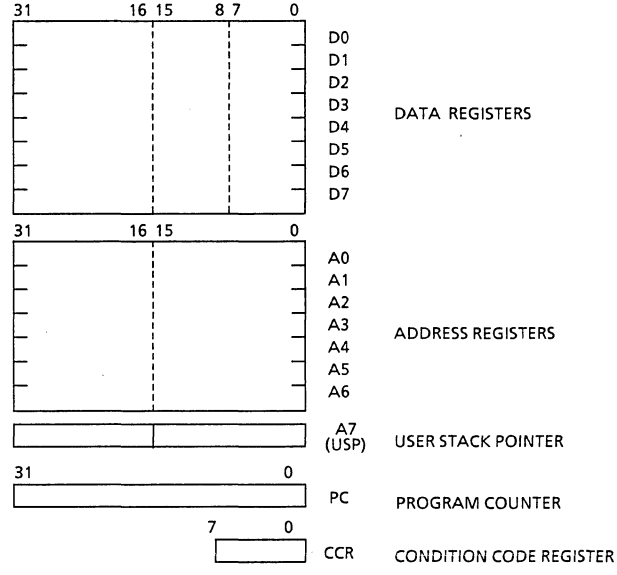


Figure 1.1 User Programmer's Model (TMP68000/TMP68008/TMP68010)

The supervisor programmer's model includes some supplementary registers in addition to the above mentioned registers. The TMP68000 and the TMP68008 contain identical supervisor mode register resources. These are shown in Figure 1.2 and include the status register (high order byte) and the supervisor stack pointer (A7').

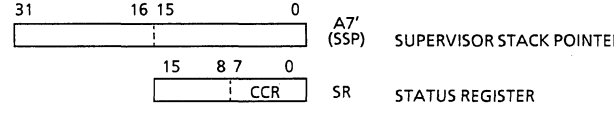


Figure 1.2 Supervisor Programmer's Model Supplement (TMP68000/TMP68008)

The supervisor programming model supplement of the TMP68010 is shown in Figure 1.3. In addition to the supervisor stack pointer and status register, it includes the vector base register and the alternate function code registers.

The vector base register is used to determine the location of the exception vector table in memory to support multiple vector tables. The alternate function code registers allow the supervisor to access user data space or emulate CPU space cycles.

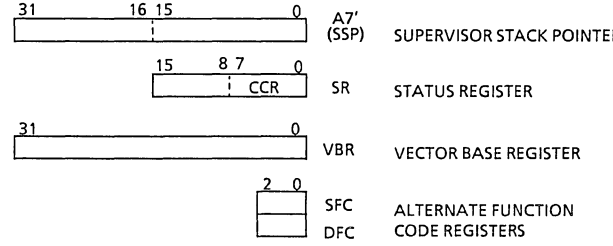


Figure 1.3 Supervisor Programmer's Model Supplement (TMP68010)

The status register, shown in Figure 1.4, contains the interrupt mask (eight levels available) as well as the condition codes: overflow (V) , zero (Z), negative (N), carry(C), and extend (X). Additional status bits indicate that the processor is in a trace (T) mode and/or in a supervisor (S) state.

Five basic data types are supported. These data types are:

- Bits
- BCD Digits (4 Bits)
- Bytes (8 Bits)
- Words (16 Bits)
- Long Words (32 Bits)

In addition, operations on other data types such as memory addresses, status word data, etc. are provided for in the instruction set.

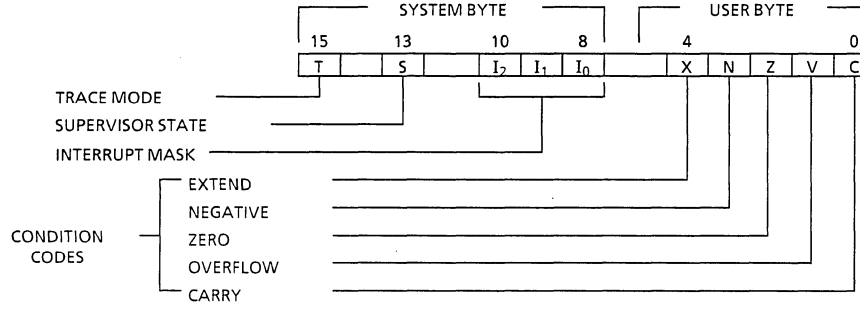


Figure 1.4 Status Register

The 14 flexibe addressing modes, shown in Table 1.1, include six basic types:

- Register Direct
- Register Indirect
- Absolute
- Immediate
- Program Counter Relative
- Implied

Included in the register indirect addressing modes is the capability to do postincrementing, predecrementing, offsetting, and indexing. Program counter relative mode can also be modified via indexing and offsetting.

Table 1.1 Data Addressing Modes

Mode	Generation
Register Direct Addressing	
Data Register Direct	EA = Dn
Address Register Direct	EA = An
Absolute data Addressing	
Absolute Short	EA = (Next Word)
Absolute Long	EA = (Next Two Words)
Program Counter Relative Addressing	
Relative with Offset	EA = (PC) + d16
Relative with Index and Offset	EA = (PC) + (Xn) + d8
Register Indirect Addresssing	
Register Indirect	EA = (An)
Postincrement Register Indirect	EA = (An), An←An + N
Predecrement Register Indirect	An←An - N, EA = (An)
Register Indirect with Offset	EA = (An) + d16
Indexed Register Indirect with Offset	EA = (An) + (Xn) + d8
Immediate Data Addressing	
Immediate	DATA = Next Word(s)
Quick Immediate	Inherent Data
Implied Addressing	
Implied Register	EA = SR, USP, SSP, PC, VBR, SFC, DFC

Notes: EA = Effective Address
An = Address Register
Dn = Data Register
Xn = Address or Data Register used as Index Register
SR = Status Register
PC = Program Counter
( ) = Contents of
d8 = 8-bit Offset (Displacement)
d16 = 16-bit Offset (Displacement)
N = 1 for byte, 2 for word, and 4 for long word. If An is the stack pointer and the operand size is byte, N = 2 to keep the stack pointer on a word boundary.
← = Replaces

The TLCS-68000 instruction set is shown in Table 1.2. Some additional instructions are variations or subsets of these and they appear in Table 1.3. Special emphasis has been given to the instruction set's support of structured high-level languages to facilitate ease of programming. Each instruction, with a few exceptions, operates on bytes, words, and long words and most instructions can use any of the 14 addressing modes. Combining instruction types, data types, and addressing modes, over 1000 useful instructions are provided. These instructions include signed and unsigned multiply and divide, "quick" arithmetic operations, BCD arithmetic, and expanded operations (through traps). Additionally, its high-symmetric, proprietary microcoded structure provides a sound, flexible base for the future.

1.3 SOFTWARE DEVELOPMENT

Many innovative features have been incorporated to make programming easier, faster, and more reliable.

1.3.1 Consistent Structure

The highly regular structure of the TLCS-68000 greatly simplifies the effort required to write programs in assembly language as well as high-level languages. Operations on integer data in registers and memory are independent of the data. Separate special instructions that operate on byte (8 bit), word (16 bit), and long word (32 bit) integers are not necessary. The programmer need only remember one mnemonic for each type of operation and then specify data size, source addressing mode, and destination addressing mode. This has helped keep the total number of instructions small.

The dual operand nature of many of the instructions significantly increases the flexibility and power of the TLCS-68000. Consistency is again maintained since all data registers and memory locations may be either a source or destination for most operations on integer data.

The addressing modes have been kept simple without sacrificing efficiency. All fourteen addressing modes operate consistently and are independent of the instruction operation itself. Additionally, all address registers may be used for the direct, register indirect, and indexed addressing modes (immediate, program counter relative, and absolute addressing by definition do not use address registers). For increased flexibility, any address or data register may be used as an index register. Address register consistency is maintained for stacking operations since any of the eight address registers may be utilized as user program stack pointers with the register indirect postincrement/predecrement addressing modes. Address register A7, however, is a special register that, in addition to is normal addressing capability, functions as the system stack pointer for stacking the program counter for subroutine calls as well as stacking the program counter and status register for traps and interrupts (while in the supervisor state).

Table 1.2 Instruction Set Summary (1 / 2)

Mnemonic	Description
ABCD*	Add Decimal with Extned
ADD*	Add
AND*	Logical AND
ASL*	Arithmetic Shift Left
ASR*	Arithmetic Shift Right
Bcc	Branch Conditionally
BCHG	Bit Test and Change
BCLR	Bit Test and Clear
BKPT	Breakpoint
BRA	Branch Always
BSET	Bit Test and Set
BSR	Branch to Subroutine
BTST	Bit Test
CHK	Check Register Against Bounds
CLR*	Clear Operand
CMP*	Compare
DBcc	Decrement and Branch Conditionally
DIVS	Signed Divide
DIVU	Unsigend Divide
EOR*	Exclusive OR
EXG	Exchange Registers
EXT	Sign Extend
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link Stack
LSL*	Logical Shift Left
LSR*	Logical Shift Right

* : These instructions available in loop mode on TMP68010.
See "APPENDIX G TMP68010 LOOP MODE OPERATIONS".

Table 1.2 Instruction Set Summary (2 / 2)

Mnemonic	Description
MOVE*	Move Source to Destination
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD*	Negate Decimal with Extend
NEG*	Negate
NOP	No Operation
NOT*	One's Complement
OR*	logical OR
PEA	Push Effective Address
RESET	Reset External Devices
ROL*	Rotate Left without Extend
ROR*	Rotate Right without Extend
ROXL*	Rotate Left with Extend
ROXR*	Rotate Right with Extend
RTD	Return and Deallocate
RTE	Return from Exception
RTR	Return and Restore
RTS	Return from Subroutine
SBCD*	Subtract Decimal with Extend
Scc	Set Conditional
STOP	Stop
SUB*	Subtract
SWAP	Swap Data Register Halves
TAS	Test and Set Operand
TRAP	Trap
TRAPV	Trap on Overflow
TST*	Test
UNLK	Unlink

Table 1.3 Variations of Instruction Types

Instruction Type	Variation	Description
ADD	ADD*	Add
	ADDA*	Add Address
	ADDQ	Add Quick
	ADDI	Add Immediate
	ADDX*	Add with extend
AND	AND*	Logical AND
	ANDI	AND Immediate
	ANDI to CCR	AND Immediate to Condition Codes
	ANDI to SR	AND Immediate to Status Regitster
CMP	CMP*	Compare
	CMPA*	Compare Address
	CMPM*	Compare Memory
	CMPI	Compare Immediate
EOR	EOR*	Exclusive OR
	EORI	Exclusive OR Immediate
	EORI to CCR	Exclusive OR Immediate to Condition Codes
	EORI to SR	Exclusive OR Immediate to Status Register
MOVE	MOVE*	Move Source to Destination
	MOVEA*	Move Address
	MOVEC	Move Control Register
	MOVEM	Move Multiple Registers
	MOVEP	Move Peripheral Data
	MOVEQ	Move Quick
	MOVES	Move Alternate Address Space
	MOVE from SR	Move from Status Register
	MOVE to SR	Move to Status Register
	MOVE from CCR	Move from Condition Codes
	MOVE to CCR	Move to Condition Codes
	MOVE USP	Move User Stack Pointer
NEG	NEG*	Negate
	NEGX*	Negate with Extend
OR	OR*	Logical OR
	ORI	OR Immediate
	ORI to CCR	OR Immediate to Condition Codes
	ORI to SR	OR Immediate to Status Register
SUB	SUB*	Subtract
	SUBA*	Subtract Addres
	SUBI	Subtract Immediate
	SUBQ	Subtract Quick
	SUBX*	Subtract with Extend

* These instructions available in loop mode on TMP68010.
See "APPENDIX G TMP68010 LOOP MODE OPERATIONS".

1.3.2 Structured Modular Programming

The art of programming microprocessors has evolved rapidly in the past few years. Numerous advanced techniques have been developed to allow easier, more consistent and reliable generation of software. In general, these techniques require that the programmer be more disciplined in observing a defined programming structure such as modular programming. Modular programming allows a required function or process to be broken down in short modules or subroutines that are concisely defined and easily programmed and tested. Such a technique is greatly simplified by the availability of advanced structured assemblers and block structured high-level languages such as Pascal. Such concepts are virtually useless, however, unless parameters are easily transferred between and within software modules that operate on a reentrant and recursive basis. (To be reentrant a routine must be usable by interrupt and non-interrupt driven programs without the loss of data. A recursive routine is one that may call or use itself.) The TLCS-68000 provides the necessary architectural features to allow efficient reentrant modular programming. The LINK and UNLK instructions reduce subroutine call overhead in two complementary instructions by allowing the manipulation of linked lists of data areas on the stack. The MOVEM (Move Multiple Register) instruction also reduces subroutine call programming overhead. This allows moving, via an effective address, multiple registers that are specified by the programmer. Sixteen software trap vector are provided with the TRAP instruction and are useful in operating system call routines or user generated macro routines. Other instructions that support modern structured programming techniques are PEA (Push Effective Address), LEA (Load Effective Address), RTR (Return and Restore), RTE (Return from Exception) as well as JSR (Jump to Subroutine), BSR (Branch to Subroutine), and RTS (Return from Subroutine).

The powerful vectored priority interrupt structure of the microprocessor allows straight-foward generation of reentrant modular input/output routines. Seven maskable levels of priority with 192 vector locations and seven autovector locations provide maximum flexibility for I/O control (a total of 255 vector locations are available for interrupts, hardware traps, and software traps).

1.3.3 Improved Software Testability

The TLCS-68000 incorporates several features that reduce the chance for errors. Some of these features, such as consistent architecture and the structured modular programming capability, have already been discussed.

Of major importance to the system programmer are features that have been incorporated specifically to detect the occurrence of programming errors or bugs. Several hardware traps, provided to indicate abnormal internal conditions, detect the following error conditions:

- Word Access with an Odd Address
- Illegal Instructions
- Unimplemented Instructions
- Illegal Memory Access (Bus Error)
- Divide by Zero
- Overflow Condition Code (Separate Instruction TRAPV)
- Register Out of Bounds (CHK Instruction)
- Spurious Interrupt

Additionally, the sixteen software TRAP instructions may utilized by the programmer to provide applications-oriented error detection or correction routines.

An additional error detection tool is the CHK (Check Register Against Bounds) instruction used for array bound checking by verifying that a data register contains a valid subscript. A trap occurs if the register contents are negative or greater than a limit.

Finally, the TLCS-68000 includes a facility that allows instruction-by-instruction tracing of a program being debugged. This trace mode results in a trap being made to a tracing routine after each instruction executed. The trace mode is available to the programmer when the microprocessor is in the supervisor state as well as the user state but may only be entered while in the supervisor state. The supervisor/user states provide an additional degree of error protection for the microprocessor by allowing memory protection of selected areas of memory when an external memory management device is used.

1.4 VIRTUAL MEMORY/MACHINE CONCEPTS

The TMP68010 introduced the virtual memory/machine concept of the TLCS-68000 architecture.

In most systems using the TMP68010 as the central processor, only a franction of the 16 megabyte address space will actually contain physical memory. However, by using virtual memory techniques the system can be made to appear to the user to have 16 megabytes of physical memory available. These techniqes have been used for several years in large mainframe computers and more recently in minicomputers and now, with the TMP68010, can be fully supported in microprocessor-based systems.

In a virtual memory system, a user program can be written as though it has a large amount of memory available to it when only a small amount of memory is physically present in the system. In a similar fashion, a system can be designed in such a manner as to allow user programs to access other types of devices that are not physically present in the system such as type drives, disk drives, printers, or CRTs. With proper software emulation, a physical system can be made to appear to a user program as any other computer system and the program may be given full access to all of the resources of that emulated system. Such an emulated system is called a virtual machine.

1.4.1 Virtual Memory

The basic mechanism for supporting virtual memory in computers is to provide only a limited amount of high-speed physical memory that can be accessed directly by the processor while maintaining an image of a much larger "virtual" memory on secondary storage devices such as large capacity disk drives. When the processor attempts to access a location in the virtual memory map that is not currently residing in physical memory (referred to as a page fault), the access to that location is temporarily suspended while the necessary data is fetched from the secondary storage and placed in physical memory; the suspended access is then completed. The TMP68010 provides hardware support for virtual memory with the capability of suspending an instruction's execution when a bus error is signaled and then completing the instruction after the physical memory has been updated as necessary.

The TMP68010 uses instruction continuation rather than instruction restart to support virtual memory. With instruction restart, the processor must remember the exact state of the system before each instruction is started in order to restore that state if a page fault occurs during its execution. Then, after the page fault has been repaired, the entire instruction that cauesd the fault is reexecuted. With instruction continuation, when a page fault occurs the processor stores its internal state and then after the page fault is repaired, restores that internal state and continues execution of the instruction. In order for the TMP68010 to utilize instruction continuation, it stores its internal state on the supervisor stack when a bus cycle is terminated with a bus error signal. It then loads the program counter from vector table entry number two (offset $008) and resumes program execution at that new address. When the bus error exception handler routine has completed execution, an RTE instruction is executed which reloads the TMP68010 with the internal state stored on the stack, re-runs the faulted bus cycle, and continues the suspended instruction. Instruction continuation has the additional advantage of allowing hardware support for virtual I/O devices. Since virtual registers may be simulated in the memory map, an access to such a register will cause a fault and the function of the register can be emulated by software.

1.4.2 Virtual Machine

One typical use for a virtual machine system is in the development of software such as an operating system for another machine with hardware also under development and not available for programming use. In such a system, the governing operating system emulates the hardware of the new system and allows the operating system to be executed and debugged as though it were running on the new hardware. Since the new operating system is controlled by the governing operating system, the new one must execute at a lower privilege level than the governing operating system, so that any attempts by the new operating system to use virtual resources that are not physically present, and should be emulated, will be trapped by the governing operating system and handled in software. In the TMP68010, a virtual machine may be fully supported by running the new operating system in the user mode and the governing operating system in the supervisor mode so that any attempts to access supervisor resources or execute privileged instructions by the new operating system will cause a trap to the governing operating system.

In order to fully support a virtual machine, the TMP68010 must protect the supervisor resources from access by user programs. The one supervisor resource that is not fully protected in the TMP68000 is the system byte of the status register. In the TMP68000 and TMP68008, the MOVE from SR instruction allows user programs to test the S bit (in addition to the T bit and interrupt mask) and thus determine that they are running in the user mode. For full virtual machine support, a new operating system must not be aware of the fact that it is running in the user mode and thus should not be allowed to access the S bit. For this reason, the MOVE from SR instruction has been added to allow user program unhindered access to the condition codes. By making the MOVE from SR instruction privileged, when the new operating system attempts to access the S bit, a trap to the governing operating system will occur, and the SR image passed to the new operating system by the governing operating system will have the S bit set.

2. DATA ORGANIZATION AND ADDRESSING CAPABILITIES

2.1 INTRODUCTION

This section describes the data organization and addressing capabilities of the TLCS-68000 architecture.

2.2 OPERAND SIZE

Operand sizes are defined as follows: a byte equals 8 bits, a word equals 16 bits, and a long word equals 32 bits. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. All explicit instructions support byte, word, or long word operands. Implicit instructions support some subset of all three sizes.

2.3 DATA ORGANIZATION IN REGISTERS

The eight data registers support data operands of 1, 8, 16, or 32 bits. The seven address registers together with the active stack pointer support address operands of 32 bits.

2.3.1 Data Registers

Each data register is 32 bits wide. Byte operands occupy the low order 8 bits, word operands the low order 16 bits, and long word operands the entire 32 bits. The least significant bit is addressed as bit zero; the most significant bit is addressed as bit 31.

When a data register is used as either a source or destination operand, only the appropriate low order portion is changed; the remaining high-order portion is neither used nor changed.

2.3.2 Address Registers

Each address register and the stack pointer is 32-bits wide and holds a full 32 bit address. Address registers do not support byte sized operands. Therefore, when an address register is used as a source operand, either the low order word or the entire long word operand is used depending upon the operation size. When an address register is used as the destination operand, the entire register is affected regardless of the operation size. If the operation size is word, any other operands are sign extended to 32 bits before the operation is performed.

2.4 DATA ORGANIZATION IN MEMORY

Bytes are individually addressable with the high order byte having an even address the same as the word as shown in Figure 2.1. The low order byte has an odd address that is one count higher than the word address. Instructions and multibyte data are accessed only on word (even byte) boundaries. If a long word datum is located at address n(n even), then the second word of that datum is located at address n + 2.

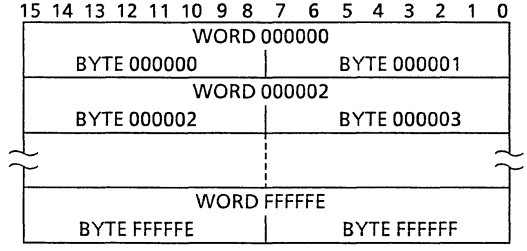


Figure 2.1 Word Organization in Memory

The data types supported by the TLCS-68000 are: bit data, integer data of 8, 16, and 32-bit addresses, and binary coded decimal data. Each of these data types is put in memory as shown in Figure 2.2. The numbers indicate the order in which the data would be accessed from the processor. For convenience, the organization of data in memory for the TMP68008 is shown in Figure 2-3. The appearance to the programmer, however, is identical to the TMP68000, and TMP68010.

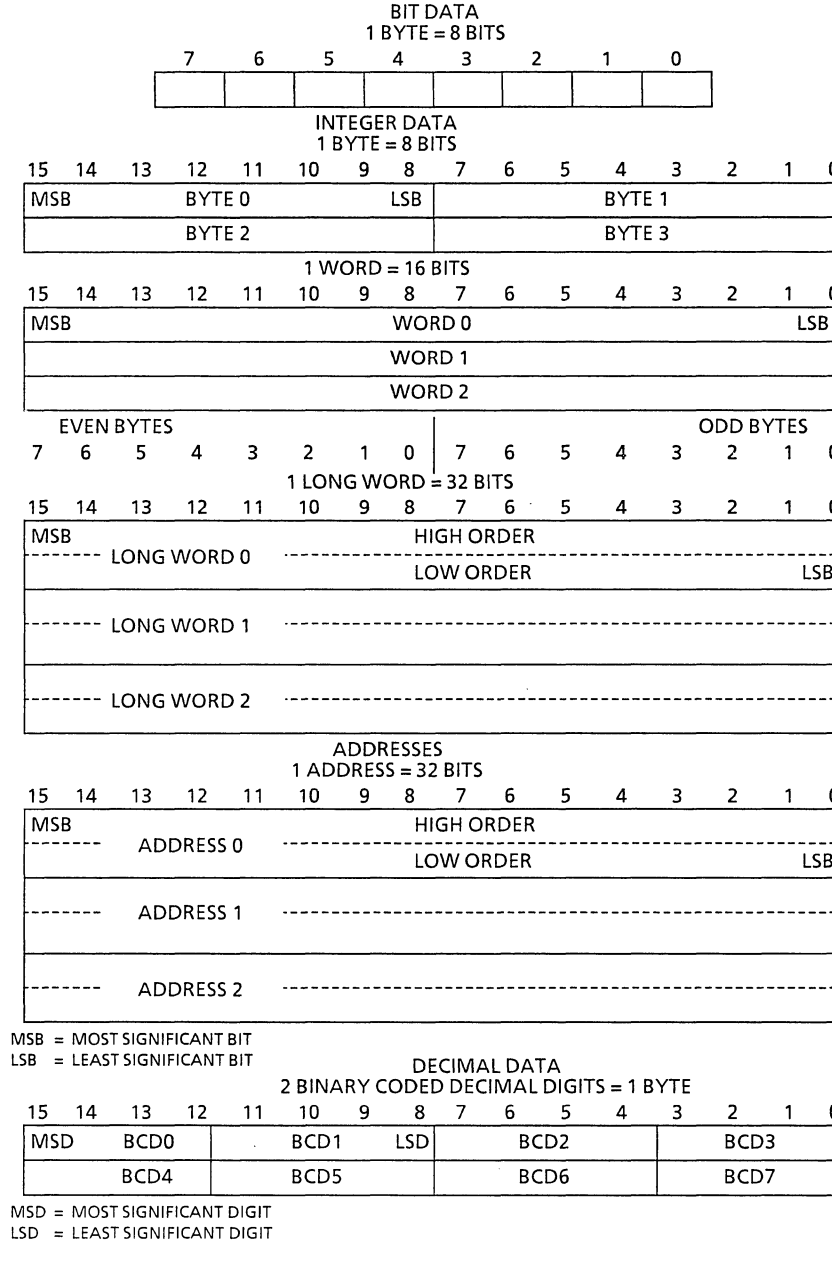


Figure 2.2 Data Organization In Memory

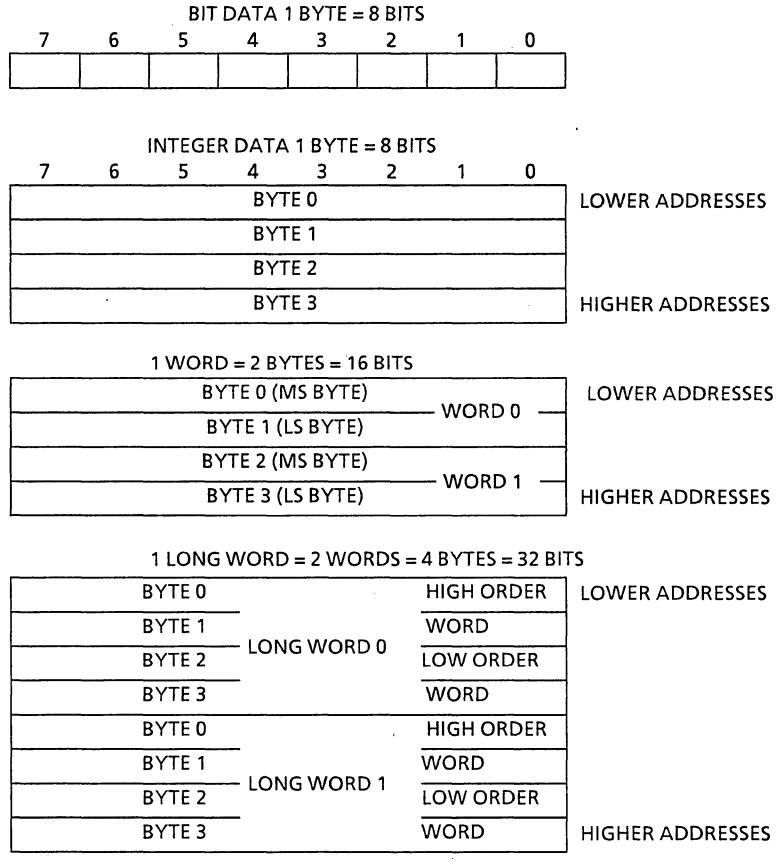


Figure 2.3 Memory Data Organization of the TMP68008

2.5 ADDRESSING

Instructions for the TLCS-68000 contain two kinds of information: the type of function to be performed and the location of the operand(s) on which to perform that function. The methods used to locate (address) the operand(s) are explained in the following paragraphs.

Instructions specify an operand location in one of three ways:

- Register Specification — the number of the register is given in the register field of the instruction.
- Effective Address — use of the different effective address modes.
- Implicit Reference — the definition of certain instructions implies the use of specific registers.

2.6 INSTRUCTION FORMAT

Instruction are from one to five words in length as shown in Figure 2.4. The length of the instruction and the operation to be performed is specified by the first word of the instruction which is called the operation word. The remaining words further specify the operands. These words are either immediate operands or extensions to the effective addres mode specified in the operation word.

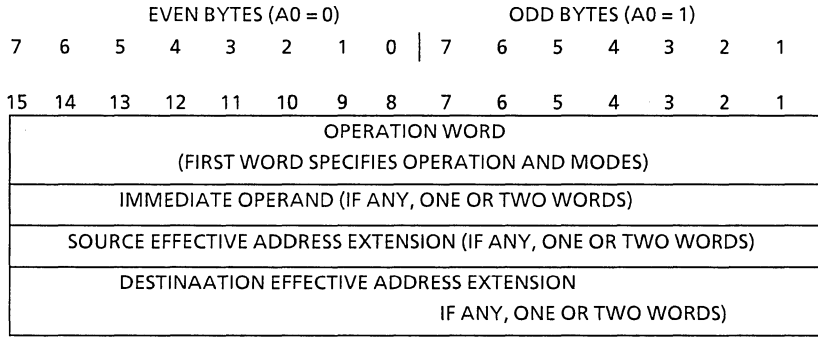


Figure 2.4 Instruction Format

2.7 PROGRAM/DATA REFERENCES

The TLCS-68000 separates memory references into two classes: program references and data references. Program references, as the name implies, are references to that section of memory that contains the program being executed. Data references refer to that section of memory that contains data. Generally, operand reads are from the data space. All operand writes are to the data space.

2.8 REGISTER NOTATION

Appendix B contains a description of each instruction operation and identifies the registers using the following mnemonics:

An – Address Register (n specifies the register number)
Dn – Data Register (n specifies the register number)
Xn – Any Register, Address or Data (n specifies the register number)
PC – Program Counter
SR – Status Register
CCR – Condition Code Half of the Status Register
SP – The Active Stack Pointer (either user or supervisor)
USP – User Stack Pointer
SSP – Supervisor Stack Pointer
d8 – 8-bit Displacement Value
d16 – 16-bit Displacement Value
disp – Displacement Value (d8 or d16)
N – Operand Size in Bytes (1, 2, 4)
SFC, DFC – Source/Destination Function Code Register
VBR – Vector Base Register

2.9 ADDRESS REGISTER INDIRECT NOTATION

When an address register is used to point to a memory location, the addressing mode is called address register indirect. The term indirect is used because the operation of the instruction is not directed to the address itself, but to the memory location pointed to by the address register. The descriptive symbol for the indirect mode is an address register designation in parenthesis, i. e., (An).

2.10 REGISTER SPECIFICATION

The register field within an instruction specifies the register to be used. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used.

2.11 EFFECTIVE ADDRESS

Most instructions specify the location of an operand by using the effective address field in the operation word. For example, Figure 2.5 shows the general format of the single effective address instruction operation word. The effective address is composed of two 3-bit fields: the mode field and the register field. The value in the mode field selects the different address modes. The register field contains the number of a register.

The effective address field may require additional information to fully specify the operand. This additional information, called the effective address extension, is contained in a following word or words and is considered part of the instruction as shown in Figure 2.4. The effective address modes are grouped into three categories: register direct, memory addressing, and special.

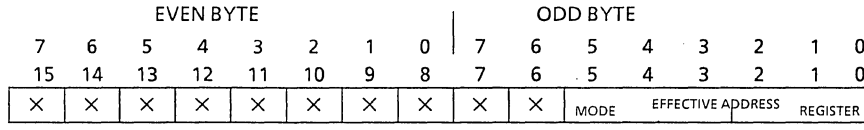


Figure 2.5 Single-Effective-Address-Instruction Operation – General Format

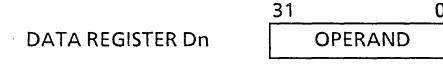
2.11.1 Register Direct Modes

These effective addressing modes specify that the operand is in one of the 16 multifunction registers.

2.11.1.1 Data Register Direct

The operand is in the data register specified by the effective address register field.

GENERATION :	EA = Dn
ASSEMBLER SYNTAX :	Dn
MODE :	000
REGISTER :	n



2.11.1.2 Address Register Direct

The operand is in the address register specified by the effective address register field.

GENERATION :	EA = An
ASSEMBLER SYNTAX :	An
MODE :	001
REGISTER :	n

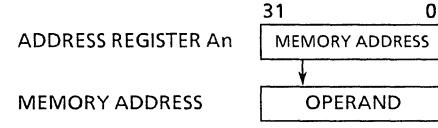
2.11.2 Memory Address Modes

These effective addressing modes specify that the operand is in memory and provide the sepcific address of the operand.

2.11.2.1 Address Register Indirect

The address of the operand is in the address register specified by the register field. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

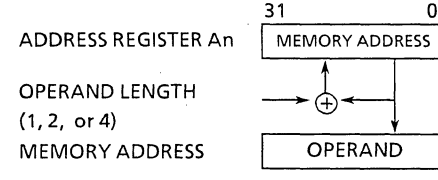
GENERATION :	EA = (An)
ASSEMBLER SYNTAX :	(An)
MODE :	010
REGISTER :	n



2.11.2.2 Address Register Indirect with Postincrement

The address of the operand is in the address register specified by the register field. After the operand address is used, it is incremented by one, two, or four depending upon whether the size of the operand is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is incremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.

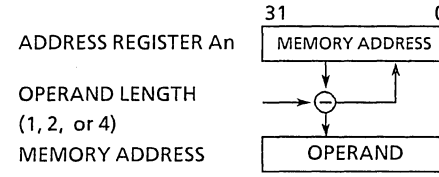
GENERATION :	EA = (An) An = An + N
ASSEMBLER SYNTAX :	(An) +
MODE :	011
REGISTER :	n



2.11.2.3 Address Register Indirect with Predecrement

The address of the operand is in the address register specified by the register field. Before the operand address is used, it is decremented by one, two, or four depending upon whether the operand size is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is decremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.

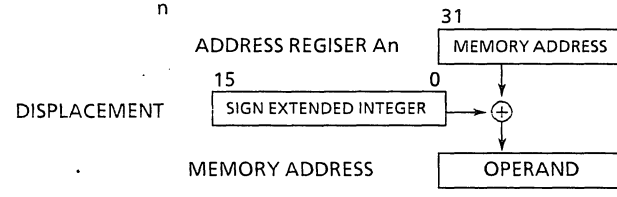
GENERATION :	AA = An – N EA = (An)
ASSEMBLER SYNTAX :	– (An)
MODE :	100
REGISTER :	n



2.11.2.4 Address Register Indirect with Displacement

This address mode requires one word of extension. The address of the operand is the sum of the address in the address register and the sign-extended 16-bit displacement integer in the extension word. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

GENERATION :	En = (An) + d16
ASSEMBLER SYNTAX :	d16 (An)
MODE :	101
REGISTER :	n



2.11.2.5 Address Register Inderect with Index

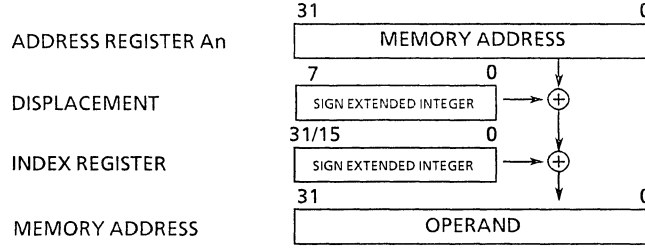
This address mode requires one word of extension formatted as shown below.

EVEN BYTE								ODD BYTE							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	REGISTER			W/L	0	0	0	DISPLACEMENT INTEGER							

Bit 15 – Index register indicator
0 – Data register
1 – Address register
Bit 14~12 – Index register number
Bit 11 – Index size
0 – Sign-extended, low order integer in index register
1 – Long value in index register

The address of the operand is the sum of the address in the address register, the sign-extended displacement integer in the low order eight bits of the extension word, and the contents of the index register. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions. The size of the index register does not affect the execution time of the instructions.

GENERATION :	En = (An) + (Xn) + d8
ASSEMBLER SYNTAX :	d8 (An, Xn.W)
	d8 (An, Xn.L)
MODE :	110
REGISTER :	n



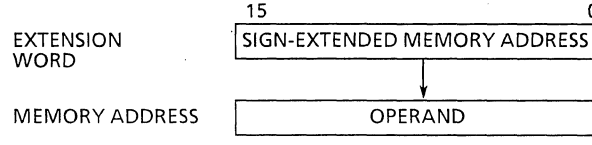
2.11.3 Special Address Modes

The special address modes use the effective address register field to specify the special addressing mode instead of a register number.

2.11.3.1 Absolute Short Address

This address mode requires one word of extension. The address of the operand is in the extension word. The 16-bit address is sign extended before it is used. The reference is classified as a reference with the exception of the jump and jump to subroutine instructions.

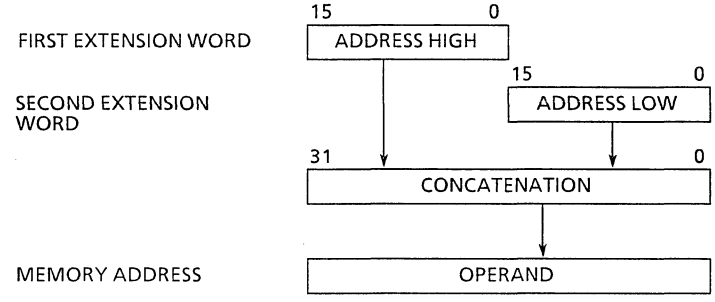
GENERATION :	EA GIVEN
ASSEMBLER SYNTAX :	Abs.W
MODE :	111
REGISTER :	000



2.11.3.2 Absolute Long Address

The address mode requires two words of extension. The address of the operand is developed by the concatenation of the extension words. The high-order part of the address is the first extension word; the low order part of the address is the second extension word. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

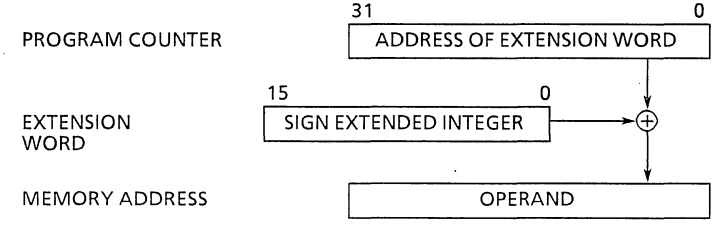
GENERATION : EA GIVEN
ASSEMBLER SYNTAX : Abs.L
MODE : 111
REGISTER : 001



2.11.3.3 Program Counter with Displacement

This address mode requires one word of extension. The address of the operand is the sum of the address in the program counter and the sign-extended 16-bit displacement integer in the extension word. The value in the program counter is the address of the extension word. The reference is classifiedas a program reference.

GENERATION : EA = (PC) + d16
ASSEMBLER SYNTAX : d16 (PC)
MODE : 111
REGISTER : 010



2.11.3.4 Program Counter with Index

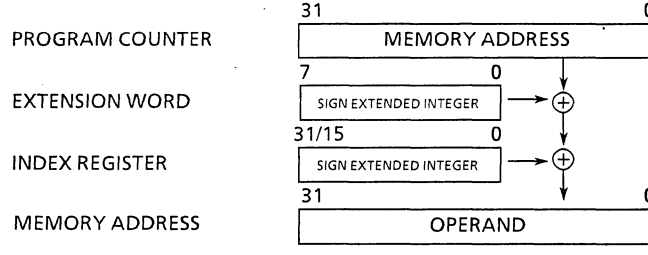
This address mode requires one word of extension formatted as shown below.

	EVEN BYTE								ODD BYTE							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	D/A	REGISTER			W/L	0	0	0	DISPLACEMENT INTEGER							

Bit 15 – Index register indicator
0 – Data register
1 – Address register
Bit 14 through 12 – Index register number
Bit 11 – Index size
0 – Sign-extended, low order integer in index register
1 – Long value in index register

The address is the sum of the address in the program counter, the sign-extended displacement integer in the lower eight bits of the extension word, and the contents of the index register. The value in the program counter is the address of the extension word. This reference is classified as a program reference. The size of the index register does not affect the execution time of the instruction.

GENERATION : En = (PC) + (Xn) + d8
ASSEMBLER SYNTAX : d8 (PC, Xn.W)
d8 (PC, Xn.L)
MODE : 111
REGISTER : 011



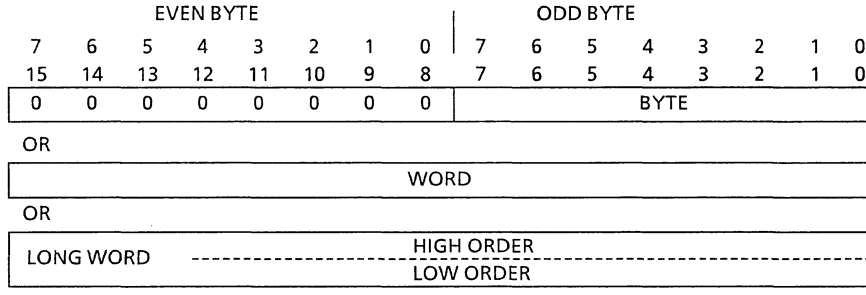
2.11.3.5 Immediate Data

This address mode requires either one or two words of extension depending on the size of the operation.

Byte Operation – operand is low order byte of extension word
Word Operation – operand is extension word
Long Word Operation – operand is in the two extension words, high order 16-bits are in the first extension word, low order 16 bits are in the second extension word.

GENERATION :	OPERAND GIVEN
ASSEMBLER SYNTAX :	#<data>
MODE :	111
REGISTER :	100

The extension word formats are shown below:



2.11.4 Effective Address Encoding Summary

Table 2.1 is a summary of the effective addressing modes discussed in the previous paragraphs.

Table 2.1 Effective Address Encoding Summary

Addressing Mode	Mode	Register
Data Regiser Direct	000	Register Number
Address Register Direct	001	Register Number
Address Register Indirect	010	Register Number
Address Register Indirect with Postincrement	011	Register Number
Address Register Indirect with Predecrement	100	Register Number
Address Register Indirect with Displacement	101	Register Number
Address Register Indirect with Index	110	Register Number
Absolute Short	111	000
Absolute Long	111	001
Program Counter with Displacement	111	010
Program Counter wiht Index	111	011
Immediate	111	100

2.12 IMPLICIT REFERENCE

Some instructions make implicit reference to the program counter (PC), the system stack pointer (SP), the supervisor stack pointer (SSP), the user stack pointer (USP), or the status register (SR). Table 2.2 provides a list of these instructions and the registers implied.

Table 2.2 Implicit Instruction Reference Summary

Instruction	Implied Register(s)
Branch Conditional (Bcc), Branch Always (BRA)	PC
Branch to Subroutine (BSR)	PC, SP
Check Register Against Bounds (CHK)	SSP, SR
Test Condition, Decrement and Branch (DBcc)	PC
Signed Divide (DIVS)	SSP, SR
Unsigned Divide (DIVU)	SSP, SR
Jump (JMP)	PC
Jump to Subroutine (JSR)	PC, SP
Link and Allocate (LINK)	PC, SP
Move Condition Codes (MOVE CCR)	SR
Move Control Register (MOVEC)	VBR, SFC, DFC
Move Alternate Address Space (MOVES)	SFC, DFC
Move Status Register (MOVE SR)	SR
Move User Stack Pointer (MOVE USP)	USP
Push Effective Address (PEA)	SP
Return and Deallocate (RTD)	PC, SP
Return from Exception (RTE)	PC, SP, SR
Return and Restore Condition Codes (RTR)	PC, SP, SR
Return from Subroutine (RTS)	PC, SP
Trap (TRAP)	SSP, SR
Trap on Overflow (TRAPV)	SSP, SR
Unlink (UNLK)	SP
Logical Immediate to CCR	SP
Logical Immediate to SR	SP

2.13 STACK AND QUEUES

In addition to supporting the array data structure with the index addressing mode, the TLCS-68000 also supports stack and queue data structures with the address register indirect postincrement and predecrement addressing modes. A stack is a last-in-first-out (LIFO) list, a queue is a first-in-first-out (FIFO) list. When data is added to a stack or queue, it is "pushed" onto the structure; when it is removed, it is "pulled" from the structure.

The system stack is used implicitly by many instructions; user stacks and queues may be created and maintained through the addressing modes.

2.13.1 System Stack

Address register seven (A7) is the system stack pointer (SP). The system stack pointer is either the supervisor stack pointer (SSP) or the user stack pointer (USP), depending on the state of the S bit in the status register. If the S bit indicates supervisor state, the SSP is the active system stack pointer and the USP cannot be referenced as an address register. If the S bit indicates user state, the USP is the active system stack pointer and the SSP cannot be referenced. Each system stack fills from high memory to low memory. The address mode $-(SP)$ creates a new item on the active system stack and the address mode $(SP)+$ deletes an item from the active system stack.

The program counter is saved on the active system stack on subroutine calls and restored from the active system stack on returns. On the other hand, both the program counter and the status register are saved on the supervisor stack during the processing of traps and interrupts. Thus, the correct execution of the supervisor state code is not dependent on the behavior of user code and user programs may use the user stack pointer arbitrarily.

In order to keep data on the system stack aligned properly, data entry on the stack is restricted so that data is always put in the stack on a word boundary. Thus, byte data is pushed on or pulled from the system stack in the high half of the word; the lower half is unchanged.

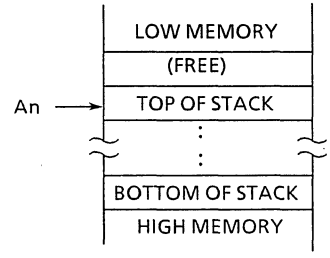
2.13.2 User Stacks

User stacks can be implemented and manipulated by employing the address register indirect with postincrement and predecrement addressing modes. Using an address register (one of A0~A6), the user may implement stacks which are filled either from high memory to low memory, or vice versa. The important things to remember are:

- using predecrement, the register is decremented before its contents are used as the pointer into the stack;
- using postincrement, the register is incremented after its contents are used as the pointer into the stack;
- byte data must be put on the stack in pairs when mixed with word or long data so that the stack will not misaligned when the data is retrieved. Word and long accesses must be on word boundary (even) addresses.

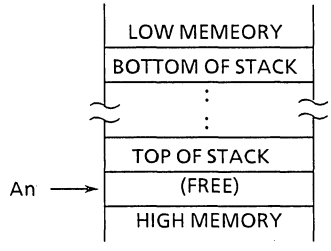
Stack growth from high to low memory is implemented with
−(An) to push data on the stack,
(An)+ to pull data from the stack.

After either a push or a pull operation, register An points to the last (top) item on the stack. This is illustrated as:



Stack growth from low to high memory is implemented with
(An)+ to push data on the stack,
−(An) to pull data from the stack.

After either a push or a pull operation, register An points to the next available space on the stack. This is illustrated as:

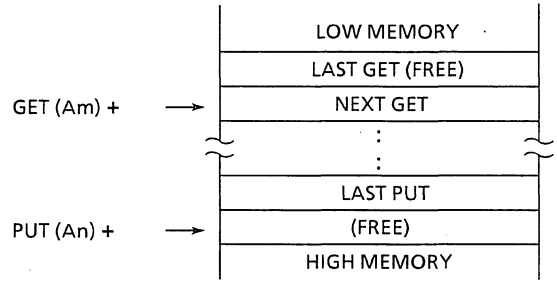


2.13.3 Queues

User queues can be implemented and mainpulated with the address register indirect with postincrement or predecrement addressing modes. Using a pair of address registers (two of A0~A6), the user may implement queues which are filled either from high memory to low memory, or vice versa. Because queues are pushed from one end and pulled from the other, two registers are used: the put and get pointers.

Queue growth from low to high memory is implemented with
(An)+ to put data into the queue,
(Am)+ to get data from the queue

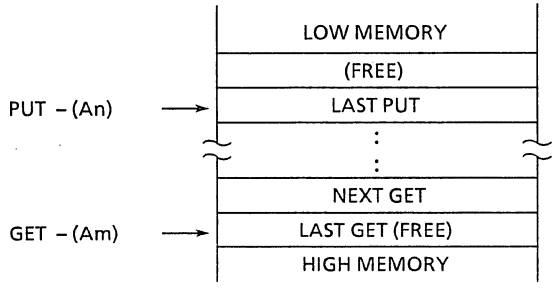
After a put operation, the put address register points to the next available space in the queue and the unchanged get address register points to the next item to remove from the queue. After a get operation, the get address register points to the next item to remove from the queue and the unchanged put address register points to the next available space in the queue. This is illustrated as:



If the queue is to be implemented as a circular buffer, the address register should be checked and, if necessary, adjusted before the put or get operation is performed. The address register is adjusted by subtracting the buffer length (in bytes).

Queue growth from high to low memory is implemented with
−(An) : to put data into the queue,
−(Am) : to get data from the queue

After a put operation, the put address register points to the last item put in the queue and the unchanged get address register points to the last item removed from the queue. After a get operation, the get address register points to the last item removed from the queue and the unchanged put address register points to the last item put in the queue. This is illustrated as:



If the queue is to be implemented as a circular buffer, the get or put operation should be performed first, and then the address register should be checked and, if necessary, adjusted. The address register is adjusted by adding the buffer length (in bytes).

3. INSTRUCTION SET SUMMARY

3.1 INTRODUCTION

This section contains an overview of the TLCS-68000 architecture instruction set. The instructions from a set of tools to perform the following operations:

Data Movement	Bit Field Manipulation
Integer Arithmetic	Binary Coded Decimal Arithmetic
Logical	Program Control
Shift and Rotate	System Control
Bit Manipulation	Multiprocessor Communications

The complete range of instruction capabilities combined with the flexible addressing modes described previously provide a very flexible base for program development. Detailed information about each instruction is given in Appendix B.

Instructions available only on the TMP68010 or which behave differently on the TMP68010 are highlighted.

The following notations will be used throughout this section.

An = any address register, A0-A7
Dn = any data register, D0-D7
Xn = any address or data register
CCR = condition code register (lower byte of status register)
cc = condition codes from CCR
SP = active stack pointer
USP = user stack pointer
SSP = supervisor stack pointer
DFC = destination function code register
SFC = source function code register
Rc = control register (VBR, SFC, DFC)
d8 = 8-bit displacement
d16 = 16-bit displacement
disp = d8 or d16
<ea> = effective address
list = list of registers, e.g., D0-D3
#<data> = immediate data; a literal integer
label = assemby program label
[7] = bit 7 of respective operand
[31:24] = bits 31~ 24 of operand; i.e., high order byte of a register
X = extend (X) bit in CCR
N = negative (N) bit in CCR
Z = zero (Z) bit in CCR

~ = invert; operand is logically complemented
Λ = logical AND
V = logical OR
⊕ = logical exclusive OR

3.2 DATA MOVEMENT OPERATIONS

The basic means of address and data manipulation (transfer and storage) is accomplished by the move (MOVE) instruction and its associated effective addressing modes. Data movement instructions allow byte, word, and long word operands to be transferred from memory to memory, memory to register, register to memory, and register to register. Address movement instructions (MOVE or MOVEA) allow word and long word operand transfers to ensure that only legal address manipulations are executed. In addition to the general MOVE instruction there are several data movement instructions: move multiple registers (MOVEM), move peripheral data (MOVEP), move quick (MOVEQ), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), unlink stack(UNLK). Table 3.1 is a summary of the data movement operations.

Table 3.1 Data Movement Operations

Instruction	Operand Size	Operation
EXG	32	Xn↔Xn
LEA	32	<ea>→An
LINK	—	An→ – (SP) SP→An SP + d16→SP
MOVE MOVEA	8, 16, 32 16, 32	(<ea>)→<ea> (<ea>)→An
MOVEC	32	Xn→Rc Rc→Xn
MOVEM	16, 32	(<ea>)→An, Dn An, Dn→<ea>
MOVES	8, 16, 32	(<ea>)→Xn Xn→<ea>
MOVEP	16, 32	(<ea>)→Dn Dn→<ea>
MOVEQ	8	#<data>→Dn
PEA	32	<ea>→ – (SP)
SWAP	32	Dn[31:16]↔Dn[15:0]
UNLK	—	An→SP (SP) + →An

3.3 INTEGER ARITHMETIC OPERATIONS

The arithmetic operations include the four basic operations of add (ADD), subtract (SUB), multiply (MUL), and divide (DIV) as well as arithmetic compare (CMP, CMPM), clear (CLR), and negate (NEG). The ADD, CMP, and SUB instructions are available for both address and data operations, with data operations accepting all operand sizes. Address operations are limited to legal address size operands (16 or 32 bits). The clear and negate instructions may be used on all sizes of data operands.

The MUL and DIV operations are available for signed and unsigned operands using word multiply to produce a long word product, and a long word dividend with word divisor to produce a word quotient with a word remainder.

Multiprecision and mixed size arithmetic can be accomplished using a set of extended instructions. These instructions are: add extended (ADDX), subtract extended (SUBX), sign extend (EXT), and negate binary with extend (NEGX).

Refer to Table 3.2 for a summary of the integer arithmetic operations.

Table 3.2 Integer Arithmetic Operations

Instruction	Operand Size	Operation
ADD	8, 12, 32	Dn + (<ea>)→Dn
		(<ea>) + Dn→<ea>
		(<ea>) + #<data>→<ea>
ADDA	16, 32	An + (<ea>)→An
ADDX	8, 16, 32	Dx + Dy + X→Dx
	16, 32	- (An) + - (An) + X→(An)
CLR	8, 16, 32	0→<ea>
CMP	8, 16, 32	Dn - (<ea>)
		(EA) - #<data>
		(Ax) + - (Ay) +
CMPA	16, 32	An - (<ea>)
DIVS	32 ÷ 16	Dn ÷ (<ea>)→Dn
DIVU	32 ÷ 16	Dn ÷ (<ea>)→Dn
EXT	8→16	(Dn)8→Dn16
	16→32	(Dn)16→Dn32
MULS	16 × 16→32	Dn × (<ea>)→Dn
MULU	16 × 16→32	Dn × (<ea>)→Dn
NEG	8, 16, 32	0 - (<ea>)→Dn
NEGX	8, 16, 32	0 - (<ea>) - X→<ea>
SUB	8, 16, 32	Dn - (<ea>)→Dn
		(<ea>) - Dn→<ea>
		(<ea>) - #<data>→<ea>
SUBA	16, 32	An - (<ea>)→An
SUBX	8, 16, 32	Dx - Dy - X→Dx
		- (Ax) - - (Ay) - X→(Ax)
TAS	8	(<ea>) - 0, 1→EA[7]
TST	8, 16, 32	(<ea>) - 0

3.4 LOGICAL OPERATIONS

Logical operation instructions AND, OR, EOR, and NOT are available for all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, EOR, and EORI) provide these logical operations with all sizes of immediate data. TST is an arithmetic comparison of the operand with zero which is then reflected in the condition codes. Table 3.3 is a summary of the logical operations.

Table 3.3 Logical Operations

Instruction	Operand Size	Operation
AND	8, 16, 32	Dn∧(<ea>) → Dn (<ea>) ∧Dn → <ea> (<ea>) ∧#<data> → <ea>
OR	8, 16, 32	Dn∨(<ea>) → Dn (<ea>) ∨Dn → <ea> (<ea>) ∨#<data> → <ea>
EOR	8, 16, 32	(<ea>) ⊕Dn → <ea> (<ea>) ⊕#<data> → <ea>
NOT	8, 16, 32	~(<ea>) → <ea>

3.5 SHIFT AND ROTATE OPERATIONS

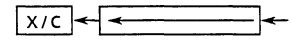
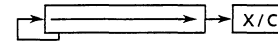
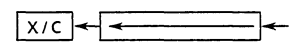
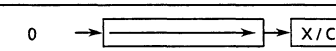
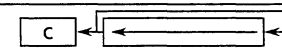
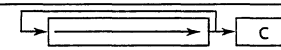
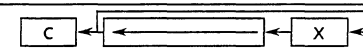
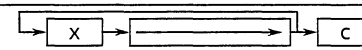
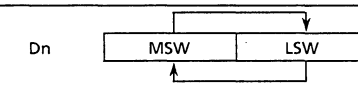
Shift operations in both directions are provided by the arithmetic shift instructions ASR and ASL, and logical shift instruction LSR and LSL. The rotate instructions (with and without extend) available are ROR, ROL, ROXR, and ROXL.

All shift and rotate operations can be performed on either registers or memory.

Register shifts and rotates support all operand sizes and allow a shift count (from one to eight) to be specified in the instruction operation word or a shift count (modulo 64) to be specified in a register.

Memory shifts and rotates are for word operands only and allow only single-bit shifts or rotates. The SWAP instruction exchanges the 16-bit halves of a register. Performance of shift/rotate instructions is enhanced so that use of the ROR or ROL instructions with a shift count of eight allows fast byte swapping.

Table 3.4 Shift and Rotate Operations

Instruction	Operand Size	Operation
ASL	8, 16, 32	
ASR	8, 16, 32	
LSL	8, 16, 32	
LSR	8, 16, 32	
ROL	8, 16, 32	
ROR	8, 16, 32	
ROXL	8, 16, 32	
ROXR	8, 16, 32	
SWAP	32	

3.6 BIT MANIPULATION OPERATIONS

Bit manipulation operations are accomplished using the following instructions: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). All bit manipulation operations can be performed on either registers or memory, with the bit number specified as immediate data or by the contents of a data register. Register operands are always 32-bits, while memory operands are always 8 bits. Table 3.5 is a summary of the bit manipulation operations. (Z is bit 2, the "zero" bit, of the status register.)

Table 3.5 Bit Manipulation Operations

Instruction	Operand Syntax	Operation
BTST	8, 32	~ (<Bit Number>of Destination)→Z
BSET	8, 32	~ (<Bit Number>of Destination) →Z;1→Bit of Destination
BCLR	8, 32	~ (<Bit Number>of Destination) →Z;0→Bit of Destination
BCHG	8, 32	~ (<Bit Number>of Destination) →Z→Bit of Destination

3.7 BINARY CODED DECIMAL OPERATIONS

Multiprecision arithmetic operations on binary coded decimal numbers are accomplished using the following instructions: add decimal with extend (ABCD), subtract decimal with extend (SBCD), and negate decimal with extend (NBCD). Table 3.6 is a summary of the binary coded decimal operations.

Table 3.6 Binary Coded Decimal Operations

Instruction	Operand Size	Operation
ABCD	8	$Dx_{10} + Dy_{10} + X \rightarrow Dx$ $-(Ax)_{10} + -(Ay)_{10} + X \rightarrow (Ax)$
SBCD	8	$Dx_{10} - Dy_{10} - X \rightarrow Dx$ $-(Ax)_{10} - -(Ay)_{10} - X \rightarrow (Ax)$
NBCD	8	$0 - (<ea>)_{10} - X \rightarrow <ea>$

3.8 PROGRAM CONTROL OPERTIONS

Program control operations are accomplished using a set of conditional and unconditional branch instructions and return instructions. These instructions are summarized in Table 3.7.

Table 3.7 Prpogram Control Operations

Instruction		Operation
Conditional		
	Bcc	If Condition True, Then PC + disp→PC
	DBcc	If Condition False, Then Dn – 1→Dn If Dn≠ – 1, Then PC + d16→PC
	Scc	If Condition True, Then 1's→Destination; Else 0's→Destination
Unconditional		
	BRA	PC + disp→PC
	BSR	SP – 4→SP; PC→(SP); PC + disp→PC
	JMP	Destination→PC
	JSR	SP – 4→SP; PC→(SP); Destinaiton→PC
	NOP	PC + 2→PC
Returns		
	RTD	(SP)→PC; SP + 4 + d16→SP
	RTR	(SP)→CCR; SP + 2→SP; (SP)→PC; SP + 4→SP
	RTS	(SP)→PC; PC + 4→SP

The conditional instructions provide testing and branching for the following conditions:

CC – carry clear	LS – low or same
CS – carry set	LT – less than
EQ – equal	MI – minus
F – never true*	NE – not equal
GE – greater or equal	PL – plus
GT – greater than	T – always true*
HI – high	VC – overflow clear
LE – less or equal	VS – overflow set

* : Not available for the Bcc instructions; use BRA for T and NOP for F.

3.9 SYSTEM CONTROL OPERATIONS

System control oprations are accomplished by using privileged instructions, trap generating instructions, and instructions that use or modify the condition code register. These instructions are summarized in Table 3.8. In the TMP68010, the MOVE from SR instruction has been made privileged and the MOVE from CCR has been added. For more detail see "1.4 VIRTUAL MEMORY/MACHINE CONCEPTS".

Table 3.8 System Control Operations

Instruction	Operation
Privilleged	
ANDI to SR	Immediate Data ∧ SR→SR
EORI to SR	Immediate Data⊕ SR→SR
ORI to SR	Immediate Data V SR→SR
MOVE EA to SR	Source→SR
MOVE SR to EA	SR→Destination
MOVE USP	USP→An
	An→USP
MOVEC	Rc→Xn
MOVES	Xn→Rc
	Xn→Destination Using DFC
	Source Using SFC→Xn
RESET	Assert $\overline{\text{RESET}}$ line
RTE	(SP)→SR; SP + 2→SP; (SP)→PC; SP + 4→SP;
	Restore Stack According to Format
STOP	Immediate Data→SR; STOP
Trap Generating	
TRAP	SSP – 2→SSP; Format and Vector OFfset→(SSP); ...
	SSP – 4→SSP; PC→(SSP);SSP – 2→SSP;
	SR→(SSP); Vector Address→PC
TRAPV	If V Then Take Overflow TRAP EXception
CHK	If Dn<0 or Dn>(ea), Then CHK Exception
BKPT	Execute Breakpoint Acknowiedge Bus Cycle; ...
	Trap as illegal Instsruction
ILLEGAL	SSP – 2→SSP; Vector Offset→(SSP); ...
	SSP – 4→SSP; PC→(SSP);
	SSP – 2→SSP; SR→(SSP);
	Illegal Instruction Vector Address→PC
Condition Code Register	
ANDI to CCR	Immediata Data ∧ CCR→CCR
EORI to CCR	Immediate Data⊕ CCR→CCR
ORI to CCR	Immediate Data V SR→SR
MOVE EA to CCR	Source→CCR
MOVE CCR to EA	CCR→Destination

3.10 MULTIPROCESSOR OPERATIONS

Communication between the TLCS-68000 Family of microprocessors is supported by the TAS instruction which executes indivisible read-modify-write bus cycles. See Table 3.9.

Table 3.9 Multiprocessor Operations

Instruction	Operand Size	Operation
TAS	8	Destination – 0; Set Condition Codes; 1→Destination (7)

4. EXCEPTION PROCESSING

4.1 INTRODUCTION

This section describes the actions of the TLCS-68000 which are outside the normal processing associated with the execution of instructions. The functions of the bits in the supervisor portion of the status register are covered: the supervisor/user bit, the trace enable bit, and the processor priority mask. Finally, the sequence of memory references and actions taken by the processor on exception conditions is detailed.

The processor is always in one of three processing states: normal, exception, or halted. The normal processing state is that associated with instruction execution; the memory references are to fetch instructions and operands, and to store results. A special case of the normal state is the stopped state which the processor enters when a STOP instruction is executed. In this state, no further memory references are made.

An additional special case of the normal state exists in the TMP68010, the loop mode, which may be entered when a DBcc instruction is executed. In loop mode, only operand fetches occur. See "APPENDIX G TMP68010 LOOP MODE OPERATION".

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

The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes that the system is unusable and halts. Only an external reset can restart a halted processor. Note that a processor in the stopped state is not in the halted state, nor vice versa.

4.2 PRIVILEGE STATES

The processor operates in one of two states of privilege: the user state or the supervisor state. The privilege state determines which operations are legal, is used by the external memory management device to control and translate accesses, and is used to choose between the supervisor stack pointer and the user stack pointer in instruction references.

The privilege state is a mechanism for providing security in a computer system. Programs should access only their own code and data areas and ought to be restricted from accessing information which they do not need and must not modify.

The privilege mechanism provides security by allowing most programs to execute in user state. In this state, the accesses are controlled and the effects on other parts of the system are limited. The operating system executes in the supervisor state, has access to all resources, and performs the overhead tasks for the user state programs.

4.2.1 Supervisor State

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

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

4.2.2 User State

The user state is the lower state of privilege. For instruction execution, the user state is determined by the S bit of the status register; if the S bit is negated (low), the processor is executing instructions in the user state.

Most instructions execute identically in user state and in the supervisor state. However, some instructions which have important system effects are mode privilege. User programs are not permitted to execute the STOP instruction or the RESET instruction. To ensure that a user program cannot enter the supervisor state except in a controlled manner, the instructions which modify the whole status register are privileged. To aid in debugging programs which are to be used as operating systems, the move to user stack pointer (MOVE to USP) and move from user stack pointer (MOVE from USP) instructions are also privileged.

To implement virtual machine concepts in the TMP68010, the move from status register (MOVE from SR), move to/from control register (MOVEC), and move alternate address space (MOVES) instructions are also privileged.

The bus cycles generated by an instruction executed in user state are classified as user state references. This allows an external memory management device to translate the address and the control access to protected portions of the address space. While the processor is in the user privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly access the user stack pointer.

4.2.3 Privilege State Changes

Once the processor is in the user state and executing instructions, only exception processing can change the privilege state. During exception processing, the current state of the S bit of the status register is saved and the S bit is asserted, putting the processor in the supervisor state. Therefore, when instruction execution resumes at the address specified to process the exception, the processor is in the supervisor privilege state.

The transition from supervisor to user state can be accomplished by any of four instructions: return from exception (RTE), move to status register (MOVE word to SR), AND immediate to status register (ANDI to SR), and exclusive OR immediate to status register (EORI to SR). The RTE instruction fetches the new status register and program counter from the supervisor stack, loads each into its respective register, and then begins the instruction fetch at the new program counter address in the privilege state determined by the S bit of the new contents of the status register. The MOVE, ANDI, and EORI to status register instructions each fetch all operands in the supervisor state, perform the appropriate update to the status register, and then fetch the next instruction at the next sequential program counter address in the privilege state determined by the new S bit.

4.2.4 Reference Classification

When the processor makes a reference, it classifies the kind of reference being made, using the encoding of the three function code output lines. This allows external translation of addresses, control of access, and differentiation of special processor states, such as interrupt acknowledge. Table 4.1 lists the classification of references.

Table 4.1 Reference Classification

Function Code Output			Address Space
FC2	FC1	FC0	
Low	Low	Low	(Undefined, Reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	(Undefined, Reserved)
High	Low	Low	(Undefined, Reserved)
High	Low	High	Supervisor Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

4.3 EXCEPTION PROCESSING

Before discussing the details of interrupts, traps, and tracing, a general description of exception processing is in order. The processing of an exception occurs in four steps, with variations for different exception causes. During the first step, a temporary copy of the status register is made and the status register is set exception processing. In the second step the exception vector is determined and the third step is the saving of the current processor context. In the fourth step a new context is obtained and the processor switches to instruction processing.

4.3.1 Exception Vectors

Exception vectors are memory locations from which the processor fetches the address of a routine which will handle that exception. All exception vectors are two words in length (Figure 4.1) except for the reset vector, which is four words. All exception vectors lie in the supervisor data space except for the reset vector which is in the supervisor program space. A vector number is an 8-bit number which, when multiplied by four, gives the offset of an exception vector. Vector numbers are generated internally or externally, depending on the cause of the exception. In the case of interrupts, during the interrupt acknowledge bus cycle, a peripheral provides an 8-bit vector number (Figure 4.2) to the processor on data bus lines D0~D7.

The processor forms the vector offset by left-shifting the vector number two bit positions and zero-filling the upper order bits to obtain a 32-bit long word vector offset. In the case of the TMP68000 and TMP68008, this offset is used as the absolute address to obtain the exception vector itself. This is shown in Figure 4.3.

In the case of the TMP68010/TMP68012, the vector offset is added to the 32-bit vector base register (VBR) to obtain the 32-bit absolute address of the exception vector. This is shown in Figure 4.4. Since the VBR is set to zero upon reset, the TMP68010, will function identically to the TMP68000 and TMP68008 until the VBR is changed via the MOVEC instruction.

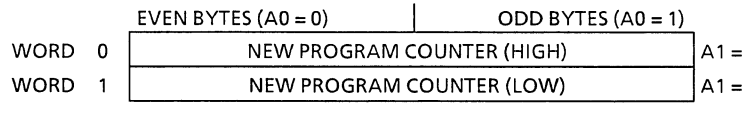
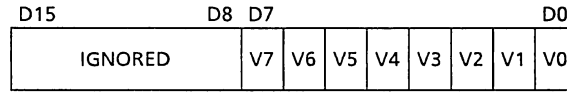


Figure 4.1 Exception Vector Format



where:
V7 is the MSB of the vector number
V0 is the LSB of the vector number

Figure 4.2 Peripheral Vector Number Format

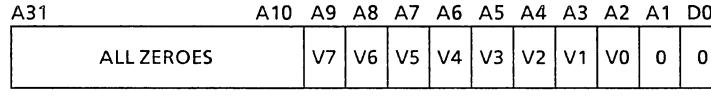


Figure 4.3 Address Translated from 8-Bit Vector Number (TMP68000/TMP68008)

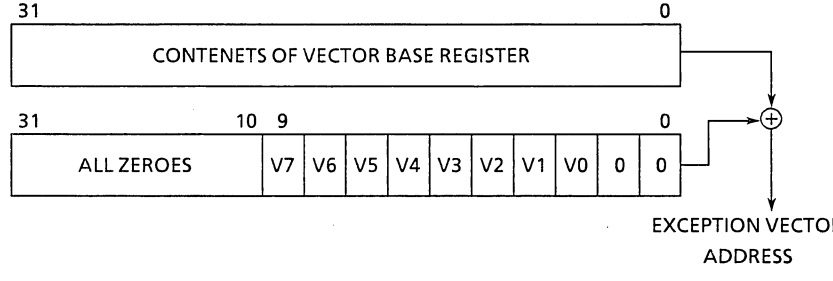


Figure 4.4 Exception Vector Address Calculation (TMP68010)

The actual address output on the address bus is truncated to the number of address bits available on the bus of the particular implementation of the TLCS-68000 architecture. In the case of the TMP68000 and the TMP68010, this is 24 bits. In the case of the TMP68008, the address is 20 bits in length.

The memory map for exception vectors is given in Table 4.2.

Table 4.2 Exception Vector Assignment

Vector Number(s)	Address		Space[6]	Assignment
	Dec	Hex		
0	0	000	SP	Reset: Initial SSP[2]
1	4	004	SP	Reset: Initial PC[2]
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruciton
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12[1]	48	030	SD	(Unassigned, Reserved)
13[1]	52	034	SD	(Unassigned, Reserved)
14	56	038	SD	Foramt Error[5]
15	60	03C	SD	Uninitialized Interrupt Vector
16~23[1]	64	040	SD	(Unassigned, Reserved)
	92	05C		—
24	96	060	SD	Spurious Interrupt[3]
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32~47	128	080	SD	TRAP Instruction Vectors[4]
	188	0BC		—
48~63[1]	192	0C0	SD	(Unassigned, Reserved)
	252	0FC		—
64~255	256	100	SD	User Interrupt Vectors
	1020	3FC		—

Notes:

1. Vector numbers 12, 13, 16~23, and 48~63 are reserved for future enhancements. No user peripheral devices should be assigned these numbers.

2. Reset vector (0) requires four words, unlike the other vectors which only require two words, and is located in the supervisor program space.
3. The spurious interrupt vector is taken when there is a bus error indication during interrupt processing. Refer to Paragraph "4.4.4 Spurious Interrupt".
4. TRAP #n uses vector number 32 + n.
5. TMP68010 only. See Return from Exception Section.
   This vector is unassigned, reserved on the TMP68000 and TMP68008.
6. SP denotes supervisor program space, and SD denotes supervisor data space.

As shown in Table 4.2, the memory layout is 512 words long (1024 bytes). It starts at address 0 (decimal) and proceeds through address 1023 (decimal). This provides 255 unique vectors; some of these are reserved for TRAPS and other system functions. Of the 255, there are 192 reserved for user interrupt vectors. However, there is no protection on the first 64 entries, so user interrupt vectors may overlap at the discretion of the systems designer.

4.3.2 Kinds of Exceptions

Exceptions can be generated by either internal or external causes. The externally generated exceptions are the interrupts, the bus error, and reset requests. The interrupts are requests from peripheral devices for processor action while the bus error and reset inputs are used for access control and processor restart. The internally generated exceptions come from instructions, or from address errors, or tracing. The trap (TRAP), trap on overflow (TRAPV), check register against bounds (CHK), and divide (DIV) instructions all can generate exceptions as part of their instruction execution. In addition, illegal instructions, word fetches from odd addresses, and privilege violations cause exceptions. Tracing behaves like a very high priority, internally generated interrupt after each instruction execution.

4.3.3 Multiple Exceptions

These paragraphs describe the processing which occurs when multiple exceptions arise simultaneously. Exceptions can be grouped accrding to their occurrence and priority. The group 0 exceptions are reset, bus error, and address error. These exceptions cause the instruction currently being executed to be aborted and the exception processing to commence within two clock cycles. The group 1 exceptions are trace and interrupt, as well as the privilege violations and illegal instructions. These exceptions allow the current instruction to execute to completion, but preempt the execution of the next instruction by forcing exception processing to occur (privilege violations and illegal instructions are detected when they are the next instruction to be executed). The group 2 exceptions occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, and zero divide exceptions are in this group. For these exceptions, the normal execution of an instruction may lead to exception processing.

Group 0 exceptions have highest priority, while group 2 exceptions have lowest priority. Within group 0, reset has highest priority, followed by address error and then bus error. Within group 1, trace has priority over external interrupts, which in turn takes priority over illegal instruction and privilege violation. Since only one instruction can be executed at a time, there is no priority relation within group 2.

The priority relation between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. Therefore, if a bus error occurs during a TRAP instruction, the bus error takes precedence, and the TRAP instriuction processing is aborted. In another example, if an interrupt request occurs during the execution of an instruction while the T bit is asserted, the trace exception has priority, and is processed first. Before instruction execution resumes, however, the interrupt exception is also processed and instruction processing commences finally in the interrupt handler routine. A summary of exception grouping and priority is given in Table 4.3.

Table 4.3 Exception Grouping and Priority

Group	Exception	Processing
0	Reset Address Error Bus Error	Exception Processing Begins Within Two Clock Cycles
1	Trace Interrupt Illegal Privilege	Exception Processing Begins Before The Next Instruction
2	TRAP, TRAPV, CHK Zero Divide	Exception Processing Is Started By Normal Instruction Execution

4.3.4 Exception Stack Frames

Exception processing saves the most volatile portion of the current processor context on the top of the supervisor stack. This context is organized in a format called the exception stack frame. Although this information varies with the particular processor and type of exception, it always includes the status register and program counter of the processor when the exception occurred.

The amount and type of information saved on the stack is determined by the processor type and type of execution. Exceptions are grouped by type according to priority of the exception. The group 0 exceptions include address error, bus error, and reset. The group 1 and 2 exceptions include interrupts, traps, illegal instructions, and trace.

The TMP68000 and TMP68008 group 1 and 2 exception stack frame is shown in Figure 4.5. Only the program counter and status register are saved. The program counter points to the next instruction to be executed after exceptions processing.

The TMP68010 exception stack frame is shown in Figure 4.6. The number of words actually stacked depends on the exception type. Group 0 exceptions (except reset) stack 29 words and group 1 and 2 exceptions stack four words. In order to support generic exception handlers, the processor also places the vector offset in the exception stack frame. The format code field allows the RTE (return from exception) instruction to identify what information is on the stack so that it may be properly restored. Table 4.4 lists the TMP68010 stack format codes. Although some formats are peculiar to a particular TLCS-68000 Family processor, the format 0000 is always legal and indicates that just the first four words of the frame are present.

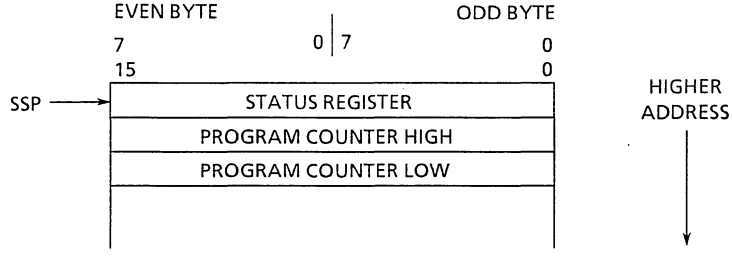


Figure 4.5 TMP68000/TMP68008 Group 1 and 2 Exception Stack Frame

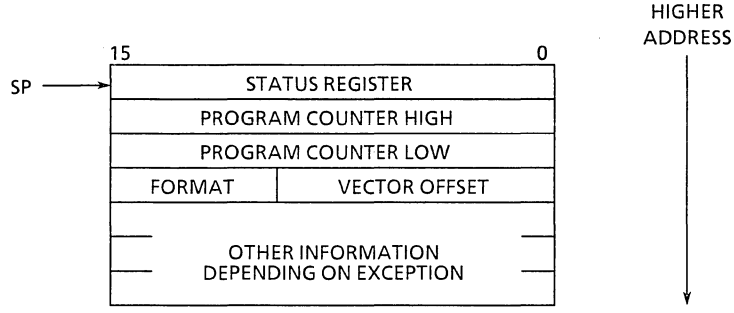


Figure 4.6 TMP68010 Stack Frame

Table 4.4 TMP68010 Format Codes

Format Code	Stacked Insformation
0000	Short Format (4 Words)
1000	Long Format (29 Words)
All Others	Unassigned, Reserved

4.3.5 Exception Processing Sequence

Exception processing occurs in four identifiable steps. In the first step, an internal copy is made of the status register. After the copy is made, the S bit is asserted, putting the processor into the supervisor privilege state. Also, the T bit is negated, which will allow the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated.

In the second step, the vector number of the exception is determined. For interrupts, the vector number is obtained by a processor fetch and classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the address of the exception vector. Group 1 and 2 exceptions use a short format exception stack frame (format = 0000 on the TMP68010). Additional information defining the current context is stacked for the bus error and address error exceptions.

The third step is to save the current processor status, except for the reset exception. The current program counter value and the saved copy of the status register are stacked using the supervisor stack pointer. The program counter value stacked usually points to the next unexecuted instruction, however for bus error and address error, the value stacked for the program counter is unpredictable and may be incremented from the address of the instruction which caused the error. Additional information defining the current context is stacked for the bus error and address error exceptions.

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

4.4 EXCEPTION PROCESSING DETAILED DISCUSSION

Exceptions have a number of sources and each exception has processing which is peculiar to it. The following paragraphs detail the source of exceptions, how each arises, and how each is processed.

4.4.1 Reset

The reset input provides the highest exception level. The processing of the reset signal is designed for system initiation and recovery from catastrophic failure. Any processing in progress at the time of the reset is aborted and cannot be recovered. The processor is forced into the supervisor state and the trace state is forced off. The processor interrupt priority mask is set at level seven. In the TMP6801, the vector base register (VBR) is forced to zero. The vector number is internally generated to reference the reset exception vector at location 0 in the supervisor program space. Because no assumptions can be made about the validity of register contents, in particular the supervisor stack pointer, neither the program counter nor the status register is saved. The address contained in the first two words of the reset exception vector is fetched as the initial supervisor stack pointer and the address in the last two words of the reset exception vector is fetched as the initial program counter. Finally, instruction execution is started at the address in the program counter. The power-up/restart code should be pointed to by the initial program counter.

The RESET instruction does not cause loading of the reset vector, but does assert the reset line to rest external devices. This allows the software to reset the system to a known state and then continue processing with the next instruction.

4.4.2 Interrupts

Seven levels of interrupt priorities are provided. In the TMP68000, and TMP68010 all seven levels are available. The TMP68008 supports three interrupt levels: two, five, and seven, level seven being the highest priority. Devices may be chained externally within interrupt priority levels, allowing an unlimited number of peripheral devices to interrupt the processor. Interrupt priority levels are numbered from one to seven, level seven being the highest priority. The status register contains a three-bit mask which indicates the current processor priority and interrupts are inhibited for all priority levels less than or equal to the current processor priority.

An interrupt request is made to the processor by encoding the interrupt request level on the interrupt request lines; a zero indicates no interrupt request. Interrupt requests arriving at the processor do not force immediate exception processing, but are made pending. Pending interrupts are detected between instruction executions. If the priority of the pending interrupt is lower than or equal to the current processor priority, execution continues with the next instruction and the interrupt exception processing is postponed.

If the priority of the pending interrupt is greater than the current processor priority, the exception processing sequence is started. A copy of the status register is saved, the privilege state is set to supervisor state, tracing is suppressed, and the processor priority level is set to the level of the interrupt being acknowledged. The processor fetches the vector number from the interrupting device, classifying the reference as an interrupt acknowledge and displaying the level number of the interrupt being acknowledged on the address bus. If external logic requests automatic vectoring, the processor internally generates a vector number which is determined by the interrupt level number. If external logic indicates a bus error, the interrupt is taken to be spurious, and the generated vector number references the spurious interrupt vector. The processor then proceeds with the usual exception processing, saving the format/offset word (TMP68010 only), program counter, and status register on the supervisor stack. The offset value in the format/offset word on the TMP68010 is the externally supplied or internally generated vector number multiplied by four. The format will be all zeroes. The saved value of the program counter is the address of the instruction which would have been executed had the interrupt not been present. The content of the interrupt vector whose vector number was previously obtained is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine.

Priority level seven is a special case. Level seven interrupts cannot be inhibited by the interrupt priority mask, thus providing a "non-maskable interrupt" capability. An interrupt is generated each time the interrupt request level changes from some lower level to level seven. Note that a level seven interrupt may still be caused by the level comparison if the request level is a seven and the processor priority is set to a lower level by an instruction.

4.4.3 Uninitialized Interrupt

An interrupting device asserts $\overline{\text{VPA}}$, $\overline{\text{BERR}}$, or provides and TLCS-68000 interrupt vector number and asserts $\overline{\text{DTACK}}$ during an interrupt acknowledge cycle by the TLCS-68000. If the vector register has not been initialized, the responding TLCS-68000 Family peripheral will provide vector number 15, the uninitialized interrupt vector. This provides a uniform way to recover from a programming error.

4.4.4 Spurious Interrupt

If during the interrupt acknowledge cycle no device responds by asserting $\overline{\text{DTACK}}$ or $\overline{\text{VPA}}$, $\overline{\text{BERR}}$ should be asserted to terminate the vector acquisition. The processor separates the processing of this error from bus error by forming a short format exception stack and fetching the spurious interrupt vector instead of the bus error vector. The processor then proceeds with the usual exception processing.

4.4.5 Instruction Traps

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

Exception processing for traps is straightforword. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is internally generated; for the TRAP instruction, part of the vector number comes from the instruction itself. The program counter and the copy of the status register are saved on the supervisor stack. The saved value of the program counter is the address of the instruction after the instruction which generated the trap. Finally, instruction execution commences at the address contained in the exception vector.

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

The signed divide (DIVS) and unsigned divide (DIVU) instructions will force an exception if a division operation is attempted with a divisor of zero.

4.4.6 Illegal and Unimplemented Instructions

Illegal instruction is the term used to refer to any of the word bit patterns which are not the bit patterns of the first word of a legal TLCS-68000 instruction. During instruction execution, if such an instruction is fetched, an illegal instruction exception occurs. Three bit patterns will always force an illegal instruction trap on all TLCS-68000 Family compatible microprocessors. They are: $4AFA, $4AFB, and $4AFC. Two of the patterns, $4AFA and $4AFB, are reserved for the system products. The third pattern, $4AFC, is reserved for customer use.

In addition to the previously defined illegal instruction opcodes, the TMP68010 defines eight breakpoint illegal instructions with the bit patterns $4848~$484F. These instructions cause the processor to enter illegal instruction exception processing as usual, but a breakpoint bus cycle is executed before the stacking operations are performed in which the function code lines (FC0~FC2)are high and address lines are all low. The processor does not accept or send any data during this cycle. Whether the breakpoint cycle is terminated with a $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, or $\overline{\text{VPA}}$ signal, the processor will continue with the illegal instruction processing. The purpose of this cycle is to provide a software breakpoint that will signal external hardware when it is executed. See TMP68010 Advanced Information data sheet.

Word patterns with bits 15~12 equaling 1010 or 1111 are distinguished as unimplemented instructions and separate exception vectors are given to these patterns to permit efficient emulation. This facility allows the operating system to detect program errors or to emulate unimplemented instructions in software.

Exception processing for illegal instructions is similar to that for traps. After the instruction is fetched and decoding is attempted, the processor determines that execution of an illegal instruction is being attempted and starts exception processing. The exception stack frame for group 2 is then pushed on the supervisor stack and the illegal instruction vector is fetched.

4.4.7 Privilege Violations

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

AND Immediate to SR	MOVE USP
EOR Immediate to SR	OR Immediate to SR
MOVE to SR	RESET
MOVE from SR*	RTE
MOVEC*	STOP
MOVES*	

*: TMP68010

Exception processing for privilege violations is nearly identical to that for illegal instructions. After the instruction is fetched and decoded, and the processor determines that a privilege violation is being attempted, the processor starts exception processing. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is generated to reference the privilege violation vector, and the current program counter and the copy of the status register are saved on the supervisor stack and, if the processor is an TMP68010, the format/offset word, is also saved. The saved value of the program counter is the address of the first word of the instruction which caused the privilege violation. Finally, instruction execution commences at the address contained in the privilege violation exception vector.

4.4.8 Tracing

To aid in program development, the TLCS-68000 includes a facility to allow instruction by instruction tracing. In the trace state, after each instruction is executed, an exception is forced, allowing a debugging program to monitor the execution of the program under test.

The trace facility uses the T bit in the supervisor portion of the status register. If the T bit is negated (off), tracing is disable and instruction execution proceeds from instruction to instruction as normal. If the T bit is asserted (on) at the beginning of the execution of an instruction, a trace exception will be generated after the execution of that instruction is completed. If the instruction is not executed, either because an interrupt is taken or the instruction is illegal or privileged, the trace exception does not occur. The trace exception also does not occur if the instruction is aborted by a reset, bus error, or address error exception. If the instruction is indeed executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception, if, during the execution of the instruction, an exception is forced by that instruction, the forced exception is processed before the trace exception. As an extreme illustration of the above rules, consider the arrival of an interrupt during the exception of a TRAP instruction while tracing is enabled. First the trap exception is processed, then the trace exception, and finally the interrupt exception. Instruction execution resumes in the interrupt handler routine.

The exception processing for trace is quite simple. After the execution of the instruction is completed and before the start of the next instruction, exception processing begins. A copy is made of the status register. The transition to supervisor privilege state is made and, as usual, the T bit of the status register is turned off, disabling further tracing. The vector number is generated to reference the trace exception vector, and the current program counter, the copy of the status register and, on the TMP68010, the format/offset word are saved on the supervisor stack. The saved value of the program counter is the address of the next instruction. Instruction execution commences at the address contained in thetrace exception vector.

4.4.9 Bus Error

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

The bus error facility is identical on the TMP68000 and TMP68008; however, the stack frame produced on the TMP68010 contains more information. This is to allow the instruction continuation facility which can be used to implement virtual memory on the TMP68010 processor. Bus error for the TMP68000/TMP68008 and for the TMP68010 are described separately below.

4.4.9.1 Bus Error (TMP68000/TMP68008)

Exception processing for bus error follows the usual sequence of steps. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is generated to refer to the bus error vector. Since the processor was not between instructions when the bus error exception request was made, the context of the processor is more detailed. To save more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are of course saved. The value saved for the program counter is advanced by some amount, two to ten bytes beyond the address of the first word of the instruction which made the reference causing the bus error. If the bus error occurred during the fetch of the next instruction, the saved program counter has a value in the vicinity of the current instruction, even if the current instruction is a branch, a jump, or a return instruction. Besides the usual information, the processor saves its internal copy of the first word of the instruction being processed and the address which was being accessed by the aborted bus cycle. Specific information about the access is also saved: whether it was a read or write, whether the processor was processing an instruction or not, and the classification displayed on the function code outputs when the bus error occurred. The processor is processing an instruction if it is in the normal state or processing a group 2 exception; the processor is not processing an instruction if it is processing a group 0 or a group 1 exception. Figure 4.7 illustrates how this information is organized on the supervisor stack. If a bus error occurs during the last step of exception processing, while either reading the exception vector or fetching the instruction, the value of the program counter is the address of the exception vector. Although this information is not sufficient in general to effect full recovery from the bus error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address contained in the vector. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

If a bus error occurs during the exception processing for a bus error, address error, or read, the processor is halted, and all processing ceases. This simplifies the detection of a catastrophic system failure, since the processor removes itself from the system rather than destroy all memory contents. Only the $\overline{\text{RESET}}$ pin can restart a halted processor.

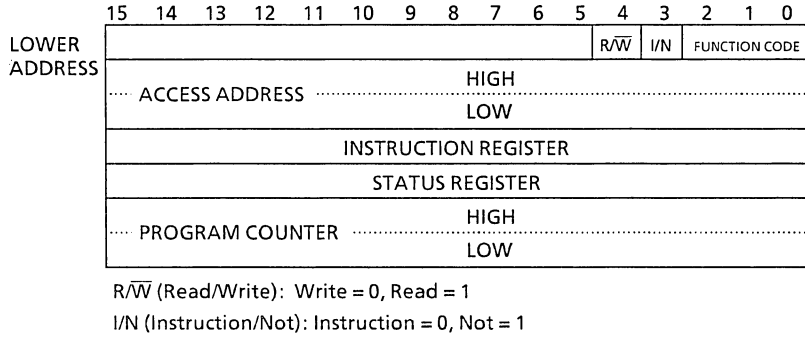
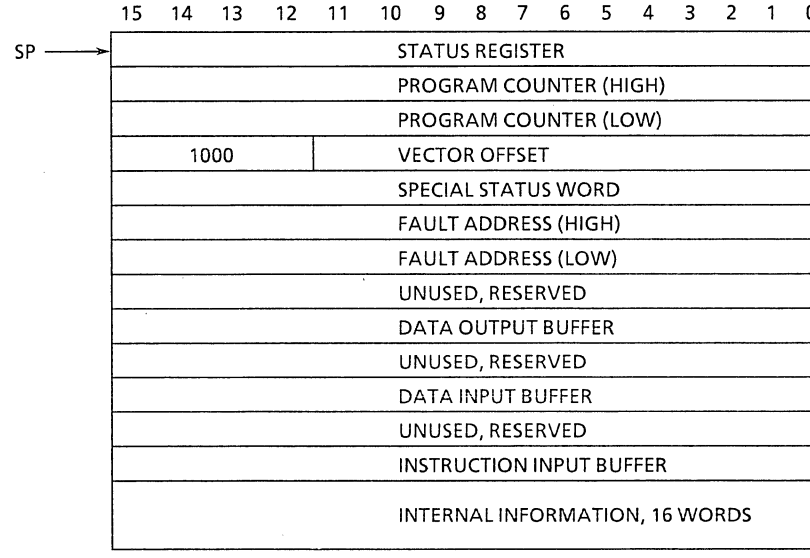


Figure 4.7 Supervisor Stack Order for Bus or Address Error Exception

4.4.9.2 BUS ERROR (TMP68010)

Exception processing for a bus error follows a slightly different sequence than the sequence for group 1 and 2 exceptions. In addition to the four steps executed during exception processing for all other exceptions, 22 words of additional information are placed on the stack. This additional information describes the internal state of the processor at the time of the bus error and is reloaded by the RTE instruction to continue the instruction that caused the error. Figure 4.8 shows the order of the stacked information.



Note: The stack pointer is decremented by 29 words, although only 26 words of information are actually written to memory. The three additional words are reserved for future use.

Figure 4.8 Exception Stack Order (Bus and Address Error)

The value of the saved program counter does not necessarily point to the instruction that was executing when the bus error occurred, but may be advanced by up to five words. This is due to the prefetch mechanism on the TMP68010 that always fetches a new instruction word as each previously fetched instruction word is used. However, enough information is placed on the stack for the bus error exception handler routine to determine why the bus fault occurred. This additional information includes the address that was being accessed, the function codes for the access, whether it was a read or a write, and what internal register was included in the transfer. The fault address can be used by an operating system to determine what virtual memory location is needed so that the requested data can be brought into phyical memory. The RTE instruction is then used to reload the processor's internal state at the time of the fault, the faulted bus cycle will then be re-run and the suspended instruction completed. If the faulted bus cycle was a read-modify-write, the entire cycle will be re-run whether the fault occurred during the read or the write operation.

An alternate method of handling a bus error is to complete the faulted access in software. In order to use this method, use of the special word, the instruction input buffer, the data input buffer, and the data output buffer image is required. The format of the special status word is shown in Figure 4.9. If the bus cycle was a write, the data at the fault address location should be written to the images of the data input buffer, instruction input buffer, or both accoding to the DF and IF bits*. In addition, for read-modify-write cycles, the status register image must be properly set to reflect the read data if the fault occurred during the read portion of the cycle and the write operation (i.e., setting the most significant bit of the memory location) must also be performed. This is because the entire read-modify-write cycle is assumed to have been completed by software. Once the cycle has been completed by software, the RR bit in the special status word is set to indicate to the processor that is should not re-run the cycle when the RTE instruction is executed. If the re-run flag is set when an RTE instruction executes, the TMP68010 still reads all of the information from the stack.

*: If the faulted access was a byte operation, the data should be moved from or to the least-significant byte of the data output or input buffer images unless the HB bit is set. This condition will only occur if a MOVEP instruction caused the fault during transfer of bits 8~15 a word or long word or bits 24~31 of a long word.

15	14	13	12	11	10	9	8	7 — 3	2 1 0
RR	*	IF	DF	RM	HB	BY	RW	*	FC2~FC0

RR — Re-run flag; 0 = processor re-run (default), 1 = softrware re – run.
IF — Instruction fetch to the Instruction Input Buffer.
DF — Data fetch to the Data Input Buffer.
RM — Read-Modify-Write cycle.
HB — High byte transfer from the Data Output Buffer or to the Data Input Buffer.
BY — Byte transfer flag; HB selects the high or low byte of the transfer register. If BY is clear, the transfer is word.
RW — Read/Write flag; 0 = write, 1 = read.
FC — The function code used during the faulted access.
* — These bits are reserved for future use and will be zero when written by the TMP68010.

Figure 4.9 Special Status Word Format

4.4.10 Address Error

Address error exceptions occur when the processor attempts to access a word or a long word operand or an instruction at an odd address. The effect is much like an internally generated bus error, so that the bus cycle is aborted, and the processor ceases whatever processing it is currently doing and begins exception processing. After exception processing commences, the sequence is the same as that for bus error including the information that is stacked, except that the vector number refers to the address error vector instead. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the processor is halted.

On the TMP68010, the address error exception stacks the same information that is stacked by a bus error exception, therefore it is possible to use the RTE instruction to continue execution of the suspended instruction. However, if the software re-run flag is not set, the fault address will be used when the cycle is re-run and another address error exception will occur. Therefore, the user must be certain that the proper corrections have been made to the stack image and user registers before attempting to continue the instruction. With proper software handling, the address error exception handler could emulate word or long word accesses to odd addresses ifdesired.

4.5 RETURN FROM EXCEPTION (TMP68010)

In addition to returning from any exception handler routine on the TMP68010, the RTE instruction is used to resume the excution of a suspended instruction by restoring all of the temporary register and control information stored during a bus error and returning to the normal processing state. For the RTE instruction to execute properly, the stack must contain valid and accessible data. The RTE instruction checks for data validity in two ways; first, by checking the format/offset word for a valid stack format code, and second, if the format code indicates the long stack format, the long stack data is checked for validity as it is loaded into the processor. In addition, the data is checked for accessibility when the processor starts reading the long data. Because of these checks, the RTE instruction executes as follows:

4.5.1 Determine The Stack Format

This step is the same for any stack format and consists of reading the status register, program counter, and format/offset word. If the format code indicates a short stack format, execution continues at the new program counter address. If the format code is not one of the TMP68010 defined stack format codes, exception processing starts for a format error.

4.5.2 Determine Data Validity

For a long stack format, the TMP68010 will begin to read the remaining stack data, checking for validity of the data. The only word checked for validity is the first of the 16 internal information words (SP+26) shown in Figure 4.8. This word contains a processor version number (in bits 10~13) in addition to proprietary internal information that must match the version number of the TMP68010 that is attempting to read the data. This validity check is used to insure that in multiprocessor systems, the data will be properly interpreted by the RTE instruction if the two processors are of different versions. If the version number is incorrect for this processor, the RTE instruction will be aborted and exception processing will begin for a format error exception. Since the stack pointer is not updated until the RTE instruction has successfully read all of the stack data, a format error occurring at this point will not stack new data over the previous bus error stack information.

4.5.3 Determine Data Accessibility

If the long stack data is valid, the TMP68010 performs a read from the last word (SP+56) of the long stack to determine data accessibility. If this read is terminated normally, the processor assumes that the remaining words on the stack frame are also accessible. If a bus error is signaled before or during this read, a bus error exception is taken as usual. After this read, the processor must be able to load the remaining data without receiving a bus error; therefore, if a bus error occurs on any of the remaining stack reads, the TMP68010 treats this as a double bus fault and enters the halted state.

APPENDIX A CONDITION CODES COMPUTATION

A.1 INTRODUCTION

This appendix provides a discussion of how the condition codes were developed, the meanings of each bit, how they are computed, and how they are represented in the instruction set details.

Two criteria were used in developing the condition codes:

- Consistency – across instruction, uses, and instances
- Meaningful Results – no change unless it provides useful information

The consistency across instructions means that instructions which are special cases of more general instructions affect the condition codes in the same way. Consistency across instances means that if an instruction ever affects a condition code, it will always affect that condition code. Consistency across uses means that whether the condition codes were set by a compare, test, or move instruction, the conditional instructions test the same situation. The tests used for the conditional instructions and the code computations are given in paragraph A.5.

A.2 CONDITION CODE REGISTER

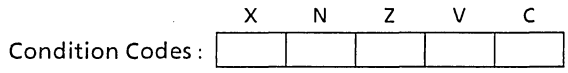
The condition code register portion of the status register contains five bits:

N – Negative
Z – Zero
V – Overflow
C – Carry
X – Extend

The first four bits are true condition code bits in that they reflect the condition of the result of a processor operation. The X bit is an operand for multiprecision computations. The carry bit (C) and the multiprecision operand extend bit (X) are separate in the TLCS–68000 Family to simplify the programming model.

A.3 CONDITION CODE REGISTER NOTATION

In the instruction set details given in "APPENDIX B", the description of the effect on the condition codes is given in the following form:



where:

N (negative)	Set if the most significant bit of the result is set. Cleared otherwise.
Z (zero)	Set if the result equals zero. Cleared otherwise.
V (overflow)	Set if there was an arithmetic overflow. This implies that the result is not representable in the operand size. Cleared otherwise.
C (carry)	Set if a carry is generated out of the most significant bit of the operands for an addition. Also, set if a borrow is generated in a subtraction. Cleared otherwise.
X (extend)	Transparent to data movement. When affected, by arithmetic operations, it is set the same as the C bit.

The convention for the notation that is used in the condition code register representation is:

- * : set according to the result of the operation
- - : not affected by the operation
- 0 : cleared
- 1 : set
- U : undefined after the operation

A.4 CONDITION CODE COMPUTATION

Most operations take a source operand and a destination operand, compute, and store the result in the destination location. Unary operations take a destination operand, compute, and store the result in the destination location. Table A.1 details how each instruction sets the condition codes.

Table A.1 Condition Code Computations

Operations	X	N	Z	V	C	Special Definition
ABCD	*	U	?	U	?	C = Decimal Carry $Z = Z \wedge \overline{Rm} \wedge \cdots \wedge \overline{R0}$
ADD, ADDI, ADDQ	*	*	*	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$
ADDX	*	*	?	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$ $Z = Z \wedge \overline{Rm} \wedge \cdots \wedge \overline{R0}$
AND, ANDI, EOR, EORI, MOVE, MOVEQ, OR, ORI, CLR, NXT, NOT, TAS, TST	–	*	*	0	0	
CHK	–	*	U	U	U	
SUB, SUBI, SUBQ	*	*	*	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
SUBX	*	*	?	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$ $Z = Z \wedge \overline{Rm} \wedge \cdots \wedge \overline{R0}$
CMP, CMPI, CMPM	–	*	*	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
DIVS, DIVU	–	*	*	?	0	V = Division Overflow
MULS, MULU	–	*	*	0	0	
SBCD, NBCD	*	U	?	U	?	C = Decimal Borrow $Z = Z \wedge \overline{Rm} \wedge \cdots \wedge \overline{R0}$
NEG	*	*	*	?	?	$V = Dm \wedge Rm$, $C = Dm \vee Rm$
NEGX	*	*	?	?	?	$V = Dm \wedge Rm$, $C = Dm \vee Rm$ $Z = Z \wedge \overline{Rm} \wedge \cdots \wedge R0$
BTST, BCHG, BSET, BCLR	–	–	?	–	–	$Z = \overline{Dm}$
ASL	*	*	*	?	?	$V = Dm \wedge (\overline{Dm\text{-}1} \vee \cdots \vee \overline{Dm\text{-}r})$ $+ \overline{Dm} \wedge (Dm\text{-}1 \vee \cdots \vee Dm\text{-}r)$ $C = \overline{Dm\text{-}r + 1}$
ASL (r = 0)	–	*	*	0	0	
LSL, ROXL	*	*	*	0	?	C = Dm-r + 1
LSR (r = 0)	–	*	*	0	0	
ROXL (r = 0)	–	*	*	0	?	C = X
ROL	–	*	*	0	?	C = Dm-r + 1
ROL (r = 0)	–	*	*	0	0	
ASR, LSR, ROXR	*	*	*	0	?	C = Dr-1
ASR, LSR (r = 0)	–	*	*	0	0	
ROXR (r = 0)	–	*	*	0	?	C = X
ROR	–	*	*	0	?	C = Dr-1
ROR (r = 0)	–	*	*	0	0	

– = Not Affected
U = Undefined, result meaningless
? = Other – See Special Definition
* = General Case
X = C
N = Rm
Z = $\overline{Rm} \wedge \cdots \wedge \overline{RO}$
Sm = Source Operand – most significant bit
Dm = Destination Operand – most significant bit
Rm = Result Operand –most significant bit
R = Register Tested
n = Bit Number
r = Shift Count
LB = Lower Bound
UB = Upper Bound
$\wedge$ = Boolean AND
$\vee$ = Boolean OR
$\overline{Rm}$ = NOT Rm

A.5 CONDITION TESTS

Table A.2 lists the condition names, encodings, and tests for the condition branch and set instructions.

The test associated with each condition is a logical formula based on the current state of the condition codes. If this formula evaluates to one, the condition succeeds, or is true. If the formula evaluates to zero, the condition is unsuccessful, or false. For example, the T condition always succeeds, while the EQ condition succeeds only if the Z bit is currently set in the condition codes.

Table A.2 Conditional Tests

Mnemonic	Condition	Encoding	Test
T*	True	0000	1
F*	False	0001	0
HI	High	0010	$\overline{C} \cdot \overline{Z}$
LS	Low or Same	0011	$C + Z$
CC (HS)	Carry Clear	0100	$\overline{C}$
CS (LO)	Carry Set	0101	C
NE	Not Equal	0110	$\overline{Z}$
EQ	Equal	0111	Z
VC	Overflow Clear	1000	$\overline{V}$
VS	Overflow Set	1001	V
PL	Plus	1010	$\overline{N}$
MI	Minus	1011	N
GE	Greater or Equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
LT	Less Than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
GT	Greater Than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
LE	Less or Equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

$\cdot$ = Bloolean AND
$+$ = Bloolean OR
$\overline{N}$ = Bloolean NOT N

* : Not available for the Bcc instruction

APPENDIX B INSTRUCTION SET DETAILS

B.1 INTRODUCTION

This appendix contains detailed information about each instruction in the TLCS−68000 instruction set. They are arranged in alphabetical order with the mnemonic heading set in large bold type for easy reference.

B.2 ADDRESSING CATEGORIES

Effective address modes may be categorized by the ways in which they may be used. The following classifications will be used in the instruction definitions.

Data : If an effective address mode may be used to refer to data operands, it is considered a data addressing effective address mode.

Memory : If an effective address mode may be used to refer to memory operands, it is considered a memory addressing effective address mode.

Alterable : If an effective address mode may be used to refer to alterable (writeable) operands, it is considered an alterable addressing effective address mode.

Control : If an effective address mode may be used to refer to memory operands without an associated size, it is considered a control addressing effective address mode.

Table B.1 shows the various categories to which each of the effective address modes belong.

Table B.1 Effective Addressing Mode Categories

Address Modes	Mode	Register	Data	Memory	Control	Alterable	Assembler Syntax
Data Register Direct	000	reg. no.	×	—	—	×	Dn
Address Register Direct	001	reg no	—	—	—	×	An
Address Register Indirect	010	reg. no.	×	×	×	×	(An)
Address Register Indirect with Postincrement	011	reg. no.	×	×	—	×	(An) +
Address Register Indirect with Predecrement	100	reg. no.	×	×	—	×	−(An)
Address Register Indirect with Displacement	101	reg. no.	×	×	×	×	d16 (An)
Address Register Indirect with Index	110	reg. no.	×	×	×	×	d8 (An, Xn)
Absolute Short	111	000	×	×	×	×	Abs. W
Absolute long	111	001	×	×	×	×	Abs. L
Program Counter Indirect with Displacement	111	101	×	×	×	—	d16 (PC)
Program Counter Indirect with Index	111	011	×	×	×	—	d8 (PC, Xn)
Immediate	111	100	×	×	—	—	#<data>

These categories may be combined so that additional, more restrictive, classifications may be defined. For example, the instruction descriptions use such classifications as alterable memory or data alterable. The former refers to those addressing modes which are both alterable and memory addresses, and the latter refers to addressing modes which are both data and alterable.

B.3 INSTRUCTION DESCRIPTION

The formats of each instruction are given in the following pages. Figure B.1 illustrates what information is given.

Instruction Name → ABCD Add Decimal

Operation Description (see "B.4 OPERATION DESCRIPTION DEFINITIONS") → Operation : $Source_{10}$ + $Destinaiton_{10}$ ·····

Assembler Syntax for this Instruction → Assembler Syntax : ABCD Dy, Dx / ABCD −(Ay), −(Ax)

Attributes : Size (Byte)

Text Description of Instruction Operation → Description : Add the source operand ·····

Condition Code Effects (see "Appendix A") → Condition Codes :

X	N	Z	V	C
*	U	*	U	*

N : Undefined.
Z : Cleared if the result is non-zero. Unchanged otherwise.
V : Undefined.
C : set if a carry (decimal) was generated. Cleared otherwise.
X : Sert the same as the carry bit

Instruction Format – Specifies the bit pattern and fields of the operation word and any other words which are part of the instruction. The effective address extensions are not explicitly illustrated. The extensions (if there are any) would follow the illustrated portions of the instructions. For the MOVE instruction, the source effective address extension is the first, followed by the destination effective address extension.

→ Instructuion Format :

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	0	0	Register Xy	1	0	0	0	0	R/M	Register Xx

Meanings and allowed values of the various fields required by the instruction format. → Instructuion Fields :

Figure B.1 Instruction Description Format

B.4 OPERATION DESCRIPTION DEFINITIONS

The following definitions are used for the operation description in the details of the instruction set.

OPERAND :

An	–	address register
Dn	–	data register
Xn	–	any data or address register
PC	–	program counter
SR	–	status register
CCR	–	condition codes (lower order byte of status register)
SSP	–	supervisor stack pointer
USP	–	user stack pointer
SP	–	active stack pointer (equivalent to A7)
X	–	extend operand (from condition codes)
N	–	negative condition code
Z	–	zero condition code
V	–	overflow condition code
C	–	carry condition code
Immediate Data	–	immediate data from the instruction
d8	–	8-bit address displacement
d16	–	16-bit address displacement
disp	–	address displacement (d8 or d16)
Source	–	source contents
Destination	–	destination contents
Vector	–	location of exception vector
ea	–	any valid effective address

SUBFIELDS AND QUALIFIERS :

<bit> of <operand>	selects a single bit of the operand
(<operand>)	the contents of the referenced location
<operand>$_{10}$	the operand is binary coded decimal ; operations are to be performed in decimal.
(<address register>)	the register indirect operator which indicates that the operand register points to the memory locatiaon of the instructuon operand.
–(<address register>)	
(<address register>)+	
#<data>	immediate data located with the instruction is the operand.

OPERATIONS: Operations are grouped into binary, unary, and other.

Binary – These operations are written <operand> <op> <operand> where <op> is one of the following:

→ :	the left operand is moved to the right operand
↔ :	the two operands are exchanged
+ :	the operands are added
− :	the right operand is subtracted from the left operand
* :	the operands are multiplied
/ :	the first operand is divided by the second operand
∧ :	the operands are logically ANDed
∨ :	the operands are logically ORed
⊕ :	the operands are logically exclusively ORed
< :	relational test, true if left operand is less than right operand
> :	relational test, true if left operand is greater than right operand
shifted by	the left operand is sfifted or rotated by the number of positions
rotated by	specified by the right operand

Unary :

~<operand>	the operand is logically complemented
<operand>sign-extended	the operand is sign extended, all bits of the upper portion are made equal to high order bit of the lower portion
<operand>tested	the operand is compared to 0, the results are used to set the condition codes

Other :

TRAP equivalent to SSP−2→SSP; Format/Offset Word→(SSP); SSP−4→SSP; PC→(SSP); SSP−2→SSP; SR→(SSP); (vector)→PC

STOP enter the stopped state, waiting for interrupts

If <condition> then <operation> else <operation>. The condition is tested. If true, the operations after the "then" are performed. If the condition is false and the optional "else" clause is present, the operations after the "else" are performed. If the condition is false and the optional "else" clause is absent, the instruction performs no operation.

; Semicolon is used to separate operations and terminatethe if/then/else operation.

ABCD Add Decimal With Extend ABCD

Operation : $Source_{10} + Destination_{10} + X \rightarrow Destination$

Assembler Syntax : ABCD Dx, Dy
: ABCD −(Ax), −(Ay)

Attributes : Size = (Byte)

Description : Add the source operand to the destination operand along with the extend bit, and store the result in the destination location. The addition is performed using binary coded decimal arithmetic. The operands may be addressed in two different ways:

1. Data register to data register: The operands are contained in the data registers specified in the instruction.
2. Memory to memory: The operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

This operation is a byte operation only.

Condition Codes :

X	N	Z	V	C
*	U	*	U	*

N : Undefined.
Z : Cleared if the result is non-zero. Unchanged otherwise.
V : Undefined.
C : Set if a carry (decimal) was generated. Cleared otherwise.
X : Set the same as the carry bit.

Note:

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

Instruction Format :

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	0	0	Register Xy	1	0	0	0	0	R/M	Register Xx

ABCD ABCD

Instruction Fields:

Register Xy field – Specifies the destination register:
If R/M = 0, specifies a data register
If R/M = 1, specifies an address register for the predecrement addressing mode

R/M field – Specifies the operand addressing mode:
0 – The operation is data register to data register
1 – The operation is memory to memory

Register Xx field – Specifies the source register:
If R/M = 0, specifies a data register
If R/M = 1, specifies an address register for the predecrement addressing mode

ADD Add Binary ADD

Operation : Source + Destination → Destination

Assembler Syntax : ADD <ea>, Dn
: ADD Dn, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Add the source operand to the destination operand using binary addition, and store the result in the destination location. The size of the operation may be specified to be byte, word, or long word. The mode of the instruction indicates which operand is the source and which is the destination as well as the operand size.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a carry is generated. Cleared otherwise.
X : Set the same as the carry bit.

The condition codes are not affected when the destination is an address register.

Instruction Format:

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	1	0	1	Register Dn	Op-Mode	Effective Address Mode	Effective Address Register

Instruction Fields :

Register field – Specifies any of the eight data registers.

Op-Mode field –

Byte	Word	Long word	Operation
000	001	010	(<ea>)+(<Dn>)→<Dn>
100	101	110	(<Dn>)+(<ea>)→<ea>

Effective Address Field – Determines addressing mode:

a. If the location specified in a source operand, the all addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An*	001	reg, number :An
(An)	010	reg, number :An
(An) +	011	reg, number :An
- (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

* : Word and Long word only.

b. If the location specified is a destination operand, then only alterable memory addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	-	-
An	-	-
(An)	010	reg, number :An
(An) +	011	reg, number :An
- (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
D8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	-	-
d8 (PC, Xn)	-	-
#<data>	-	-

Notes: 1. If the destination is a data register, then it cannot be specified by using the destination <ea> mode, but must use the destination Dn mode instead.

2. ADDA is used when the destination is an address register. ADDI and ADDQ are used when the source is immediate data. Most assemblers automatically make this distinction.

ADDA Add Address ADDA

Operation : Source + Destination → Destination

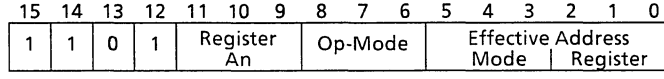
Assembler Syntax : ADDA <ea>, An

Attributes : Size = (Word, Long word)

Description : Add the source operand to the destination address register, and store the result in the address register. The size of the operation may be specified to be word or long word. The entire destination address register is used regardless of the operation size.

Condition Codes : Not affected

Instruction Format:



Instruction Fields :

Register field – Specifies any of the eight address registers. This is always the destination.

Op-Mode field – Specifies the size of the operation:

011 – word operation. The source operand is sign-extended to a operand and the operation is performed on the address register using all 32 bits.

111 – long word operation.

Effective Address field – Specifies the source operand. All addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	001	reg, number :An
(An)	010	reg, number :An
(An) +	011	reg, number :An
- (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

ADDI Add Immediate ADDI

Operation : Immediate Data + Destination → Destination

Assembler Syntax : ADDI #<data>, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Add the immediate data to the destination operand, and store the result in the destination location. The size of the operation may be specified to be byte, word, or long word. The size of the immediate data matches the operation size.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a carry is generated. Cleared otherwise.
X : Set the same as the carry bit.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	Size		Effective Address Mode			Register		
Word Data								Byte Data							
Long Word Data (Includes Previous Word)															

Instruction Fields :

Size field – Specifies the size of the operation:
- 00 – byte operation.
- 01 – word operation.
- 10 – long word operation.

Effective Address field – Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Immediate field – (Data immediately following the instruction):

If size = 00, then the data is the low order byte of the immediate word.

If size = 01, then the data is the entire immediated word.

If size = 10, then the data is the next two immediated words.

ADDQ Add Quick ADDQ

Operation : Immediate Data + Destination → Destination

Assembler Syntax : ADDQ #<data>, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Add the immediate data to the operand at the destination location. The data range is from 1 to 8. The size of the operation may be specified to be byte, word, or long word. Word and long word operations are also allowed on the address registers, in which case the condition codes are not affected. When adding to address registers, the entire destination address register is used, regardless of the operation size.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a carry is generated. Cleared otherwise.
X : Set the same as the carry bit.
The condition codes are not affected if the destination is an address register.

Instruction Format:

15	14	13	12	11 10 9	8	7 6	5 4 3	2 1 0
0	1	0	1	Data	0	Size	Effective Address Mode	Register

Instruction Fields :

Data field – Three bits of immediate data, 0, 1~7 representing a range of 8, 1 to 7 respectively.

Size field – Specifies the size of the operation:

00 – byte operation.
01 – word operation.
10 – long word operation.

ADDQ ADDQ

Effective Address field – Specifies the destination location. Only alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An*	001	reg, number :An
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

* : Word and Long word Only.

ADDX Add Extended ADDX

Operation : Source + Destination + X → Destination

Assembler Syntax : ADDX Dx, Dy
ADDX −(Ax), −(Ay)

Attributes : Size = (Byte, Word, Long word)

Description : Add the source operand to the destination operand along with the extend bit and store the result in the destination location. The operands may be addressed in two different ways:

1. Data register to data register: the operands are contained in data registers specified in the instruction.
2. Memory to memory: the operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

The size of the operation may be specified to be byte, word, or long word.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Cleared if the result is non-zero. Unchanged otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a carry is generated. Cleared otherwise.
X : Set the same as the carry bit.

(NOTE)

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

Instruction Format :

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	0	1	Regisater Xy	1	Size	0	0	R/M	Register Xx

Instruction Fields :

Register Xy field – Specifies the destination register:
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

Size field – Specifies the size of the operation:
00 – byte operation.
01 – word operation.
10 – long word operation.

R/M field – Specifies the operand addressing mode:
0 – The operation is data register to data register.
1 – The operation is memory to memory.

Register Xx field – Specifies the source register:
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

AND — AND Logical — AND

Operation : Source ∧ Destination → Destination

Assembler Syntax : AND <ea>, Dn
: AND Dn, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : AND the source operand to the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long word. The contents of an address register may not be used as an operand.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the most significant bit of the result is set. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Always cleared.
C : Always cleared.
X : Not affected.

Instruction Format:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	1	0	0	Register Dn	0p-Mode	Effective Mode | Register

Instruction Fields :

Register field – Specifies any of the eight data registers.

Op-Mode field –

Byte	Word	Long word	Operation
000	001	010	(<ea>) ∧ (<Dn>) → <Dn>
100	101	110	(<Dn>) ∧ (<ea>) → <ea>

Effective Address field – Determines addressing mode:

If the location specified is a source operand then only data addressing modes are allowed as shown:

AND AND

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

If the location specified is a destination operand then only alterable memory addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Notes: 1. If the destination is a data register, then it cannot be specified by using the destination <ea> mode, but must use the destination Dn mode instead.
2. ANDI is used when the source is immediate data. Most assemblers automatically make this distinction.

ANDI AND Immediate ANDI

Operation : Immediate Data ∧ Destination → Destination

Assembler Syntax : ANDI #<data>, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : AND the immediate data to the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the most significant bit of the result is set. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Always cleared.
C : Always cleared.
X : Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	Size		Effective Address Mode			Register		
Word Data								Byte Data							
Long Word Data (Includes Previous Word)															

Instruction Fields :

Size field – Specifies the size of the operation:
00 – byte operation.
01 – word operation.
10 – long word operation.

Effective Address field – Specifies the destination operand. Only data alterable addressing modes are allowes as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Immediate field – (Data immediately following the instruction):

If size = 00, then the data is the low order byte of the immediate word.

If size = 01, then the data is the entire immediate word.

If size = 10, then the data is the next two immediate words.

ANDI to CCR ANDI to CCR

AND Immediate to Condition Codes

Operation : Source ∧ CCR → CCR

Assembler Syntax : ANDI #<data>, CCR

Attributes : Size = (Byte)

Description : AND the immediate operand with the condition codes and store the result in the low-order byte of the status register.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Cleared if bit 3 of immediate operand is zero. Unchanged otherwise.
Z : Cleared if bit 2 of immediate operand is zero. Unchanged otherwise.
V : Cleared if bit 1 of immediate operand is zero. Unchanged otherwise.
C : Cleared if bit 0 of immediate operand is zero. Unchanged otherwise.
X : Cleared if bit 4 of immediate operand is zero. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	Byte Data (8 Bits)							

ANDI to SR AND Immediate to the Status Register (Privileged Instruction) ANDI to SR

Operation : If supervisor state
then Source ∧ SR → SR
else TRAP;

Assembler Syntax : ANDI #<data>, SR

Attributes : Size = (Word)

Description : AND the immediate operand with the contents of the status register and store the result in the status register. All bits of the status register are affected.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Cleared if bit 3 of immediate operand is zero. Unchanged otherwise.
Z : Cleared if bit 2 of immediate operand is zero. Unchanged otherwise.
V : Cleared if bit 1 of immediate operand is zero. Unchanged otherwise.
C : Cleared if bit 0 of immediate operand is zero. Unchanged otherwise.
X : Cleared if bit 4 of immediate operand is zero. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0
Word Data (16 Bits)															

ASL ASR — Arithmetic Shift — ASL ASR

Operation : Destination Shifted by <count> → Destination

Assemble Syntax :
ASd Dx, Dy
ASd #<data>, Dy
ASd <ea>
where d is direction, L or R

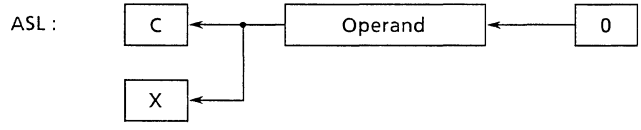
Attributes : Size = (Byte, Word, Long word)

Description : Arithmetically shift the bits of the operand in the direction (L or R) specified. The carry bit receives the last bit shifted out of the operand. The shift count for the shifting of a register may be specified in two different ways:

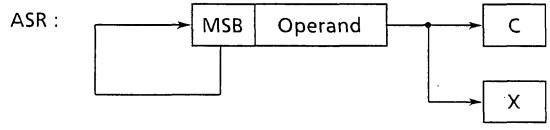
1. Immedicate: the shift count is specified in the instruction (shift range, 1~8).
2. Register: the shift count is contained in a data register specified in the instruction (shift count is modulo 64).

The size of the operation may be specified to be byte, word, or long. The content of memory may be shifted one bit only, and the operand size is restricted to a word.

For ASL, the operand is shifted left; the number of positions shifted is the shift count. Bits shifted out of the high order bit go to both the carry and the extend bits; zeroes are shifted into the low order bit. The overflow bit indicates if any sign changes occur during the shift.



For ASR, the operand is shifted right; the number of positions shifted is the shift count. Bits shifted out of the low order bit go to both the carry and the extend bits; the sign bit (MSB) is replicated into the high order bit.



ASL ASL
ASR ASR

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if the most significant bit of the result is set. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if the most significant bit is changed at any time during the shift operation. Cleared otherwise.
C : Set according to the last bit shifted out of the operand. Cleared for a shift count of zero.
X : Set according to the last bit shifted out of the operand. Unaffected for a shift count of zero.

Instruction Format (Register Shifts) :

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Count Register	dr	Size	i/r	0	0	Register

Instruction Fields (Register Shifts) :

Count/Register field – Specifies shift count or register where count is located:
If i/r = 0, the shift count is specified in this field. The values 0, 1~7 represent a range of 8, 1 to 7 respectively.
If i/r = 1, the shift count (modulo 64) is contained in the data register specified in this field.

dr field – Specifies the direction of the shift:
0 – shift right.
1 – shift left.

Size field – Specifies the size of the operation:
00 – byte operation.
01 – word operation.
10 – long operation.

ASL ASL
ASR ASR

i/r field –
If i/r = 0, specifies immediate shift count.
If i/r = 1, specifies register shift count.

Register field – Specifies a data register whose content is to be shifted.

Instruction Format (Memory Shifts):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	dr	1	1	Effective Address Mode			Register		

Instruction Fields (Memory Shifts):

dr field – Specifies the direction of the shift:
0 – shift right
1 – shfit left

Effective Address field – Specifies the operand to be shifted. Only memory alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Bcc Branch Conditionally Bcc

Operation : If (condition true) then PC + disp → PC;

Assembler Syntax : Bcc <label>

Attributes : Size = (Byte, Word)

Description : If the specified condition is met, program execution continues at location (PC) + displacement. The displacement is a twos complement integer which counts the relative distance in bytes. The value in the PC is the sign-extended instruction location plus two. If the 8-bit displacement in the instruction word is zero, then the 16-bit displacement (word immediately following the instruction) is used. "cc" may specify the following conditions:

CC	carry clear	0100	$\overline{C}$
CS	carry set	0101	C
EQ	equal	0111	Z
GE	greater or equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
GT	greater than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
HI	high	0010	$\overline{C} \cdot \overline{Z}$
LE	less or equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$
LS	low or same	0011	$C + Z$
LT	less than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
MI	minus	1011	N
NE	not equal	0110	$\overline{Z}$
PL	plus	1010	$\overline{N}$
VC	overflow clear	1000	$\overline{V}$
VS	overflow set	1001	V

• : Boolean AND + : Boolean OR $\overline{N}$: Boolean NOT N

Condition Codes : Not affected.

Instruction Format:

15	14	13	12	11 10 9 8	7 6 5 4 3 2 1 0
0	1	1	0	Condition	8-bit Displacement
16-Bit Displacement if 8-Bit Displacement = $00					

Instruction Fields :

Condition field – One of fourteen conditions discussed in description.

8-Bit Displacement field – Twos complement integer specifying the relative distance (in bytes) between the branch instruction and the next instruction to be executed if the condition is met.

16-Bit Displacement field – Allows a larger displacement than 8 bits. Uesd only if the 8-bit displacement is equal to $00.

Note: A short branch to the immediately following instruction cannot be generated, because it would result in a zero offset, which forces a word branch instruction definition.

BCHG Test a Bit and Change BCHG

Operation : ~(<bit number> of Destination) → Z;
~(<bit number> of Destination) → <bit number> of Destination

Assembler Syntax : BCHG Dn, <ea>
BCHG #<data>, <ea>

Attributes : Size = (Byte, Long word)

Description : A bit in the destination operand is tested and the state of the specified bit is reflected in the Z condition code. After the test, the state of the specified bit is changed in the destination. If a data register is the destination, then the bit numbering is modulo 32 allowing bit manipulation on all bits in a data register. If a memory location is the destination, a byte is read from that location, the bit operation is performed using the bit number, modulo 8, and the byte is written back to the location. In all cases, bit zero refers to the least significant bit. The bit number for this operation may be specified in two different ways:

1. Immediate – the bit number is specified in a second word of the instruction.
2. Register – the bit number is contained in a data register specified in the instruction.

Condition Codes :

X	N	Z	V	C
-	-	*	-	-

N : Not affected.
Z : Set if the bit tested is zero. Cleared otherwise.
V : Not affected.
C : Not affected.
X : Not affected.

Instruction Format (Bit Number Dynamic specified by a register) :

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	0	0	Register Dn	1	0	1	Effective Address Mode	Effective Address Register

BCHG BCHG

Instruction Fields (Bit Number Dynamic) :

Register field – Specifies the data register whose content is the bit number.

Effective Address field – Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn *	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

* : Long word only; all others are byte only.

Instruction Format (Bit Number Static, Specified as immedicate data) :

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
0	0	0	0	1	0	0	0	0	1	Effective Address Mode	Register
0	0	0	0	0	0	0	0	Bit Number			

Instruction Fields (Bit Number Static) :

Effective Address field – Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn *	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

* : Long word only; all others are byte only.

Bit Number field – Specifies the bit number.

BCLR Test a Bit and Clear BCLR

Operation : ~(<bit number> of Destination) → Z;
0 → <bit number> of Destination

Assembler Syntax : BCLR Dn, <ea>
: BCLR #<data>, <ea>

Attributes : Size = (Byte, Long word)

Description : A bit in the destination operand is tested and the state of the specified bit is reflected in the Z condition code. After the test, the specified bit is cleared in the destination. If a data register is the destination, then the bit numbering is modulo 32 allowing bit manipulation on all bits in a data register. If a memory location is the destination, a byte is read from that location, the bit operation performed using the bit number, modulo 8, and the byte written back to the location. In all cases, bit zero refers to the least significant bit. The bit number for this operation may be specified in two different ways:

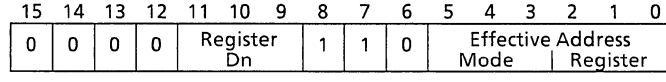
1. Immedicate – the bit number is specified in a second word of the instruction.
2. Register – the bit number is contained in a data register specified in the instruction.

Condition Codes :

X	N	Z	V	C
-	-	*	-	-

N : Not affected.
Z : Set if the bit tested is zero. Cleared otherwise.
V : Not affected.
C : Not affected.
X : Not affected.

Instruction Format (Bit Number Dynamic, specified in a register) :



Instruction Fields (Bit Number Dynamic) :

Register field – Specifies the data register whose content is the bit number.

Effective Address field — Specifies the destination location.
Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn *	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

* : Long only; all others are byte only.

Instruction Format (Bit Number Static, specified as immediate data):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	Effective Address Mode			Register		
0	0	0	0	0	0	0	0	Bit Number							

Instruction Fields (Bit Number Static) :
Effective Address field — Specifies the destination location.
Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn *	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

* : Long only; all others are byte only.

Bit Number field — Specifies the bit number.

BKPT Breakpoint BKPT

Operation : Execute breakpoint acknowledge bus cycle;
Trap as illegal instruction

Assembler Syntax : BKPT #<data>

Attributes : Unsized

Description : This instruction is used to support the program breakpoint function for debug monitors and real-time hardware emulators, and the operation will be dependent on the implementation. Execution of this instruction will cause the TMP68010 to run a breakpoint acknowledge bus cycle (all function codes driven high) and zeros on all address lines.

Whether the breakpoint acknowledge bus cycle is terminated with $\overline{\text{DTACK}}$, BERR, or $\overline{\text{VPA}}$, the processor always takes an illegal instruction exception. During exception processing, a debug monitor can distinguish eight different software breakpoints by decoding the field in the BKPT instruction.

For the TMP68000 and TMP68008, this instruction causes an illegal instruction exception but does not run the breakpoint acknowledge bus cycle.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields :

BKPT # = Immediate data (value = 0~7), encodes 8 software breakpoints.

BRA Branch Always BRA

Operation : PC + disp → PC

Assembler Syntax : BRA <label>

Attributes : Size = (Byte, Word)

Description : Program execution continues at location (PC) + displacement. The displacement is a twos complement integer, which counts the relative distance in bytes. The value in the PC is the instruction location plus two. If the 8-bit displacement in the instruction word is zero, then the 16-bit displacement (word immediately following the instruction) is used.

Condition Codes : Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
0	1	1	0	0	0	0	0	8-Bit Displacement
16-Bit Displacement if 8-Bit Displacement = $00								

Instruction Fields :

8-Bit Displacement field – Two complement integer specifying the relative distance (in bytes) between the branch instruction and the next instruction to be executed

16-Bit Displacement field – Allows a larger displacementthan 8 bits. Used only if the 8-bit displacement is equal to $00.

Note: A short branch to the immediately following instruction cannot be generated because it would result in a zero offset, which forces a word branch instruction definition.

BSET Test a Bit and Set BSET

operation : ~(<bit number> of Destination) → Z;
1 → <bit number> of Destination

Assembler Syntax : BSET Dn, <ea>
: BSET #<data>, <ea>

Attributes : Size = (Byte, Long word)

Description : A bit in the destination operand is tested, and the state of the specified bit is reflected in the Z condition code. After the test, the specified bit is set in the destination. If a data register is the destination, then the bit numbering is modulo 32, allowing bit manipulation on all bits in a data register. If a memory location is the destination, a byte is read from that location, the bit operation performed using the bit number, modulo 8, and the byte written back to the location. Bit zero refers to the least significant bit. The bit number for this operation may be specified in two different ways:

1. Immediate – the bit number is specified in a second word of the instruction.
2. Register – the bit number is contained in a data register specified in the instruction.

Condition Codes :

X	N	Z	V	C
-	-	*	-	-

N : Not affected.
Z : Set if the bit tested is zero. Cleared otherwise.
V : Not affected.
C : Not affected.
X : Not affected.

Instruction Format (Bit Number Dynamic, specified in a register) :

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

Instruction Fields (Bit Number Dynamic) :

Register field – Specifies the data register whose content is the bit number.
Effective Address field – Specifies the destination location. Only data alterable addressing modes are allowed as shown:

BSET BSET

Addr. Mode	Mode	Register
Dn *	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

* : Long word only; all others are byte only.

Instruction Format (Bit Number Static, specified as immediate data) :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	1	Effective Address Mode			Register		
0	0	0	0	0	0	0	0	Bit Number							

Instruction Fields (Bit Number Static) :

Bit Number field – Specifies the bit number.

Effective Address field – Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn *	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

* : Long word only; all others are byte only.

BSR Branch to subroutine BSR

Operation : SP − 4 → SP; PC → (SP); PC + disp → PC

Assembler Syntax : BSR <label>

Attributes : Size = (Byte, Word)

Description : The long word address of the instruction immediately following the BSR instruction is pushed onto the system stack. Program execution then continues at location (PC) + displacement. The displacement in a twos complement integer which counts the relative distances in the bytes. The value in the PC is the instruction location plus two. If the 8-bit displacement in the instruction word is zero, then the 16-bit displacement (word immediately following the instruction) is used.

Condition Codes : Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8-Bit Displacement							
16-Bit displacement if 8-Bit Displacement = $00															

Instruction Fields :

8-Bit Displacement field − Twos complement integer specifying the relative distance (in bytes) between the branchinstruction and the next instruction to be executed.

16-Bit Displacement field − Allows a larger displacement than 8 bits. Used only if the 8-bit displacement is equal to $00.

Note: A short subroutine branch to the immediately following instruction cannot be generated because it would result in a zero offset, which forces a word branch instruction definition.

BTST Test a Bit BTST

Operation : ~(< bit number > of Destination) → Z;

Assembler Syntax : BTST Dn, <ea>
: BTST #<data>, <ea>

Attributes : Size = (Byte, Long word)

Description : A bit in the destination operand is tested, and the state of the specified bit is reflected in the Z condition code. If a data register is the destination, then the bit numbering is modulo 32, allowing bit manipulation on all bits in a data register. If a memory location is the destination, a byte is read from that location, and the bit operation performed using the bit number, modulo 8, with zero referring to the least significant bit. The bit number for this operation may be specified in two different ways:

1. Immediate – the bit number is specified in a second word of the instruction.
2. Register – the bit number is contained in a data register specified in theinstruction.

Condition Codes :

X	N	Z	V	C
-	-	*	-	-

N : Not affected.
Z : Set if the bit tested is zero. Cleared otherwise.
V : Not affected.
C : Not affected.
X : Not affected.

Instruction Format (Bit Number Dynamic, specified in a register) :

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	0	0	Register Dn	1	0	0	Effective Address Mode	Register

Instruction Fields (Bit Number Dynamic) :

Register field – Specifies the data register whose content is the bit number.

Effective Address field – Specifies the destination location. Only data addressing modes are allowed as shown:

BTST BTST

Addr. Mode	Mode	Register
Dn *	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

* : Long word only; all others are byte only.

Instruction Format (Bit Number Static, specified as immediate data) :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	Effective Address Mode			Register		
0	0	0	0	0	0	0	0	Bit Number							

Instruction Fields (Bit Number Static) :

Bit Number field —Specifies the bit number.

Effective Address field —Specifies the destination location. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn *	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	–	–

* : Long word only; all others are byte only.

CHK Check Register Against Bounds CHK

Operation : If Dn < 0 or Dn > Source then TRAP;

Assembler Syntax : CHK <ea>, Dn

Attributes : Size = (Word)

Description : The content of the low order word in the data register specified in the instruction is examined and compared to the upper bound. The upper bound is a twos complement integer. If the register value is less than zero or greater than the upper bound, then the processor initiates exception processing. The vector number is generated to reference the CHK instruction exception vector.

Condition Codes :

X	N	Z	V	C
-	*	U	U	U

N : Set if Dn < 0; cleared if Dn > Source. Undifined otherwise.
Z : Undefined.
V : Undefined.
C : Undefined.
X : Not affected.

Instruction Format:

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	1	0	0	Register Dn	1	1	0	Effective Address Mode	Register

Instruction Fields :

Register field — Specifies the data register whose content is checked.
Effective Address field — Specifies the upper bound operand word.
Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	-	-
(An)	010	reg, number :An
(An) +	011	reg, number :An
- (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

CLR Clear an Operand CLR

Operation : 0→Destination

Assembler Syntax : CLR <ea>

Attributes : Size = (Byte, Word, Long word)

Description : The destination is cleared to all zero. The size of the operation may be specified to be byte, word, or long word.

Condition Codes :

X	N	Z	V	C
-	0	1	0	0

N : Always cleared.
Z : Always set.
V : Always cleared.
C : Always cleared.
X : Not affected.

Instruction Format:

15	14	13	12	11	10	9	8 7	6 5 4 3 2 1 0
0	1	0	0	0	1	0	Size	Effective Address Mode | Register

Instruction Fields :

Size field – Specifies the size of the operation.
00 – byte operation.
01 – word operation.
10 – long word operation.

Effective Address field – Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Note : A memory destination is read before it is written to.

CMP Compare CLP

Operation : Destination − Source

Assembler Syntax : CMP <ea>, Dn

Attributes : Size = (Byte, Word, Long word)

Description : Subtract the source operand from the specified data register and set the condition codes according to the result; the data register is not changed. The size of the operation may be byte, word, or long word.

Condition Codes :

X	N	Z	V	C
-	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a borrow is generated. Cleared otherwise.
X : Not affected.

Instruction Format:

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	1	1	Register Dn	Op-Mode	Effective Address Mode	Register

Instruction Fields :

Register field − Specifies the destination data register.

Op-Mode field −

Byte	Word	Long word	Operation
000	001	010	Dn - (<ea>)

Effective Address field − Specifies the source operand. All addressing modes are allowed as shown:

CMP CMP

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An *	001	reg, number :An
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

* : Word and Long word only.

Note: CMPA is used when the destination is an address register. CMPI is used when the source is immediate data. CMPM is used for memory to memory compares. Most assemblers automatically make this distinction.

CMPA Compare Address CMPA

Operation : Destination − Source

Assembler Syntax : CMPA <ea>, An

Attributes : Size = (Word, Long word)

Description : Subtract the source operand from the destination address register and set the condition codes according to the result; the address register is not changed. The size of the operation may be specified to be word or long word. Word length source operands are sign extended to 32-bit quantities before the operation is done.

Condition Codes :

X	N	Z	V	C
-	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a borrow is generated. Cleared otherwise.
X : Not affected.

Instruction Format:

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	1	1	Register An	Op-Mode	Effective Address Mode	Register

Instruction Fields :

Register field − Specifies the destination data register.

Op-Mode field −Specifies the size of the operation:

011 − word operation. The source operand is sign-extened to a long operand and the operation is performed on the address register using all 32 bits.

111 − long operation.

Effective Address field −Specifies the source operand. All addressing modes are allowed as shown:

CMPA CMPA

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	001	reg, number :An
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

CMPI Compare Immediate CMPI

Operation : Destination − Immediate Data

Assembler Syntax : CMPI #<data>, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Subtract the immediate data from the destination operand and set the condition codes according to the result; the destination location is not changed. The size of the operation may be specified to be byte, word, or long word. The size of the immediate data matches the operation size.

Condition Codes :

X	N	Z	V	C
-	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a borrow is generated. Cleared otherwise.
X : Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	Size		Effective Address Mode			Register		
Word Data								Byte Data							
Long word Data															

Instruction Fields :

Size field − Specifies the size of the operation:
00 − byte operation.
01 − word operation.
10 − long word operation.

Effective Address field − Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

CMPI CMPI

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Immediate field – (Data immediately following the instruction) :

If size = 00, then the data is the low order byte of the immediate word.

If size = 01, then the data is the entire immeidate word.

If size = 10, then the data is the next two immediate words.

CMPM Compare Memory CMPM

Operation : Destination − Source

Assembler Syntax : CMPM (Ax)+,(Ay)+

Attributes : Size = (Byte, Word, Long word)

Description : Subtract the source operand form the destination operand, and set the condition codes according to the results; the destination location is not changed. The operands are always addressed with the postincrement addressing mode, using the address registers specified in the instruction. The size of the operation may be specified to be byte, word, or long word.

Condition Codes :

X	N	Z	V	C
-	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a borrow is generated. Cleared otherwise.
X : Not affected.

Instruction Format:

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

Instruction Fields :

Register Ay field −(always the destination) Specifies an address register for the postincrement addressing mode.

Size field −Specifies the size of the operation:

00 − byte operation.
01 − word operation.
10 − long word operation.

Register Ax field −(always the source) Specifies an address register for the postincrement addressing mode.

DBcc Test Condition, Decrement, and Branch DBcc

Operation : If condition false then (Dn −1 → Dn; if Dn ≠ −1 then PC + disp → PC):

Assembler Syntax : DBcc Dn, <label>

Attributes : Size = (Word)

Description : This instruction is a looping primitive of three parameters: a condition, a counter (data register), and a displacement. The instruction first tests the condition to determine if the termination condition for the loop has been met, and if so, no operation is performed. If the termination condition is not true, the low order 16-bits of the counter data register are decremented by one. If the result is −1, the counter is exhausted and execution continues with the next instruction. If the result is not equal to −1, execution continues at the location indicated by the current value of the PC plus the sign-extended 16-bit displacement. The value in the PC is the current instruction location plus two.

"cc" may specify the following conditions:

CC	carry clear	0100	$\overline{C}$
CS	carry set	0101	C
EQ	equal	0111	Z
F	never true	0001	0
GE	greater or equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
GT	greater than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
HI	high	0010	$\overline{C} \cdot \overline{Z}$
LE	less or equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$
LS	low or same	0011	C + Z
LT	less than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
MI	minus	1011	N
NE	not equal	0110	$\overline{Z}$
PL	plus	1010	$\overline{N}$
T	always true	0000	1
VC	overflow clear	1000	$\overline{V}$
VS	overflow set	1001	V

• = Boolean AND + = Boolean OR $\overline{N}$ = Boolean NOT N

DBcc DBcc

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields :

Condition field – One of the sixteen conditions discussed in discription.
Register field – Specifies the data register which is the counter.
Displacement field – Specifies the distance of the branch (in bytes).

Notes:

1. The terminating condition is like that definedby the UNTIL loop constructs of high-level languages. For example: DBMI can be stated as "decrement and branch until minus".
2. Most assemblers accept DBRA for DBF for use when no condition is required for termination of a loop.
3. There are two basic ways of entering a loop: at the beginning or by branching to the trailing DBcc instruction. If a loop structure terminated with DBcc is entered at the beginning, the control index count must be one less than the number of loop executions desired. This count is useful for indexed addressing modes and dynamically specified bit operations. However, when entering a loop by branching directly to the trailing DBcc instruction, the control index should equal the loop execution count. In this case, if a zero count occurs, the DBcc instruction will not branch, causing a complete bypass of the main loop.

DIVS Signed Divide DIVS

Operation : Destination/Source → Destination

Assembler Syntax : DIVS <ea>, Dn 32/16→16r:16g

Attributes : Size = (Word)

Description : Divide the destination operand by the source and store the result in the destination. The operation is performed using signed arithmetic.
The destination operand is a long word and the source operand is a word. The result is 32-bits, such that the quotient is in the lower word (least significant 16 bits)of the destination and the remainder is in the upper word (most significant 16 bits) of the destination. Note that the sign of the remainder is the same as the sign of the dividend.
Two special conditions may arise during the operation:

1. Division by zero causes a trap.
2. Overflow may be detected and set before completion of the instruction. If overflow is detected, the condition is flagged but the operands are unaffected.

Condition Codes :

X	N	Z	V	C
-	*	*	*	0

N : Set if the quotient is negative. Cleared otherwise. Undefined if overflow or divide by zero.

Z : Set if the quotient is zero. Cleared otherwise. Undefined if overflow or divide by zero.

V : Set if division overflow is detected. Cleared otherwise.

C : Always cleared.

X : Not affected.

Instruction Format :

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	0	0	0	Register Dn	1	1	1	Effective Address Mode	Register

DIVS DIVS

Instruction Fields :

Register field – Specifies any of the eight data registers. This field always specifies the destination operand.

Effective Address field – Specifies the source operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

Note: Overflow occurs if the quotient is larger than a 16-bit signed integer.

DIVU Unsigned Divide DIVU

Operation : Destination/Source → Destination

Assembler Syntax : DIVU <ea>, Dn 32/16 → 16r:16q

Attributes : Size = (Word)

Description : Divide the destination operand by the source and store the result in the destination. The operation is performed using unsigned arithmetic.
The destination operand is a long word and the source operand is a word. The result is 32-bits, such that the quotient is in the lower word (least significant 16-bits) of the destination and the remainder is in the upper word (most significant 16 bits) of the destination. Note that the sign of the remainder is the same as the sign of the dividend.
Two special conditions may arise during the operation:

1. Division by zero causes a trap.
2. Overflow may be detected and set before completion of the instruction. If overflow is detected, the condition is flagged but the operands are unaffected.

Condition Codes :

X	N	Z	V	C
-	*	*	*	0

N : Set if the quotient is negative. Cleared otherwise. Undefined if overflow or divide by zero.

Z : Set if the quotient is zero. Cleared otherwise. Undefined if overflow or divideby zero.

V : Set if division overflow is detected. Cleared otherwise.

C : Always cleared.

X : Not affected.

Instruction Format:

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	0	0	0	Register Dn	0	1	1	Effective Address Mode	Register

DIVU DIVU

Instruction Fields :

Register field – Specifies any of the eight data registers. This field always spesifies the destination operand.

Effective Address field – Specifies the source operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

Note: Overflow occurs if the quotient is larger than a 16-bit unsigned integer.

EOR Exclusive OR Logical EOR

Operation : Source ⊕ Destination → Destination

Assembler Syntax : EOR Dn, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Exclusive OR the source operand to the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long word. This operation is restricted to data registers as the source operand. The destination operand is specified in the effective address field.

Condition Codes :

X	N	Z	V	C
-	*	*	*	0

N : Set if the most significant bit of the result is set. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Always cleared.
C : Always cleared.
X : Not affected.

Instruction Format (word form) :

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	1	1	Register Dn	Op-Mode	Effective Address Mode	Effective Address Register

Instruction Fields :

Register field – Specifies any of the eight data registers.

Op-Mode field –

Byte	Word	Long word	Operation
100	101	110	<ea> ⊕ <Dx> → <ea>

Effective Address field – Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

EOR EOR

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

Note: Memory to data register operations are not allowed. EORI is used when the source is immediate data. Most assemblers automatically make this distinction.

EORI Exclusive OR Immediate EORI

Operation : Immediate Data ⊕ Destination → Destination

Assembler Syntax : EORI #<data>, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Exclusive OR the immediate data to the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long word. The immediate data matches the operation size.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the most significant bit of the result is set. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Always cleared.
C : Always cleared.
X : Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	Size		Effective Address Mode			Register		
Word Data (16 Bits)								Byte Data (8Bits)							
Long word Data (32 Bits, including Previous Word)															

Instruction Fields :

Size field – Specifies the size of the operation:
00 – byte operation.
01 – word operation.
10 – long word operation.

Effective Address field – Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Immediate field – (Data immediately following the instruction):

If size = 00, then the data is the low order byte of the immediate word.

If size = 01, then the data is the entire immediate word.

If size = 10, then the data is next two immediate words.

EORI to CCR — Exclusive OR Immediate to Condition Code — EORIto CCR

Operation : Source ⊕ CCR → CCR

Assembler Syntax : EORI #<data>, CCR

Attributes : Size = (Byte)

Description : Exclusive OR the immediate operand with the condition codes and store the result in the low-order byte of the status register.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Changed if bit 3 of immediate operand is one. Unchanged otherwise.
Z : Changed if bit 2 of immediate operand is one. Unchanged otherwise.
V : Changed if bit 1 of immediate operand is one. Unchanged otherwise.
C : Changed if bit 0 of immediate operand is one. Unchanged otherwise.
X : Changed if bit 4 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	Byte Data (8 Bits)							

EORI to SR — Exclusive OR Immediate to the Status Register (Privileged Instruction) — EORI to SR

Operation : If supervisor state
then Source ⊕ SR → SR
else TRAP;

Assembler Syntax : EORI #<data>, SR

Attributes : Size = (Word)

Description : Exclusive OR the immediate operand with the contents of the status register and store the result in the status register. All bits of the status register are affected.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Changed if bit 3 of immediate operand is one. Unchanged otherwise.
Z : Changed if bit 2 of immediate operand is one. Unchanged otherwise.
V : Changed if bit 1 of immediate operand is one. Unchanged otherwise.
C : Changed if bit 0 of immediate operand is one. Unchanged otherwise.
X : Changed if bit 4 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	0
Word Data (16 Bits)															

EXG Exchange Register EXG

Operation : Xx ↔ Xy

Assembler Syntax :
EXG Dx, Dy
EXG Ax, Ay
EXG Dx, Ay

Attributes : Size = (Long word)

Description : Exchange the contents of two registers. This exchange is always a long (32-bit) operation. Exchange works in three modes:

1. Exchange data registers.
2. Exchange address registers.
3. Exchange a data register and an address register.

Condition Codes : Not affected.

Instruction Format:

15	14	13	12	11 10 9	8	7 6 5 4 3	2 1 0
1	1	0	0	Register Xx	1	Op-Mode	Register Xy

Instruction Fields :

Register Xx field – Specifies either a data register or an address register depending on the mode. If the exchange is between data and addrress registers, this field always specifies the data register.

Op-Mode field – Specifies whether exchanging:

01000 – data registers.
01001 – address registers.
10001 – data register and address register.

Register Xy field – Specifies either a data register or an address register depending on the mode. If the exchange is between data and address registers, this field always specifies the address register.

EXT Sign Extend EXT

Operation : Destination Sign-extended → Destination

Assembler Syntax : EXT Dn

Attributes : Size = (Word, Long word)

Description : Extend the sign bit of a data register from a byte to a word, or from a word to a long word, depending on the size selected. If the operation is word, bit [7] of the designated data register is copied to bits [15:8] of that data register. If the operation is long, bit [15] of the designated data register is copied to bits [31:16] of the data register.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Always cleared.
C : Always cleared.
X : Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	Op-Mode			0	0	0	Register Dn		

Instruction Fields :

Op-Mode field – Specifies the size of the sign-extension operation:
010 – Sign-extend low order byte of data register to word.
011 – Sign-extend low order word of data register to long word.

Register field – Specifies the data register whose content is to be sign-extended.

ILLEGAL — Take Illegal Instruction Trap — ILLEGAL

Operation : SSP − 2 → SSP; Vector Offset → (SSP);
SSP − 4 → SSP; PC → (SSP);
SSP − 2 → SSP; SR → (SSP);
Illegal Instruction Vector Address → PC

Assembler Syntax : ILLEGAL

Attributes : Unsized

Description : This bit pattern causes an illegal instruction exception. All other illegal instruction bit patterns are reserved for future exception of the instruction set.
The TMP68010 will first write the exception vector offset and format code to the system stack followed by the PC and SR to complete a 4-word exception stack frame.

Condition Codes : Not affected.

Instruction Format:

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

Operation : Destination Address → PC

Assembler Syntax : JMP <ea>

Attributes : Unsized

Description : Program execution continues at the effective address specified by the instruction. The address is specified by the control addressing modes.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields :

Effective Address field – Specifies the address of the next instruction. Only control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg, number :An
(An) +	–	–
– (An)	–	–
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	–	–

JSR Jump to Subroutine JSR

Operation : SP −4 → SP; PC → (SP);
Destination Address → PC

Assembler
Syntax : JSR <ea>

Attributes : Unsized

Description : The long word address of the instruction immediately following the JSR instruction is pushed onto the system stack. Program execution then continues at the address specified in the instruction.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields :

Effective Address field − Specifies the address of the next instruction. Only control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	−	−
An	−	−
(An)	010	reg, number : An
(An) +	−	−
− (An)	−	−
d16 (An)	101	reg, number : An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number : An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	−	−

LEA Load Effective Address LEA

Operation : <ea> → An

Assembler Syntax : LEA <ea>, An

Attributes : Size = (Long word)

Description : The effective address is loaded into the specified address register. All 32 bits of the address register are affected by this instruction.

Condition Codes : Not affected.

Instruction Format:

15	14	13	12	11 10 9	8	7	6	5 4 3 2 1 0
0	1	0	0	Register An	1	1	1	Effective Address Mode | Register

Instruction Fields :

Register field – Specifies the address register which is to be loaded with the effective address.

Effective Address field – Specifies the address to be loaded into the address register. Only control addressingmodes are allowed as shown:

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg, number :An
(An) +	–	–
– (An)	–	–
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	–	–

LINK Link and Allocate LINK

Operation : SP − 4 → SR; An → (SR);
SP → An; SP + disp → SP

Assembler
Syntax : LINK An, #<displacement>

Attributes : Size = Unsized

Description : The current content of the specified address register is pushed onto the stack. After the push, the address register is loaded from the updated stack pointer. Finally, the 16-bit sign-extended displacement operand is added to the stack pointer. The content of the address register occupies one long word on the stack. A negative displacement is specified to allocate stack area.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields :

Register field − Specifies the address register through which the link is to be constructed.

Displacement field − Specifies the twos complement integer which is to be added to the stack pointer.

Note: LINK and UNLK can be used to maintain a linked list of local data and parameter areas on the stack for nested subroutine calls.

LSL LSR — Logical Shift — LSL LSR

Operation : Destination Shifted by <count> → Destination

Assembler Syntax :
LSd Dx, Dy
LSd #<data>, Dy
LSd <ea>
where d is direction, L or R

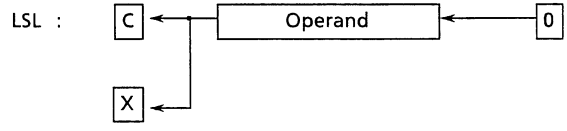
Attributes : Size = (Byte, Word, Long word)

Description : Shift the bits of the operand in the direction (L or R) specified. The carry bit receives the last bit shifted out of the operand. The shift count for the shifting of a register may be specified in two different ways:

1. Immediate – the shift count is specified in the instruction (shift range 1~8).
2. Registe – the shift count is contained in a data register specified in the instruction (shift count modulo 64).

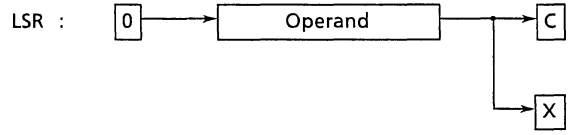
The size of the operation may be specified to be byte, word, or long word. The content of memory may be shifted one bit only, and the operand size is restricted to a word.

For LSL, the operand is shifted left; the number of positions shifted is the shift count. Bits shifted out of the high order bit go to both the carry and the extend bits; zeroes are shifted into the low order bit.



For LSR, the operand is shifted right; the number of positions shifted is the shift count. Bits shifted out of the low order bit go to both the carry and the extend bits; zeroes are shifted into the high order bit.

LSL LSL
LSR LSR



Condition Codes :

X	N	Z	V	C
*	*	*	0	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Always cleared.
C : Set according to the last bit shifted out of the operand. Cleared for a shift count of zero.
X : Set according to the last bit shifted out of the operand. Unaffected for a shift count of zero.

Instruction Format (Register Shifts) :

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Count/ Register	dr	Size	i/r	0	1	Register

Instruction Field (Register Shifts) :

Count/Register field –
If i/r = 0, he shift count is specified in this field. The values 0, 1~7 represent a range of 8, 1 to 7 respectively.
If i/r = 1, the shift count (modulo 64) is contained in the data register specified in this field.

dr field – Specifies the direction of the shift:
0 – shift right.
1 – shift left.

Size field – Specifies the size of the operation:
00 – byte operation.
01 – word operation.
10 – long word operation

i/r field –
If i/r = 0, Specifies immediate shift count.
If i/r = 1, Specifies register shift count.

Register field – Specifies a data register whose content is to be shifted.

LSL LSL
LSR LSR

Instruction Format (Memory Shifts) :

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
1	1	1	0	0	0	1	dr	1	1	Effective Address Mode	Register

instruction Fields (Memory Shifts) :

dr field – Specifies the direction of the shift:

0 – shift right.

1 – shift left.

Effective Address field – Specifies the operand to be shifted. Only memory alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

MOVE Move Data from Source to Destination MOVE

Operation : Source→Destination

Assembler Syntax : MOVE <ea>, <ea>

Attributes : Size = (Byte, Word, Long word)

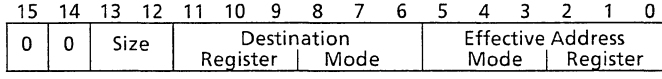
Description : Move the content of the source to the destination location. The data is examined as it is moved, and the condition codes set accordingly. The size of the operation may be specified to be byte, word, or long word.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Always cleared.
C : Always cleared.
X : Not affected.

Instruction Format:



Instruction Fields :

Size field – Specifies the size of the operand to be moved:
01 – byte operation.
11 – word operation.
10 – long word operation.

Destination Effective Address field – Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Source Effective Address field – Specifies the source operand. All addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An*	001	reg, number :An
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

* : For byte size operation, address register direct is not allowed.

Notes: 1. MOVEA is used when the destination is an address register. Most assemblers automatically make this distinction.

2. MOVEQ can also be used for certain operations on data registers.

MOVE from CCR — Move from the Condition Code Register — MOVE from CCR

Operation : CCR→Destination

Assembler Syntax : MOVE CCR, <ea>

Attributes : Size = (Word)

Description : The content of the status register is moved to the destination location. The source operand is a word, but only the low order byte contains the conditioncodes. The upper byte is all zeroes.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields :

Effective Address field – Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Notes: MOVE from CCR is a word operation. ANDI, ORI, and EORI to CCR are byte operations.

MOVE to CCR Move to the Condition Code Register MOVE to CCR

Operation : Source→CCR

Assembler Syntax : MOVE <ea>, CCR

Attributes : Size = (Word)

Description : The content of the source operand is moved to the condition codes. The source operand is a word, but only the low order byte is used to update the condition codes. The upper byte is ignored.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set the same as bit 3 of the source operand.
Z : Set the same as bit 2 of the source operand.
V : Set the same as bit 1 of the source operand.
C : Set the same as bit 0 of the source operand.
X : Set the same as bit 4 of the source operand.

Instruction Format:

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

Instruction Fields :

Effective Address field – Specifies the location of the source operand. Only data addressing modes are allowed as shown:

MOVE to CCR

MOVE to CCR

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

Notes: MOVE to CCR is a word operation. ANDI, ORI, and EORI to CCR are byte operations.

MOVE to SR — Move to the Status Register (Privileged Instruction) — MOVE to SR

Operation : If supervisor stsate
then Source→SR
else TRAP;

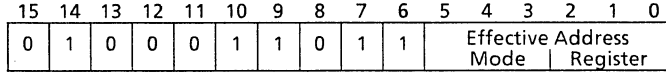
Assembler
Syntax : MOVE <ea>, SR

Attributes : Size = (Word)

Description : The content of the source operrand is moved to the status registe. The sourrce operarnd is a word and all bits of the status register are affected.

Condition Codes : Not affected.

Instruction Format:



Instruction Fields :

Effective Address field – Specifies the locaation of the source operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

MOVE from SR Move from the Status Register MOVE from SR

Operation : SR→Destination

Assembler
Syntax : MOVE SR, <ea>

Attributes : Size = (Word)

Description : The content of the status register is moved to the destination location. The operand size is a word.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields :

Effective Address field – Specifies the destination location. Only data alterrable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg, number :Dn
An	–	–
(An)	010	reg, number :An
(An) +	011	reg, number :An
– (An)	100	reg, number :An
d16 (An)	101	reg, number :An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg, number :An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Note: A memory destination is read beforre it is wrritten to.

Move from SR — Move from the Status Register (Privileged Instruction) — Move from SR

Operation : If supervisor state
then SR → Destination
else TRAP;

Assembler
Syntax : MOVE SR, <ea>

Attributes : Size = (Word)

Description : The content of the status register is moved to the destination location. The operand size is a word.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields :

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Add. Mode	Mode	Register
Dn	000	reg. Number:An
An	–	–
(An)	010	reg. Number:An
(An) +	011	reg. Number:An
– (An)	100	reg. Number:An
d16 (An)	101	reg. Number:An

Add. Mode	Mode	Register
d8 (An, Xn)	110	reg. Number:An
Abs. W	111	000
Abs. L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Note: Use the MOVE from CCR instruction to access only the condition codes.

Move USP — Move User Stack Pointer (Privileged Instruciton) — Move USP

Operation : If supervisor state
then USP → An or An → USP
else TRAP;

Assembler Syntax : MOVE USP, An
: MOVE An, USP

Attributes : Size = (Long word)

Description : The contents of the user stack pointer are transferred to or from the specified address register.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields :

dr field – Specifies the direction of transfer:
0 – transfer the address register to the USP.
1 – transfer the USP to the address register.

Register field – Specifies the address register to or from which the user stack pointer is to be transferred.

MOVEA Move Address MOVEA

Operation : Source→Destination

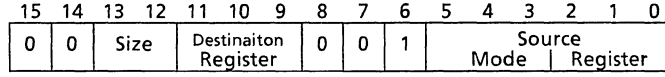
Assembler Syntax : MOVEA <ea>, An

Attributes : Size = (Word, Long word)

Description : Move the content of the source to the destination address register. The size of the operation may be specified to be word or long word. Word size source operands are sign extended to 32-bit quantities before the operation is done.

Condition Codes : Not affected.

Instruction Format:



Instruction Fields :

Size field – Specifies the size of the operand to be moved:

11 – Word operation. The source operand is sign-extended to a long operand and all 32 bits are loaded into the address register.

10 – Long word operation.

Destination Register field – Specifies the destination address register.

Source Effective Address field – Specifies the location of source operand. All addressing modes are allowed as shown:

Add. Mode	Mode	Register
Dn	000	–
An	001	–
(An)	010	reg. Number:An
(An) +	011	reg. Number:An
– (An)	100	reg. Number:An
d16 (An)	101	reg. Number:An

Add. Mode	Mode	Register
d8 (An, Xn)	110	reg. Number:An
Abs. W	111	000
Abs. L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

MOVEC — Move to/from Control Register (Privileged Instruction) — MOVEC

Operation : If supervisor state
then Rc → Xn or Xn → Rc
else TRAP;

Assembler Syntax : MOVEC Rc, Xn
MOVEC Xn, Rc

Attributes : Size = (Long word)

Description : Copy the contents of the specified control register (Rc) to the specified general register or copy the contents of the specified general register to the specified control register. This is always a 32-bit transfer even though the control register may be implemented with fewer bits. Unimplemented bits are read as zeros.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields:

dr field – Specifies the direction of transfer:
0 – control register to general register
1 – general register to control register

A/D field – Specifies the type of general register:
0 – data register
1 – address register

Register field – Specifies the register number.

Control Register field – Specifies the control register.

Hex	Control Register
000	Source Function Code (SFC) register.
001	Destination Function Code (DFC) register.
800	User Stack Pointer (USP).
801	Vector Base Register (VBR).

All other codes cause an illegal instruction exception.

MOVEM Move Multiple Registers MOVEM

Operation : Registers → Destination
Source → Registers

Assembler Syntax : MOVEM register list, <ea>
: MOVEM <ea>, register list

Attributes : Size = (Word, Long word)

Description : Selected registers are transferred to or from consecutive memory locations starting at the location specified by the effective address. A register is transferred if the bit corresponding to that register is set in the mask field. The instruction selects how much of each register is transferred; either the entire long word can be moved or just the low order word. In the case of a word transfer to the registers, each word is sign-extended to 32 bits (including data registers) and the resulting long word loaded into the associated register. MOVEM allows three forms of address modes: the control modes, the predecrement mode, or the postincrement mode. If the effective address is in one of the control modes, the registers are transferred starting at the specified address and up through higher addresses. The order of transfer is from data register 0 to data register 7, then from address register 0 to address register 7.

If the effective address is the predecrement mode, only a register to memory operation is allowed. The registers are stored starting at the specified address minus the operand length (2 or 4) and down through lower addresses. The order of storing is from address register 7 to address register 0, then from data register 7 to data register 0. The decremented address register is updated to contain the address of the last word stored.

If the effective address is the postincrement mode, only a memory to register operation is allowed. The registers are loaded starting at the specified address and up through higher addresses. The order of loading is the same as for the control mode addressing. The incremented address register is updated to contain the address of the last word loaded plus the operand length (2 or 4).

Condition Codes : Not affected.

MOVEM

MOVEM

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	dr	0	0	1	Sz	Effective Address Mode			Register		
Register List Mask															

Instruction Fields :

dr field – Specifies the direction of the transfer:
0 – register to memory.
1 – memory to register.

Sz field – Specifies the size of the registers being transferred:
0 – word transfer.
1 – long word transfer.

Effective Address field – Specifies the memory address to or from which the registers are to be moved. For register to memory transfers, only control alterable addressing modes or the predecrement addressing mode are allowed as shown:

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg. number:An
(An) +	–	–
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8(An, Xn)	110	reg. number:An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

For memory to register transfers, only control addressing modes or the postincrement addressing mode are allowed as shown:

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	–	–
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8(An, Xn)	110	reg. number:An
Abs.W	111	000
Abs.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	–	–

Register List Mask field – Specifies which registers are to be transferred. The low order bit corresponds to the first register to be transferred; the high bit corresponds to the last register to be transferred. Thus, both for control modes and for the postincrement mode addresses, the mask correspondence is

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

while for the predecrement mode addresses, the mask correspondence is

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7

Note :

An extra read bus cycle occurs for memory operands. This accesses an operand at one address higher than the last register image required.

MOVEP Move Peripheral Data MOVEP

Operation : Source → Destination

Assembler Syntax :
```
MOVEP   Dx, d16 (Ay)
MOVEP   d16 (Ay), Dx
```

Attributes : Size = (Word, Long word)

Description : Data is transferred between a data register and alternate bytes of memory, starting at the location specified and incrementing by two. The high order byte of the data register is transferred first and the low order byte is transferred last. The memory address is specified using the address register indirect plus 16-bit displacement addressing mode. This instruction is designed to work with 8-bit peripherals on a 16-bit data bus. If the address is even, all the transfers are made on the high order half of the data bus; if the address is odd, all the transfers are made on the low order half of the data bus. On an 8-bit or 32-bit bus, the instruction still accesses every other byte.

Example : Long transfer to/from an even address.

Byte organization in register

31 24	23 16	15 8	7 0
Hi-Order	Mid-Upper	Mid-Lower	Low-Order

Byte organization in memory (low address at top)

15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
Hi-Order	
Mid-Upper	
Mid-Lower	
Low-Order	

MOVEP

MOVEP

Example : Word transfer to/from an odd address.

Byte organization in register

31 24	23 16	15 8	7 0
		High-Order	Low-Order

Byte organization in memory (low address at top)

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

Condition Codes : Not affected.

Instruction Format:

15	14	13	12	11 10 9	8 7 6	5	4	3	2 1 0
0	0	0	0	Data Register	Op-Mode	0	0	1	Address Register
Displacement									

Instruction Fields : Data Register field – Specifies the data register to or from which the data is to be transferred.

Op-Mode field – Specifies the direction and size of the operation:

100 – transfer word from memory to register.

101 – transfer long from memory to register.

110 – transfer word from register to memory.

111 – transfer long from register to memory.

Address Register field – Specifies the address register which is used in the address register indirect plus displacement addressing mode.

Displacement field – Specifies the displacement which is used in calculating the operand address.

MOVEQ Move Quick MOVEQ

Operation : Immediate Data → Destination

Assembler Syntax : MOVEQ #<data>, Dn

Attributes : Size = (Long word)

Description : Move immediate data to a data register. The data is contained in an 8-bit field within the operation word. The data is sign-extended to a long word operand and all 32 bits are transferred to the data register.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the result is negative. Cleared otherwise.

Z : Set if the result is zero. Cleared otherwise.

V : Always cleared.

C : Always cleared.

X : Not affected.

Instruction Format:

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

Instruction Fields : Register field – Specifies the data register to be loaded.

Data field – 8 bits of data which are sign extended to a long word operand.

MOVES Move Alternate Address Space (Privileged Instruction) MOVES

Operation : If supervisor state
then Xn → Destination [DFC] or Source [SFC] → Xn
else TRAP;

Assembler Syntax : MOVES Xn, <ea>
MOVES <ea>, Xn

Attributes : Size = (Byte, Word, Long word)

Description : Move the byte, word, or long word operand from the specified general register to a location within the address space specified by the destination function code (DFC) register. Or, move the byte, word, or long word operand from a location within the address space specified by the source function code (SFC) register to the specified general register.
If the destination is a data register, the source operand replaces the corresponding low-order bits of that data register. If the destination is an address register, the source operand is sign-extended to 32 bits and then loaded into that address register.

Condition Codes : Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	Size		Effective Address Mode			Register		
A/D	Register			dr	0	0	0	0	0	0	0	0	0	0	0

Instruction Fields : Size field – Specifies the size of the operation:
00 – byte operation.
01 – word operation.
10 – long word operation.
Effective Address field – Specifies the source or destination location within the alternate address space. Only alterable memory addressing modes are allowed as shown:
A/D field – Specifies the type of general register:
0 – data register.
1 – address register.

Register field – Specifies the register number.
dr field – Specifies the direction of the transfer:
0 – from <ea> to general register.
1 – from general register to <ea>.
MOVES.x An, (An)+
or
MOVES.x An, –(An)
where An is the same address register for both source and destination and is an undefined operation. The value stored in memory is undefined.

NOTE

On the TMP68010 implementations, the value stored is the incremented or the decremented value of An. This implementation may not appear on future devices.

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

MULS Signed Multiply MULS

Operation : Source *Destination → Destination

Assembler Syntax : MULS <ea>, Dn 16 × 16 → 32

Attributes : Size = (Word)

Description : Multiply two signed operands yielding a signed result. The operation is performed using signed arithmetic.
The multiplier and multiplicand are both word operands and the result is long word operand. A register operand is taken from the low order word, the upper word is unused. All 32 bits of the product are saved in the destination data register.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the result is negative. Cleared otherwise.

Z : Set if the result is zero. Cleared otherwise.

V : Always cleared.

C : Always cleared.

X : Not affected.

Instruction Format:

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	1	0	0	Register Dn	1	1	1	Effective Address Mode	Register

MULS MULS

Instruction Fields : Register field – Specifies one of the data registers. This field always specifies the destination.

Effective Address field – Specifies the source operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

MULU Unsigned Multiply MULU

Operation : Source *Destination → Destination

Assembler Syntax : MULS <ea>, Dn 16 × 16 → 32

Attributes : Size = (Word)

Description : Multiply two unsigned operands yielding a unsigned result. The operation is performed using unsigned arithmetic.
The multiplier and multiplicand are both word operands and the result is a long word operand. A register operand is taken from the low order word, the upper word is unused. All 32 bits of the product are saved in the destination data register.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the result is negative. Cleared otherwise.

Z : Set if the result is zero. Cleared otherwise.

V : Always cleared.

C : Always cleared.

X : Not affected.

Instruction Format:

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	1	0	0	Register Dn	0	1	1	Effective Address Mode	Register

MULU MULU

Instruction Fields : Register field – Specifies one of the data registers. This field always specifies the destination.

Effective Address field – Specifies the source operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

NBCD Negate Decimal with Extend NBCD

Operation : $0 - \text{Destination}_{10} - X \rightarrow \text{Destination}$

Assembler Syntax : NBCD <ea>

Attributes : Size = (Byte)

Description : The operand addressed as the destination and the extend bit are subtracted from zero. The operation is performed using decimal arithmetic. The result is saved in the destination location. This instruction produces the tens complement of the destination if the extend bit is clear, the nines complement if the extend bit is set. This is a byte operation only.

Condition Codes :

X	N	Z	V	C
*	U	*	U	*

N : Undefined.

Z : Cleared if the result is non-zero. Unchanged otherwise.

V : Undefined.

C : Set if a borrow (decimal) was generated. Cleared otherwise.

X : Set the same as the carry bit.

Note :

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple precision operations.

Instruction Format :

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

NBCD NBCD

Instruction Fields : Effective Address field – Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

NEG Negate NEG

Operation : 0 − Destination → Destination

Assembler Syntax : NEG <ea>

Attributes : Size = (Byte, Word, Long word)

Description : The operand addressed as the destination is subtracted from zero. The result is stored in the destination location. The size of the operation may be specified to be byte, word, or long word.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if the result is negative. Cleared otherwise.

Z : Set if the result is zero. Cleared otherwise.

V : Set if an overflow is generated. Cleared otherwise.

C : Cleared if the result is zero. Set otherwise.

X : Set the same as the carry bit.

Instruction Format:

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	1	0	0	0	1	0	0	Size	Effective Address Mode	Register

NEG NEG

Instruction Fields : Size field — Specifies the size of the operation.

00 — byte operation

01 — word operation

10 — long word operation

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
- (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	-	-
d8 (PC, Xn)	-	-
#<data>	-	-

NEGX Negate with Extend NEGX

Operation : 0 − Destination − X → Destination

Assembler Syntax : NEGX <ea>

Attributes : Size = (Byte, Word, Long word)

Description : The operand addressed as the destination and the extend bit are subtracted from zero. The result is stored in the destination location. The size of the operation may be specified to be byte, word, or long word .

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if the result is negative. Cleared otherwise.

Z : Cleared if the result is non-zero. Unchanged otherwise.

V : Set if overflow is generated. Cleared otherwise.

C : Set if a borrow is generated. Cleared otherwise.

X : Set the same as the carry bit.

Note :

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

Instruction Format :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	Size		Effective Address Mode			Register		

NEGX NEGX

Instruction Fields : Size field – Specifies the size of the operation:

00 – byte operation.

01 – word operation.

10 – long operation.

Effective Address field – Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

NOP No Operation NOP

Operation : None

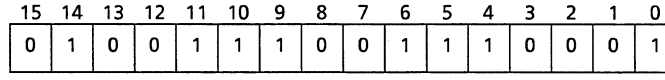
Assebler Syntax : NOP

Attributes : Unsized

Description : No operation occurs. The processor state, other than the program counter, is unaffected. Execution continues with the instruction following the NOP instruction. The NOP instruction does not complete execution until all pending bus cycles are completed. This allows synchronization of the pipeline to be accomplished, and prevents instruction overlap.

Condition Codes : Not affected.

Instruction Format:



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

NOT Logical Complement NOT

Operation : ~Destination → Destination

Assembler Syntax : NOT <ea>

Attributes : Size = (Byte, Word, Long word)

Description : The ones complements of the destination operand is taken and the result is stored in the destination location. The size of the operation may be specified to be byte, word, or long word.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the result is negative. Cleared otherwise.

Z : Set if the result is zero. Cleared otherwise.

V : Always cleared.

C : Always cleared.

X : Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	1	0	0	0	1	1	0	Size	Effective Address Mode	Register

NOT NOT

Instruction Fields : Size field – Specifies the size of the operation.

00 – byte operation.

01 – word operation.

10 – long word operation.

Effective Address field – Specifies the destination operand.

Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

OR — Inclusive OR Logical — OR

Operation : Source V Destination → Destination

Assembler Syntax : OR <ea>, Dn
OR Dn, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Inclusive OR the source operand to the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long word. The contents of an adress register may not be used as an operand.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the most significant bit of the result is set. Cleared otherwise.

Z : Set if the result is zero. Cleared otherwise.

V : Always cleared.

C : Always cleared.

X : Not affected.

Instruction Format:

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	0	0	Register	Op-Mode	Effective Address Mode	Register

OR OR

Instruction Fields : Register field – Specifies any of the eight data registers.
Op-Mode field –

Byte	Word	Long word	Operation
000	001	010	(<ea>)V(<Dn>)→<Dn>
100	101	110	(<Dn>)V(<ea>)→<ea>

Effective Address field –
If the location specified is a source operand then only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

If the location specified is a destination operand then only memory alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Notes: 1. If the destination is a data register, then it cannot be specified by using the destination <ea> mode, but must use the destination Dn mode instead.

2. ORI is used when the source is immediate data. Most assemblers automatically make this distinction.

ORI Inclusive OR Immediate ORI

Operation : Immediate Data V Destination → Destination

Assembler Syntax : ORI #<data>, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Inclusive OR the immediate data to the destination operand and store the result in the destination location. The size of the operation may be specified tobe byte, word, or long word. The size of the immediate data matches the operation size.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the most significant bit of the result is set. Cleared otherwise.

Z : Set if the result is zero. Cleared otherwise.

V : Always cleared.

C : Always cleared.

X : Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	Size		Effective Address Mode			Register		
Word Data								Byte Data							
Long word Data															

ORI ORI

Instruction Fields : Size field – Specifies the size of the operation.

00 – byte operation.

01 – word operation.

10 – long operation.

Effective Address field – Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Immediate field – (Data immediately following the instruction):

If size = 00, then the data is the low order byte of the immediate word.

If size = 01, then the data is the entire immediate word.

If size = 10, then the data is the next two immediate words.

ORI to CCR Inclusive OR Immediate to Condition Codes ORI to CCR

Operation : Source V CCR → CCR

Assembler Syntax : ORI #<data>, CCR

Attributes : Size = (Byte)

Description : Inclusive OR the immediate operand with the condition codes and store the result in the low-order byte of the status register.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if bit 3 of immediate operand is one. Unchanged otherwise.

Z : Set if bit 2 of immediate operand is one. Unchanged otherwise.

V : Set if bit 1 of immediate operand is one. Unchanged otherwise.

C : Set if bit 0 of immediate operand is one. Unchanged otherwise.

X : Set if bit 4 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	Byte Data (8 Bits)							

ORI to SR ORI to SR

Inclusive OR Immediate to the Status Register (Privileged Instruction)

Operation : If supervisor state
then Source V SR → SR
else TRAP;

Assembler Syntax : ORI #<data>, SR

Attributes : Size = (Word)

Description : Inclusive OR the immediate operand with the contents of the status register and store the result in the status register. All bits of the status register are affected.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if bit 3 of immediate operand is one. Unchanged otherwise.

Z : Set if bit 2 of immediate operand is one. Unchanged otherwise.

V : Set if bit 1 of immediate operand is one. Unchanged otherwise.

C : Set if bit 0 of immediate operand is one. Unchanged otherwise.

X : Set if bit 4 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
Word Data (16 Bits)															

PEA Push Effective Address PEA

Operation : SP − 4 → SP; EA → (SP)

Assembler Syntax : PEA <ea>

Attributes : Size = (Long word)

Description : The effective address is computed and pushed onto the stack. A long word address is pushed onto the stack.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields : Effective Address field − Specifies the address to be pushed on to the stack. Only control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	−	−
An	−	−
(An)	010	reg. number:An
(An) +	−	−
− (An)	−	−
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	−	−

RESET — Reset External Devices (Privileged Instruction) — RESET

Operation : If supervisor state
then Assert $\overline{\text{RESET}}$ Line
else TRAP;

Assembler Syntax : RESET

Attributes : Unsized

Description : The reset line is asserted for 124 clocks, causing all external devices to be reset. The processor state, other than the program counter, is unaffected and execution continues with the next instruction.

Condition Codes : Not affected.

Instruction Format:

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

ROL / ROR — Rotate (Without Extend)

ROL ROR

Operation : Destination Rotated by <count> → Destination

Assembler Syntax :
ROd Dx, Dy
ROd #<data>, Dy
ROd <ea>
where d is direction, L or R

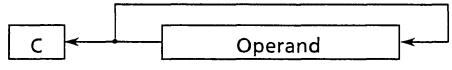
Attributes : Size = (Byte, Word, Long word)

Description : Rotate the bits of the operand in the direction (L or R) specified. The extend bit is not included in the rotation. The rotate count for the rotation of a register may be specified in two different ways:

1. Immediate – the rotate count is specified in the instruction (rotate range, 1~8).
2. Register – the rotate count is contained in a data register specified in the instriuction.

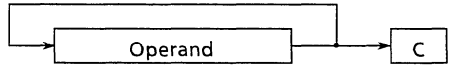
The size of the operation may be specified to be byte, word, or long word. The content of memory may be rotated by one bit only and the operand size is restricted to a word.
For ROL, the operand is rotated left; the number of positions rotated is the rotate count. Bits rotated out of the high order bit go to both the carry bit and back into the low order bit. The extend bit is not modified or used.

ROL :



For ROR, the operand is rotated right; the number of positions rotated is the rotate count. Bits shifted out of the low order bit go to both the carry bit and back into the high order bit. The extend bit is not modified or used.

ROR :



ROL ROL
ROR ROR

Condition Codes :

X	N	Z	V	C
-	*	*	0	*

N : Set if the most significant bit of the result is set. Cleared otherwise.

Z : Set if the result is zero. Cleared otherwise.

V : Always cleared.

C : Set according to the last bit rotated out of the operand. Cleared for a rotate count of zero.

X : Not affected.

Instruction Format (Register Rotate) :

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Rotate/ Register	dr	Size	i/r	1	1	Register

Instruction Fields (Register Rotate) :

Rotate/Register field –
If i/r = 0, the rotate count is specified in this field. The values 0, 1~7 represent a range of 8, 1 to 7 respectively.
If i/r = 1, the rotate count (modulo 64) is contained in the data register specified in this field.

dr field – Specifies the direction of the rotate:
0 – rotate right
1 – rotate left

Size field – Specifies the size of the operation:
00 – byte operation
01 – word operation
10 – long operation

i/r field –
If i/r = 0, Specifies immediate rotate count.
If i/r = 1, Specifies register rotate count.

Register field – Specifies a data register whose content is to be rotated.

ROL ROL

ROR ROR

Instruction Format (Memory Rotate) :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	dr	1	1	Effective Address Mode			Register		

Instruction Fields (Memory Rotate) :

dr field – Specifies the direction of the rotate:

0 – rotate right

1 – rotate left

Effective Address field – Specifies the operand to be rotated.

Only memory alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

ROXL / ROXR — Rotate with Extend — ROXL / ROXR

Operation : Destination Rotated with X by <count> → Destination

Assembler Syntax :
ROXd Dx, Dy
ROXd #<data>, Dy
ROXd <ea>
where d is direction, L or R

Attributes : Size = (Byte, Word, Long word)

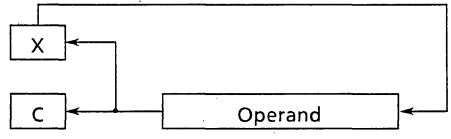
Description : Rotate the bits of the destination operand in the direction specified. The extend bit (X) is included in the rotation. The rotate count for the rotation of a register may be specified in two different ways:

1. Immediate – the rotate count is specified in the instruction (rotate range, 1~8).
2. Register – the rotate count (modulo 64) is contained in a data register specified in the instriuction.

The size of the operation may be specified to be byte, word, or long word. The content of memory may be rotated one bit only and the operand size is restricted to a word.

For ROXL, the operand is rotated left; the number of positions rotated is the rotate count. Bits rotated out of the high order bit go to both the carry and extend bits; the previous value of the extend bit is rotated into the low order bit.

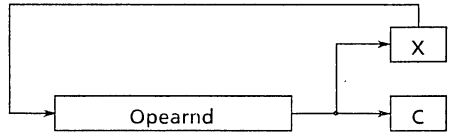
ROXL :



For ROXR, the operand is rotated right; the number of positions shifted is the rotate count. Bits rotated out of the low order bit go to both the carry and extend bits; the previous value of the extend bit is rotated into the high order bit.

ROXL ROXL
ROXR ROXR

ROXR :



Condition Codes :

X	N	Z	V	C
*	*	*	0	*

N : Set if the most significant bit of the result is set. Cleared otherwise.

Z : Set if the result is zero. Cleared otherwise.

V : Always cleared.

C : Set according to the last bit rotated out of the operand. Set to the value of the extend bit for a rotate count of zero.

X : Set according to the last bit rotated out of the operand. Unaffected for a rotate count of zero.

Instruction Format (Register Rotate) :

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Rotate/ Register	dr	Size	i/r	1	0	Register

Instruction Fields (Register Rotate) :

Rotate/Register field –
If i/r = 0, the rotate count is specified in this field. The values 0, 1~7 represent a range of 8, 1 to 7 respectively.
If i/r = 1, the rotate count (modulo 64) is contained in the data register specified in this field.

dr field – Specifies the direction of the rotate:
0 – rotate right
1 – rotate left

ROXL ROXL
ROXR ROXR

Size field – Specifies the size of the operation:
00 – byte operation.
01 – word operation.
10 – long word operation.
i/r field –
If i/r = 0, specifies immediate rotate count.
If i/r = 1, specifies register rotate count.
Register field – Specifies a data register whose content is to be rotated.

Instruction Format (Memory Rotate) :

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
1	1	1	0	0	1	0	dr	1	1	Effective Address Mode	Register

Instruction Fields (Memory Rotate) :

dr field – Specifies the direction of the rotate:
0 – rotate right
1 – rotate left
Effective Address field – Specifies the operand to be rotated. Only memory alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

RTD Return and Deallocate Parameters RTD

Operation : (SP) → PC; SP + 4 + d16 → SP

Assembler Syntax : RTD #<displacement>

Attributes : Unsized

Description : The program counter is pulled from the stack. The previous program counter value is lost. After the program counter is read from the stack, the displacement value (16 bits) is sign-extended to 32 bits and added to the stack pointer.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields : Displacement field –
Specifies the twos complement integer which is to be sign-extended and added to the stack pointer.

RTE Return from Exception (Privileged Instruction) RTE

Operation : If supervisor stat
then ((SP) → SR; SP + 2 → SP; (SP) → PC; SP + 4 → SP;)
else TRAP;

Assembler Syntax : RTE

Attributes : Unsized

Description : The status register and program counter are pulled from the system stack. The previous status register and program counter are lost. All bits in the status register aare affected.

Condition Codes : Set according to the content of the word on the stack.

Instruction Format:

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

RTE Return from Exception (Privileged Instruction) RTE

Operation : If supervisor stat
then ((SP) → SR; SP + 2 → SP; (SP) → PC; SP + 4 → SP;
restore state and deallocate
stack according to (SP))
else TRAP;

Assembler Syntax : RTE

Attributes : Unsized

Description : The processor state information in the exception stack frame on top of the stack is loaded into the processor. The stack format field in the format/offset word is examined to determine how much information must be restored.

Condition Codes : Set according to the content of the word on the stack.

Instruction Format:

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

Format/Offset Word (in stack frame) :

15 – 12	11	10	9 – 0
Format	0	0	Vector Offset

Instruction Fields : Format field – This 4-bit field defines the amount of information to be restored.

0000 – Short Format, only four words are to be removed from the top of the stack. The status register and program counter are loaded from the stack frame.

1000 – TMP68010 Long Format, 29 words are removed from the top of the stack.

Any others – the processor takes a format error exception.

RTR Return and Restore Condition Codes RTR

Operation : (SP) → CCR; SP + 2 → SP;
(SP) → PC; SP + 4 → SP

Assembler Syntax : RTR

Attributes : Unsized

Description : The condition codes and program counter are pulled from the stack. The previous condition codes and program counter are lost. The supervisor portion of the status register is unaffected.

Condition Codes : Set according to the content of the word on the stack.

Instruction Format:

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

RTS Return from Subroutine RTS

Operation : (SP) → PC; SP + 4 → SP

Assembler Syntax : RTS

Attributes : Unsized

Description : The program counter is pulled from the stack. The previous program counter is lost.

Condition Codes : Not affected.

Instruction Format :

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

SBCD Subtract Decimal with Extend SBCD

Operation : $Destination_{10} - Source_{10} - X \rightarrow Destination$

Assembler Syntax :
SBCD Dx, Dy
SBCD −(Ax), −(Ay)

Attributes : Size = (Byte)

Description : Subtract the source operand from the distination operand with the extend bit and store the result in the destination location. The subtraction is performed using decimal arithmetic. The operands may be addressed in two different ways:

1. Data register to data register: The operands are contained in the data registers specified in the instruction.
2. Memory to memory: The operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

This operation is a byte operation only.

Condition Codes :

X	N	Z	V	C
*	U	*	U	*

N : Undefined.

Z : Cleared if the result is non-zero. Unchanged otherwise.

V : Undefined.

C : Set if a borrow (decimal) is generated. Cleared otherwose.

X Set the same as the carry bit.

Note:

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operation.

Instruction Format:

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	0	0	0	Register Dy/Ay	1	0	0	0	0	R/M	Register Dx/Ax

SBCD SBCD

Instruction Fields : Register Dy/Ay field – Specifies the destination register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.
R/M filed – Specifies the operand addressing mode:
0 – The operation is data register to data register
1 – The operation is memory to memory
Register Dx/Ax field – Specifies the source register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

Scc — Set According to Condition — Scc

Operation : If Condition True
then 1s → Destination
else 0s → Destination

Assembler Syntax : Scc <ea>

Attributes : Size = (Byte)

Description : The specified condition code is tested; if the condition is true, the byte specified by the effective address is set to TRUE (all ones), otherwise that byte is set to FALSE (all zeroes). "cc" may specify the following conditions:

CC	carry clear	0100	$\overline{C}$
CS	carry set	0101	C
EQ	equal	0111	Z
F	never true	0001	0
GE	greater or equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
GT	greater than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
HI	high	0010	$\overline{C} \cdot \overline{Z}$
LE	less or equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$
LS	low or same	0011	$C + Z$
LT	less than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
MI	minus	1011	N
NE	not equal	0110	$\overline{Z}$
PI	plus	1010	$\overline{N}$
T	always true	0000	1
VC	overflow clear	1000	$\overline{V}$
VS	overflow set	1001	V

· = Boolean AND + = Boolean OR $\overline{N}$ = Boolean NOT N

Condition Codes : Not affected.

Instruction Format:

15	14	13	12	11 10 9 8	7	6	5 4 3	2 1 0
0	1	0	1	Condition	1	1	Effective Address Mode	Register

Scc Scc

Instruction Fields : Condition field – One of sixteen conditions discussed in description. Effective Address field – Specifies the location in which the true/false byte is to be stored. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	–	–
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
d16 (An)	101	reg. number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. number:An
Ads.W	111	000
Ads.L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Note: 1. An arithmetic one and zero result may be generated by following the Scc instruction with a NEG instruction.

STOP — Load Status Register and Stop (Privileged Instruction) — STOP

Operation : If supervisor state
then (Immediate Data → SR; STOP)
else TRAP ;

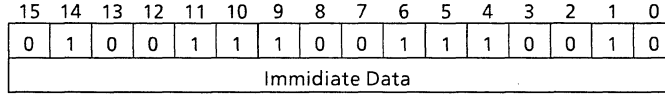
Assembler Syntax : STOP #<data>

Attributes : Unsized

Description : The immediate operand is moved into the entire status register; the program counter is advanced to point to the next instruction and the processor stops fetching and executing instructions. Execution of instructions resumes when a trace, interrupt, or reset exception occurs. A trace exception will occur if the trace state is on when the STOP instruction begins execution. If an interrupt request is asserted with a priority higher than the priority level set by the immediate data, an interrupt exception occurs, otherwise, the interrupt request has no effect. If the bit of the immediate data corresponding to the S-bit is off, execution of the instruction will cause a privilege violation. External reset will always initiate reset exception processing.

Condition Codes : Set according to the immediate operand.

Instruction Format:



Instruction Fields : Immediate field – Specifies the data to be loaded into the status register.

SUB Subtract Binary SUB

Operation : Destination − Source → Destination

Assembler Syntax : SUB <ea>, Dn
SUB Dn, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Subtract the source operand from the destination operand and store the result in the destination. The size of the operation may be specified to be byte, word, or long. The mode of the instruction indicates which operand is the source and which is the destination as well as the operand size.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a borrow is generated. Cleared otherwise.
X : Set the same as the carry bit.

The condition codes are not affected if a subtraction from an address register is made.

Instruction Format:

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	0	1	Register	Op-Mode	Effective Address Mode	Effective Address Register

Instruction Fields :

Register field − Specifies any of the eight data registers.

Op-Mode field −

Byte	Word	Long word	Opration
000	001	010	<Dn> − <ea> → <Dn>
100	101	110	<ea> − <Dn> → <ea>

Effective Address field − Determines addressing mode:
If the location specified is a source operand, then all addressing modes are allowed as shown:

SUB SUB

Addr. Mode	Mode	Register
Dn	000	reg. Number:Dn
An*	001	reg. Number:An
(An)	010	reg. Number:An
(An) +	011	reg. Number:An
– (An)	100	reg. Number:An
d16 (An)	101	reg. Number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. Number:An
Abs. W	111	000
Abs. L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

* : For byte size operaiton, address register direct is not allowed.

If the location specified is a destination operand, then only alterable memory addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	–	–
An	–	–
(An)	010	reg. Number:An
(An) +	011	reg. Number:An
– (An)	100	reg. Number:An
d16 (An)	101	reg. Number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. Number:An
Abs. W	111	000
Abs. L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Notes:
1. If the destination is a data register, then it cannot be specified by using the destination <ea> mode, but must use the destination Dn mode instead.
2. SUBA is used when the destination is an address register. SUBI and SUBQ are used when the source is immediate data. Most assemblers automatically make this distinction.

SUBA Subtract Address SUBA

Operation : Destination − Source → Destination

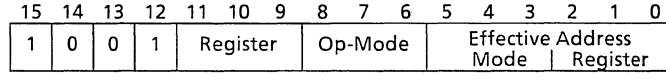
Assembler Syntax : SUBA <ea>, An

Attributes : Size = (Word, Long word)

Description : Subtract the source operand from the destination address register and store the result in the address register. The size of the operation may be specified to be word or long word. Word size source operands are sign extended to 32 bit quantities before the operation is done.

Condition Codes : Not affected.

Instruction Format:



Instruction Fields :

Register field − Specifies any of the eight address registers. This is always the destination.

Op-Mode field − Specifies the size of the operation:

011 − Word operation. The source operand is sign-extended to a long word operand and the operation is performed on the address register using all 32 bits.

111 − Long word operations.

Effective Address field − Specifies the source operand. All addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. Number:Dn
An	001	reg. Number:An
(An)	010	reg. Number:An
(An) +	011	reg. Number:An
− (An)	100	reg. Number:An
d16 (An)	101	reg. Number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. Number:An
Abs. W	111	000
Abs. L	111	001
d16 (PC)	111	010
d8 (PC, Xn)	111	011
#<data>	111	100

SUBI Subtract Immediate SUBI

Operation : Destination − Immediate Data → Destination

Assembler Syntax : SUBI #<data>, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Subtract the immediate data from destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long word. The size of the immediate data matches the operation size.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a borrow is generated. Cleared otherwise.
X : Set the same as the carry bit.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	Size		Effective Address Mode			Register		
Word Data								Byte Data							
Long Data															

Instruction Fields :

Size field − Specifies the size of the operation.
00 − byte operation.
01 − word operation.
10 − long word operation.
Effective Address field − Specifies the destination operand.
Only data alterable addressing modes are allowed as shown:

SUBI SUBI

Addr. Mode	Mode	Register
Dn	000	reg. Number:Dn
An	–	–
(An)	010	reg. Number:An
(An) +	011	reg. Number:An
– (An)	100	reg. Number:An
d16 (An)	101	reg. Number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. Number:An
Abs. W	111	000
Abs. L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Immediate field – (Data immediately following the instruction)

If size = 00, then the data is the low order byte of the immediate word.

If size = 01, then the data is the entire immediate word.

If size = 10, then the data is the next two immediate words.

SUBQ Subtract Quick SUBQ

Operation : Destination − Immediate Data → Destination

Assembler Syntax : SUBQ #<data>, <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Subtract the immediate data from the destination operand. The data range is from 1~8. The size of the operation may be specified to be byte, word, or long word. Word and long word operations are also allowed on the address registers and the condition codes are not affected. When subtracting from address registers, the entire destination address register is used, regardless of the operation size.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a borrow is generated. Cleared otherwise.
X : Set the same as the carry bit.

The condition codes are not affected if a subtraction from an address register is made.

Instruction Format:

15	14	13	12	11 10 9	8	7 6	5 4 3	2 1 0
0	1	0	1	Data	1	Size	Effective Address Mode	Register

Instruction Fields :

Data field − Three bits of immediate data, 0, 1~7 representing a range of 8, 1 to 7 respectively.

Size field − Specifies the size of the operation.

00 − byte operation.
01 − word operation.
10 − long word operation.

Effective Address field − Specifies the destination location. Only data alterable addressing modes are allowed as shown:

SUBQ SUBQ

Addr. Mode	Mode	Register
Dn	000	reg. Number:Dn
An*	001	reg. Number:An
(An)	010	reg. Number:An
(An) +	011	reg. Number:An
– (An)	100	reg. Number:An
d16 (An)	101	reg. Number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. Number:An
Abs. W	111	000
Abs. L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

* : Word and long only.

SUBX Subtract with Extend SUBX

Operation : Destination − Source − X → Destination

Assembler Syntax :
SUBX Dx, Dy
SUBX −(Ax), −(Ay)

Attributes : Size = (Byte, Word, Long word)

Description : Subtract the source operand from the destination operand along with the extend bit and store the result in the destination location. The operands may be addressed in two different ways:

1. Data register to data register: The operands are contained in data registers specified in the instruction.
2. Memory to memory. The operands are contained in memory and addressed with the predecrement addressing mode using the address registers specified in the instruction.

The size of the operand may be specified to be byte, word, or long word.

Condition Codes :

X	N	Z	V	C
*	*	*	*	*

N : Set if the result is negative. Cleared otherwise.
Z : Set if the result is zero. Cleared otherwise.
V : Set if an overflow is generated. Cleared otherwise.
C : Set if a borrow is generated. Cleared otherwise.
X : Set the same as the carry bit.

Note:

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

Instruction Format:

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	0	0	1	Register Xy	1	Size	0	0	R/M	Register Xx

SUBX SUBX

Instruction Fields :

Register Xy field – Specifies the destination register:
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

Size field – Specifies the size of the operation:
00 – byte operation
01 – word operation
10 – long operation

R/M filed – Specifies the operand addressing mode:
0 – The operation is data register to data register
1 – The operation is memory to memory

Register Xx field – Specifies the source register:
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

SWAP Swap Register Halves SWAP

Oeration : Register [31:16] ↔ Register [15:0]

Assembler Syntax : SWAP Dn

Attributes : Size = (Word)

Descriotion : Exchange the 16-bit halves of a data register.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the most significant bit of the 32-bit resultis set. Cleared otherwise.

Z : Set if the 32-bit result is zero. Cleared otherwise.

V : Always cleared.

C : Always cleared.

X : Not affected.

Instruction Format:

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

Instruction Fields :

Register field – Specifies the data register to swap.

TAS Test and Set an Operand TAS

Operation : Destination Tested → Condition Codes; 1 → bit 7 of Destination

Assembler Syntax : TAS <ea>

Attributes : Size = (Byte)

Description : Test and set the byte operand addressed by the effective address field. The current value of the operand is tested and N and Z are set accordingly. The high order bit of the operand is set. The operation is indivisible (using a read-modify-write memory cycle) to allow synchronisation of several processors.

Condition Codes:

X	N	Z	V	C
-	*	*	0	0

N : Set if the most significant bit of the operand was set. Cleared otherwise.
Z : Set if the operand was zero. Cleared otherwise.
V : Always cleared.
C : Always cleared.
X : Not affected.

Instruction Format:

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

Instruction Fields :

Effective Address field – Specifies the location of the tested operand. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. Number:Dn
An	–	–
(An)	010	reg. Number:An
(An) +	011	reg. Number:An
– (An)	100	reg. Number:An
d16 (An)	101	reg. Number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. Number:An
Abs. W	111	000
Abs. L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

Note: Bus error retry is inhibited on the read portion of the TAS read-modify-write bus cycle to ensure system integrity. The bus error exception is always taken.

TRAP Trap TRAP

Operation : SSP − 2 → SSP; Format/Vector Offset → (SSP);
SSP − 4 → SSP; PC → (SSP); SSP − 2 → SSP;
SR → (SSP); Vector Address → PC

Assembler Syntax : TRAP #<vector>

Attributes : Unsized

Description : The processor initiates exception processing. The vector number is generated to reference the TRAP instruction exception vector specified by the low order four bits of the instruction. Sixteen TRAP instruction vectors (0~15) are available.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields : Vector field − Specifies which trap vector contains the new program counter to be loaded.

TRAPV Trap on Overflow TRAPV

Operation : If V then TRAP

Assembler Syntax : TRAPV

Attributes : Unsized

Description : If the overflow condition is set, the processor initiates exception processing. The vector number is generated to reference the TRAPV exception vector. If the overflow condition is clear, no operation is performed and execution continues with the next instruction in sequence.

Condition Codes : Not affected.

Instruction Format:

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

TST Test an Operand TST

Operation : Destination Tested → Condition Codes

Assembler Syntax : TST <ea>

Attributes : Size = (Byte, Word, Long word)

Description : Compare the operand with zero. No results are saved; however, the condition codes are set according to results of the test. The size of the operation maybe specified to be byte, word, or long word.

Condition Codes :

X	N	Z	V	C
-	*	*	0	0

N : Set if the operand is negative. Cleared otherwise.
Z : Set if the operand is zero. Cleared otherwise.
V : Always cleared.
C : Always cleared.
X : Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	Size		Effective Address Mode			Register		

Instruction Fields :

Size field – Specifies the size of the operation:
00 – byte operation
01 – word operation
10 – long word operation

Effective Address field – Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. Number:Dn
An	–	–
(An)	010	reg. Number:An
(An) +	011	reg. Number:An
– (An)	100	reg. Number:An
d16 (An)	101	reg. Number:An

Addr. Mode	Mode	Register
d8 (An, Xn)	110	reg. Number:An
Abs. W	111	000
Abs. L	111	001
d16 (PC)	–	–
d8 (PC, Xn)	–	–
#<data>	–	–

UNLK Unlink UNLK

Operation : An → SP; (SP) → An; SP + 2 → SP

Assembler Syntax : UNLK An

Attributes : Unsized

Description : The stack pointer is loaded from the specified address register. The address register is then loaded with the long word pulled from the top of the stack.

Condition Codes : Not affected.

Instruction Format:

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

Instruction Fields : Register field – Specifies the address register through which the unlinking is to be done.

APPENDIX C INSTRUCTION FORMAT SUMMARY

C.1 INSTRODUCTION

This appendix provides a summary of the primary words in each instruction of the instruction set. The complete instruction definition consists of the primary words followed by the addressing mode operands such as immediate data fields, displacements, and index operands. Table C.1 is an operand code (opcode) map which illustrates how bits 15 ~ 12 are used to specify the operations.

Table C.1 Operation Code Map

Bits 15 through 12	Operation
0000	Bit Manipulation / MOVEP / Immediate
0001	Move Byte
0010	Move Long word
0011	Move Word
0100	Miscellaneous
0101	ADDQ / SUBQ / Scc / DBcc
0110	Bcc / BSR
0111	MOVEQ
1000	OR / DIV / SBCD
1001	SUB / SUBX
1010	(Unassigned, Reserved)
1011	CMP / EOR
1100	AND / MUL / ABCD / EXG
1101	ADD / ADDX
1110	Shift / Rotate
1111	(Unassigned, Reserved)

Table C.2. Effective Addressing Mode Categories

Address Modes	Mode	Register
Data Register Direct	000	reg. no.
Address Register Direct	001	reg. no.
Address Register Indirect	010	reg. no.
Address Register Indirect with Postincrement	011	reg. no.
Address Register Indirect with Predecrement	100	reg. no.
Address Register Indirect with Displacement	101	reg. no.
Address Register Indirect with Index	110	reg. no.
Absolute Short	111	000
Absolute Long	111	001
Program Counter Indirect with Displacement	111	010
Program Counter Indirect with Index	111	011
Immediate	111	100

Table C.3. Conditional Tests

Mnemonic	Condition	Encoding	Test
T*	True	0000	1
F*	False	0001	0
HI	High	0010	$\overline{C} \cdot \overline{Z}$
LS	Low or Same	0011	$C + Z$
CC (HS)	Carry Clear	0100	$\overline{C}$
CS (LO)	Carry Set	0101	C
NE	Not Equal	0110	$\overline{Z}$
EQ	Equal	0111	Z
VC	Overflow Clear	1000	$\overline{V}$
VS	Overflow Set	1001	V
PL	Plus	1010	$\overline{N}$
MI	Minus	1011	N
GE	Greater or Equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
LT	Less Than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
GT	Greater Than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
LE	Less or Equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

• = Boolean AND + = Boolean OR $\overline{N}$ = Boolean NOT N

* : Not available for the Bcc instruction

STANDARD INSTRUCTIONS

ORI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	Size		Effective Address Mode			Register		

Size field : 00 = byte
01 = word
10 = long word

ORI to CCR

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

ORI to SR

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

Dynamic Bit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Data Register			1	Type		Effective Address Mode			Register		

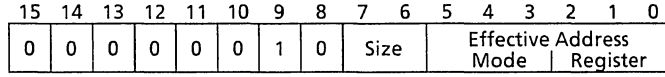
Type field : 00 = TST
01 = CHG
10 = CLR
11 = SET

MOVEP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Data Register			OP -Mode			0	0	1	Address Register		

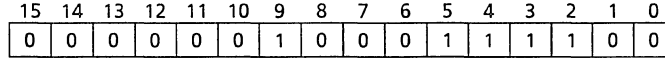
Op-Mode field : 100 = transfer word from memory to register
101 = transfer long from memory to register
110 = transfer word from register to memory
111 = transfer long from register to memory

ANDI

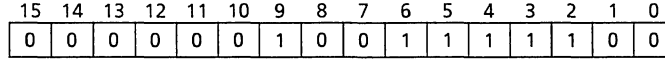


Size field : 00 = byte
01 = word
10 = long word

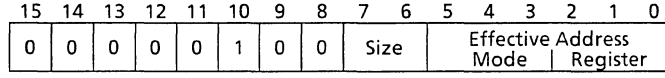
ANDI to CCR



ANDI to SR

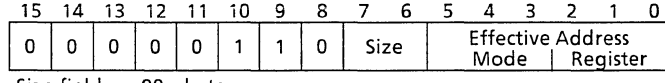


SUBI



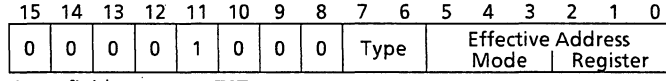
Size field : 00 = byte
01 = word
10 = long word

ADDI



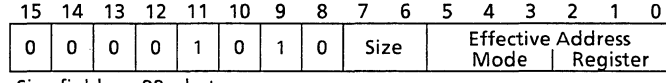
Size field : 00 = byte
01 = word
10 = long word

Static Bit



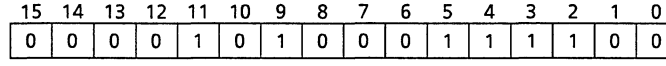
Type field : 00 = TST
01 = CHG
10 = CLR
11 = SET

EORI

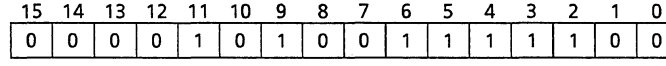


Size field : 00 = byte
01 = word
10 = long word

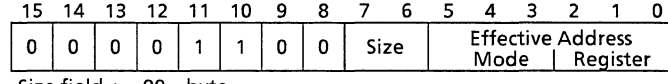
EORI to CCR



EORI to SR

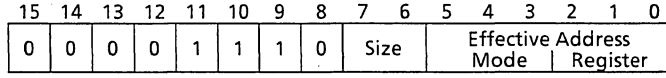


CMPI



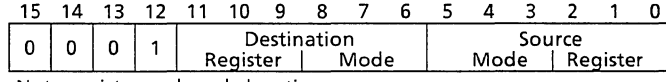
Size field : 00 = byte
01 = word
10 = long word

MOVES



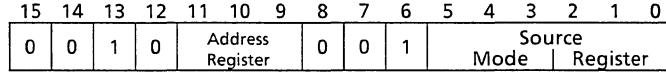
Size field : 00 = byte
01 = word
10 = long word

MOVE (Byte)

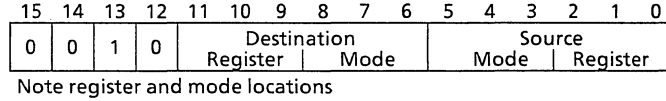


Note register and mode locations

MOVEA (Long word)

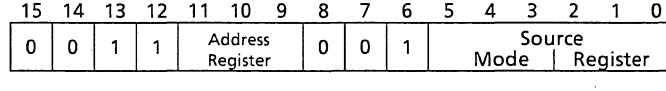


MOVE (Long word)

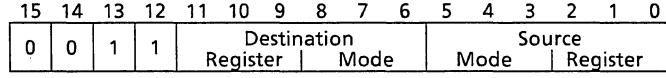


Note register and mode locations

MOVEA (Word)

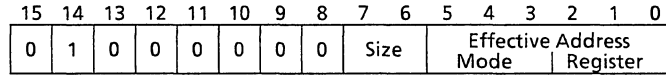


MOVE (Word)



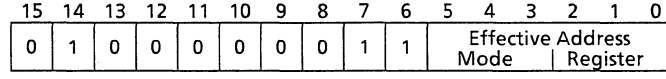
Note register and mode locations

NEGX

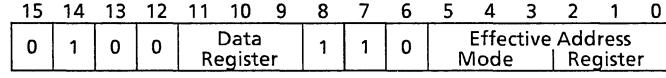


Size field : 00 = byte
01 = word
10 = long word

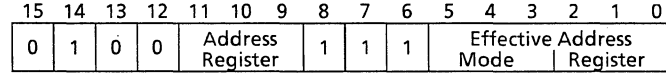
MOVE from SR



CHK



LEA



CLR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	Size		Effective Address Mode			Register		

Size field : 00 = byte
01 = word
10 = long word

MOVE from CCR

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

NEG

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	Size		Effective Address Mode			Register		

Size field : 00 = byte
01 = word
10 = long word

MOVE to CCR

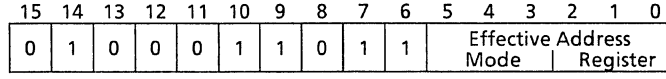
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	Effective Address Mode			Register		

NOT

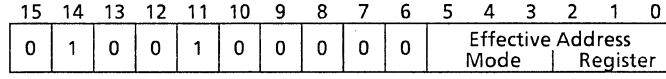
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	Size		Effective Address Mode			Register		

Size field : 00 = byte
01 = word
10 = long word

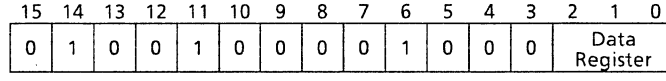
MOVE to SR



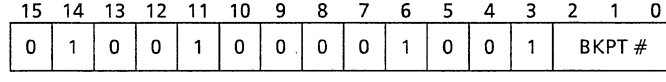
NBCD



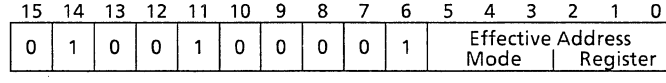
SWAP



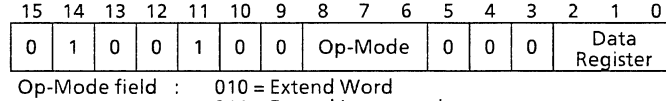
BKPT



PEA

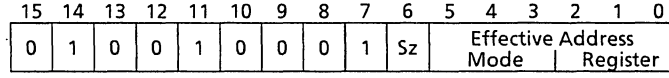


EXT



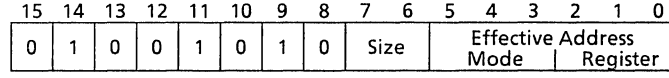
Op-Mode field : 010 = Extend Word
011 = Extend Long word

MOVEM (Registers to EA)



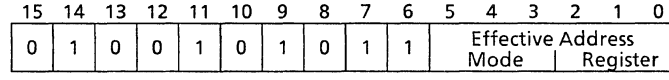
Sz field : 0 = word transfer
1 = long word transfer

TST

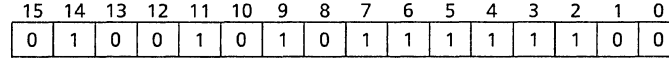


Size field : 00 = byte
01 = word
10 = long word

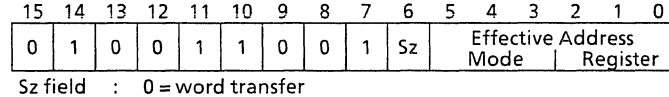
TAS



ILLEGAL

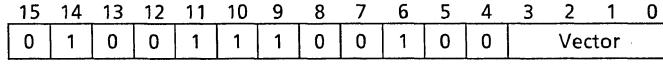


MOVEM (EA to Registers)

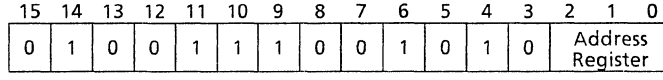


Sz field : 0 = word transfer
1 = long word transfer

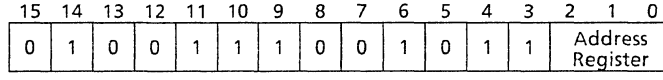
TRAP



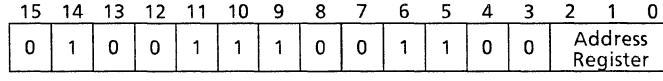
LINK



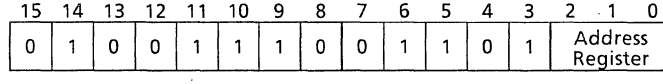
UNLK



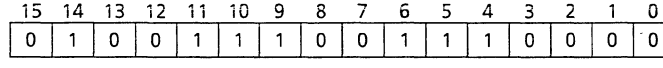
MOVE to USP



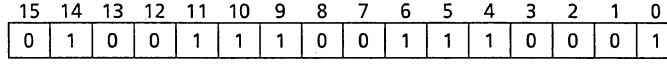
MOVE from USP



RESET

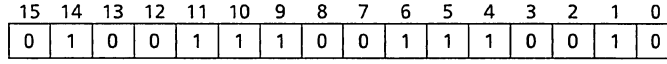


NOP



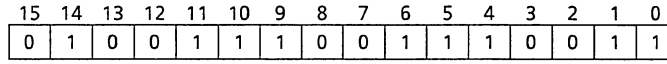
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

STOP



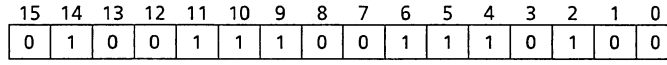
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0

RTE



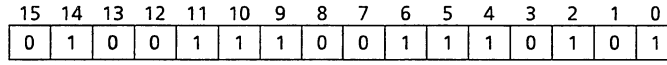
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

RTD



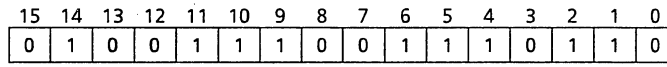
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	0

RTS



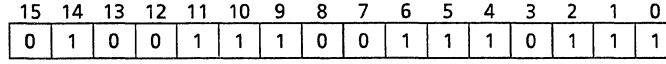
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

TRAPV



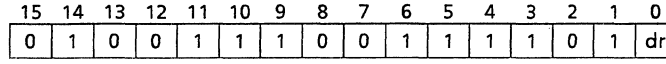
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

RTR



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

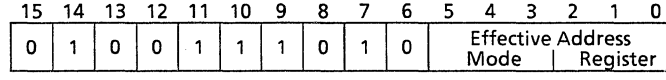
MOVEC



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

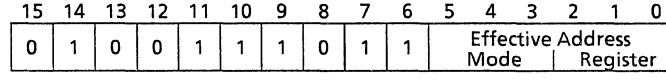
dr field : 0 = control register to general register
1 = general register to control register

JSR



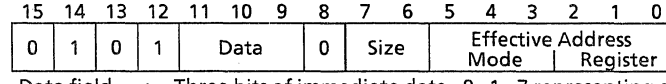
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	Effective Address Mode			Register		

JMP



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

ADDQ



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Data			0	Size		Effective Address Mode			Register		

Data field : Three bits of immediate data, 0, 1~7 representing a range of 8, 1 to 7 respectively.

Size field : 00 = byte
01 = word
10 = long word

Scc

15	14	13	12	11 10 9 8	7	6	5 4 3	2 1 0
0	1	0	1	Condition	1	1	Effective Address Mode	Register

Condition field :

0000 = always true	1000 = overflow clear
0001 = never true	1001 = overflow set
0010 = high	1010 = plus
0011 = low or same	1011 = minus
0100 = carry clear	1100 = greater or equal
0101 = carry set	1101 = less than
0110 = not equal	1110 = greater than
0111 = equal	1111 = less or equal

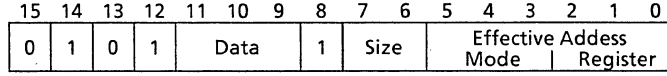
DBcc

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

Condition field :

0000 = always true	1000 = overflow clear
0001 = never true	1001 = overflow set
0010 = high	1010 = plus
0011 = low or same	1011 = minus
0100 = carry clear	1100 = greater or equal
0101 = carry set	1101 = less than
0110 = not equal	1110 = greater than
0111 = equal	1111 = less or equal

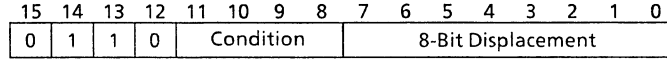
SUBQ



Data field : Three bits of immediate data, 0, 1~7representing a range of 8, 1 to 7 respectively.

Size field . : 00 = byte
01 = word
10 = long word

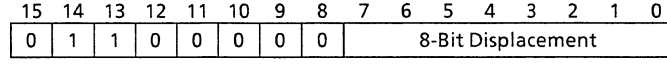
Bcc



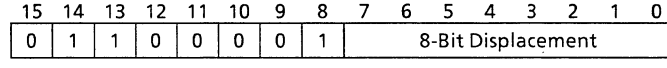
Condition field :

0010 = high
0011 = low or same
0100 = carry clear
0101 = carry set
0110 = not equal
0111 = equal
1000 = overflow clear
1001 = overflow set
1010 = plus
1011 = minus
1100 = greater or equal
1101 = less than
1110 = greater than
1111 = less or equal

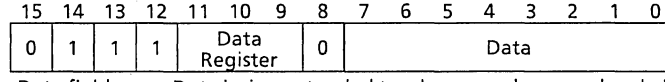
BRA



BSR

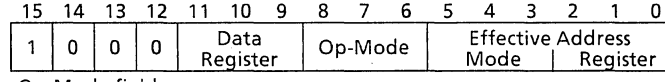


MOVEQ



Data field : Data is sign extended to a long word operand and all 32 bits are transferred to the data register.

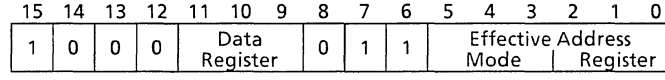
OR



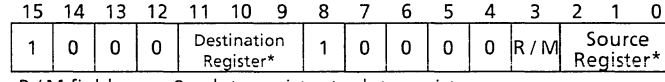
Op-Mode field:

byte	Word	Long word	Operation
000	001	010	<ea> V <Dn> → <Dn>
100	101	110	<Dn> V <ea> → <ea>

DIVU



SBCD



R / M field : 0 = data register to data register
1 = memory to memory

* If R/M = 0, specifies a data register
If R/M = 1, specifies an address register for the predecrement addressing mode.

DIVS

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	0	0	0	Data Register	1	1	1	Effective Address Mode	Register

SUB

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	0	1	Data Register	Op-Mode	Effective Address Mode	Register

Op-Mode field:

Byte	Word	Long word	Operation
000	001	010	(<Dn>)–(<ea>) → <Dn>
100	101	110	(<ea>)–(<Dn>) → <ea>

SUBA

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	0	1	Address Register	Op-Mode	Effective Address Mode	Register

Op-Mode field:

Word	Long word	Operation
000	111	(<ea>)–(<An>)→<An>

SUBX

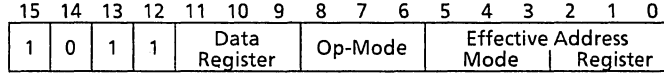
15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	0	0	1	Destination Register*	1	Size	0	0	R/M	Source Register*

Size field : 00 = byte
01 = word
10 = long word

R / M field : 0 = data register to data register
1 = memory to memory

* : If R/M = 0, specifies a data register
If R/M = 1, specifies an address register for the predecrement addressing mode.

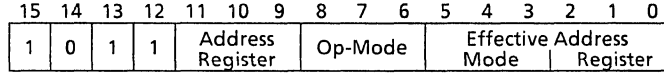
CMP



Op-Mode field:

Byte	Word	Long word	Operation
000	001	010	(<Dn>)–(<ea>)

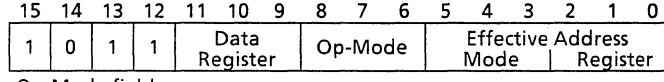
CMPA



Op-Mode field:

Word	Long word	Operation
011	001	(<An>)–(<ea>)

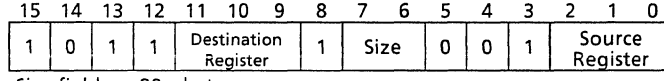
EOR



Op-Mode field:

Byte	Word	Long word	Operation
100	101	110	(<ea>)⊕(<Dn>) → <ea>

CMPM



Size field : 00 = byte
01 = word
10 = long word

AND

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	1	0	0	Data Register	Op-Mode	Effective Address Mode	Effective Address Register

Op-Mode field:

Byte	Word	Long word	Operation
000	001	010	(<ea>)/\(<Dn>) → <Dn>
100	101	110	(<Dn>)/\(<ea>) → <ea>

MULU

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	1	0	0	Data Register	0	1	1	Effective Address Mode	Effective Address Register

ABCD

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	0	0	Destination Register*	1	0	0	0	0	R/M	Source Register*

R / M field : 0 = data register to data register
1 = memory to memory

* If R/M = 0, specifies a data register
If R/M = 1, specifies an address register for the predecrement addressing mode.

EXG (Data Registers)

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

EXG (Address Registers)

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

EXG (Data Register and Address Register)

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

MULS

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	1	0	0	Data Register	1	1	1	Effective Address Mode	Register

ADD

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	1	0	1	Data Register	Op-Mode	Effective Address Mode	Register

OP-Mode field:

Byte	Word	Long word	Operation
000	001	010	(<ea>) + (<Dn>) → <Dn>
100	101	110	(<Dn>) + (<ea>) → <ea>

ADDA

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	1	0	1	Address Register	Op-Mode	Effective Address Mode	Register

OP-Mode field:

Word	Long word	Operation
011	111	(<ea>) + (<An>) → <An>

ADDX

15	14	13	12	11 10 9	8	7 6	5 4	3	2 1 0
1	1	0	1	Destination Register*	1	Size	0 0	R / M	Source Register*

Size field : 00 = byte
01 = word
10 = long word

R/M field : 0 = data register to data register
1 = memory to memory

* : If R/M = 0, specifies a data register
If R/M = 1, specifies an address register for the predecrement addressing mode.

SHIFT/ROTATE (Register)

15	14	13	12	11 10 9	8	7 6	5	4 3	2 1 0
1	1	1	0	Count/ Register	dr	Size	i/r	Type	Data Register

Count/Register field : If i/r field = 0, specifies shift count
If i/r field = 1, specifies a data register that contains the shift count

dr field : 0 = right
1 = left

Size field : 00 = byte
01 = word
10 = long word

i/r field : 0 = immediate shift count
1 = register shift count

Type field : 00 = arithmetic shft
01 = logical shift
10 = rotate with extend
11 = rotate

SHIFT/ROTATE (Memory)

15	14	13	12	11	10 9	8	7	6	5 4 3	2 1 0
1	1	1	0	0	Type	dr	1	1	Effective Address Mode	Register

Type field : 00 = arithmetic shift
01 = logical shift
10 = rotate with extend
11 = rotate

dr field : 0 = right
1 = left

APPENDIX D TMP68000 INSTRUCTION EXECUTION TIMES

D.1 INTRODUCTION

This Appendix contains listings of the instruction execution times in terms of external clock (CLK) periods. In this data, it is assumed that both memory read and write cycle times are four clock periods. A longer memory cycle will cause the generation of wait states which must be added to the total instruction time.

The number of bus read and write cycles for each instruction is also included with the timing data. This data is enclosed in parenthesis following the number of clock periods and is shown as: (r/w) where r is the number of read cycles and w is the number of write cycles included in the clock period number. Recalling that either a read or write cycle requires four clock periods, a timing number given as 18(3/1) relates to 12 clock periods for the three read cycles, plus 4 clock periods for the one write cycle, plus 2 cycles required for some internal function of the processor.

Note:

The number of periods includes instruction fetch and all applicable operand fetches and stores.

D.2 OPERAND EFFECTIVE ADDRESS CALCULATION TIMES

Table D.1 lists the number of clock periods required to compute an instruction's effective address. It includes fetching of any extension words, the address computation, and fetching of the memory operand. The number of bus read and write cycles is shown in parenthesis as (r/w). Note there are no write cycles involved in processing the effective address.

Table D.1 Effective Address Calculation Times

Addressing Mode		Byte, Word	Long word
	Register		
Dn	Data Register Direct	0 (0/0)	0 (0/0)
An	Address Register Direct	0 (0/0)	0 (0/0)
	Memory		
(An)	Address Register Indirect	4 (1/0)	8 (2/0)
(An) +	Address Register Indirect with Postincrement	4 (1/0)	8 (2/0)
-(An)	Address Register Indirect with Predecrement	6 (1/0)	10 (2/0)
d16 (An)	Address Register Indirect with Displacement	8 (2/0)	12 (3/0)
d8 (An, Xn)*	Address Register Indirect with Index	10 (2/0)	14 (3/0)
Abs.W	Absolute Short	8 (2/0)	12 (3/0)
Abs. L	Absolute Long	12 (3/0)	16 (4/0)
d16 (PC)	Program Counter with Displacement	8 (2/0)	12 (3/0)
d16 (PC, Xn)*	Program Counter with Index	10 (2/0)	14 (3/0)
#<data>	Immediate	4 (1/0)	8 (2/0)

* : The size of the index register (Xn) does not affect execution time.

D.3 MOVE INSTRUCTION EXECUTION TIMES

Table D.2 and D.3 indicate the number of clock periods for the move instruction. This data includes instruction fetch, operand reads, and operand writes. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table D.2 Move Byte and Word Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An) +	- (An)	d16 (An)	d8 (An, Xn)*	Abs. W	Abs. L
Dn	4 (1 / 0)	4 (1 / 0)	8 (1 / 1)	8 (1 / 1)	8 (1 / 1)	12 (2 / 1)	14 (2 / 1)	12 (2 / 1)	16 (1 / 3)
An	4 (1 / 0)	4 (1 / 0)	8 (1 / 1)	8 (1 / 1)	8 (1 / 1)	12 (2 / 1)	14 (2 / 1)	12 (2 / 1)	16 (1 / 3)
(An)	8 (2 / 0)	8 (2 / 0)	12 (2 / 1)	12 (2 / 1)	12 (2 / 1)	16 (3 / 1)	18 (3 / 1)	16 (3 / 1)	20 (4 / 1)
(An) +	8 (2 / 0)	8 (2 / 0)	12 (2 / 1)	12 (2 / 1)	12 (2 / 1)	16 (3 / 1)	18 (3 / 1)	16 (3 / 1)	20 (4 / 1)
- (An)	10 (2 / 0)	10 (2 / 0)	14 (2 / 1)	14 (2 / 1)	14 (2 / 1)	18 (3 / 1)	20 (3 / 1)	18 (3 / 1)	22 (1 / 4)
d16 (An)	12 (3 / 0)	12 (3 / 0)	16 (3 / 1)	16 (3 / 1)	16 (3 / 1)	20 (4 / 1)	22 (4 / 1)	20 (4 / 1)	24 (5 / 1)
d8 (An, Xn)*	14 (3 / 0)	14 (3 / 0)	18 (3 / 1)	18 (3 / 1)	18 (3 / 1)	22 (4 / 1)	24 (4 / 1)	22 (4 / 1)	26 (5 / 1)
Abs. W	12 (3 / 0)	12 (3 / 0)	16 (3 / 1)	16 (3 / 1)	16 (3 / 1)	20 (4 / 1)	22 (4 / 1)	20 (4 / 1)	24 (5 / 1)
Abs. L	16 (4 / 0)	16 (4 / 0)	20 (4 / 1)	20 (4 / 1)	20 (4 / 1)	24 (5 / 1)	26 (5 / 1)	24 (5 / 1)	28 (6 / 1)
d16 (PC)	12 (3 / 0)	12 (3 / 0)	16 (3 / 1)	16 (3 / 1)	16 (3 / 1)	20 (4 / 1)	22 (4 / 1)	20 (4 / 1)	24 (5 / 1)
d8 (PC, Xn)*	14 (3 / 0)	14 (3 / 0)	18 (3 / 1)	18 (3 / 1)	18 (3 / 1)	22 (4 / 1)	24 (4 / 1)	22 (4 / 1)	26 (5 / 1)
#<data>	8 (2 / 0)	8 (2 / 0)	12 (2 / 1)	12 (2 / 1)	12 (2 / 1)	16 (3 / 1)	18 (3 / 1)	16 (3 / 1)	20 (4 / 1)

* : The size of the index register (Xn) does not affect execution time.

Table D.3 Move Long Word Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An) +	- (An)	d16 (An)	d8 (An, Xn)*	Abs. W	Abs. L
Dn	4 (1 / 0)	4 (1 / 0)	12 (1 / 2)	12 (1 / 2)	12 (1 / 2)	16 (2 / 2)	18 (2 / 2)	16 (2 / 2)	20 (3 / 2)
An	4 (1 / 0)	4 (1 / 0)	12 (1 / 2)	12 (1 / 2)	12 (1 / 2)	16 (2 / 2)	18 (2 / 2)	16 (2 / 2)	20 (3 / 2)
(An)	12 (3 / 0)	12 (3 / 0)	20 (3 / 2)	20 (3 / 2)	20 (3 / 2)	24 (4 / 2)	26 (4 / 2)	24 (4 / 2)	28 (5 / 2)
(An) +	12 (3 / 0)	12 (3 / 0)	20 (3 / 2)	20 (3 / 2)	20 (3 / 2)	24 (4 / 2)	26 (4 / 2)	24 (4 / 2)	28 (5 / 2)
- (An)	14 (3 / 0)	14 (3 / 0)	22 (3 / 2)	22 (3 / 2)	22 (3 / 2)	26 (4 / 2)	28 (4 / 2)	26 (4 / 2)	30 (5 / 2)
d16 (An)	16 (4 / 0)	16 (4 / 0)	24 (4 / 2)	24 (4 / 2)	24 (4 / 2)	28 (5 / 2)	30 (5 / 2)	28 (5 / 2)	32 (6 / 2)
d8 (An, Xn)*	18 (4 / 0)	18 (4 / 0)	26 (4 / 2)	26 (4 / 2)	26 (4 / 2)	30 (5 / 2)	32 (5 / 2)	30 (5 / 2)	34 (6 / 2)
Abs. W	16 (4 / 0)	16 (4 / 0)	24 (4 / 2)	24 (4 / 2)	24 (4 / 2)	28 (5 / 2)	30 (5 / 2)	28 (5 / 2)	32 (6 / 2)
Abs. L	20 (5 / 0)	20 (5 / 0)	28 (5 / 2)	28 (5 / 2)	28 (5 / 2)	32 (6 / 2)	34 (6 / 2)	32 (6 / 2)	36 (7 / 2)
d16 (PC)	16 (4 / 0)	16 (4 / 0)	24 (4 / 2)	24 (4 / 2)	24 (4 / 2)	28 (5 / 2)	30 (5 / 2)	28 (5 / 2)	32 (5 / 2)
d8 (PC, Xn)*	18 (4 / 0)	18 (4 / 0)	26 (4 / 2)	26 (4 / 2)	26 (4 / 2)	30 (5 / 2)	32 (5 / 2)	30 (5 / 2)	34 (6 / 2)
#<data>	12 (3 / 0)	12 (3 / 0)	20 (3 / 2)	20 (3 / 2)	20 (3 / 2)	24 (4 / 2)	26 (4 / 2)	24 (4 / 2)	28 (5 / 2)

* : The size of the index register (Xn) does not affect execution time.

D.4 STANDARD INSTRUCTION EXECUTION TIMES

The number of clock periods shown in Table D.4 indicates the time required to perform the operations, store the results, and read the next instruction. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table D.4 the headings have the following meanings: An = address register operand, Dn = data register operand, ea = an operand specified by an effective address, and M = memory effective address operand.

Table D.4 Standard Instruction Execution Times

Instruction	Size	op<ea>, An^	op<ea>, Dn	op Dn, <M>
ADD / ADDA	Byte, Word	6(1/0) +**	4(1/0) +	8(1/1) +
	Long word	8(1/0) +	6(1/0) +**	12(1/2) +
AND	Byte, Word	—	4(1/0) +	8(1/1) +
	Long word	—	6(1/0) +**	12(1/2) +
CMP / CMPA	Byte, word	6(1/0) +	4(1/0) +	—
	Long word	6(1/0) +	6(1/0) +	—
DIVS	—	—	158(1/0) +*	—
DIVU	—	—	140(1/0) *	—
EOR	Byte, Word	—	4(1/0) ***	8(1/1) +
	Long word	—	8(1/0) ***	12(1/2) +
MULS	—	—	70(1/0) +*	—
MULU	—	—	70(1/0) +*	—
OR	Byte, Word	—	4(1/0) +	8(1/1) +
	Long word	—	6(1/0) +**	12(1/2) +
SUB	Byte, Word	8(1/0) +	4(1/0) +	8(1/1) +
	Long word	6(1/0) +**	6(1/0) +**	12(1/2) +

Note :

+ : add effective address calculation time

^ : word or long word only

* : indicates maximum basic value added to word effective address time.

** : The base time of six clock periods is increased to eight if the effective address mode is register direct or immediate (effective address time should also be added).

*** : Only available effective address mode is data register direct.

DIVS, DIVU - The divide algorithm used by the TMP68000 provides less than 10% difference between the best and worst case timings.

MULS, MULU - The multiply algorithm requires 38 + 2n clocks where n is defines as:

MULS: n = the number of ones in the <ea>

MULU: n = concatanate the <ea> with a zero as the LSB;n is the resultant number of 10 or 01 patterns in the 17-bit source; i.e., worst case happens when the source is $5555.

D.5 IMMEDIATE INSTRUCTION EXECUTION TIMES

The number of clock periods shown in Table D.5 includes the time to fetch immediate operands, perform the operations, store the results, and read the next operation. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table D.5, the headings have the following meanings:

# = immediate operand,
Dn = data register operand,
An = address register operand,
M = memory operand.
SR = status register.

Table D.5 Immediate Instruction Execution Times

Instruction	Size	op #, Dn	op #, An	op #, M
ADDI	Byte, Word	8 (2 / 0)	–	12 (2 / 1) +
	Long word	16 (3 / 0)	–	20 (3 / 2) +
ADDQ	Byte, Word	4 (1 / 0)	8 (1 / 0)*	8 (1 / 1) +
	Long word	8 (1 / 0)	8 (1 / 0)	12 (1 / 2) +
ANDI	Byte, Word	8 (2 / 0)	–	12 (2 / 1) +
	Long word	16 (3 / 0)	–	20 (3 / 1) +
CMPI	Byte, Word	8 (2 / 0)	–	8 (2 / 0) +
	Long word	14 (3 / 0)	–	12 (3 / 0) +
EORI	Byte, Word	8 (2 / 0)	–	12 (2 / 1) +
	Long word	16 (3 / 0)	–	20 (3 / 2) +
MOVEQ	Long word	4 (1 / 0)	–	–
ORI	Byte, Word	8 (2 / 0)	–	12 (2 / 1) +
	Long word	16 (3 / 0)	–	20 (3 / 2) +
SUBI	Byte, Word	8 (2 / 0)	–	12 (2 / 1)
	Long word	16 (3 / 0)	–	20 (3 / 2) +
SUBQ	Byte, Word	4 (1 / 0)	8 (1 / 0)*	8 (1 / 1) +
	Long word	8 (1 / 0)	8 (1 / 0)	12 (1 / 2) +

+ : add effective address calculation time

* : word only

D.6 SINGLE OPERAND INSTRUCTION EXECUTION TIMES

Table D.6 indicates the number of clock periods for the single operand instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table D.6 Single Operand Instruction Execution Times

Instruction	Size	Register	Memory
CLR	Byte, Word	4 (1 / 0)	8 (1 / 1) +
	Long word	6 (1 / 0)	12 (1 / 2) +
NBCD	Byte	6 (1 / 0)	8 (1 / 1) +
NEG	Byte, Word	4 (1 / 0)	8 (1 / 1) +
	Long word	6 (1 / 0)	12 (1 / 2) +
NEGX	Byte, Word	4 (1 / 0)	8 (1 / 1) +
	Long word	6 (1 / 0)	12 (1 / 2) +
NOT	Byte, Word	4 (1 / 0)	8 (1 / 1) +
	Long word	6 (1 / 0)	12 (1 / 2) +
Scc	Byte, Word	4 (1 / 0)	8 (1 / 1) +
	Long word	6 (1 / 0)	8 (1 / 1) +
TAS	Byte	4 (1 / 0)	10 (1 / 1) +
TST	Byte, Word	4 (1 / 0)	4 (1 / 0) +
	Long word	4 (1 / 0)	4 (1 / 0) +

+ : add effective address calculation time

D.7 SHIFT/ROTATE INSTRUCTION EXECUTION TIMES

Table D.7 indicates the number of clock periods for the shift and rotate instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table D.7 Shift/Rotate Instruction Execution Times

Instruction	Size	Register	Memory
ASR, ASL	Byte, Word	6 + 2n (1 / 0)	8 (1 / 1) +
	Long word	8 + 2n (1 / 0)	—
LSR, LSL	Byte, Word	6 + 2n (1 / 0)	8 (1 / 1) +
	Long word	8 + 2n (1 / 0)	—
ROR, ROL	Byte, Word	6 + 2n (1 / 0)	8 (1 / 1) +
	Long word	8 + 2n (1 / 0)	—
ROXR, ROXL	Byte, Word	6 + 2n (1 / 0)	8 (1 / 1) +
	Long word	8 + 2n (1 / 0)	—

+ : add effective address calculation time for word operands

n : the shift count

D.8 BIT MANIPULATION INSTRUCTION EXECUTION TIMES

Table D.8 indicates the number of clock periods required for the bit manipulation instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table D.8 Bit Manipulation Instruction Execution Times

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	–	8 (1 / 1) +	–	12 (2 / 1) +
	Long word	8 (1 / 0)*	–	12 (2 / 0)	–
BCLR	Byte	–	8 (1 / 1) +	–	12 (2 / 1) +
	Long word	10 (1 / 0)	–	14 (2 / 0)*	–
BSET	Byte	–	8 (1 / 1) +	–	12 (2 / 1) +
	Long word	8 (1 / 0)*	–	12 (2 / 0)*	–
BTST	Byte	–	4 (1 / 0) +	–	8 (2 / 0) +
	Long word	6 (1 / 0)	–	10 (2 / 0)	–

\+ : add effective address calculation time

* : indicates maximum value; data addressing mode only

D.9 CONDITIONAL INSTRUCTION EXECUTION TIMES

Table D.9 indicates the number of clock periods required for the conditional instructions. The number of bus read and write cycles is indicated in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table D.9 Conditional Instruction Execution Times

Instruction	Displacement	Branch Taken	Branch Not Taken
Bcc	Byte	10 (2 / 0)	8 (1 / 0)
	Word	10 (2 / 0)	12 (2 / 0)
BRA	Byte	10 (2 / 0)	–
	Word	10 (2 / 0)	–
BSR	Byte	18 (2 / 2)	–
	Word	18 (2 / 2)	–
DBcc	cc true	–	12 (2 / 0)
	cc false, Count Not Expired	10 (2 / 0)	–
	cc false, Count Expired	–	14 (3 / 0)

\+ : add effective address calculation time

* : indicates maximum base value

D.10 JMP, JSR, LEA, PEA, AND MOVEM INSTRUCTION EXECUTIONTIMES

Table D.10 indicates the number of clock periods required for the jump, jump-to-subroutine, load effective address, push effective address, and move multiple registers instructions. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table D.10 JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

In-struction	Size	(An)	(An) +	- (An)	d16 (An)	d8 (An,Xn)*	Abs. W	Abs. L	d16 (PC)	d8 (Pc, Xn)*
JMP	–	8 (2/0)	–	–	10 (2/0)	14 (3/0)	10 (2/0)	12 (3/0)	10 (2/0)	14 (3/0)
JSR	–	16 (2/2)	–	–	18 (2/2)	22 (2/2)	18 (2/2)	20 (3/2)	18 (2/2)	22 (2/2)
LEA	–	4 (1/0)	–	–	8 (2/0)	12 (2/0)	8 (2/0)	12 (3/0)	8 (2/0)	12 (2/0)
PEA	–	12 (1/2)	–	–	16 (2/2)	20 (2/2)	16 (2/2)	20 (3/2)	16 (2/2)	20 (2/2)
MOVEM M→R	Word	12 + 4n (3 + n/0)	12 + 4n (3 + n/0)	–	16 + 4n (4 + n/0)	18 + 4n (4 + n/0)	16 + 4n (4 + n/0)	20 + 4n (5 + n/0)	16 + 4n (4 + n/0)	18 + 4n (4 + n/0)
	Long	12 + 8n (3 + 2n/0)	12 + 8n (3 + 2n/0)	–	16 + 8n (4 + 2n/0)	18 + 8n (4 + 2n/0)	16 + 8n (4 + 2n/0)	20 + 8n (5 + 2n/0)	16 + 8n (4 + 2n/0)	18 + 8n (4 + 2n/0)
MOVEM R→M	Word	8 + 4n (2/n)	–	8 + 4n (2/n)	12 + 4n (3/n)	14 + 4n (3/n)	12 + 4n (3/n)	16 + 4n (4/n)	– –	– –
	Long	8 + 8n (2/2n)	– –	8 + 8n (2/2n)	12 + 8n (3/2n)	14 + 8n (3/2n)	12 + 8n (3/2n)	16 + 8n (4/2n)	– –	– –

n : the number of registers to move

* : the size of the index register (Xn) does not affect the instruction's execution time

D.11 MULTI-PRECISION INSTRUCTION EXECUTION TIMES

Table D.11 indicates the number of clock periods for the multi-precision instructions. The number of clock periods includes the time to fetch both operands, perform the operations, store the results, and read the next instructions. The number of bus read and write cycles is shown in parenthesis as (r/w).

In Table D.11, the headings have the following meaning: Dn = data register operand and M = memory operand.

Table D.11 Multi-Precision Instruction Execution Times

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte, Word	4 (1/0)	18 (3/1)
	Long word	8 (1/0)	30 (5/2)
CMPM	Byte, Word	–	12 (3/0)
	Long word	–	20 (5/0)
SUBX	Byte, Word	4 (1/0)	18 (3/1)
	Long word	8 (1/0)	30 (5/2)
ABCD	Byte	6 (1/0)	18 (3/1)
SBCD	Byte	6 (1/0)	18 (3/1)

D.12 MISCELLANEOUS INSTRUCTION EXECUTION TIMES

Table D.12 and D.13 indicate the number of clock periods for the folowing miscellaneous instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods plus the number of read and write cycles must be added to those of the effective address calculation where indicated.

Table D.12 Miscellaneous Instruction Execution Times

Instruction	Size	Register	Memory
ANDI to CCR	Byte	20 (3 / 0)	–
ANDI to SR	Word	20 (3 / 0)	–
CHK (No Trap)	–	10 (1 / 0) +	–
EORI to CCR	Byte	20 (3 / 0)	–
EORI to SR	Word	20 (3 / 0)	–
ORI to CCR	Byte	20 (3 / 0)	–
ORI to SR	Word	20 (3 / 0)	–
MOVE from SR	–	6 (1 / 0)	8 (1 / 1) +
MOVE to CCR	–	12 (1 / 0)	12 (2 / 0) +
MOVE to SR	–	12 (1 / 0)	12 (2 / 0) +
EXG	–	6 (1 / 0)	–
EXT	Word	4 (1 / 0)	–
	Long word	4 (1 / 0)	–
LINK	–	16 (2 / 2)	–
MOVE from USP	–	4 (1 / 0)	–
MOVE to USP	–	4 (1 / 0)	–
NOP	–	4 (1 / 0)	–
RESET	–	132 (1 / 0)	–
RTE	–	20 (5 / 0)	–
RTR	–	20 (5 / 0)	–
RTS	–	16 (4 / 0)	–
STOP	–	4 (0 / 0)	–
SWAP	–	4 (1 / 0)	–
TRAPV	–	4 (1 / 0)	–
UNLK	–	12 (3 / 0)	–

+ : add effective address calculation time

Table D.13 Move Peripheral Instruction Execution Times

Instruction	Size	Register → Memory	Memory→Register
MOVEP	Word	16 (2 / 2)	16 (4 / 0)
	Long word	24 (2 / 4)	24 (6 / 0)

D.13 EXCEPTION PROCESSING EXECUTION TIMES

Table D.14 indicates the number of clock periods for exception processing. The number of clock periods includes the time for all stacking, the vector fetch, and the fetch of the first two instruction words of the handler routine. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table D.14 Exception Processing Execution Times

Exception	Periods
Address Error	50 (4 / 7)
Bus Error	50 (4 / 7)
CHK Instruction	44 (5 / 4) +
Divide by Zero	42 (5 / 4)
Ilegal Instruction	34 (4 / 3)
Interrupt	44 (5 / 3)*
Privilege Violation	34 (4 / 3)
$\overline{\text{RESET}}$**	40 (6 / 0)
Trace	34 (4 / 3)
TRAP Instruction	38 (4 / 4)
TRAPV Instruction	34 (4 / 3)

\+ : add effective address calculation time

* : The interrupt acknowledge cycle is assumed to take four clock periods.

** : Indicates the time from when $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are first sampled as negated to when instruction execution satrts.

APPENDIX E TMP68008 INSTRUCTION EXECUTION TIMES

E.1 INTRODUCTION

This Appendix contains listings of the instruction execution times in terms of external clock (CLK) periods. In this data, it is assumed that both memory read and write cycle times are four clock periods. A longer memorycycle will cause the generation of wait states which must be added to the total instruction time.

The number of bus read and write cycles for each instruction is also included with the timing data. This data is enclosed in parenthesis following the number of clock periods and is shown as: (r/w) where r is the number of read cycles and w is the number of write cycles includedin the clock period number. Recalling that either a read or write cycle requires four clock periods, a timing number given as 18 (3/1) relates to 12 clock periods for the three read cycles, plus 4 clock periods for the one write cycle, plus 2 cycles required for some internal function of the processor.

Note:

The number of periods includes instruction fetch and all applicable operand fetches and stores.

E.2 OPERAND EFFECTIVE ADDRESS CALCULATION TIMES

Table E.1 lists the number of clock periods required to compute an instruction's effective address. It includes fetching of any extension words, the address computation, and fetching of the memory operand. The number of bus read and write cycles is shown in parenthesis as (r/w). Note there are no write cycles involved in processing the effective address.

Table E.1 Effective Address Calculation Times

Addressing Mode		Byte	Word	Long word
	Register			
Dn	Data Register Direct	0 (0/0)	0 (0/0)	0 (0/0)
An	Address Register Direct	0 (0/0)	0 (0/0)	0 (0/0)
	Memory			
(An)	Data Register Indirect	4 (1/0)	8 (2/0)	16 (4/0)
(An) +	Address Register Indirect with Postincrement	4 (1/0)	8 (2/0)	16 (4/0)
-(An)	Address Register Indirect with Predecrement	6 (1/0)	10 (2/0)	18 (4/0)
d16 (An)	Address Register Indirect with Displacement	12 (3/0)	16 (4/0)	24 (6/0)
d8 (An, Xn)*	Address Register Indirect with Index	14 (3/0)	18 (4/0)	26 (6/0)
Abs.W	Absolute Short	12 (3/0)	16 (4/0)	24 (6/0)
Abs.L	Absolute Long	20 (5/0)	24 (6/0)	32 (8/0)
d16 (PC)	Program Counter with Displacement	12 (3/0)	16 (4/0)	24 (6/0)
d8 (PC, Xn)*	Program Counter with Index	14 (3/0)	18 (4/0)	26 (6/0)
#<data>	Immediate	8 (2/0)	8 (2/0)	16 (4/0)

* : The size of the index register (Xn) does not affect execution time.

E.3 MOVE INSTRUCTION EXECUTION TIMES

Table E.2, E.3 and E.4 indicate the number of clock periods for the move instruction. This data includes instruction fetch, operand reads, and operand writes. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table E.2 Move Byte Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An) +	-(An)	d16 (An)	d8 (An,Xn)*	Abs.W	Abs.L
Dn	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	20 (4/1)	22 (4/1)	20 (4/1)	28 (6/1)
An	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	20 (4/1)	22 (4/1)	20 (4/1)	28 (6/1)
(An)	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	24 (5/1)	26 (5/1)	24 (5/1)	32 (7/1)
(An) +	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	24 (5/1)	26 (5/1)	24 (5/1)	32 (7/1)
-(An)	14 (3/0)	14 (3/0)	18 (3/1)	18 (3/1)	18 (3/1)	26 (5/1)	28 (5/1)	26 (5/1)	34 (7/1)
d16 (An)	20 (5/0)	20 (5/0)	24 (5/1)	24 (5/1)	24 (5/1)	32 (7/1)	34 (7/1)	32 (7/1)	40 (9/1)
d8 (An, Xn)*	22 (5/0)	22 (5/0)	26 (5/1)	26 (5/1)	26 (5/1)	34 (7/1)	36 (7/1)	34 (7/1)	42 (9/1)
Abs.W	20 (5/0)	20 (5/0)	24 (5/1)	24 (5/1)	24 (5/1)	32 (7/1)	34 (7/1)	32 (7/1)	40 (9/1)
Abs.L	28 (7/0)	28 (7/0)	32 (7/1)	32 (7/1)	32 (7/1)	40 (9/1)	42 (9/1)	40 (9/1)	48 (11/1)
d16 (PC)	20 (5/0)	20 (5/0)	24 (5/1)	24 (5/1)	24 (5/1)	32 (7/1)	34 (7/1)	32 (7/1)	40 (9/1)
d8 (PC, Xn)*	22 (5/0)	22 (5/0)	26 (5/1)	26 (5/1)	26 (5/1)	34 (7/1)	36 (7/1)	34 (7/1)	42 (9/1)
#<data>	16 (4/0)	16 (4/0)	20 (4/1)	20 (4/1)	20 (4/1)	28 (6/1)	30 (6/1)	28 (6/1)	36 (8/1)

* : The size of the index register (Xn) does not affect execution time.

Table E.3 Move Word Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An) +	-(An)	d16 (An)	d8 (An,Xn)*	Abs.W	Abs.L
Dn	8 (2/0)	8 (2/0)	16 (2/2)	16 (2/2)	16 (2/2)	24 (4/2)	26 (4/2)	20 (4/2)	32 (6/2)
An	8 (2/0)	8 (2/0)	16 (2/2)	16 (2/2)	16 (2/2)	24 (4/2)	26 (4/2)	20 (4/2)	32 (6/2)
(An)	16 (4/0)	16 (4/0)	24 (4/2)	24 (4/2)	24 (4/2)	32 (6/2)	34 (6/2)	32 (6/2)	40 (8/2)
(An) +	16 (4/0)	16 (4/0)	24 (4/2)	24 (4/2)	24 (4/2)	32 (6/2)	34 (6/2)	32 (6/2)	40 (8/2)
-(An)	18 (4/0)	18 (4/0)	26 (4/2)	26 (4/2)	26 (4/2)	34 (6/2)	32 (6/2)	34 (6/2)	42 (8/2)
d16 (An)	24 (6/0)	24 (6/0)	32 (6/2)	32 (6/2)	32 (6/2)	40 (8/2)	44 (8/2)	40 (8/2)	48 (10/2)
d8(An, Xn)*	26 (6/0)	26 (6/0)	34 (6/2)	34 (6/2)	34 (6/2)	42 (8/2)	44 (8/2)	42 (8/2)	50 (10/2)
Abs.W	24 (6/0)	24 (6/0)	32 (6/2)	32 (6/2)	32 (6/2)	40 (8/2)	42 (8/2)	40 (8/2)	48 (10/2)
Abs.L	32 (8/0)	32 (8/0)	40 (8/2)	40 (8/2)	40 (8/2)	48 (10/2)	50 (10/2)	48 (10/2)	56 (12/2)
d16(PC)	24 (6/0)	24 (6/0)	32 (6/2)	32 (6/2)	32 (6/2)	40 (8/2)	42 (8/2)	40 (8/2)	48 (10/2)
d8(PC, Xn)*	26 (6/0)	26 (6/0)	34 (6/2)	34 (6/2)	34 (6/2)	42 (8/2)	44 (8/2)	42 (8/2)	50 (10/2)
#<data>	16 (4/0)	16 (4/0)	24 (4/2)	24 (4/2)	24 (4/2)	32 (6/2)	34 (6/2)	32 (6/2)	40 (8/2)

* : The size of the index register (Xn) does not affect execution time.

Table E.4 Move Long Word Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An) +	-(An)	d16 (An)	d8 (An,Xn)*	Abs.W	Abs.L
Dn	8 (2/0)	8 (2/0)	24 (2/4)	24 (2/4)	24 (2/4)	32 (4/4)	34 (4/4)	32 (4/4)	40 (6/4)
An	8 (2/0)	8 (2/0)	24 (2/4)	24 (2/4)	24 (2/4)	32 (4/4)	34 (4/4)	32 (4/4)	40 (6/4)
(An)	24 (6/0)	24 (6/0)	40 (6/4)	40 (6/4)	40 (6/4)	48 (8/4)	50 (8/4)	48 (8/4)	56 (10/4)
(An) +	24 (6/0)	24 (6/0)	40 (6/4)	40 (6/4)	40 (6/4)	48 (8/4)	50 (8/4)	48 (8/4)	56 (10/4)
-(An)	26 (6/0)	26 (6/0)	42 (6/4)	42 (6/4)	42 (6/4)	50 (8/4)	52 (8/4)	50 (8/4)	58 (10/4)
d16 (An)	32 (8/0)	32 (8/0)	48 (8/4)	48 (8/4)	48 (8/4)	56 (10/4)	58 (10/4)	56 (10/4)	64 (12/4)
d8 (An, Xn)*	34 (8/0)	34 (8/0)	50 (8/4)	50 (8/4)	50 (8/4)	58 (10/4)	60 (10/4)	58 (10/4)	66 (12/4)
Abs.W	32 (8/0)	32 (8/0)	48 (8/4)	48 (8/4)	48 (8/4)	56 (10/4)	58 (10/4)	56 (10/4)	64 (12/4)
Abs.L	40 (10/0)	40 (10/0)	56 (10/4)	56 (10/4)	56 (10/4)	64 (12/4)	66 (12/4)	64 (12/4)	72 (14/4)
d16 (PC)	32 (8/0)	32 (8/0)	48 (8/4)	48 (8/4)	48 (8/4)	56 (10/4)	58 (10/4)	56 (10/4)	64 (12/4)
d8(PC, Xn)*	34 (8/0)	34 (8/0)	50 (8/4)	50 (8/4)	50 (8/4)	58 (10/4)	60 (10/4)	58 (10/4)	66 (12/4)
#<data>	24 (6/0)	24 (6/0)	40 (6/4)	40 (6/4)	40 (6/4)	48 (8/4)	50 (8/4)	48 (8/4)	56 (10/4)

* : The size of the index register (Xn) does not affect execution time.

E.4 STANDARD INSTRUCTION EXECUTION TIMES

The number of clock periods shown in Table E.5 indicates the time required to perform the operations, store the results, and read the next instruction. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated. In Table E.5 the headings have the following meanings: An = address register operand, Dn = data register operand, ea =an operand specified by an effective address, and M = memory effective address operand.

Table E.5 Standard Instruction Execution Times

Instruction	Size	op<ea>, An	op<ea>, Dn	op Dn, <M>
ADD / ADDA	Byte	–	8 (2/0) +	12 (2/1) +
	Word	12 (2/0) +	8 (2/0) +	16 (2/2) +
	Long word	10 (2/0) + **	10 (2/0) + **	24 (2/4) +
AND	Byte	–	8 (2/0) +	12 (2/1) +
	Word	–	8 (2/0) +	16 (2/2) +
	Long word	–	10 (2/0) + **	24 (2/4) +
CMP / CMPA	Byte	–	8 (2/0) +	–
	Word	10 (2/0) +	8 (2/0) +	–
	Long word	10 (2/0) +	10 (2/0) +	–
DIVS		–	162 (2/0) *	–
DIVU		–	144 (2/0) *	–
EOR	Byte	–	8 (2/0) + ***	12 (2/1) +
	Word	–	8 (2/0) + ***	16 (2/2) +
	Long word	–	12 (2/0) + ***	24 (2/4) +
MULS		–	74 (2/0) + *	–
MULU		–	74 (2/0) + *	–
OR	Byte	–	8 (2/0) +	12 (2/1) +
	Word	–	8 (2/0) +	16 (2/2) +
	Long word	–	10 (2/0) + **	24 (2/4) +
SUB	Byte	–	8 (2/0) +	12 (2/1) +
	Word	12 (2/0) +	8 (2/0) +	16 (2/2) +
	Long word	10 (2/0) + **	10 (2/0) **	24 (2/4) +

Notes:

\+ : add effective address calculation time

* : Indicates maximum base value added to word effective address time.

** : The base time of 10 clock periods is increased to 12 if the effective address mode is register direct or immediate (effective address time should also be added).

*** Only available effective address mode is data register direct.

DIVS, DIVU – The divide algorithm used by the TMP68008 provides less than 10% difference between the best and worst case timings.

MULS, MULU – The multiply algorithm requires 42+2n clocks where n is defined as:

MULS n=tag the <ea> with a zero as the MSB; n is the resultant number of 10 or 01 patterns in the 17-bit source, i.e., worst case happens when the source is $5555.

MULU n=the number of ones in the <ea>

E.5 IMMEDIATE INSTRUCTION EXECUTION TIMES

The number of clock periods shown in Table E.6 includes the time to fetch immediate operands, perform the operations,store the results, and read the next operation. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated. In Table E.6, the headings have the follwing meanings: # = immediate operand, Dn = data register operand, An= address register operand, and M = memory operand.

Table E.6 Immediate Instruction Execution Times

Instruction	Size	op #, Dn	op #, An	op #, M
ADDI	Byte	16 (4/0)	—	20 (4/1) +
	Word	16 (4/0)	—	24 (4/2) +
	Long word	28 (6/0)	—	40 (6/4) +
ADDQ	Byte	8 (2/0)	—	12 (2/1) +
	Word	8 (2/0)	12 (2/0)	16 (2/2) +
	Lon word	12 (2/0)	12 (2/0)	24 (2/4) +
ANDI	Byte	16 (4/0)	—	20 (4/1) +
	Word	16 (4/0)	—	24 (4/2) +
	Lon word	28 (6/0)	—	40 (6/4) +
CMPI	Byte	16 (4/0)	—	16 (4/0) +
	Word	16 (4/0)	—	16 (4/0) +
	Lon word	26 (6/0)	—	24 (6/0) +
EORI	Byte	16 (4/0)	—	20 (4/1) +
	Word	16 (4/0)	—	24 (4/2) +
	Lon word	28 (6/0)	—	40 (6/4) +
MOVEQ	Lon word	8 (2/0)	—	—
ORI	Byte	16 (4/0)	—	20 (4/1) +
	Word	16 (4/0)	—	24 (4/2) +
	Lon word	28 (6/0)	—	40 (6/4) +
SUBI	Byte	16 (4/0)	—	12 (2/1) +
	Word	16 (4/0)	—	16 (2/2) +
	Lon word	28 (6/0)	—	24 (2/4) +
SUBQ	Byte	8 (2/0)	—	20 (4/1) +
	Word	8 (2/0)	12 (2/0)	24 (4/2) +
	Lon word	12 (2/0)	12 (2/0)	40 (6/4) +

+ : add effective address calculation time

E.6 SINGLE OPERAND INSTRUCTION EXECUTION TIMES

Table E.7 indicates the number of clock periods for the single operand instructions. The number of bus read andwrite cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table E.7 Single Operand Instruction Execution Times

Instruction	Size	Register	Memory
CLR	Byte	8 (2/0)	12 (2/1) +
	Word	8 (2/0)	16 (2/2) +
	Long word	10 (2/0)	24 (2/4) +
NBCD	Byte	10 (2/0)	12 (2/1) +
NEG	Byte	8 (2/0)	12 (2/1) +
	Word	8 (2/0)	16 (2/2) +
	Long word	10 (2/0)	24 (2/4) +
NEGX	Byte	8 (2/0)	12 (2/1) +
	Word	8 (2/0)	16 (2/2) +
	Long word	10 (2/0)	24 (2/4) +
NOT	Byte	8 (2/0)	12 (2/1) +
	Word	8 (2/0)	16 (2/2) +
	Long word	10 (2/0)	24 (2/4) +
Scc	Byte, False	8 (2/0)	12 (2/1) +
	Byte, True	10 (2/0)	12 (2/1) +
TAS	Byte	8 (2/0)	14 (2/1) +
TST	Byte	8 (2/0)	8 (2/0) +
	Word	8 (2/0)	8 (2/0) +
	Long word	8 (2/0)	8 (2/0) +

+ : add effective address calculation time.

E.7 SHIFT/ROTATE INSTRUCTION EXECUTION TIMES

Table E.8 indicates the number of clock periods for the shift and rotate instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table E.8 Shift/Rotate Instruction Execution Times

Instruction	Size	Register	Memory
ASR, ASL	Byte	10 + 2n (2/0)	–
	Word	10 + 2n (2/0)	16 (2/2) +
	Long word	12 + 2n (2/0)	–
LSR, LSL	Byte	10 + 2n (2/0)	–
	Word	10 + 2n (2/0)	16 (2/2) +
	Long word	12 + 2n (2/0)	–
ROR, ROL	Byte	10 + 2n (2/0)	–
	Word	10 + 2n (2/0)	16 (2/2) +
	Long word	12 + 2n (2/0)	–
ROXR, ROXL	Byte	10 + 2n (2/0)	–
	Word	10 + 2n (2/0)	16 (2/2) +
	Long word	12 + 2n (2/0)	–

+ : add effective address calculation time for word operands
n : is the shift count

E.8 BIT MANIPULATION INSTRUCTION EXECUTION TIMES

Table E.9 indicates the number of clock periods required for the bit manipulation instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table E.9 Bit Manipulation Instruction Execution Times

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	–	12 (2/1) +	–	20 (4/1) +
	Long word	12 (2/0) *	–	20 (4/0)*	–
BCLR	Byte	–	12 (2/1) +	–	20 (4/1) +
	Long word	14 (2/0) *	–	22 (4/0)*	–
BSET	Byte	–	12 (2/1) +	–	20 (4/1) +
	Long word	12 (2/0) *	–	20 (4/0)*	–
BTST	Byte	–	8 (2/0) +	–	16 (4/0) +
	Long word	10 (2/0) *	–	18 (4/0)*	–

\+ : add effective address calculation time
* : Indicates maximum value; data addressing mode only

E.9 CONDITIONAL INSTRUCTION EXECUTION TIMES

Table E.10 indicates the number of clock periods required for the conditional instructions. The number of bus read and write cycles is indicated in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table E.10 Conditional Instruction Execution Times

Instruction	Displacement	Trap or Branch Taken	Trap or Branch Not taken
Bcc	Byte	18 (4/0)	12 (2/0)
	Word	18 (4/0)	20 (4/0)
BRA	Byte	18 (4/0)	–
	Word	18 (4/0)	–
BSR	Byte	34 (4/4)	–
	Word	34 (4/4)	–
DBcc	cc True	–	20 (4/0)
	cc False	18 (4/0)	26 (6/0)
CHK	–	68 (8/6) + *	14 (2/0) +
TRAP	–	62 (8/6)	–
TRAPV	–	66 (10/6)	8 (2/0)

\+ : add effective address calculation time for word operand
* : indicates maximum base value

E.10 JMP, JSR, LEA, PEA, AND MOVEM INSTRUCTION EXECUTION TIMES

Table E.11 indicates the number of clock periods required for the jump, jump-to-subroutine, load effective address, push effective address, and move multiple registers instructions. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table E.11 JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

Instruction	Size	(An)	(An)+	-(An)	d16(An)	d8(An, Xn)*	Abs,W	Abs,L	d16(PC)	d8(PC, Xn)*
JMP	–	16 (4/0)	–	–	18 (4/0)	22 (4/0)	18 (4/0)	24 (6/0)	18 (4/0)	22 (4/0)
JSR	–	32 (4/4)	–	–	34 (4/4)	38 (4/4)	34 (4/4)	40 (6/4)	34 (4/4)	32 (4/4)
LEA	–	8 (2/0)	–	–	16 (4/0)	20 (4/0)	16 (4/0)	24 (6/0)	16 (4/0)	20 (4/0)
PEA	–	24 (2/4)	–	–	32 (4/4)	36 (4/4)	32 (4/4)	40 (6/4)	32 (4/4)	36 (4/4)
MOVEN N→R	Word	24+8n (6+2n/0)	24+8n (6+2n/0)	– –	32+8n (8+2n/0)	34+8n (8+2n/0)	32+8n (10+n/0)	40+8n (10+2n/0)	32+8n (8+2n/0)	34+8n (8+2n/0)
	Long word	24+16n (6+4n/0)	24+16n (6+4n/0)	– –	32+16n (8+4n/0)	34+16n (8+4n/0)	32+16n (8+4n/0)	40+16n (8+4n/0)	32+16n (8+4n/0)	32+16n (8+4n/0)
MOVEN R→N	Word	16+8n (4/2n)	– –	16+8n (4/2n)	24+8n (6/2n)	26+8n (6/2n)	24+8n (6/2n)	32+8n (8/2n)	– –	– –
	Long word	16+16n (4/4n)	– –	16+16n (4/4n)	24+16n (6/4n)	26+16n	24+16n (8/4n)	32+16n (6/4n)	– –	– –

n : the number of registers to move

* : the size of the index register (Xn) does not affect the instruction's execution time

E.11 MULTI-PRECISION INSTRUCTION EXECUTION TIMES

Table E.12 indicates the number of clock periods for the multi-precision instructions. The number of clock periods includes the time to fetch both operands, perform the operations, store the results, and read the next instructions. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table E.12 Multi-Precision Instruction Execution Times

Instruction	Size	OP Dn, Dn	OP M, M
ADDX	Byte	8 (2/0)	22 (4/1)
	Word	8 (2/0)	50 (6/2)
	Long word	12 (2/0)	58 (10/4)
CMPM	Byte	–	16 (4/0)
	Word	–	24 (6/0)
	Long word	–	40 (10/0)
SUBX	Byte	8 (2/0)	22 (4/1)
	Word	8 (2/0)	50 (6/2)
	Long word	12 (2/0)	58 (10/4)
ABCD	Byte	10 (2/0)	20 (4/1)
SBCD	Byte	10 (2/0)	20 (4/1)

E.12 MISCELLANEOUS INSTRUCTION EXECUTION TIMES

Table E.13 and E.14 indicate the number of clock periods for the folowing miscellaneous instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods plus the number of read and write cycles must be added to those of the effective address calculation where indicated.

Table E.13 Miscellaneous Instruction Execution Times

Instruction	Register	Memory
ANDI to CCR	32 (6/0)	–
ANDI to SR	32 (6/0)	–
EORI to CCR	32 (6/0)	–
EORI to SR	32 (6/0)	–
EXG	10 (2/0)	–
EXT	8 (2/0)	–
LINK	32 (4/4)	–
MOVE to CCR	18 (4/0)	18 (4/0) +
MOVE to SR	18 (4/0)	18 (4/0) +
MOVE from SR	10 (2/0)	16 (2/2) +
MOVE to USP	8 (2/0)	–
MOVE from USP	8 (2/0)	–
NOP	8 (2/0)	–
ORI to CCR	32 (6/0)	–
ORI to SR	32 (6/0)	–
RESET	136 (2/0)	–
RTE	40 (10/0)	–
RTR	40 (10/0)	–
RTS	32 (8/0)	–
STOP	4 (0/0)	–
SWAP	8 (2/0)	–
UNLK	24 (6/0)	–

+ : add effective address calculation time for word operand

Table E.14 Move Peripheral Instruction Execution Times

Instruction	Size	Register→Memory	Memory→Register
MOVEP	Word	24 (4/2)	24 (6/0)
	Long word	32 (4/4)	32 (8/0)

+ : add effective address calculation time

E.13 EXCEPTION PROCESSING EXECUTION TIMES

Table E.15 indicates the number of clock periods for exception processing. The number of clock periods includes the time for all stacking, the vector fetch, and the fetch of the first instruction of the handler routine. The number of bus read and write cycles is shown in parenthesis as: (r/w).

Table E.15 Exception Processing Execution Times

Exception	Periods
Address Error	94 (8/14)
Bus Error	94 (8/14)
CHK Instruction	68 (8/6) +
Divide by Zero	66 (8/6) +
Interrupt	72 (9/6)*
Illegal Instruction	62 (8/6)
Privileged Instruction	62 (8/6)
$\overline{\text{RESET}}$**	64 (12/0)
Trace	62 (8/6)
TRAP Instruction	62 (8/6)
TRAPV Instruction	66 (10/6)

\+ : add effective address calculation time

** : Indicates the time from when $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are first sampled as negated to when instruction execution starts.

APPENDIX F TMP68010 INSTRUCTION EXECUTION TIMES

F.1 INTRODUCTION

This Appendix contains listings of the instruction execution times in terms of external clock (CLK) periods. In this data, it is assumed that both memory read and write cycle times are four clock periods. A longer memory cycle will cause the generation of wait states which must be added to the total instruction time.

The number of bus read and write cycles for each instruction is also included with the timing data. This data is enclosed in parenthesis following the number of clock periods and is shown as: (r/w) where r is the number of read cycles and w is the number of write cycles included in the clock period number. Recalling that either a read or write cycle requires four clock periods, a timing number given as 18(3/1) relates to 12 clock periods for the three read cycles, puls 4 clock periods for the one write cycle, puls 2 cycles required for some internal function of the processor.

Note:

The number of periods includes instruction fetch and all applicable operand fetches and stores.

F.2 OPERAND EFFECTIVE ADDRESS CALCULATION TIMES

Table F.1 lists the number of clock periods required to compute an instruction's effective address. It includes fetching of any extension words if necessary. Several instructions do not need the operand at the effective address to be fetched and thus require fewer clock periods to calculate a given effective address than the instructions that fetch the effective address operand. The number of bus read and write cycles is shown in parenthesis as (r/w). Note there are no write cycles involved in processing the effective address.

Table F.1 Effective Address Calculation Times

Addressing Mode		Byte, Word		Long word	
		Fetch	No Fetch	Fetch	No Fetch
	Register				
Dn	Data Register Direct	0 (0/0)	-	0 (0/0)	-
An	Address Register Direct	0 (0/0)	-	0 (0/0)	-
	Memory				
(An)	Address Register Indirect	4 (1/0)	2 (0/0)	8 (2/0)	2 (2/0)
(An) +	Address Register Indirect with Postincrement	4 (1/0)	4 (0/0)	8 (2/0)	4 (0/0)
-An	Address Register Indirect with Predecrement	6 (1/0)	4 (0/0)	10 (2/0)	4 (0/0)
d16 (An)	Address Register Indirect with Displacement	8 (2/0)	4 (0/0)	12 (3/0)	4 (1/0)
d8 (An, Xn)*	Address Register Indirect with Index	10 (2/0)	8 (1/0)	14 (3/0)	8 (1/0)
Abs.W	Absolute Short	8 (2/0)	4 (1/0)	12 (3/0)	4 (1/0)
Abs.L	Absolute Long	12 (3/0)	8 (2/0)	16 (4/0)	8 (2/0)
d16 (PC)	Program Counter with Displacement	8 (2/0)	-	12 (3/0)	-
d8 (PC, Xn)*	Program Counter with Index	10 (2/0)	-	14 (3/0)	-
#<data>	Immediate	4 (1/0)	-	8 (2/0)	-

*: The size of the index register (Xn) does not affect execution time.

F.3 MOVE INSTRUCTION EXECUTION TIMES

Table F.2, F.3, F.4 and F.5 indicate the number of clock periods for the move instruction. This data includes instruction fetch, operand reads, and operand writes. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table F.2 Move Byte and Word Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An) +	- (An)	d16 (An)	d8 (An, Xn)*	Abs.W	Abs.L
Dn	4 (1/0)	4 (1/0)	8 (1/1)	8 (1/1)	8 (1/1)	12 (2/1)	14 (2/1)	12 (2/1)	16 (3/1)
An	4 (1/0)	4 (1/0)	8 (1/1)	8 (1/1)	8 (1/1)	12 (2/1)	14 (2/1)	12 (2/1)	16 (3/1)
(An)	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	16 (3/1)	18 (3/1)	16 (3/1)	20 (4/1)
(An) +	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	16 (3/1)	18 (3/1)	16 (3/1)	20 (4/1)
- (An)	10 (2/0)	10 (2/0)	14 (2/1)	14 (2/1)	14 (2/1)	18 (3/1)	20 (3/1)	18 (3/1)	22 (4/1)
d16 (An)	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	20 (4/1)	22 (4/1)	20 (4/1)	24 (4/1)
d8 (An, Xn) *	14 (3/0)	14 (3/0)	18 (3/1)	18 (3/1)	18 (3/1)	22 (4/1)	24 (4/1)	22 (4/1)	26 (5/1)
Abs.W	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	20 (4/1)	22 (4/1)	20 (4/1)	24 (5/1)
Abs.L	16 (4/0)	16 (4/0)	20 (4/1)	20 (4/1)	20 (4/1)	24 (5/1)	26 (5/1)	24 (5/1)	28 (6/1)
d16 (PC)	12 (3/0)	12 (3/0)	16 (3/1)	16 (3/1)	16 (3/1)	20 (4/1)	22 (4/1)	20 (4/1)	24 (5/1)
d8 (PC, Xn) *	14 (3/0)	14 (3/0)	18 (3/1)	18 (3/1)	18 (3/1)	22 (4/1)	24 (4/1)	22 (4/1)	26 (5/1)
#<data>	8 (2/0)	8 (2/0)	12 (2/1)	12 (2/1)	12 (2/1)	16 (3/1)	18 (3/1)	16 (3/1)	20 (4/1)

*: The size of the index register (Xn) does not affect execution time.

Table F.3 Move Byte and Word Instruction Loop Mode Execution Times

Source	Loop Continued			Loop Terminated					
	Valid Count, cc False			Valid Count, cc True			Expired Count		
	Destination								
	(An)	(An) +	- (An)	(An)	(An) +	- (An)	(An)	(An) +	- (An)
Dn	10 (0 / 1)	10 (0 / 1)	–	18 (2 / 1)	18 (2 / 1)	–	16 (2 / 1)	16 (2 / 1)	–
An*	10 (0 / 1)	10 (0 / 1)	–	18 (2 / 1)	18 (2 / 1)	–	16 (2 / 1)	16 (2 / 1)	–
(An)	14 (1 / 1)	14 (1 / 1)	16 (1 / 1)	20 (3 / 1)	20 (3 / 1)	22 (3 / 1)	18 (3 / 1)	18 (3 / 1)	20 (3 / 1)
(An) +	14 (1 / 1)	14 (1 / 1)	16 (1 / 1)	20 (3 / 1)	20 (3 / 1)	22 (3 / 1)	18 (3 / 1)	18 (3 / 1)	20 (3 / 1)
- (An)	16 (1 / 1)	16 (1 / 1)	18 (1 / 1)	22 (3 / 1)	22 (3 / 1)	24 (3 / 1)	20 (3 / 1)	20 (3 / 1)	22 (3 / 1)

*:

Table F.4 Move Long Word Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An) +	- (An)	d16 (An)	d8 (An,Xn)*	Abs.W	Abs.L
Dn	4 (1 / 0)	4 (1 / 0)	12 (1 / 2)	12 (1 / 2)	14 (1 / 2)	16 (2 / 2)	18 (2 / 2)	16 (2 / 2)	20 (3 / 2)
An	4 (1 / 0)	4 (1 / 0)	12 (1 / 2)	12 (1 / 2)	14 (1 / 2)	16 (2 / 2)	18 (2 / 2)	16 (2 / 2)	20 (3 / 2)
(An)	12 (3 / 0)	12 (3 / 0)	20 (3 / 2)	20 (3 / 2)	20 (3 / 2)	24 (4 / 2)	26 (4 / 2)	24 (4 / 2)	28 (5 / 2)
(An) +	12 (3 / 0)	12 (3 / 0)	20 (3 / 2)	20 (3 / 2)	20 (3 / 2)	24 (4 / 2)	26 (4 / 2)	24 (4 / 2)	28 (5 / 2)
- (An)	14 (3 / 0)	14 (3 / 0)	22 (3 / 2)	22 (3 / 2)	22 (3 / 2)	26 (4 / 2)	28 (4 / 2)	26 (4 / 2)	30 (5 / 2)
d16 (An)	16 (4 / 0)	16 (4 / 0)	24 (4 / 2)	24 (4 / 2)	24 (4 / 2)	28 (5 / 2)	30 (5 / 2)	28 (5 / 2)	32 (6 / 2)
d8 (An, Xn) *	18 (4 / 0)	18 (4 / 0)	26 (4 / 2)	26 (4 / 2)	26 (4 / 2)	30 (5 / 2)	32 (5 / 2)	30 (5 / 2)	34 (6 / 2)
Abs.W	16 (4 / 0)	16 (4 / 0)	24 (4 / 2)	24 (4 / 2)	24 (4 / 2)	28 (5 / 2)	30 (5 / 2)	28 (5 / 2)	32 (6 / 2)
Abs.L	20 (5 / 0)	20 (5 / 0)	28 (5 / 2)	28 (5 / 2)	28 (5 / 2)	32 (6 / 2)	34 (6 / 2)	32 (6 / 2)	36 (7 / 2)
d16 (PC)	16 (4 / 0)	16 (4 / 0)	24 (4 / 2)	24 (4 / 2)	24 (4 / 2)	28 (5 / 2)	30 (5 / 2)	28 (5 / 2)	32 (6 / 2)
d8 (PC, Xn) *	18 (4 / 0)	18 (4 / 0)	26 (4 / 2)	26 (4 / 2)	26 (4 / 2)	30 (5 / 2)	32 (5 / 2)	30 (5 / 2)	34 (6 / 2)
#<data>	12 (3 / 0)	12 (3 / 0)	20 (3 / 2)	20 (3 / 2)	20 (3 / 2)	24 (4 / 2)	26 (4 / 2)	24 (4 / 2)	28 (5 / 2)

*: The size of the index register (Xn) does not affect execution

Table F.5 Move Long Word Instruction Loop Mode Execution Times

Source	Loop Continued			Loop Terminated					
	Valid Count, cc False			Valid Count, cc True			Expired Count		
	Destination								
	(An)	(An) +	- (An)	(An)	(An) +	- (An)	(An)	(An) +	- (An)
Dn	14 (0 / 2)	14 (0 / 2)	–	20 (2 / 2)	20 (2 / 2)	–	18 (2 / 2)	18 (2 / 2)	–
An	14 (0 / 2)	14 (0 / 2)	–	20 (2 / 2)	20 (2 / 2)	–	18 (2 / 2)	18 (2 / 2)	–
(An)	22 (2 / 2)	22 (2 / 2)	24 (2 / 2)	28 (4 / 2)	28 (4 / 2)	30 (4 / 2)	24 (4 / 2)	24 (4 / 2)	26 (4 / 2)
(An) +	22 (2 / 2)	22 (2 / 2)	24 (2 / 2)	28 (4 / 2)	28 (4 / 2)	30 (4 / 2)	24 (4 / 2)	24 (4 / 2)	26 (4 / 2)
- (An)	24 (2 / 2)	24 (2 / 2)	26 (2 / 2)	30 (4 / 2)	30 (4 / 2)	32 (4 / 2)	26 (4 / 2)	26 (4 / 2)	28 (4 / 2)

F.4 STANDARD INSTRUCTION EXECUTION TIMES

The number of clock periods shown in Tables F.6 and F.7 indicate the time required to perform the operations, store the results, and read the next instruction. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Tables F.6 and F.7 the headings have the following meanings: An = address register operand, Dn = data register operand, ea. = an operand specified by an effective address, and M = memory effective address operand.

Table F.6 Standard Instruction Execution Times

Instruction	Size	op<ea>, An***	op<ea>, Dn	op Dn, <M>
ADD / ADDA	Byte, Word	8 (1 / 0) +	4 (1 / 0) +	8 (1 / 1) +
	Long word	6 (1 / 0) +	6 (1 / 0) +	12 (1 / 2) +
AND	Byte, Word	—	4 (1 / 0) +	8 (1 / 1) +
	Long	—	6 (1 / 0) +	12 (1 / 2) +
CMP / CMPA	Byte, Word	6 (1 / 0) +	4 (1 / 0) +	—
	Long word	6 (1 / 0) +	6 (1 / 0) +	—
DIVS	—	—	122 (1 / 0) +	—
DIVU	—	—	108 (1 / 0) +	—
EOR	Byte, Word	—	4 (1 / 0) **	8 (1 / 1) +
	Long word	—	6 (1 / 0) **	12 (1 / 2) +
MULS	—	—	42 (1 / 0) +	—
MULU	—	—	40 (1 / 0) +	—
OR	Byte, Word	—	4 (1 / 0) +	8 (1 / 1) +
	Long word	—	6 (1 / 0) +	12 (1 / 2) +
SUB / SUBA	Byte, Word	8 (1 / 0) +	4 (1 / 0) +	8 (1 / 1) +
	Long word	6 (1 / 0) +	6 (1 / 0) +	12 (1 / 2) +

Notes :

+	:	add effective address calculation time
*	:	Indicates maximum value
**	:	only available addressing mode is data register direct
***	:	word or long word only

Table F.7 Standard Instruction Loop Mode Execution Times

Instruction	Size	Loop Continued			Loop Terminated					
		Valid Count, cc False			Valid Count cc True			Expired Count		
		op<ea> An*	op<ea> Dn	op Dn <ea>	op<ea> An*	op<ea> Dn	op Dn <ea>	op<ea> An*	op<ea> Dn	op Dn <ea>
ADD	Byte, Word	18 (1 / 0)	18 (1 / 0)	16 (1 / 1)	24 (3 / 0)	22 (3 / 0)	22 (3 / 1)	22 (3 / 0)	20 (3 / 0)	20 (3 / 1)
	Long word	22 (2 / 0)	22 (2 / 0)	24 (2 / 2)	28 (4 / 0)	28 (4 / 0)	30 (4 / 2)	26 (4 / 0)	26 (4 / 0)	28 (4 / 2)
AND	Byte, Word	–	16 (1 / 1)	16 (1 / 1)	–	22 (3 / 0)	22 (3 / 1)	–	20 (3 / 0)	20 (3 / 1)
	Long word	–	22 (2 / 0)	24 (2 / 2)	–	28 (4 / 0)	30 (4 / 2)	–	26 (4 / 0)	28 (4 / 2)
CMP	Byte, Word	12 (1 / 0)	12 (1 / 0)	–	18 (3 / 0)	18 (3 / 0)	–	16 (3 / 0)	16 (4 / 0)	
	Long word	18 (2 / 0)	18 (1 / 0)	–	24 (4 / 0)	24 (4 / 0)	–	20 (4 / 1)	20 (4 / 0)	
EOR	Byte, Word	–	–	16 (1 / 0)	–	–	22 (3 / 1)	–	–	20 (3 / 1)
	Long word	–	–	24 (2 / 2)	–	–	30 (4 / 2)	–	–	28 (4 / 2)
OR	Byte, Word	–	16 (1 / 0)	16 (1 / 0)	–	22 (3 / 0)	22 (3 / 1)	–	20 (3 / 0)	20 (3 / 1)
	Long word	–	22 (2 / 0)	24 (2 / 2)	–	28 (4 / 0)	30 (4 / 2)	–	26 (4 / 0)	28 (4 / 2)
SUB	Byte, Word	18 (1 / 0)	16 (1 / 0)	16 (1 / 1)	24 (3 / 0)	22 (3 / 0)	22 (3 / 1)	22 (3 / 0)	20 (3 / 0)	20 (3 / 1)
	Long word	22 (2 / 0)	20 (2 / 0)	24 (2 / 2)	28 (4 / 0)	26 (4 / 0)	30 (4 / 2)	26 (4 / 1)	24 (4 / 0)	28 (4 / 2)

* : Word or long only. <ea> may be (An), (An)+ or –(An) only. Add two clock periods to the table value if <ea> is –(An).

F.5 IMMEDIATE INSTRUCTION EXECUTION TIMES

The number of clock periods shown in Table F.8 includes the time to fetch immediate operands, perform the operations, store the results, and read the next operation. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table F.8, the headings have the follwing meanings: # = immediate operand, Dn = data register operand, An= address register operand, and M = memory operand.

Table F.8 Immediate Instruction Execution Times

Instruction	Size	op #, Dn	op #, An	op #, M
ADDI	Byte, Word	8 (2 / 0)	–	12 (2 / 1) +
	Long word	14 (3 / 0)	–	20 (3 / 2) +
ADDQ	Byte, Word	4 (1 / 0)	4 (1 / 0) *	8 (1 / 1) +
	Long word	8 (1 / 0)	8 (1 / 0)	12 (1 / 2) +
ANDI	Byte, Word	8 (2 / 0)	–	12 (2 / 1) +
	Long word	14 (3 / 0)	–	20 (3 / 1) +
CMPI	Byte, Word	8 (2 / 0)	–	8 (2 / 0) +
	Long word	12 (3 / 0)	–	12 (3 / 0) +
EORI	Long word	8 (2 / 0)	–	12 (2 / 1) +
	Byte, Word	14 (3 / 0)	–	20 (3 / 2) +
MOVEQ	Long word	4 (1 / 0)	–	–
ORI	Byte, Word	8 (2 / 0)	–	12 (2 / 1) +
	Long word	14 (3 / 0)	–	20 (3 / 2) +
SUBI	Byte, Word	8 (2 / 0)	–	12 (2 / 1) +
	Long word	14 (3 / 0)	–	20 (3 / 2) +
SUBQ	Byte, Word	4 (1 / 0)	4 (1 / 0) *	8 (1 / 1) +
	Long word	8 (/ 0)	8 (1 / 0)	12 (1 / 2) +

\+ : add effective address calculation time
* : word only

F.6 SINGLE OPERAND INSTRUCTION EXECUTION TIMES

Tables F.9, F.10, and F.11 indicate the number of clock periods for the single operand instructions. The numberof bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table F.9 Single Operand Instruction Execution Times

Instruction	Size	Register	Memory
NBDC	Byte	6 (1 / 0)	8 (1 / 1) +
NEG	Byte, Word	4 (1 / 0)	8 (1 / 1) +
	Long word	6 (1 / 0)	12 (1 / 2) +
NEGX	Byte, Word	4 (1 / 0)	8 (1 / 1) +
	Long word	6 (1 / 0)	12 (1 / 2) +
NOT	Byte, Word	4 (1 / 0)	8 (1 / 1) +
	Long word	6 (1 / 0)	12 (1 / 2) +
Scc	Byte, Faise	4 (1 / 0)	8 (1 / 1) + *
	Byte, True	4 (1 / 0)	8 (1 / 1) + *
TAS	Byte	4 (1 / 0)	14 (2 / 1) + *
TST	Byte, Word	4 (1 / 0)	4 (1 / 0) +
	Long word	4 (1 / 0)	4 (1 / 0) +

\+ : add effective address calculation time.
* : Use non-fetching effective address calculation time.

Table F.10 Clear Instruction Execution Times

Inst.	Size	Dn	An	(An)	(An) +	– (An)	d16 (An)	d8 (An, Xn)*	Abs. w	Abs.L
CLR	Byte, Word	4 (1/0)	—	8 (1 / 1)	8 (1 / 1)	10 (1 / 1)	12 (1 / 1)	16 (2 / 1)	12 (2 / 1)	16 (3 / 1)
	Long word	6 (1/0)	—	12 (1 / 2)	12 (1 / 2)	14 (1 / 2)	16 (2 / 2)	20 (2 / 2)	16 (2 / 2)	20 (3 / 2)

* : The size of the index register (Xn) does not affect execution time.

Table F.11 Single Operand Instruction Loop Mode Execution Times

		Loop Continued			Loop Terminated					
		Valid Count, cc False			Valid Count, cc True			Expired Count		
Inst.	Size	(An)	(An) +	– (An)	(An)	(An) +	– (An)	(An)	(An) +	– (An)
CLR	Byte, Word	10 (0 / 1)	10 (0 / 1)	12 (0 / 1)	18 (2 / 1)	18 (2 / 1)	20 (2 / 0)	16 (2 / 1)	16 (2 / 1)	18 (2 / 1)
	Long word	14 (0 / 2)	14 (0 / 2)	16 (0 / 2)	22 (2 / 2)	22 (2 / 2)	24 (2 / 2)	20 (2 / 2)	20 (2 / 2)	22 (2 / 2)
NBCD	Byte	18 (1 / 1)	18 (1 / 1)	20 (1 / 1)	24 (3 / 1)	24 (3 / 1)	26 (3 / 1)	22 (3 / 1)	22 (3 / 1)	24 (3 / 1)
NEG	Byte, Word	16 (1 / 1)	16 (1 / 1)	18 (2 / 2)	22 (3 / 1)	22 (3 / 1)	24 (3 / 1)	20 (3 / 1)	20 (3 / 1)	22 (3 / 1)
	Long word	24 (2 / 2)	24 (2 / 2)	26 (2 / 2)	30 (4 / 2)	30 (4 / 2)	32 (4 / 2)	28 (4 / 2)	28 (4 / 2)	30 (4 / 2)
NEGX	Byte, Word	16 (1 / 1)	16 (1 / 1)	18 (2 / 2)	22 (3 / 1)	22 (3 / 1)	24 (3 / 1)	20 (3 / 1)	20 (3 / 1)	22 (3 / 1)
	Long word	24 (2 / 2)	24 (2 / 2)	26 (2 / 2)	30 (4 / 2)	30 (4 / 2)	32 (4 / 2)	28 (4 / 2)	28 (4 / 2)	30 (4 / 2)
NOT	Byte, Word	16 (1 / 1)	16 (1 / 1)	18 (2 / 2)	22 (3 / 1)	22 (3 / 1)	24 (3 / 1)	20 (3 / 1)	20 (3 / 1)	22 (3 / 1)
	Long word	24 (2 / 2)	24 (2 / 2)	26 (2 / 2)	30 (4 / 2)	30 (4 / 2)	32 (4 / 2)	28 (4 / 2)	28 (4 / 2)	30 (4 / 2)
TST	Byte, Word	12 (1 / 0)	12 (1 / 0)	14 (1 / 0)	18 (3 / 0)	18 (3 / 0)	20 (3 / 0)	16 (3 / 1)	16 (3 / 1)	18 (3 / 0)
	Long word	18 (2 / 0)	18 (2 / 0)	20 (2 / 0)	24 (4 / 0)	24 (4 / 0)	26 (4 / 0)	20 (4 / 0)	20 (4 / 0)	22 (4 / 0)

F.7 SHIFT/ROTATE INSTRUCTION EXECUTION TIMES

Tables F.12 and F.13 indicate the number of clock periods for the shift and rotate instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table F.12 Shift/Rotate Instruction Execution Times

Instruction	Size	Register	Memory*
FSR, ASL	Byte, Word	6 + 2n (1 / 0)	8 (1 / 1) +
	Long word	8 + 2n (1 / 0)	–
LSR, LSL	Byte, Word	6 + 2n (1 / 0)	8 (1 / 1) +
	Long word	8 + 2n (1 / 0)	–
ROR, ROL	Byte, Word	6 + 2n (1 / 0)	8 (1 / 1) +
	Long word	8 + 2n (1 / 0)	–
ROXR, ROXL	Byte, Word	6 + 2n (1 / 0)	8 (1 / 1) +
	Long word	8 + 2n (1 / 0)	–

\+ : add effective address calculation time
n : the shift or rotate count
* : word only

Table F.13 Shift/Rotate Instruction Loop Mode Execution Times

		Loop Continued			Loop Terminated					
		Valid Count, cc False			Valid Count, cc True			Expired Count		
Instruction	Size	(An)	(An) +	– (An)	(An)	(An) +	– (An)	(An)	(An) +	– (An)
ASR, ASL	Word	18 (1 / 1)	18 (1 / 1)	20 (1 / 1)	24 (3 / 1)	24 (3 / 1)	26 (3 / 1)	22 (3 / 1)	22 (3 / 1)	24 (3 / 1)
LSR, LSL	Word	18 (1 / 1)	18 (1 / 1)	20 (1 / 1)	24 (3 / 1)	24 (3 / 1)	26 (3 / 1)	22 (3 / 1)	22 (3 / 1)	24 (3 / 1)
ROR, ROL	Word	18 (1 / 1)	18 (1 / 1)	20 (1 / 1)	24 (3 / 1)	24 (3 / 1)	26 (3 / 1)	22 (3 / 1)	22 (3 / 1)	24 (3 / 1)
ROXR, ROXL	Word	18 (1 / 1)	18 (1 / 1)	20 (1 / 1)	24 (3 / 1)	24 (3 / 1)	26 (3 / 1)	22 (3 / 1)	22 (3 / 1)	24 (3 / 1)

F.8 BIT MANIPULATION INSTRUCTION EXECUTION TIMES

Table F.14 indicates the number of clock periods required for the bit manipulation instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table F.14 Bit Manipulation Instruction Execution Times

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	—	8 (1/1) +	—	12 (2/1) +
	Long word	8 (1/0) *	—	12 (2/0) *	—
BCLR	Byte	—	10 (1/1) +	—	14 (2/1) +
	Long word	10 (1/0) *	—	14 (2/0) *	—
BSET	Byte	—	8 (1/1) +	—	12 (2/1) +
	Long word	8 (1/0) *	—	12 (2/0) *	—
BTST	Byte	—	4 (1/0) +	—	8 (2/0) +
	Long word	6 (1/0) *	—	10 (2/0) *	—

+ : add effective address calculation time.

* : indicates maximum value

F.9 CONDITIONAL INSTRUCTION EXECUTION TIMES

Table F.15 indicates the number of clock periods required for the conditional instructions. The number of bus read and write cycles is indicated in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table F.15 Conditional Instruction Execution Times

Instruction	Displacement	Branch Taken	Branch Not Taken
Bcc	Byte	10 (2 / 0)	6 (1 / 0)
	Word	10 (2 / 0)	10 (2 / 0)
BRA	Byte	10 (2 / 0)	—
	Word	10 (2 / 0)	—
BSR	Byte	18 (2 / 2)	—
	Word	18 (2 / 2)	—
DBcc	cc true	—	10 (2 / 2)
	cc false	10 (2 / 2)	16 (3 / 0)

+ : add effective address calculation time.

* : indicates maximum value

F.10 JMP, JSR, LEA, PEA, AND MOVEM INSTRUCTION EXECUTION TIMES

Table F.16 indicates the number of clock periods required for the jump, jump-to-subroutine, load effective address, push effective address, and move multiple registers instructions. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table F.16 JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

Inst.	Size	An	(An) +	- (An)	d16 (An)	d8 (An,Xn)*	Abs.W	Abs.L	d16 (PC)	d8 (PC,Xn)*
JMP	–	8 (2/0)	–	–	10 (2/0)	14 (3/0)	10 (2/0)	12 (3/0)	10 (2/0)	14 (3/0)
JSR	–	16 (2/2)	–	–	18 (2/2)	22 (2/2)	18 (2/2)	20 (3/2)	18 (2/2)	22 (2/2)
LEA	–	4 (1/0)	–	–	8 (2/0)	12 (2/0)	8 (2/0)	12 (3/0)	8 (2/0)	12 (2/0)
PEA	–	12 (1/2)	–	–	16 (2/2)	20 (2/2)	16 (2/2)	20 (3/2)	16 (2/2)	20 (2/2)
MOVEM M→R	Word	12 + 4n (3 + n/0)	12 + 4n (3 + n/0)	–	16 + 4n (4 + n/0)	18 + 4n (4 + n/0)	16 + 4n (4 + n/0)	20 + 4n (5 + n/0)	16 + 4n (4 + n/0)	18 + 4n (4 + n/0)
	Long word	12 + 8n (3 + 2n/0)	12 + 8n (3 + 2n/0	–	16 + 8n (4 + 2n/0)	18 + 8n (4 + 2n/0)	16 + 8n (4 + 2n/0)	20 + 8n (5 + n/0)	16 + 8n (4 + 2n/0)	16 + 8n (4 + 2n/0)
MOVEM R→M	Word	8 + 4n (2/n)	–	8 + 4n (2/n)	12 + 4n (3/n)	14 + 4n (3 + n)	12 + 4n (3/n)	16 + 4n (4/n)	– –	– –
	Long word	8 + 8n (2/2n)	– –	8 + 8n (2/2.ı)	12 + 8n (3/2n)	14 + 8n (3/2n)	12 + 8n (3/2n)	16 + 8n (4/n)	– –	– –
MOVES M→R	Byte, Word	18 (3/0)	20 (3/0)	20 (3/0)	20 (4/0)	24 (4/0)	20 (4/0)	24 (5/0)	–	–
	Long word	22 (4/0)	24 (4/0)	24 (4/0)	24 (5/0)	28 (5/0)	24 (5/0)	28 (6/0)	–	–
MOVES R→M	Byte, Word	18 (2/1)	20 (2/1)	20 (2/1)	20 (3/1)	24 (3/1)	20 (3/1)	24 (4/1)	–	–
	Long word	22 (2/2)	24 (2/2)	24 (2/2)	24 (3/2)	28 (3/21)	24 (3/2)	28 (4/2)	–	–

n : The number of registers to move.
* : The size of the index register (ix) does not affect the instruction's execution time.

F.11 MULTI-PRECISION INSTRUCTION EXECUTION TIMES

Table F.17 indicates the number of clock periods for the multi-precision instructions. The number of clock periods includes the time to fetch both operands, perform the operations, store the results, and read the next instructions. The number of read and write cycles is shown in parenthesis as (r/w).

In Table F.17, the headings have the following meanings: Dn = data register operand and M = memory operand.

Table F.17 Multi-Precision Instruction Execution Times

		Non-Looped		Loop Mode		
				Continued	Terminated	
				Valid Count, cc False	Valid Count, cc True	Expired Count
Instruction	Size	op Dn, Dn	op M, M *			
ADDX	Byte, Word	4 (1 / 0)	18 (3 / 1)	22 (2 / 1)	28 (4 / 1)	26 (4 / 1)
	Long word	6 (1 / 0)	30 (5 / 2)	32 (4 / 2)	38 (6 / 2)	36 (6 / 2)
CMPM	Byte, Word	—	12 (3 / 0)	14 (2 / 0)	20 (4 / 0)	18 (4 / 0)
	Long word	—	20 (5 / 0)	24 (4 / 0)	30 (6 / 0)	26 (6 / 0)
SUBX	Byte, Word	4 (1 / 0)	18 (3 / 1)	22 (2 / 1)	28 (4 / 1)	26 (4 / 1)
	Long word	6 (1 / 0)	30 (5 / 2)	32 (4 / 2)	38 (6 / 2)	36 (6 / 2)
ASCD	Byte	6 (1 / 0)	18 (3 / 1)	24 (2 / 1)	30 (4 / 1)	28 (4 / 1)
SBCD	Byte	6 (1 / 0)	18 (3 / 1)	24 (2 / 1)	30 (4 / 1)	28 (4 / 1)

* : Source and destination ea is (An) + for CMPM and – (An) for all others.

F.12 MISCELLANEOUS INSTRUCTION EXECUTION TIMES

Table F.18 indicates the number of clock periods for the following miscellaneous instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods plus the number of read and write cycles must be added to those of the effective address calculation where indicated.

Table F.18 Miscellaneous Instruction Execution Times

Instruction	Size	Register	Memory	Register** → Destination	Source** → Register
ANDI to CCR	–	16 (2/0)	–	–	–
ANDI to SR	–	16 (2/0)	–	–	–
CHK	–	8 (1/0) +	–	–	–
EORI to CCR	–	16 (2/0)	–	–	–
EORI to SR	–	16 (2/0)	–	–	–
EXG	–	6 (1/0)	–	–	–
EXT	Word	4 (1/0)	–	–	–
	Long word	4 (1/0)	–	–	–
LINK	–	16 (2/2)	–	–	–
MOVE from CCR	–	4 (1/0)	8(1/1) + *	–	–
MOVE to CCR	–	12 (2/0)	12(2/0) +	–	–
MOVE from SR	–	4 (1/0)	8(1/1) + *	–	–
MOVE to SR	–	12 (2/0)	12(2/0) +	–	–
MOVE from USP	–	6 (1/0)	–	–	–
MOVE to USP	–	6 (1/0)	–	–	–
MOVEC	–	–	–	10(2/0)	12(2/0)
MOVEP	Word	–	–	16(2/2)	16(4/0)
	Long word	–	–	24(2/4)	24(6/0)
NOP	–	4 (1/0)	–	–	–
ORI to CCR	–	16 (2/0)	–	–	–
ORI to SR	–	16 (2/0)	–	–	–
RESET	–	130 (1/0)	–	–	–
RTD	–	16 (4/0)	–	–	–
RTE	Short	24 (4/0)	–	–	–
	Long word, Retry Read	112(27/10)	–	–	–
	Long word, Retry Write	112 (26/1)	–	–	–
	Long word, No Retry	110 (26/0)	–	–	–
RTR	–	20 (5/0)	–	–	–
RTS	–	16 (4/0)	–	–	–
STOP	–	4 (0/0)	–	–	–
SWAP	–	4 (1/0)	–	–	–
TRAPV	–	4 (1/0)	–	–	–
UNLK	–	12 (3/0)	–	–	–

+ : add effective address calculation time.
* : use non-feching effective address calculation time.
** : Source or destination is a memory location for the MOVEP instruction and a control register for the MOVEC instruction.

F.13 EXCEPTION PROCESSING EXECUTION TIMES

Table F.19 indicates the number of clock periods for exception processing. The number of clock periods includes the time for all stacking, the vector fetch, and the fetch of the first two instruction words of the handler routine. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table F.19 Exception Processing Execution Times

Exception	
Address Error	126 (4 / 26)
Breadpoint Instruction*	42 (5 / 4)
Bus Error	126 (4 / 26)
CHK Instruction*	44 (5 / 4) +
Divide By Zero	42 (5 / 4) +
Illegal Instruction	38 (5 / 4)
Interrupt*	46 (5 / 4)
MOVEC, Illegal Control Register**	46 (5 / 4)
Privilege Violation	38 (5 / 4)
Reset***	40 (6 / 0)
RTE, Illegal Format	50 (7 / 4)
RTE, Illegal Revision	70 (12 / 4)
Trace	38 (4 / 4)
TRAP Instruction	38 (4 / 4)
TRAPV Instruction	38 (5 / 4)

\+ : Add effective address calculation time.
* : The interrupt acknowledge and breakpoint cycles are assumed to take four clock periods.
** : Indicates maximum value.
*** : Indicates the time from when $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are first sampled as negated to when instruction execution starts.

APPENDIX G TMP68010 LOOP MODE OPERATION

The TMP68010 has several features that provide efficient execution of program loops. One of these features is the DBcc looping primitive instruction. The DBcc instruction operates on three operands, a loop counter, a branch condition, and a branch displacement. When the DBcc is executed in loop mode, the contents of low order word of the register specified as the loop counter is decremented by one and compared to minus one. If equal to minus one, the result of the decrement is placed back into the count register and the next sequential instruction is executed, otherwise the condition code register is checked against the specified branch condition. If the condition is true, the result of the decrement is discarded and the next sequential instruction is executed. Finally, if the count register is not equal to minus one and the branch condition is false, the branch displacement is added to the program counter and instruction execution continues at that new address. Note that this is slightly different than non-looped execution; however, the results are the same.

An example of using the DBcc instruction in a simple loop for moving a block of data is shown in Figure G.1. In this program, the block of data `LENGTH' words long at address `SOURCE' is to be moved to address `DEST" provided that none of the words moved are equal to zero. When the effect of instruction prefetch on this loop is examined it can be seen that the bus activity during the loop execution would be:

1. Fetch the MOVE.W instruction,
2. Fetch the DBEQ instruction,
3. Read the operand where A0 points,
4. Write the operand where A1 points,
5. Fetch the DBEQ branch displacement, and
6. If loop conditions are met, return to step 1.

```
        LEA       SOURCE, A0      Load A Pointer To Source Data
        LEA       DEST, A1        Load A Pointer To Destination
        MOVE.W    #LENGTH, D0     Load The Counter Register
LOOP    MOVE.W    (A0)+, (A1)+    Loop To Move The Block Of Data
        DBEQ      D0, LOOP        Stop If Data Word Is Zero
```

Figure G.1 DBcc Loop Mode Program Example

During this loop, five bus cycles are executed; however, only two bus cycles perform the data movement. Since the TMP68010 has a two word prefetch queue in addition to a one word instruction decode register, it is evident that the three instruction fetches in this loop could be eliminated by placing the MOVE.W word in the instruction decode register and holding the DBEQ instruction and its branch displacement in the prefetch queue. The TMP68010 have the ability to do this by entering the loop mode of operation. During loop mode operation, all opcode fetches are suppressed and only operand reads and writes performed until an exit loop condition is met.

Loop mode operation is transparent to the programmer, with only two conditions required for the TMP68010 to enter the loop mode. First, a DBcc instruction must be executed with both branch conditions met and a branch displacement of minus four; which indicates that the branch is to a one word instruction preceding the DBcc instruction. Second, when the processor fetches the instruction at the branch address, it is checked to determine whether it is one of the allowed looping instructions. If it is, the loop mode is entered. Thus, the single word looped instruction and the first word of the DBcc instruction will each be fetched twice when the loop is entered; but no instruction fetches will occur again until the DBcc loop conditions fail.

In addition to the normal termination conditions for a loop, there are several conditions that will cause the TMP68010 to exit loop mode operation. These conditions are interrupts, trace exceptions, reset errors, and bus errors. Interrupts are honored after each execution of the DBcc instruction, but not after execution of the looped instruction. If an interrupt exception occurs, loop mode operation is terminated and can be restarted on return from the interrupt handler. If the T bit is set, trace exceptions will occur at the end of both the loop instruction and the DBcc instruction and thus loop mode operation is not available. Reset will abort all processing, including the loop mode. Bus errors during the loop mode will be treated the same as in normal processing; however, when the RTE instruction is used to continue the execution of the looped instruction, the three word loop will not be re-fetched.

The loopable instructions available on the TMP68010 are listed in Table G.1. These instructions may use the three address register indirect modes to form one word looping instructions; (An), (An)+, and −(An).

Table G.1 TMP68010 Loopable Instructions

Opcodes	Applicable Addressing Modes
MOVE [BWL]	(Ay)to(Ax) -(Ay)to(Ax) (Ay)to(Ax)+ -(Ay)to(Ax)+ (Ay)to-(Ax) -(Ay)to-(Ax) (Ay)+to(Ax) Xy to (Ax) (Ay)+to (Ax)+Xy to (Ax)+ (Ay)+to -(Ax)
ADD [BWL] AND [BWL] CMP [BWL] OR [BWL] SUB [BWL]	(Ay) to Dx (Ay)+to Dx - (Ay) to Dx
ADDA [WL] CMPA [WL] SUBA [WL]	(Ay) to Ax - (Ay) to Ax (Ay)+to Ax
ADD [BWL] AND [BWL] EOR [BWL] OR [BWL] SUB [BWL]	Dx to (Ay) Dx to (Ay)+ Dx to -(Ay)

Opcodes	Applicable Addressing Modes
ABCD [B] ADDX [BWL] SBCD [B] SUBX [BWL]	-(Ay) to -(Ax)
CMPM [BWL]	(Ay)+ to (Ax)+
CLR [BWL] NEG [BWL] NEGX [BWL] NOT [BWL] TST [BWL] NBCD [B]	(Ay) (Ay)+ -(Ay)
ASL [W] ASR [W] LSL [W] LSR [W] ROL [W] ROR [W] ROXL [W] ROXR [W]	(Ay) by #1 (Ay)+ by #1 -(Ay) by #1

Note : [B, W, or L] indicate an operand size of byte, word or long word.

Postscript

This manual describes functions and characteristics of each LSI in TLCS-68000 family.

All examples employed herein are used as reference for the purpose of explanation. Toshiba and Motorola disclaim all responsibilities for problems that may result from using any of these examples. The information contained herein is subject to change without prior notice as a result of future technical advancement.

This manual is made by :

Toshiba corporation
Integrated Circuit Div.
High End Microprocessor Engineering Sec.
580-1, Horikawa-cho, Saiwai-ku,
Kawasaki-city, Kanagawa 210
JAPAN
PHONE : Japan (81) 44-548-2190

OVERSEAS OFFICES

São Paulo:
Toshiba Brasileira Representacoes Ltda.
Av., Paulista, 807, 21 Andar Cjto 2106,
Cerqueira Cesar,
Cep. 01311-São Paulo-S.P.-Brasil
Tel.: 283-4511, 4714, 4964 Fax: (11) 251-4104

Athens:
Toshiba Corporation Athens Office
Athens Tower Bldg. A, 2-4 Mesogion Ave.,
Athens 115-27, Greece
Tel.: 7799828-9, 7791824 Telex: 21-6502 TSBA GR
Cable: TOSHIBA ATHENS

Tehran:
Toshiba Corporation Iran Office
No. 79 Bucharest Ave., 3rd Floor, Argentine
Square, Tehran, Iran
P.O. Box 15745-343, Tehran, Iran
Tel.: 624729 Telex: 212531 TSBAIR
Cable: TOSHIBACO TEHERAN

Beijing:
Toshiba Corporation Beijing Office
Room 1622/1624 Beijing Hotel, Dong Chang An Jie,
Beijing, The People's Republic of China
Tel.: 500-7766 (EX. 1622, 1624, 3857) 55-4179, 4768
(Direct)
Telex: 22807 TOSPK CN Cable: TOSHIBA PEKING

Guangzhou:
Toshiba Corporation Guangzhou Office
Room 609, Office Tower, China Hotel, Liu Hua, Lu,
Guangzhou, The People's Republic of China
Tel.: 663388 (EX. 2609) 67427 (Direct)
Fax: (20) 67-7427 Telex: 44585 TSBGZ CN
Cable: TOSHIBA GUANGZHOU

Shanghai:
Toshiba Corporation Shanghai Office
Room 2705 ~ 8, Shanghai Union Building Yanan Rd.,
East/Sichuan Rd., Shanghai The People's Republic of
China
Tel.: 200156, 200157, 200076 Fax: (21) 200075

Wellington:
Toshiba Corporation Representative
in New Zealand
12th Floor, Paxus House, 79 Boulcott Street
P.O. Box 3549, Wellington, New Zealand
Tel.: 721865, 726001 Fax: (4) 731-394
Telex: 3433 TOSHIBA NZ
Cable: TOSHIBA WELLINGTON

SALES SUBSIDIARIES

Toshiba America, Inc.
Electronic Components Business Sector
Irvine Head Office
(MOS IC Div., Semiconductor Div.)
9775 Toledo Way, Irvine, CA 92718, U.S.A.
Tel.: (714) 455-2000 Fax: (714) 859-3963
Eastern Area Office
(MOS IC Div., Semiconductor Div.)
25 Mall Road, 5th Floor,
Burlington, MA 01803, U.S.A.
Tel.: (617) 272-4352 Fax: (617) 272-3089
South Eastern Regional Office
(MOS IC Div.)
4025 Pleasantdale Rd., Suite 320 Atlanta, GA 30340,
U.S.A.
Tel.: (404) 493-4240 Fax: (404) 493-4401
San Jose Office (Electron Tubes & Devices Div.)
Western Area Office (MOS IC Div.)
2021 The Alameda Suite 220
San Jose, CA 95126, U.S.A.
Tel.: 408-244-4070 Fax: 408-248-5370
Southwestern Regional Office
(MOS IC Div.)
1400 Quail St., Suite 100 Newport Beach, CA 92660,
U.S.A.
Tel.: (714) 752-0373
Chicago Office (Electron Tubes & Devices Div.,
Semiconductor Div.)
Central Area Office (MOS IC Div.)
1101A Lake Cook RD. Deerfield, IL 60015, U.S.A.
Tel.: (312) 945-1500 Fax: (312) 945-1044
South Central Regional Office
(MOS IC Div.)
1750 North Collins Blvd., Suite #116
Richardson, Texas 75080, U.S.A.
Tel.: (214) 480-0470 Fax: (214) 235-4114
Poughkeepsie Sales Office
(MOS IC Div.)
RR 1 Box 6E
Windsor Park Fishkill, New York 12524, U.S.A.
Tel.: (914) 896-6500 Fax: (914) 297-2605
Boca Raton Sales Office
(MOS IC Div.)
1200 N. Federal Highway Suite 407
Boca Raton, FL 33432, U.S.A.
Tel.: (305) 394-3004 Fax: (305) 394-3006
Detroit Office
(Automotiv Div.)
26533 Evergreen Road,
Suite 420 Southfield, MI 48076, U.S.A.
Tel.: (313) 827-7700 Fax: (313) 827-4444

Toshiba (UK) Limited
Electronic Components Group
Riverside Way, Camberley,
Surrey GU 15 3YA, U.K.
Tel.: 0276-694600 Fax: 0276-691583

Toshiba Electronics Scandinavia AB
Gustavslundsvagen 141, 4th Floor
S-161 15 Bromma, Sweden
Tel.: 46-8-704 0900 Fax: 46-8-80 8459
Telex: 14169 TSBSTK S
(Mailing Address: P.O. Box 15031
S-161 15 Bromma Sweden)

Toshiba Electronics Europe GmbH
Düsseldorf Head Office
Hansaallee 181, 4000 Düsseldorf 11,
F.R. Germany
Tel.: (0211) 5296-0 Fax: (0211) 5296-400
Telex: 8582685
Liaison Offices
Stuttgart: Vertriebsburo Baden-Wurttemberg
Eltinger Str. 61
D-7250 Leonberg F.R. Germany
Tel.: (07152) 21061-66 Fax: (07152) 27658
Telex: 7245706
München: Buro München Arabellastr. 33/v.
8000 München 81, F.R. Germany
Tel.: (089) 928091-0 Fax: 089-9280942
Telex: 5-212363

Toshiba Electronics Italiana S.R.L.
Centro Direzionale Colleoni Palazzo Orione-
Ingresso 3 (3° Piano) 20041 Agrate
Brianza (Milano), Italy
Tel.: 039-638891 Fax: 039-638892
Telex: 326423 SIAVBC

Toshiba Electronics España, S.A.
Torres Heron Plaza Colón No.2 Torre II
Planta 6-Pte-2
28046 Madrid, Spain
Tel.: (1) 53-25-846 Fax: (1) 41-91-266
Telex: 44672 TOSHE (E)

Toshiba Electronics Taiwan Corp.
Taipei Head Office
8F, Min Sheng Chen Kou Bldg., 348-350 Min Sherg
East Road Taipei, Taiwan
Tel.: 02-502-9641 Fax: 02-503-7964
Telex: 26874 TETTP
Kaohsiung Office
16F-A Chung-Cheng Building,
No. 2 Chung-Cheng 3rd. 80027 Kaohsiung
Tel.: (07) 241-0826 Fax: (07) 282-7446

Toshiba Electronics Asia, Ltd.
Hong Kong Head Office
Suite 501, Hong Kong Hotel, Canton Road,
Tsimshatsui, Kowloon, Hong Kong
Tel.: 3-671-141 ~ 4, 3-721-6111 Fax: 3-739-8969
Telex: 38501 TSBEH HX
Seoul Branch Office
Room 1061, Chamber Building, 45, 4-KA
Namdaemun-Ro, Chung-Ku, Seoul, Korea
Tel.: (2) 757-2472 ~ 3 Fax: (2) 757-2475
Singapore Branch Office
460 Alexandra Road #26-01, PSA Bldg,
Singapore 0511
Tel.: 2785252 Fax: 2735368
Telex: RS 23892 TOSHIBA

Toshiba (Australia) Pty. Ltd.
84-92 Talavera Road, North Ryde, N.S.W. 2113,
Australia
Tel.: (2) 887-3322 Fax: 2-887-3201 Telex: AA27235

MANUFACTURING SUBSIDIARIES AND JOINT VENTURES

Toshiba America, Inc.
Microelectronics Center
1220, Midas Way, Sunnyvale, CA 94086, U.S.A.
Tel.: 408-739-0560 Fax: 408-746-0577
Telex: 346378

Toshiba Westinghouse Electronics Corporation
Westinghouse Circle, Horseheads, New York
14845, U.S.A.
Tel.: (607) 796-3500

Industria Mexicana Toshiba, S.A.
Calzada de Guadalupe No. 303, Cuatitlan,
Edo de Mexico, Mexico
Tel.: 5-65-00-88 Telex: 017-72-560
Cable: Toshiba Mexico

Semp Toshiba Amazonas S.A.
Rua Ica No. 500, Distrito Industrial de Suframa, Manaus,
CEP 69000, Amazonas, Brazil
Tel.: (092) 237-2366
Telex: 38922197 SEMP BR Cable: SEMP AMAZON

Toshiba Semiconductor G.m.b.H.
Grotrian-Steinweg Str. 10, 3300
Braunschweig, F.R. Germany
Tel.: (0531) 31-0060 Fax: (0531) 31006139
Telex: 952368 TSCD

Toshiba Electronics Malaysia Sdn. Bhd.
42057 Telok Panglima, Grang, 15KM Klang-Banting
road, Kuala Langat Selangor, Malaysia
Tel.: 03-352-6001-7 Fax: 03-352-6139
Telex: TOEL MA 39506
Penang Branch Office
Lot 2.08 2nd Floor, Wisma Chocolate Products,
41 Adoo Sittle Lane Panany 10400 Malaysia
Tel.: 04-368523, 04-368529 Fax: 04-368515

Toshiba Singapore Pte. Ltd.
20 Pasir Panjang Road, 09-18/26 PSA Multi-Storey
Complex, Singapore 0511
Tel.: 2718066 Telex: RS 36592 TSPMFG

TOSHIBA

TOSHIBA CORPORATION
INTERNATIONAL OPERATIONS—ELECTRONIC COMPONENTS
1-1 SHIBAURA 1-CHOME, MINATO-KU, TOKYO, 105, JAPAN
Tel.: 457-3495 Fax: 451-0576 Telex: J22587
TOSHIBA CABLE: TOSHIBA TOKYO

MANUAL NO. 4416

'88-8(CK)03Printed in Japan