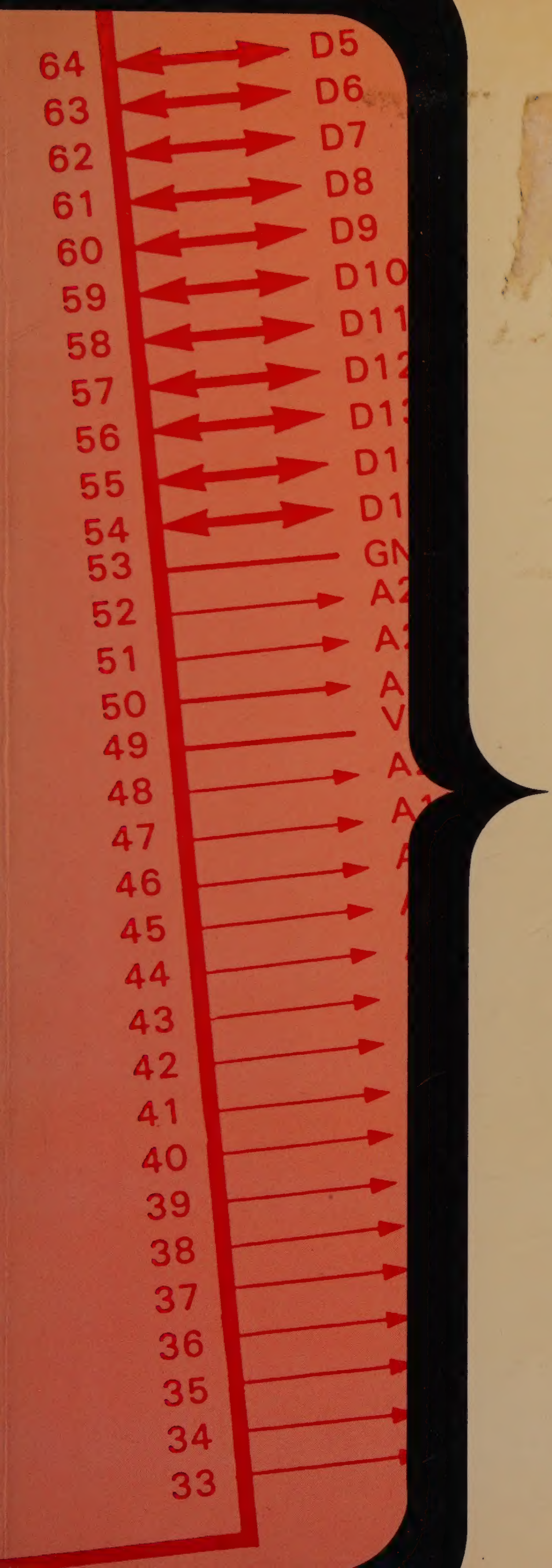


The 68000 Hardware and Software

Patrick Jaulent

1950

ᚢ

The 68000

Hardware and Software

The 68000 Hardware and Software

Patrick Jaulent
Microprocessor System Training Engineer
Microprocess
Puteaux, France

MACMILLAN

Authorised English Language edition
(with additions and revisions) of
Le Microprocesseur 68000 et sa programmation
by Patrick Jaulent, first published 1983 by
Editions EYROLLES, 61 boulevard Saint-Germain,
75005 Paris

Translated by M. J. Stewart

First published 1985

Published by
Higher and Further Education Division
MACMILLAN PUBLISHERS LTD
Houndmills, Basingstoke, Hampshire RG21 2XS
and London
Companies and representatives
throughout the world

Printed and bound in Great Britain at
The Camelot Press Ltd, Southampton

British Library Cataloguing in Publication Data
Jaulent, Patrick
The 68000 hardware and software.—(Macmillan computer science series)
1. Motorola 68000 (Microprocessor)
I. Title
001.64'04 QA76.8.M67

ISBN 0–333–38574–8

A mes parents — René et Antoinette

Contents

Acknowledgements

Both the author and the publishers are grateful to Motorola Inc., Chicago, USA and to Monolithic Memories Inc., Santa Clara, USA for permission to reproduce data and diagrams from their publications, through the kind assistance of their United Kingdom offices.

Special thanks are also due to Richard Mitchell for his help in the preparation of the English language edition of this book.

1 Pin Assignment of the MC 68000 and MC 68010

TECHNICAL HISTORY

The 16-bit MC 68000 microprocessor is the result of the MACSS project (Motorola Advanced Computer System on Silicon), which was begun in 1976 with the objective of developing a monolithic microprocessor whose performance would be based on the two main criteria of simplicity and orthogonality (that is, the internal registers would be general purpose with regard to addressing modes and instructions).

From the software point of view, the aim was to simplify programming by drawing upon the best of the modern programming techniques that enable the use of high-level languages, such as FORTRAN, Pascal, COBOL and Ada.

The package would also need to be able to function in a multiprocessor configuration, while remaining hardware compatible with the 6800 family; this requirement imposed serious constraints at the design stage.

The HMOS technology used for its production was required to reduce by a factor of 2 to 3 the area of an elementary cell, and divide by 4 the associated quality factor (that is, the result of the product of the consumption times the speed), which in turn would give a consumption of 1 picojoule per cell at 8 MHz.

The first samples were offered to industry in 1979 with the majority of these aims having been realised.

PIN ASSIGNMENT OF THE MC 68000 AND THE MC 68010

Data Bus (D0-D15) (Tristate logic)

These 16 bidirectional lines, which are not multiplexed, can transfer two types of data

16-bit word, or

8-bit byte, via the lower line D0-D7, or via the upper line D8-D15.

It was necessary to be able to transfer 8-bit data in order to be hardware compatible with the 6800 family. Note also that on a vectored interrupt the lower line (D0-D7) is used to transfer a vectored number.

Address Bus (A1-A23) (Tristate logic)
The non-multiplexed address bus allows direct addressing of 2^{24} combinations; that is, 16 777 216 bytes or 8 388 608 words. We should note that the 68000 does not have an A0 output address line whose role would be to select an even address (A0 = 0) or an odd address (A0 = 1) for a byte data item.

Such a single line would in fact be insufficient to satisfy the following three combinations

1. Selection of an exclusively even address for a word data item.
2. Selection of an odd address for a byte data item.
3. Selection of an even address for a byte data item.

Upper Data Strobe $\overline{\text{UDS}}$ (Tristate logic)
Lower Data Strobe LDS (Tristate logic)
These two signals _work in conjunction with the Read/Write line (R/W) and control the different lines for word or byte read/write operations. Table 1.1 shows how they work.

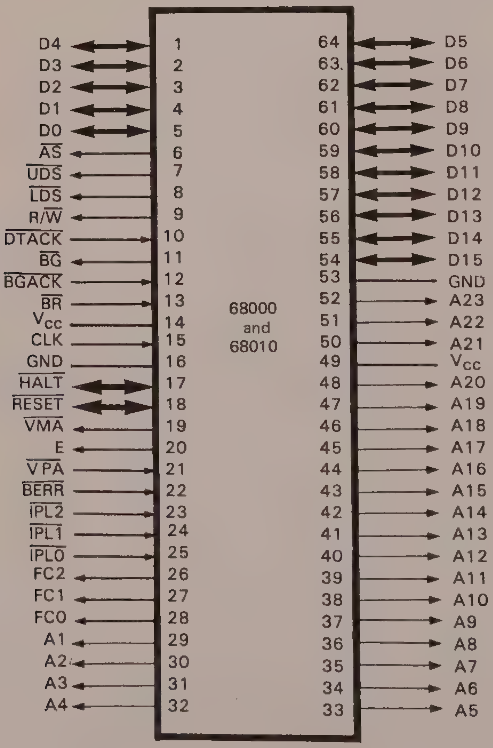


Figure 1.1

Table 1.1

A0	$R/\overline{W}$	$\overline{LDS}$	$\overline{UDS}$	Lower line (D0-D7)	Upper line (D8-D15)	Operation	Address
1	0	0	1	En	Dis	Write Byte	Odd
1	1	0	1	En	Dis	Read Byte	Odd
0	0	1	0	Dis	En	Write Byte	Even
0	1	1	0	Dis	En	Read Byte	Even
0	0	0	0	En	En	Write Word	Even
0	1	0	0	En	En	Read Word	Even

Read/Write $R/\overline{W}$ (Tristate logic)
This signal determines the direction of the transfer on the data bus; that is, a read cycle ($R/\overline{W} = 1$) or a write cycle ($R/\overline{W} = 0$).

Note the use of the line over the $\overline{W}$ in $R/\overline{W}$. Here it indicates that when the voltage is low, the data bus is to be used for a write cycle. This convention is also used for all the other lines.

Address Strobe $\overline{AS}$ (Tristate logic)
The pulse along this line to the external hardware signals that the address currently present on the address bus is electrically stable.

This signal is for example required by dynamic RAM systems. Some examples are RAS Row Address Strobe, MUX Multiplexors (type 74LS157 or PAL), and CAS Column Address Strobe.

Data Transfer Acknowledge $\overline{DTACK}$
When this input line is asserted ($\overline{DTACK} = 0$) by a memory or peripheral device, the processor is informed that a data transfer is acknowledged.

Recognition of the DTACK signal at low state during a read cycle indicates that the data transmitted on the data bus is latched, or that it has been received during a write cycle. This feature, resulting from the asynchronous operation of the 68000, is of particular value for the synchronisation of slow memory or peripheral devices.

Processor Status : Function Codes FC0, FC1, FC2 (Tristate logic)
These three fixed output lines are set by the processor at the beginning of a bus cycle and indicate that status of the processor to the external hardware.

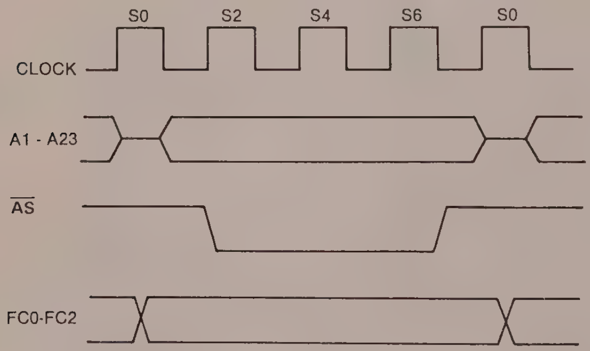
In particular, they show whether the processor is operating in supervisor mode (FC2 = 1) or in user mode (FC2 = 0), whether the information being executed is "data" type or "program" type, or whether it has acknowledged an interrupt (table 1.2).

Table 1.2

FC2	FC1	FC0	Status	Mode
0	0	0	Reserved	User
0	0	1	Data	User
0	1	0	Program	User
0	1	1	Reserved	User
1	0	0	Reserved	Supervisor
1	0	1	Data	Supervisor
1	1	0	Program	Supervisor
1	1	1	Interrupt	Supervisor

These lines therefore constitute an additional security for the system, while also making it possible to increase the addressing capacity of the 68000 from 16 megabytes to 64 megabytes by using the noted reserved combinations.

The timing diagram shown in figure 1.2 compares the electrical relationships between the AS signals, FC0, FC1, FC2 and the address bus.



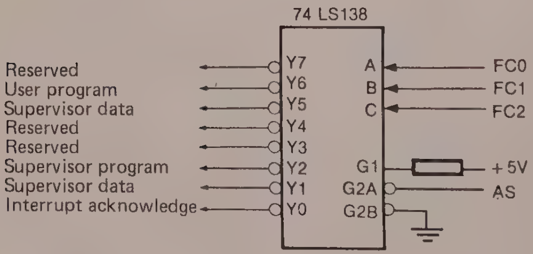


Figure 1.2

BUS ARBITRATION CONTROL

The three lines that ensure arbitration of the bus are described below.

Bus Request $\overline{\mathrm{BR}}$
This input, at low state, informs the processor that an external device requires the bus (for example, the DDMA 68440, DMAC 68450 or SBC 68430).

Bus Grant $\overline{\mathrm{BG}}$
While authorising the calling circuit to take control of the bus, the 68000 alerts its surrounding circuitry that it will surrender the bus at the end of the current bus cycle.

Bus Grant Acknowledge $\overline{\mathrm{BGACK}}$
The input confirms to the processor that the calling circuit has taken control of the bus. This line can only be enabled by the caller if the following four conditions are satisfied.

1.	$\overline{\mathrm{BG}}$	asserted	($\overline{\mathrm{BG}}$ = 0)
2.	$\overline{\mathrm{AS}}$	invalid	($\overline{\mathrm{AS}}$ = 1)
3.	$\overline{\mathrm{DTACK}}$	invalid	($\overline{\mathrm{DTACK}}$ = 1)
4.	$\overline{\mathrm{BGACK}}$	invalid	($\overline{\mathrm{BGACK}}$ = 1)

INTERRUPTS

Interrupt Request : Interrupt Priority Level

$\overline{\mathrm{IPL2}}$, $\overline{\mathrm{IPL1}}$, $\overline{\mathrm{IPL0}}$

The logical state of lines $\overline{\mathrm{IPL2}}$, $\overline{\mathrm{IPL1}}$ and $\overline{\mathrm{IPL0}}$ indicates to the processor the level of the waiting interrupt request. IPL0 represents the LSB and IPL2 the MSB.

Level 7 codes the highest level priority while level 0 indicates that there is no waiting interrupt request.

If the logical state on these lines is greater than the level of the interrupt mask set by the programmer in the status register, the processor accepts the interrupt request.

These lines must remain stable until the processor sets FC0 to FC2 to 1 and the address lines A4-A23 to high.

SYSTEM CONTROL

Bus Error : $\overline{\mathrm{BERR}}$
This input is controlled by external hardware, for

example by a memory management unit (MMU). It informs the processor that there is a hardware error in course of execution of the bus cycle.

Example
Absence of the $\overline{\text{DTACK}}$ signal during a reading or writing operation in working memory after a time delay fixed by the designer.

Enabling of the BERR input leads either to a sequential rerouting, called a trap, or to a rerun cycle.

Reset : Bidirectional Line

Reset on input : Initialisation of the 68000
When this line, which is set to input at power up, is held for 100 ms at the low state by means of the HALT line, the stack system and the program counter are loaded. This is the initialisation phase of the 68000.

Reset on output
Execution by the processor of the RESET instruction sets the reset line to the low state for 124 clock cycles. Handling this instruction does not affect the internal registers of the processor.

For example, this instruction is used to initialise a system or to program a peripheral circuit (PIA, timer, etc).

$\overline{\text{HALT}}$: Bidirectional Line

HALT on input
1. Initialisation of the 68000
2. Halting the processor
1. On input the HALT line follows the state of the RESET line (on input) throughout the entire initialisation phase.
2. When the HALT input is asserted, the processor terminates its bus cycle, then sets the three status lines at high impedance before moving to stop.

$\overline{\text{HALT}}$ on output
An example is the display of a double bus error. This follows a double error on the bus (for example, a hardware fault).

If during the initialisation phase ($\overline{\text{RESET}}$ and HALT on input at low state) a hardware or software anomaly occurs, the 68000 takes this to be catastrophic for the remainder of the program. In such a case it places itself at the halt state and alerts the outside world via the HALT output line.

Only an action on the RESET and $\overline{\text{HALT}}$ pins will cause the 68000 to leave the HALT condition.

6800 PERIPHERAL COMMANDS

The three signals defined below allow a dialogue between an asynchronous processor like the MC 68000 and the synchronous peripherals of the 6800 family.

Valid Peripheral Address : $\overline{\text{VPA}}$

1. When a 6800 family peripheral device wishes to converse with the 68000, the request circuit enables the VPA signal (VPA = 0) in order to alert the (asynchronous) processor that it should now transfer data according to the clock E. This is the synchronisation phase.

2. If VPA is at the low state during the interrupt acknowledge phase, the 68000 will identify the interrupt as coming from a 6800 peripheral device. From then on, the processor will move to autovectorisation by distributing a vector number according to the state of lines IPL2 to IPL0 (see chapter 4 on interrupts).

Enable E

This periodic signal, which is generated from a floating clock internal to the 68000, represents the time reference for all exchanges with the synchronous circuits of the 6800 family.

The period of signal E is equal to 10 periods of the signal fed to the input clock of the 68000, and has the form of 6 low states and 4 high states, as shown in figure 1.3.

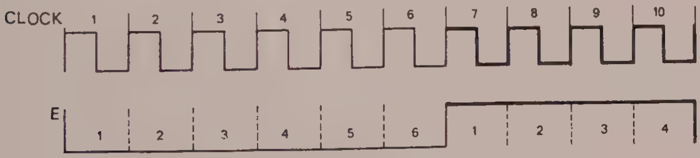


Figure 1.3

Valid Memory Address : $\overline{\text{VMA}}$

On receiving the VPA signal ($\overline{\text{VPA}}$ = 0) the 68000 synchronises itself before asserting the address sent on the address bus, by setting the VMA output to zero, when the clock E is at the low state (two cycles before E moves to the high state).

The $\overline{VMA}$ signal is used in the logical equation which ensures selection of the 6800 peripheral circuits (chip select).

Clock : CLK
The 68000 is able to produce the different signals required to allow it to function (for example, the E clock of the 6800 family), beginning from the clock signal fed to the 68000 CLK input.

The TTL compatible signal must be perfectly stable and adhere to the manufacturer´s specifications as set out in table 1.3.

Table 1.3

	4 MHz		6 MHz		8 MHz		10 MHz		12.5 MHz		
	68000L4		68000L6		68000L8		68000L10		68000L12		
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	
Freq	2	4	2	6	2	8	2	10	2	12.5	MHz
Prd	250	500	167	500	125	500	100	500	80	250	ns
pr -ve	115	250	75	250	55	250	45	250	35	125	ns
pr +ve	115	250	75	250	55	250	45	250	35	125	ns

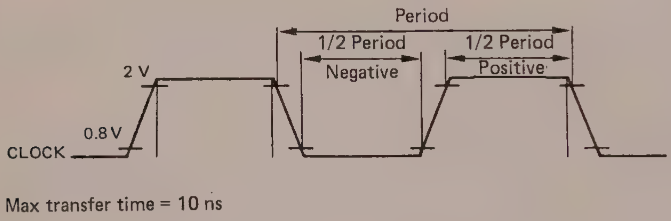


Figure 1.4

2 Internal Organisation of the 68000

As shown in figure 2.1, the MC 68000 contains the following.

8	32-bit data registers	(D0-D7)
7	32-bit address registers	(A0-A6)
1	32-bit user stack pointer	A7(or USP)
1	32-bit supervisor stack pointer	A7´(or SSP)
1	16-bit status register	(SR)
1	24-bit program counter	(PC)

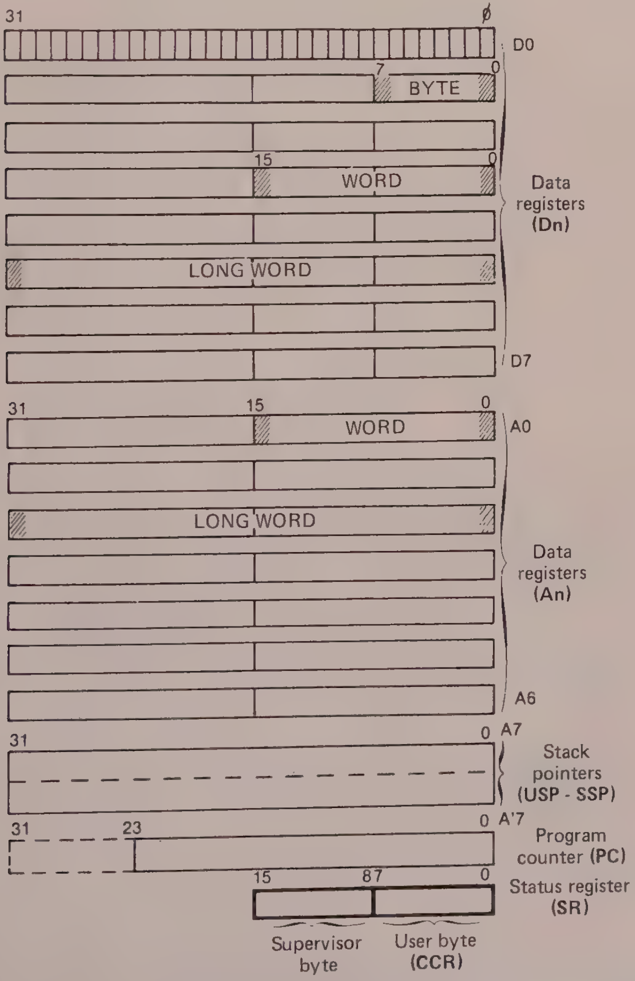


Figure 2.1 Programmer´s model (68000 and 68008)

STATUS REGISTER (SR)

It is no coincidence that our description of the 68000 programming model begins with a examination of the status register, since it is, as we shall see, the real heart of the microprocessor.

As figure 2.2 shows, the 16 bits of the status register form the user and supervisor bytes.

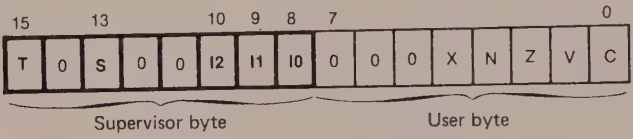


Figure 2.2

The 68000´s method of operation is determined by the logical state of bit S, which fixes the processor in supervisor mode when S = 1 and in user mode when S = 0. The availability of these modes facilitates the setting up of the operating system and makes multitasking and multi-user operation possible. In this case, memory management and logical protection will need to be carried out by a type MC68451 memory management unit (see figure 2.3b), or by an electronic equivalent.

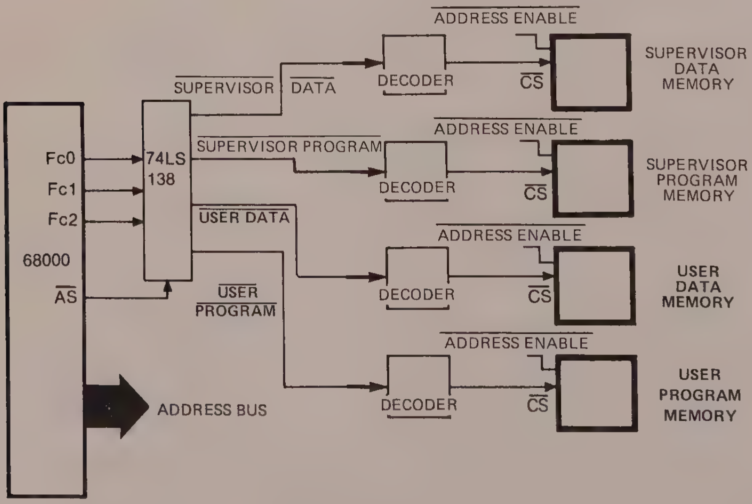


Figure 2.3a Principles of memory organisation (without MMU)

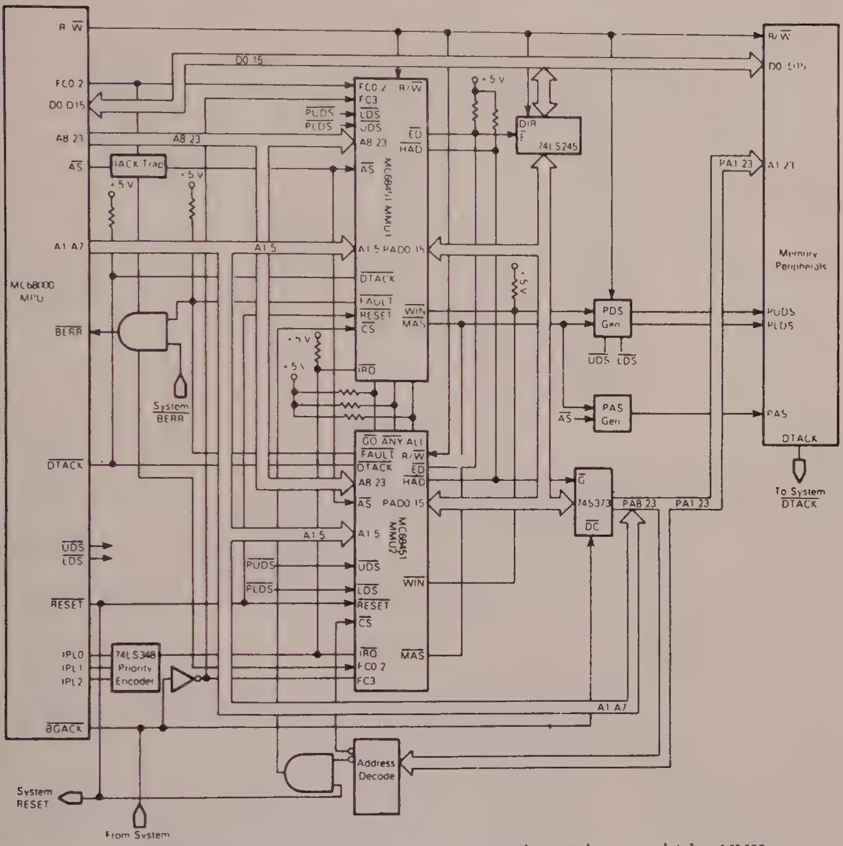


Figure 2.3b Memory organisation with MMU

SUPERVISOR MODE S = 1

Also called the security system, the supervisor mode (S = 1) allows the programmer access to all resources, both software and hardware. For example, he can choose from any of the following

1. Use all the 68000 instructions
2. Address the data, program, supervisor and user memory locations
3. Access the complete status register (supervisor and user bytes)
4. If required, select the supervisor stack (SSP) and user stack (USP) pointers (see section 2.4).

The supervisor is a set of programs that allows the user program to be initialised and then chained with other user programs, all without the operator having to take any action.

Figure 2.4 shows the supervisor byte that can only be accessed by the programmer when in supervisor mode.

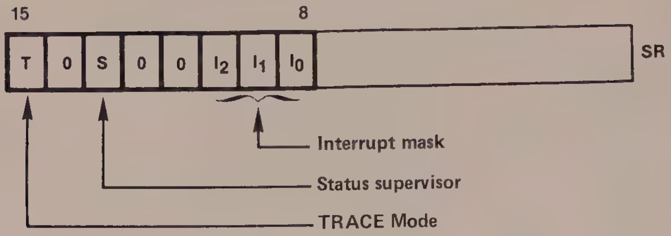


Figure 2.4

Trace Mode
After each instruction the processor tests internally whether bit T of the status register is enabled (T = 1).

When T = 1, a program can be traced, instruction by instruction. It is the software equivalent to the single step operation carried out in 8-bit microprocessors. The trace function can be used to debug a program, whether in supervisor or user mode.

Processor Status

S = 1 fixes the processor in supervisor mode
S = 0 fixes the processor in user mode

Interrupt Mask
The 68000 has seven interrupt levels that can be programmed by bits I2, I1 and I0, as shown by the table in figure 2.5.

These three bits fix the interrupt mask, and also the priority level of the interrupt currently being handled.

Increasing priorities

Level	I_2	I_1	I_0	
7	1	1	1	Highest priority level (NMI type interrupt)
6	1	1	0	
5	1	0	1	
4	1	0	0	
3	0	1	1	
2	0	1	0	
1	0	0	1	Lowest priority level
0	0	0	0	No priority (no request)

Figure 2.5

Priority occurs in the increasing order, so that level 1 represents the lowest priority (level 0 corresponds to no priority) and level 7 the highest, being equivalent to a non-maskable interrupt.

In order to understand how the interrupt is handled, let us assume that the interrupt mask is at level 5 (the logical state of bits I2, I1 and I0 represent the mask).

Any interrupt request less than or equal to the mask (1 to 5 inclusive) is ignored, whereas a higher level request (6 or 7) will be accepted by the processor. Note, though, that level 7 is non-maskable.

USER MODE S = 0

The user mode (S = 0) is the lowest priority mode. This means that a program executed in this mode can only access the data and the user program memories.

Those instructions that are able to change the functioning mode of the processor are called privileged.

On executing a privileged instruction the 68000 also causes a trap which is called a privilege violation. It is therefore impossible, except in a manner controlled by the central unit, for a program executing in user mode to be able to change the functioning mode of the processor or address a data or supervisor program area. The latter would still require a memory management unit (MMU) circuit.

Flag Identification

As figure 2.6 shows, only the first five bits of the user byte are of importance for the programmer.

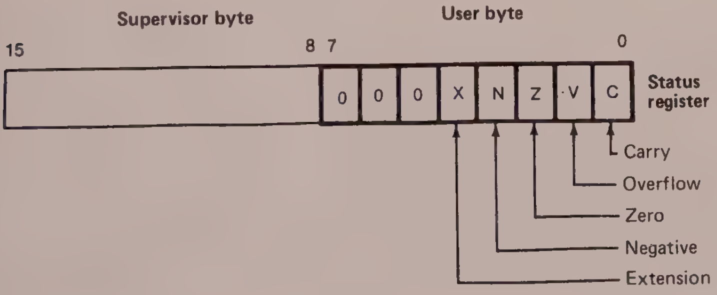


Figure 2.6

These flags, which are found in the majority of 8-bit microprocessors, inform the programmer of the status of the microprocessor after an arithmetical or logical instruction has been carried out.

For example, flag X is required by the processor when carrying out decimal addition and subtraction.

Arithmetic processing

Instruction source	Logical state of flags after instruction execution
MOVE.B #$FF,D2	N = 1; Z = 0; V = set at 1; C = set at 0
ADDI.B #$01,D2	N = 0; Z = 1; V = 0; C = 1

Logical processing

The instruction LSL.W #1,D4 causes a logical shift one position to the left.

Figure 2.7 shows the flags requested by the instruction.

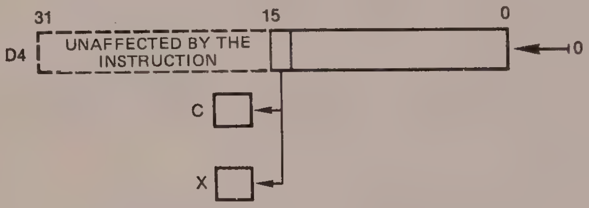


Figure 2.7

DATA REGISTERS

Figure 2.8 illustrates how the eight data registers accept operands, whether of 8, 16 or 32 bits.

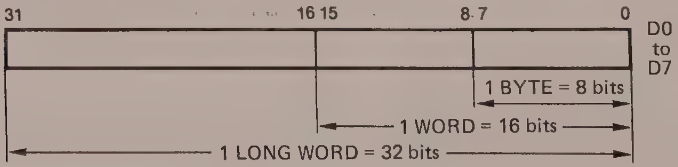


Figure 2.8

When a data register is used for a process (instruction) as the source or the destination, only the bits specified by the size are involved; the other bits are not affected.

Example
The instruction CLR.B D2 resets bits 0-7 of register D2 without altering bits 8-31, as shown in figure 2.9.

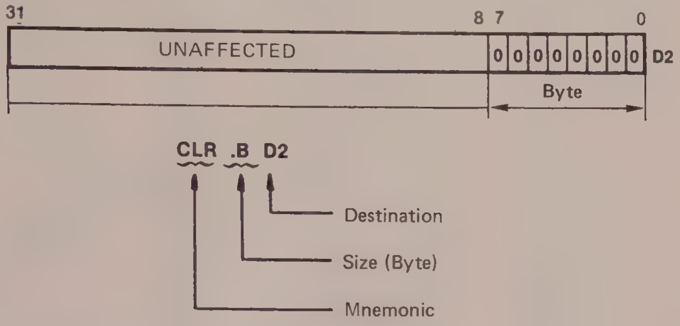


Figure 2.9

The size is specified in the instruction source by the letters B for Byte (8 bits), W for Word (16 bits), and L for Long Word (32 bits).

Example

```
        CLR .B D2 resets bits 0-7  of D2
        CLR .W D2 resets bits 0-15 of D2
        CLR .L D2 resets bits 0-31 of D2
```

ADDRESS REGISTERS (A0-A6)

The seven address registers (A0-A6) handle operands of 16 or 32 bits. In fact, they do not accept 8-bit byte type operands.

If an address register An is used as destination, the whole register is affected, even if the size specified by the instruction is a word type.

Example

```
    MOVEA.W # $8000, A5
                             Destination register
                             Hexadecimal  operand
                             16-bit word size
                             Mnemonic
```

Since the operand is negative (bit 15 = 1), the 32-bit sign extension sets register A5 to the value $FFFF8000 as shown in figure 2.10.

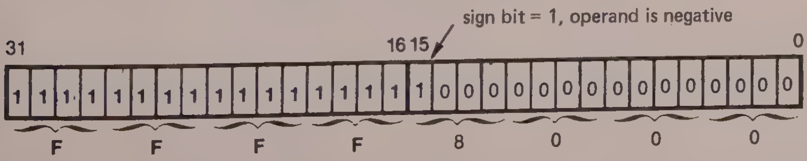


Figure 2.10

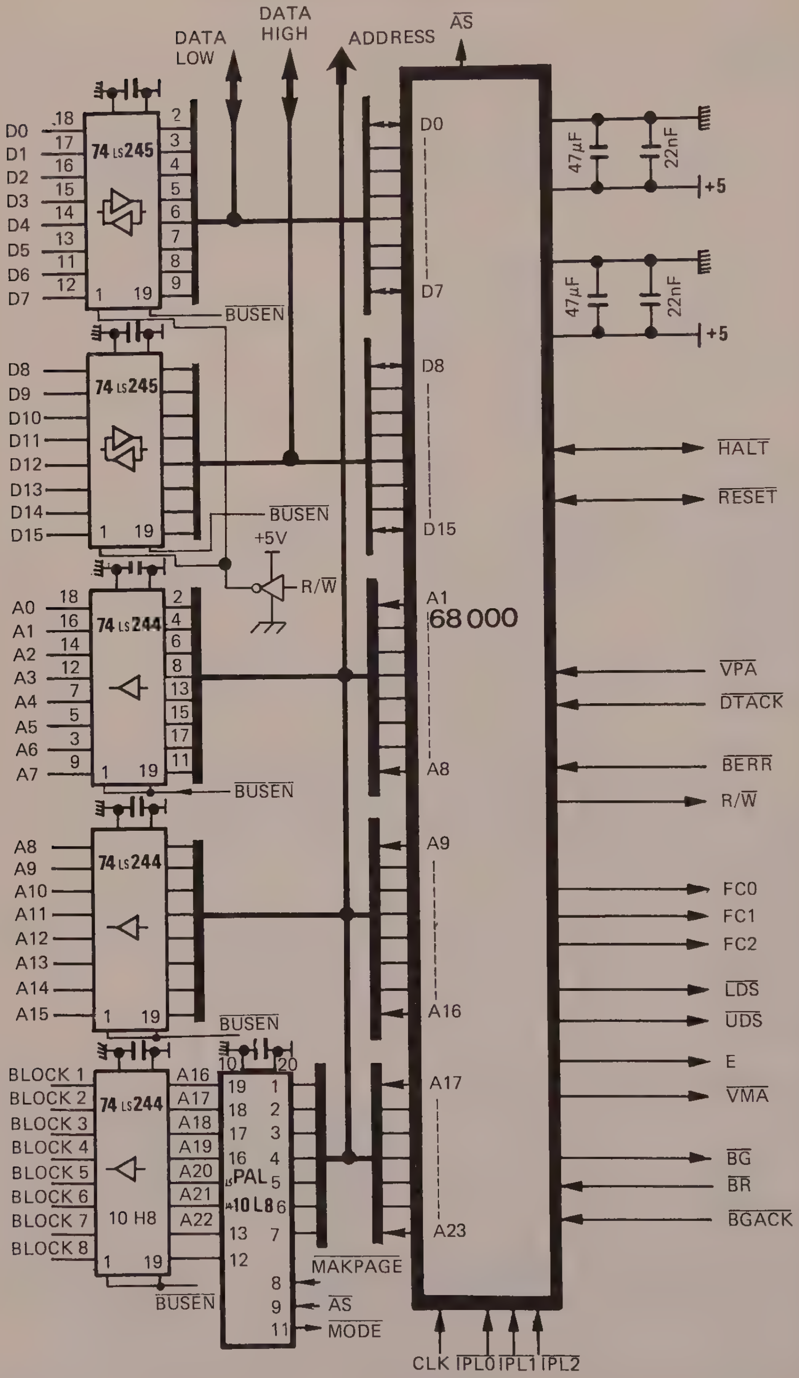


Figure 2.11 Shows the interface between the 68000 and MAKBUS. The 74LS244/5 devices buffer the 68000 signals, and the PAL10L8 acts as memory management unit (see also Appendix 5). Note that the availability of the MODE signal allows two possible configurations, such as MAKBUS and MAKBUS+. (Copyright Microprocess)

Problem
What is the content of register A5 after the processor has executed the instruction MOVEA.W #$2A00,A5? Can you explain your answer?

STACK POINTER (SP)

All the rules explained in the previous section apply equally to the stack pointers.

The CPU automatically uses address register A7 as stack pointer (SP) when subprograms are called, such as for exception handling or implicitly for certain instructions (RTS, PEA, or RTD, for example).

The system stack pointer can be one of the following

1. The supervisor stack (SSP) when bit S of the status register has the value 1;
2. The user stack pointer (USP) where S = 0.

By convention, the user stack pointer (USP) is designated A7, while the supervisor stack pointer (SSP) is A7´.

Note that this chapter is equally valid for the 8-bit MC 68008.

3 Bus Operation

INTRODUCTION

What follows is intended to show future designers of applications based on the MC 68000 how its bus functions during such operations as the following

data transfer
rerun cycle
bus allocation arbitration
halt or single-step operation
dialogue with synchronous circuits.

DATA TRANSFER

Figure 3.1 illustrates the different command and control signals that are requested during data transfer operations.

1 Read Cycle
During a read cycle the processor receives an item of data from memory or peripheral circuits. The 68000 always reads a byte type data item using an internal bit A0, in order to determine which line the data item should follow.

If A0 = 1 and the $\overline{\mathrm{LDS}}$ signal is asserted, the data item can be read on the lower line D0-D7, to an odd address.

If A0 = 0 and the $\overline{\mathrm{UDS}}$ signal is asserted, the data item can be read on the upper line D8-D15, to an even address.

On the other hand, when the instruction code indicates reading of a word (or a long word), the 68000 processor simultaneously enables $\overline{\mathrm{LDS}}$ and $\overline{\mathrm{UDS}}$, while A0 = 0, since in this case the specified address can only be even.

The reader is urged to study closely the timings for reading a word, as illustrated in figures 3.2 and 3.3, referring where necessary to table 3.1 for further details.

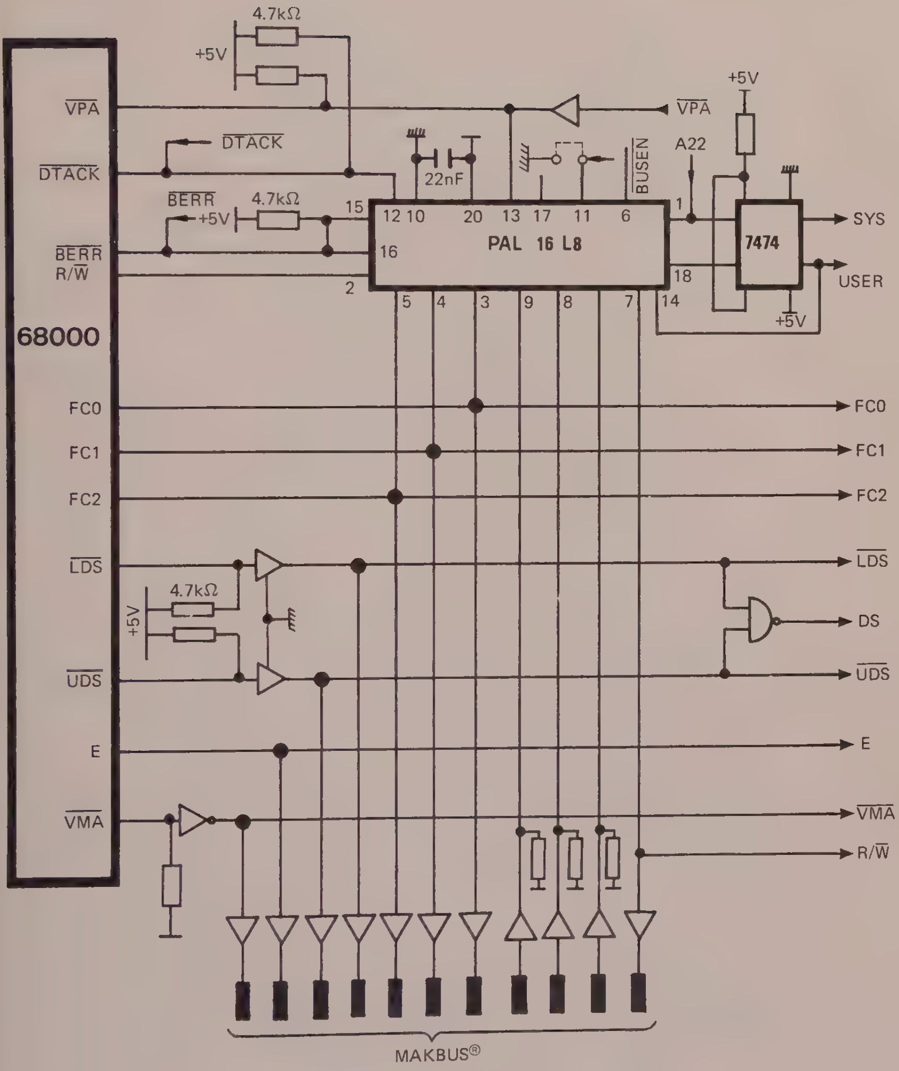


Figure 3.1 In this circuit the PAL16L8 buffers microprocessor and bus signals. It also provides the control for multiprocessor configurations. (Copyright Microprocess)

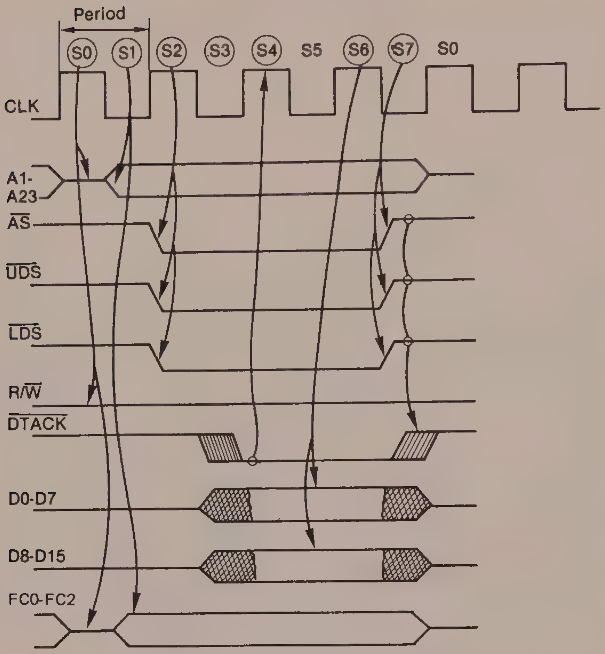


Figure 3.2 Read bus cycle for a word

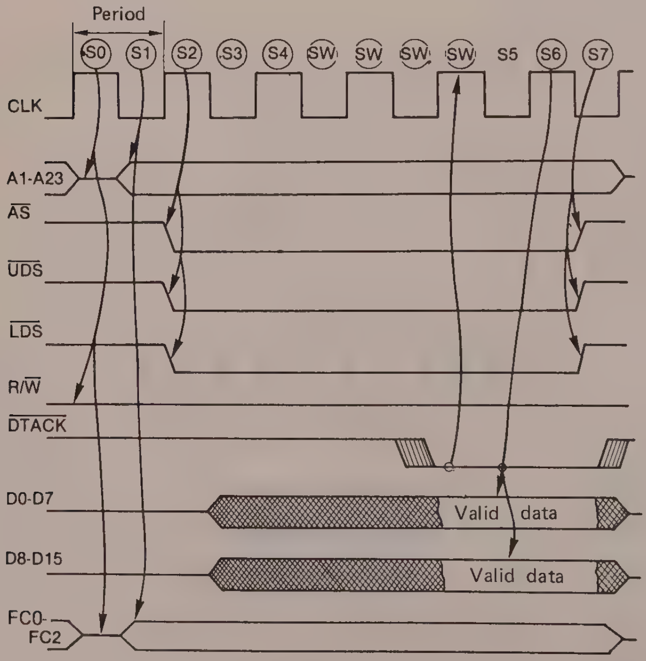


Figure 3.3 Slow read bus cycle for a word

Table 3.1 Reading a Word Within the Bus Cycle

State 0 (S0)	Address lines A1-A23, the read/write line (R/$\overline{W}$) and the processor status outputs FC0, FC1 and FC2, are all high.
State 1 (S1)	The processor places the address on its bus A1-A23 and reports its electronic status externally, by means of lines FC0-FC2.
State 2 (S2)	Output $\overline{AS}$ is enabled from the beginning of S2, which allows a decoding circuit to use this signal to latch the address sent on the bus. $\overline{UDS}$ and $\overline{LDS}$ are also asserted during S2.
State 3 (S3/4)	Wait for signal $\overline{DTACK}$. 1. If $\overline{DTACK}$ arrives during S3, it will be recognised by the processor when S4 is high. 2. If $\overline{DTACK}$ is not present (for example, on slow read), the processor sets wait states until the arrival of $\overline{DTACK}$ (see figure 3.3).
State 5 (S5)	Inactive in our example.
State 6 (S6)	The data are recognised and latched in the input register (DBIN) of the processor.
State 7 (S7)	Signals $\overline{AS}$, $\overline{UDS}$ and $\overline{LDS}$ are disabled causing the signal $\overline{DTACK}$ to be set high by an external electronic source. During the following state (S0) or, at the end of S7, lines FC0 to FC2, the address bus, the data bus and the R/$\overline{W}$ line are disabled.

2 Write Cycle

For a write cycle, the MC 68000 processor places the data on the bus, to be sent to an addressable area (memories or peripherals).

In a manner similar to the read operation, the data written by the processor is byte type. We therefore do not need to go into the conditions that lead to its being sent on the lower or upper line. Of course, if the operation code specifies a word or long word data item, the two lines are enabled by means of LDS and UDS signals (with A0 = 0), as shown in table 3.2.

Figure 3.4 shows the timing of a write cycle.

Table 3.2 Writing a Word Within the Bus Cycle

State 0 (S0)	The address bus is high (A1-A23). The read/write line (R/W)is at read (end of previous cycle). The processor status lines (FC0, FC1, FC2) are enabled.
State 1 (S1)	The current address is placed on the bus (A1-A23).
State 2 (S2)	Sampling of the address is asserted (AS = 0), line $R/\overline{W}$ is positioned at write while the data bus is disabled.
State 3 (S3)	The data is placed on the bus (D0-D15) by the processor.
State 4 (S4)	Outputs $\overline{UDS}$ and $\overline{LDS}$ are asserted. The data is latched. The processor recognises $\overline{DTACK}$.
State 5 (S5)	Confirmation that the write bus cycle is terminated as DTACK is present (in the opposite case, the processor introduces state waits).
State 6 (S6)	Inactive in our example.
State 7 (S7)	Outputs $\overline{AS}$, $\overline{UDS}$ and $\overline{LDS}$ are deactivated, bringing about an incrementation of the DTACK signal via an external electronic source.

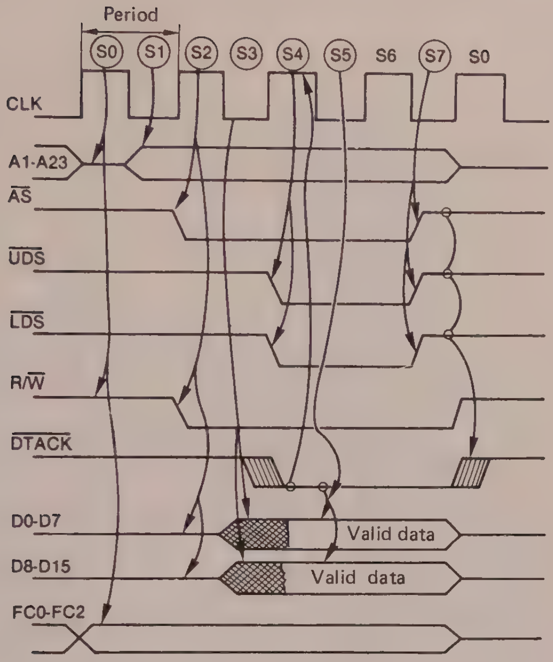


Figure 3.4 Timing of a write cycle

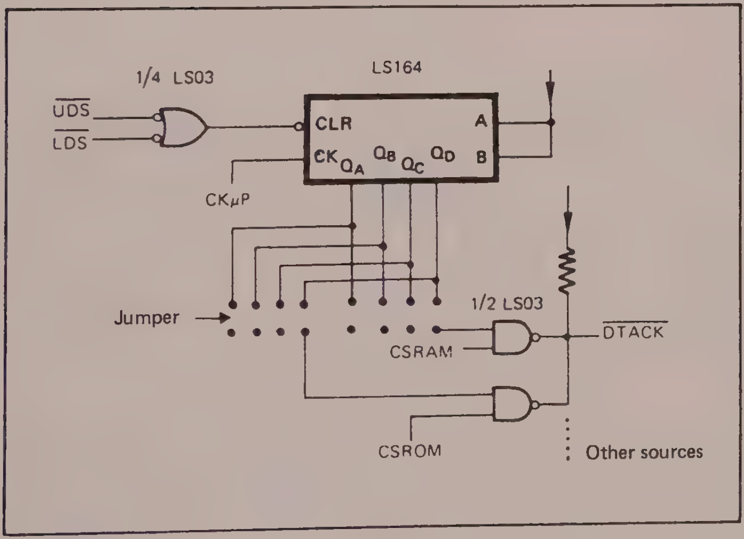


Figure 3.5a Traditional method of producing $\overline{\text{DTACK}}$ signal

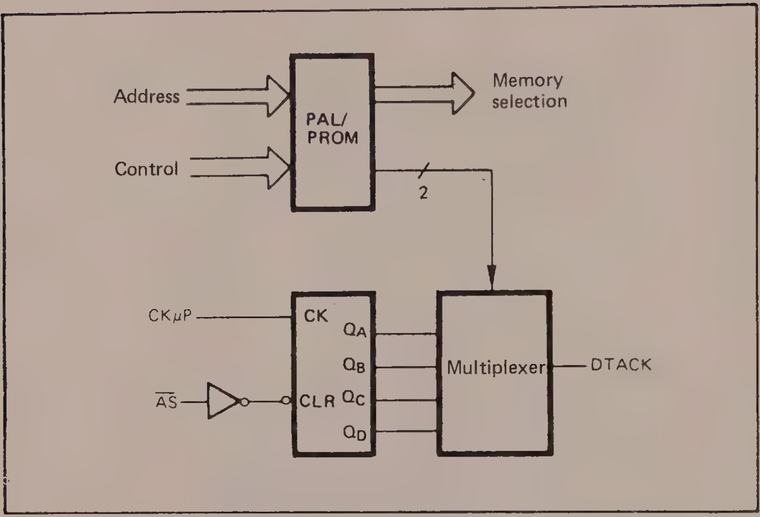


Figure 3.5b Producing $\overline{\text{DTACK}}$ signal using PROM and/or PAL technologies

3 Relationship Between $\overline{\text{DTACK}}$ and the Data Bus
If you want a system to function at the maximum permitted speed, using RAM memory locations, the relationships between DTACK and the data bus are important.

We recall that
1. DTACK, when recognised by the processor during a read cycle, indicates that the data is latched and that the bus cycle is terminated.
2. DTACK, when recognised during a write cycle, indicates that the bus cycle is terminated.
Most systems use a timer (counter) to detect a nonexistent addressable area (removed or damaged memory locations). If no DTACK signal has been received when the timer reaches a timeout value, a bus error signal is generated (see next section).

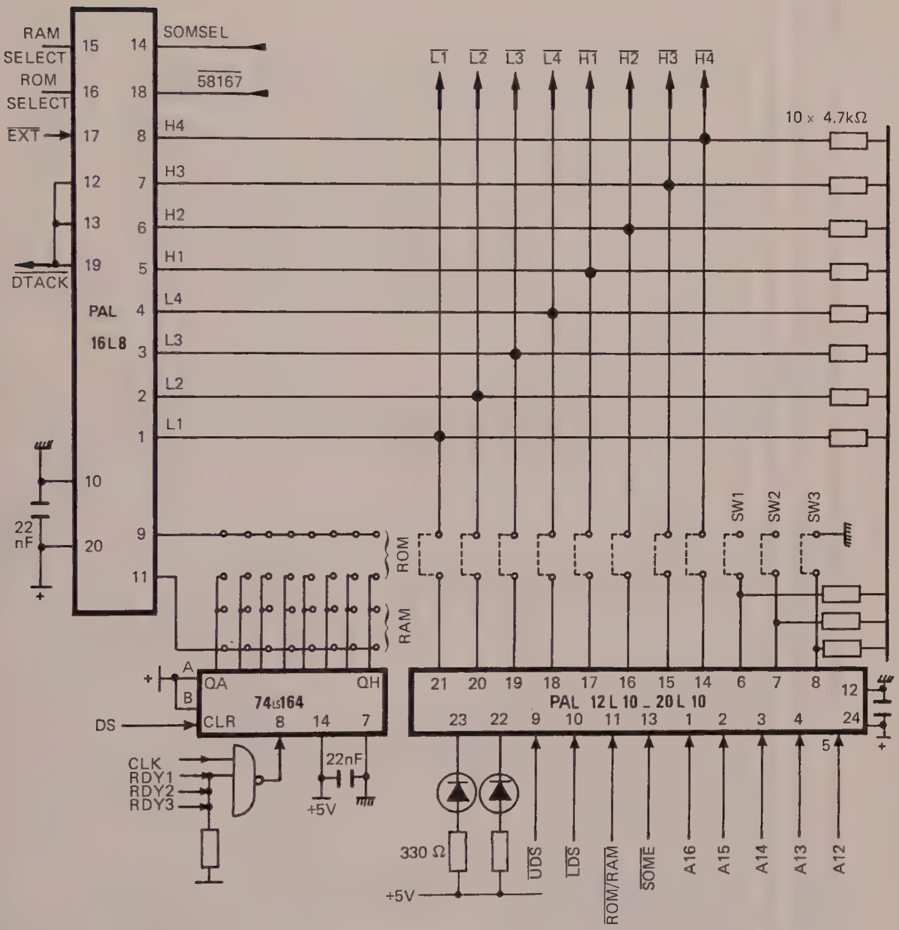


Figure 3.6 The PAL12L10 controls the decoding of the 8 memory packages on the card. It can be reprogrammed to be used with memory packages of a different size. The PAL16L8 controls the memory DTACK signal(s), both on the CPU-M card and external to it. (Copyright Microprocess)

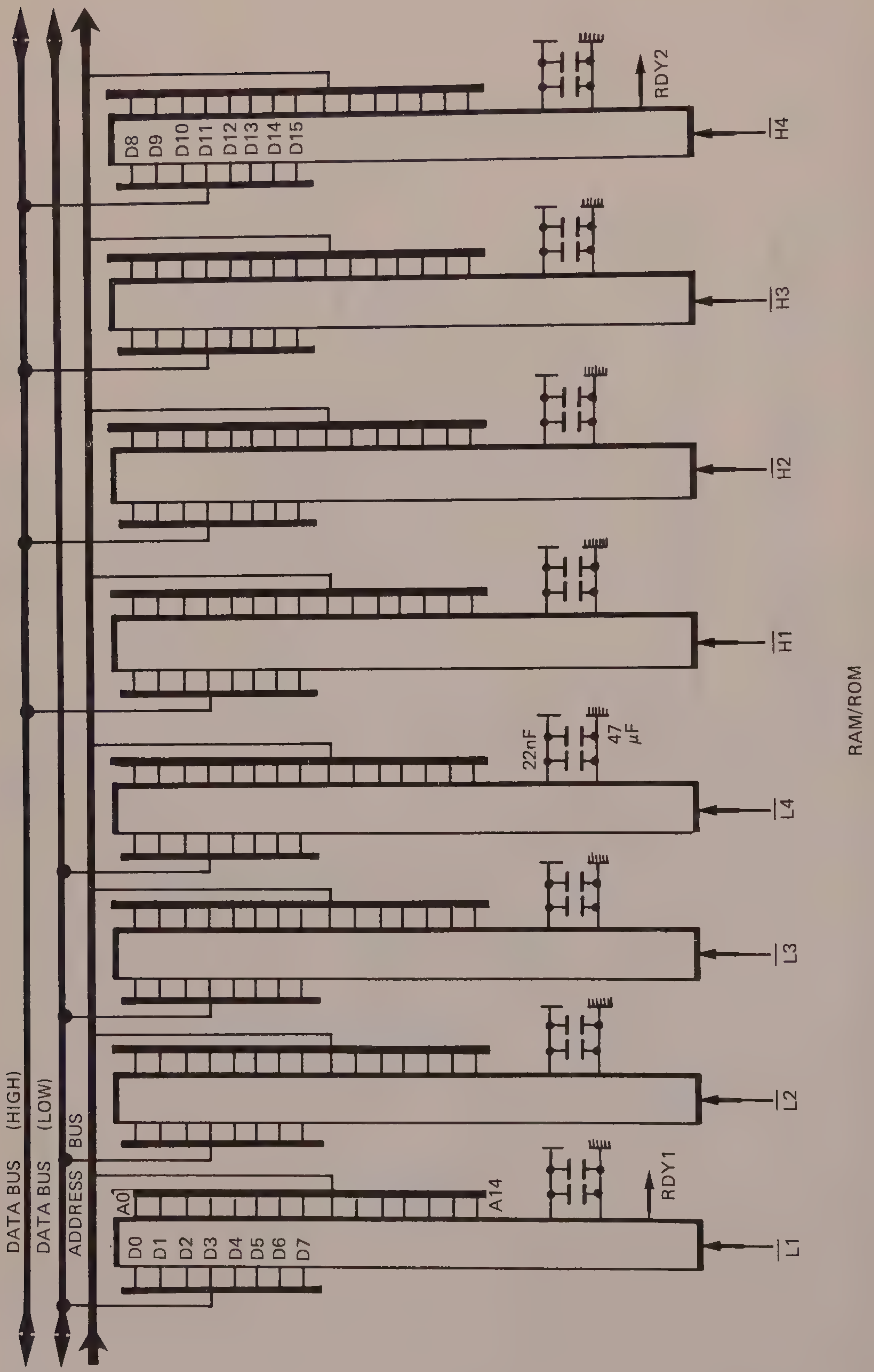


Figure 3.7 Part of figure 3.6. (Copyright Microprocess)

RERUN CYCLE

If during execution of a bus cycle an external circuit activates lines BERR and HALT (BERR = 0 and HALT = 0), the processor is alerted that the current bus cycle will not be correctly completed or terminated, and that it must initiate the rerun cycle procedure.

Figure 3.8 shows the timing of a rerun cycle whose different stages, set out in the flowchart of figure 3.7, are described below.

1. Inputs BERR and HALT are asserted during a bus cycle.

2. The MC 68000 processor terminates its bus cycle before placing the address and data lines at high. While the HALT line is held at low by an external circuit, the processor is halted.

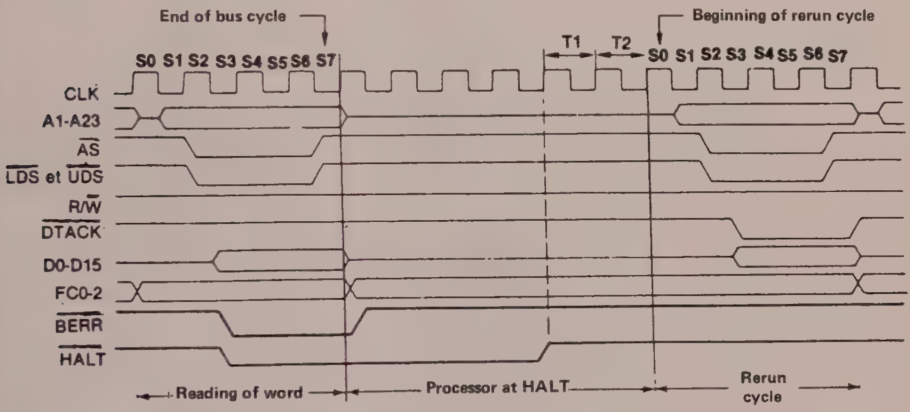


Figure 3.8 Timing diagram of rerun cycle
(Courtesy of Motorola)

Rerun Condition

If the aborted bus cycle is not indivisible (TAS instruction : read/modify/write cycle) and, if the BERR line is again positioned at high, a rerun cycle can be envisaged.

3. As soon as the HALT line is disabled by the external circuitry (HALT = 1), the 68000 moves to the preceding rerun cycle using the address and data transmitted by their respective buses which have previously been set to high.

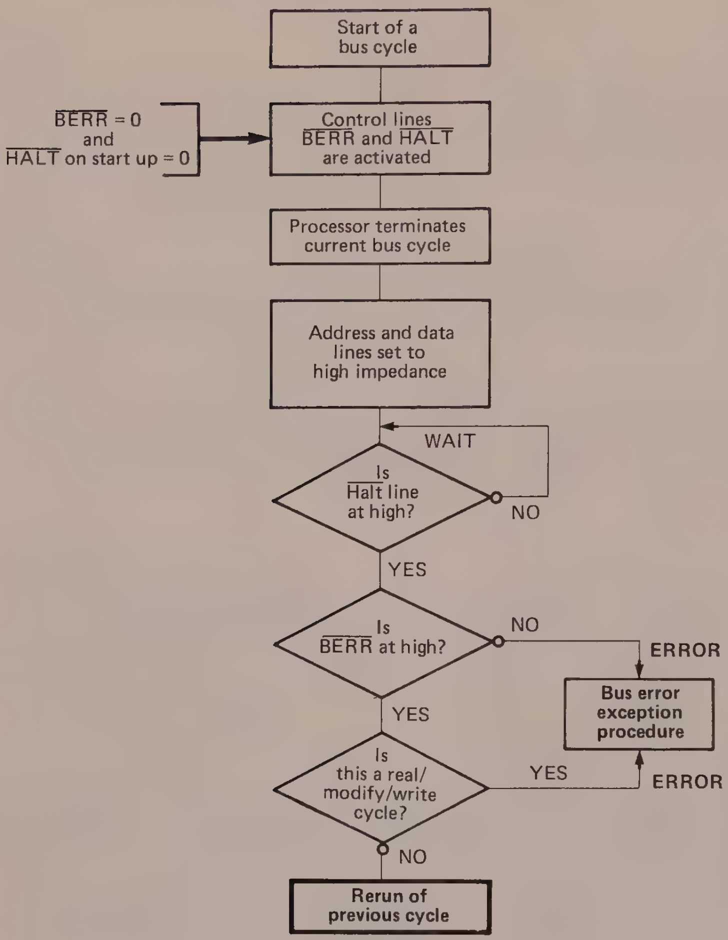

Start of a bus cycle
$\overline{BERR}$ = 0 and $\overline{HALT}$ on start up = 0
Control lines $\overline{BERR}$ and $\overline{HALT}$ are activated
Processor terminates current bus cycle
Address and data lines set to high impedance
WAIT
Is $\overline{Halt}$ line at high?
NO
YES
Is $\overline{BERR}$ at high?
NO
ERROR
Bus error exception procedure
YES
Is this a real/modify/write cycle?
YES
ERROR
NO
Rerun of previous cycle

Figure 3.9 Rerun cycle

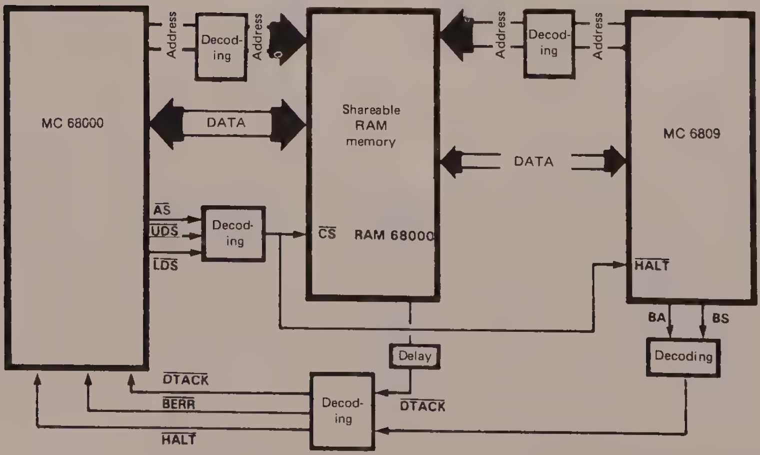


Figure 3.10 Simplified multiprocessor system

Example
In the multiprocessor system shown in figure 3.10, the MC 6809 microprocessor uses RAM when the MC 68000 processor wishes to access it (CS RAM 68000 enabled). While waiting for the memory to be available to the 68000, the former executes n rerun cycles.

BUS ALLOCATION ARBITRATION

1 Bus Request

When an external unit wishes to take control of the bus, whether in a multiprocessor context by means of the BAM 68452 bus arbitration module or for a direct memory access via a DMA controller, it makes its request of the MC 68000 with the enabling signal BR (Bus Request).

The 68000 then confirms that it has received the signal by replying with BG (Bus Grant). The acknowledgement, that is, BG at low, occurs between 1.5 and 3.5 clock periods after the request (BR at low), which will lead us quite naturally to propose several consequences.

2 Agreement to Transfer Bus Control

The transfer of the bus to the requesting circuit only becomes effective if the following four conditions are satisfied: the AS output of the 68000 is high, thus confirming the end of the current bus cycle; the DTACK input is high, indicating that no addressable circuit (peripherals, memories, etc) is communicating with the processor; the BGACK input is high, indicating that the bus is not being controlled by a main bus control circuit; BR remains low.

If these four conditions are fulfilled, at the end of the current bus cycle the 68000 sets the address, data and exchange control lines to the high impedance state, and awaits confirmation of bus control from the requesting circuit (BGACK at low).

3 Recognition of Bus Control by Requesting Circuit

The requesting circuit confirms that it has assumed control of the bus by enabling the BGACK signal, and this remains so for as long as BGACK is at low. It therefore becomes superfluous for the BR output to remain low, and the 68000 accordingly repositions BG at the high impedance state.

4 Methods of Operation
We shall examine three examples of requests that might occur in practice.

Bus request during bus cycle
If the BR input is enabled after AS is at low, the 68000 acknowledges receipt of the bus control request by asserting BG (at low) 1.5 clock periods later. This is the optimum.

If BR is asserted before AS, that is, during S0 or S1 (AS is asserted during S2), the 68000 decides that it has not advanced sufficiently far in its bus cycle to be able to respond as before, namely 1.5 cycles after the request. As a consequence, confirmation that the 68000 has received a bus request (BR at low) will be made 1 clock period after AS has been set at low (see figure 3.11).

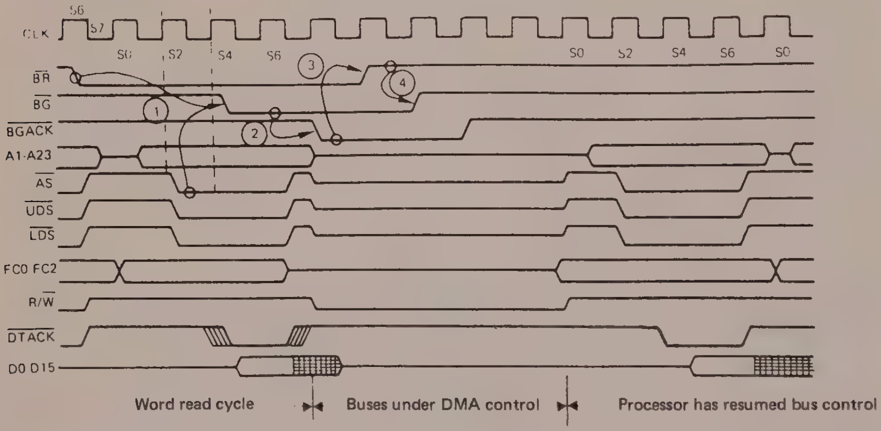


Figure 3.11 Timing of various signals when bus request arrives during execution of a bus cycle

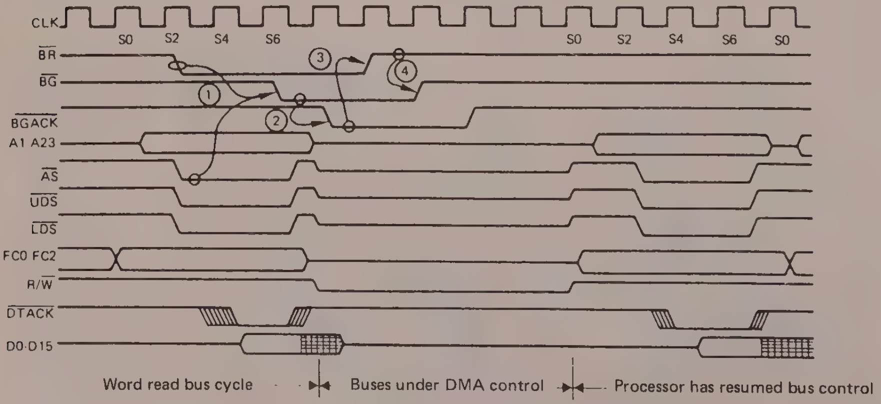


Figure 3.12 Timing of various signals when the bus request arrives at the beginning of a bus cycle

Bus request at beginning of bus cycle
If BR is asserted at the end of a bus cycle (S6 or during S6 or S7) and is therefore present at the beginning of the next bus cycle, the 68000 will position BG at low after AS is enabled by the processor (see figure 3.12).

Bus request when not in use
If BR is asserted when the 68000 is not using the bus (that is, when the 68000 is working internally), the processor will confirm receipt of the bus request by setting BG at low 2.5 periods after the request. Also, the bus control will only be effective a further 1 period later, that is, 3.5 periods after BR is set at low (see figure 3.13).

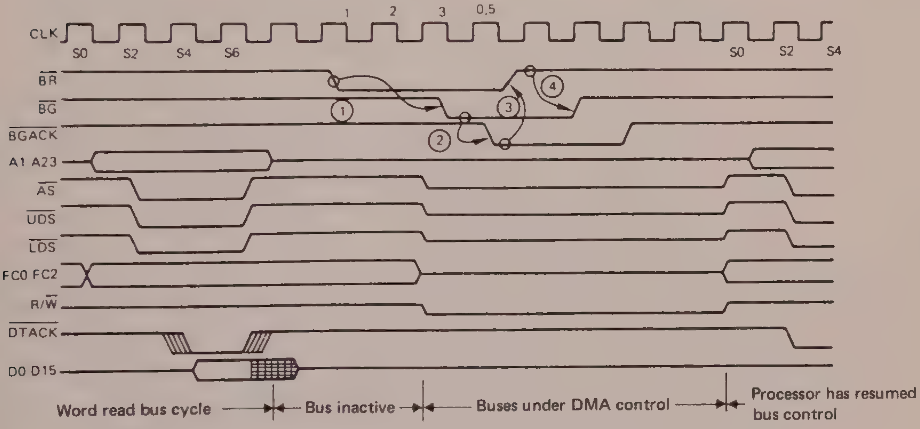


Figure 3.13 Timing of various signals when bus request arrives while bus is inactive

Bus Control Circuits
There are at least three DMA controllers in the 68000 family (to which new circuits are continually being added). These are as follows

SBC	68430	(1 channel) from Philips/Signetics
DDMAC	68440	(2 channels) from Motorola
DMA	68450	(4 channels) from Hitachi

Motorola have also produced a bus arbitration module BAM 68452 that is capable of handling eight bus requests via the device bus request lines DBR0 to DBR7, where DBR7 carries the highest priority request and DBR0 the lowest. This system of priority is only invoked if two or more requests arrive at the 68452 simultaneously.

These eight inputs have assigned to them eight device bus grant outputs, DBG0 to DBG7, whose role is to advise the requesting circuit that the request has been received by the 68000 via the BR output of the 68452. Once this has been done, the requesting circuit positions the BGACK line connected to the BAM 68452 and to the 68000 (see figure 3.14).

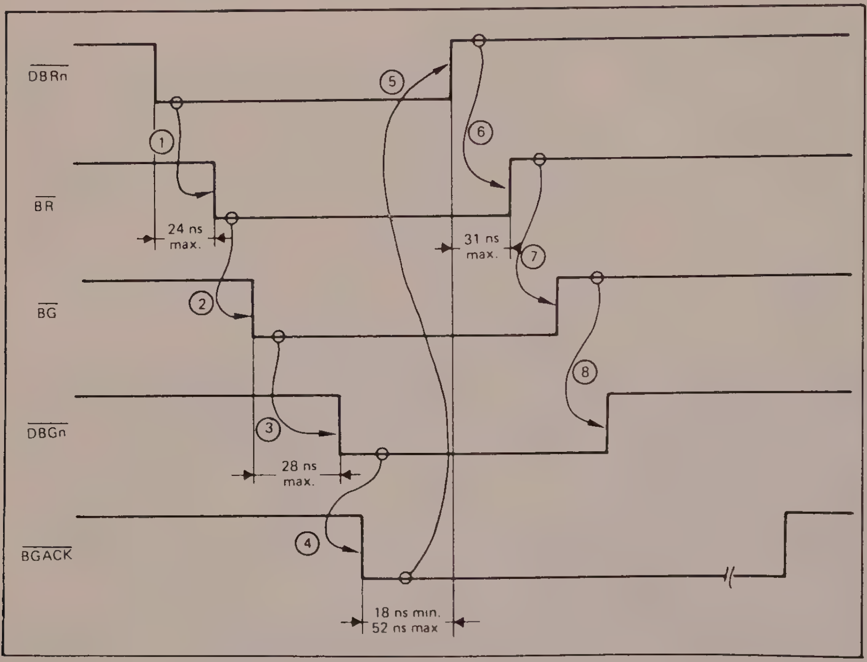


Figure 3.14 Communication between requesting circuit and processor via bus arbitration circuit

HALT AND SINGLE STEP OPERATION

1 Functions of the Bidirectional HALT Line

1.1 On input

Keeping the RESET and HALT lines at low causes a processor RESET (initialisation phase of the 68000) or return to a system function following a double error bus.

If an external circuit activates the HALT line during handling of a bus cycle, the processor terminates its cycle (AS = 1 at the end of the cycle) before being stopped on the next bus cycle to be executed. This next cycle places the address and data lines at the high impedance state.

During the entire halt period the control lines UDS, LDS, AS and R/W are inactive while the bus arbitration lines, namely BR, BG and BGACK are available.

The halt or single step mode allows the instruction executing program to be debugged, bus cycle by bus cycle. This function is complementary to the trace mode which authorises the processor to execute the program instruction by instruction.

Once the HALT line (on input) is again at high, two clock periods (T1 and T2) are necessary before the processor can resume the remainder of the program (see figure 3.15).

If an external circuit enables inputs BERR and HALT (BERR and HALT at low), the processor terminates its bus cycle before proceeding, subject to certain conditions (non-indivisible cycle and HALT line reset at high), to the rerun of the preceding cycle.

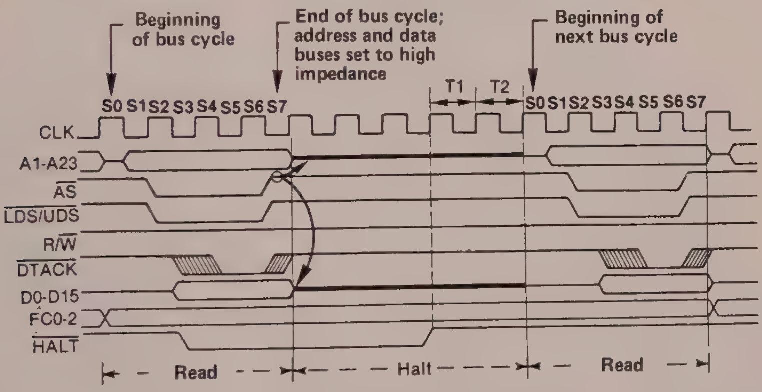


Figure 3.15 Timing diagram to show processor HALT (Courtesy of Motorola)

1.2 On output

When the HALT line is asserted on output (HALT at low), the outside world is alerted that the processor is halted, following a hardware of software event. Only an action on the RESET pin will cause it to leave this state.

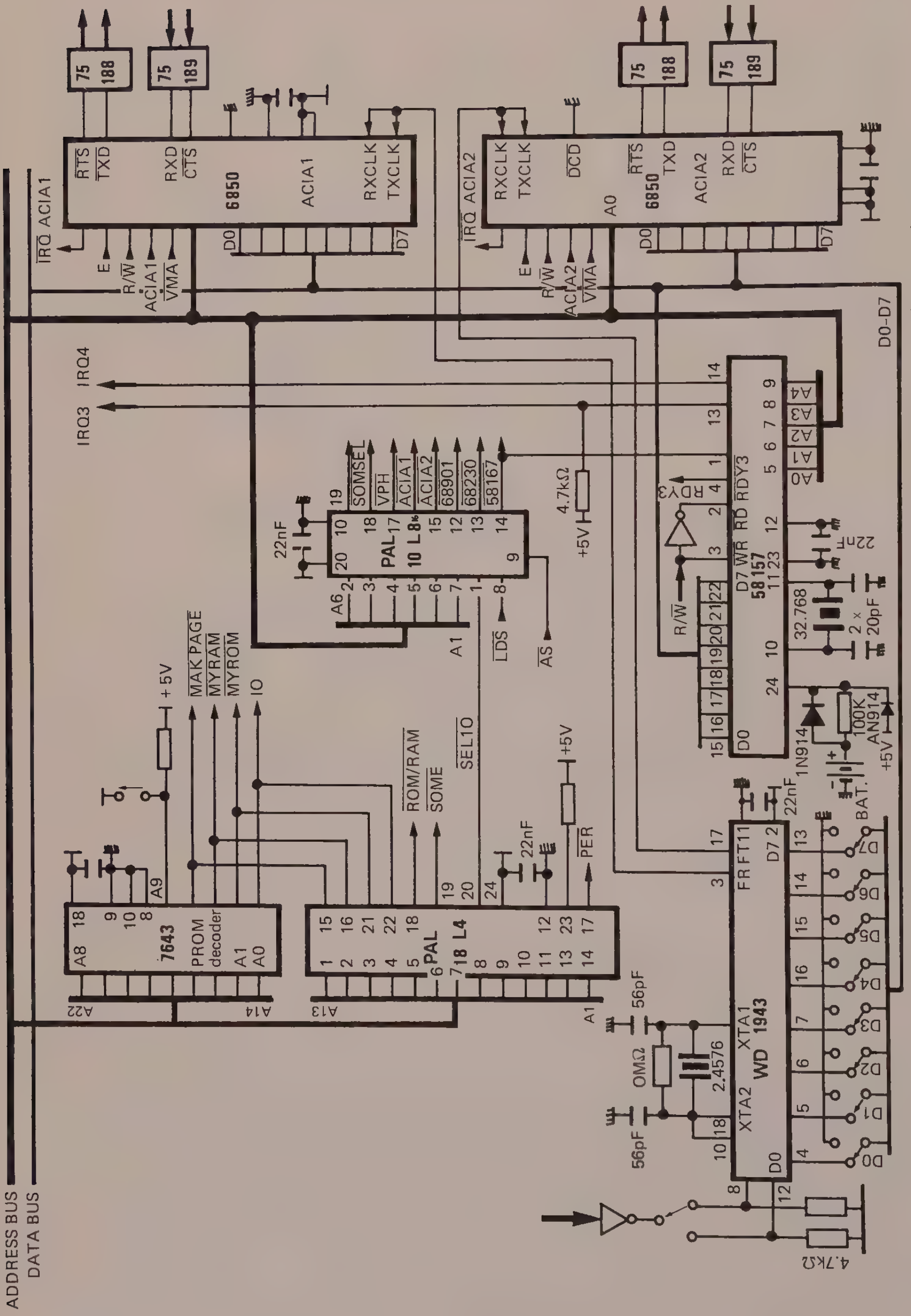


Figure 3.16 Use of programmable array logic devices for input/output decoding. (Copyright Microprocess)

INTERACTION WITH SYNCHRONOUS CIRCUITS (6800 FAMILY)

Interfacing with 6800 Family

1.1 Review

Although synchronous, the 6800 family is hardware compatible with the MC 68000, something that was not easily achieved, as the reader may imagine.

The solution adopted by Motorola of providing on the MC 68000 the means for the exchange of important signals with the MC 6800 is probably the most satisfactory.

Three lines from the MC 68000 ensure interfacing with the peripheral circuits of the 6800 family

clock E (Enable)
validation of a peripheral address
($\overline{\text{VPA}}$: Valid Peripheral Address)
validation of an addressed position
($\overline{\text{VMA}}$: Valid Memory Address).

The clock signal E, which is equal to one-tenth of that fed to the clock input of the MC 68000, has a cyclic relationship of 60/40 (six periods of input clock at low and four clock periods at high). Note that this cyclic relationship for example avoids the need for the 68000 to have to resynchronise itself after each movement of bytes when handling such instructions as MOVEP.W (MOVEP.L). This feature allows two successive VPAs for consecutive E periods.

It quickly occurs to the user that certain 6800 applications may require a clock frequency greater than that available at the output E of the MC 68000 (E = 1/10 of the frequency on the CLK input of the MC 68000), such as for example the advanced communications controller circuit, ADLC MC 6854, which in order to effect a high speed data transfer, most usually requires a clock frequency of 2 MHz.

In this case, the user will have to solve the problem by hardware means; for example, by "refabricating" the clock E.

1.2 Exchange Protocol (MC 68000/MC 6800)

We assume that the 6800 peripheral interacts with the 68000 processor by means of the lower line D0-D7, enabled by $\overline{\text{LDS}}$ = 0 (or via the upper line D8-D15, enabled by $\overline{\text{UDS}}$ = 0).

State 0 : S0

line R/$\overline{\text{W}}$ is at read (preceding cycle)
address lines are at high
lines FC0-FC2 show the processor status.

State 1 : S1

address bus is freed from the high impedance state
the processor places the current address on lines A1-A23.

State 2 : S2

the 68000 address strobe $\overline{\text{AS}}$ is asserted, indicating that there is a valid address on the bus.

Write
Line R/$\overline{\text{W}}$
is set to
write (R/$\overline{\text{W}}$ = 0)

Read
Output $\overline{\text{LDS}}$
low, enabling
channel D0-D7
(or $\overline{\text{UDS}}$ for
channel D8-D15)

State 3 : S3
The processor presents
the data item on channel D0-D7,
then one half clock cycle
later, asserts $\overline{\text{LDS}}$
($\overline{\text{LDS}}$ = 0) confirming
the validity of
the data on D0-D7.
(Alternatively, on D8-D15 by
asserting $\overline{\text{UDS}}$.)

State 3 : S3

At this point, the processor waits for the $\overline{\text{VPA}}$ signal at low level, by inserting waits W. In fact, the VPA input informs the MC 68000 that the current address belongs to a 6800 family circuit or to a memory field reserved for the MC 6800. When the 68000 recognises the VPA signal on the low state of E, it is aware that a synchronous peripheral wishes to interact at the E clock rate after synchronisation. The VMA output, which is used for decoding 6800 peripheral circuits (selection or deselection), is asserted by the 68000 (VMA = 0), two clock periods before the E signal moves to high.

Write
States 5 and 6
(S5 and S6)
The data item is
latched when clock
E moves from
high to low.

Read
States 5 and 6
(S5 and S6)
The processor
carries out a read
at the high state
of signal E.

State 7 : S7

Output $\overline{LDS}$ (or $\overline{UDS}$) is disabled by the 68000.
Outputs $\overline{AS}$ and $\overline{VMA}$ are set high, which authorises the 6800 peripheral circuit to disable $\overline{VPA}$($\overline{VPA}$ = 1).
Note that DTACK must on no account be enabled at the same time as $\overline{VPA}$.

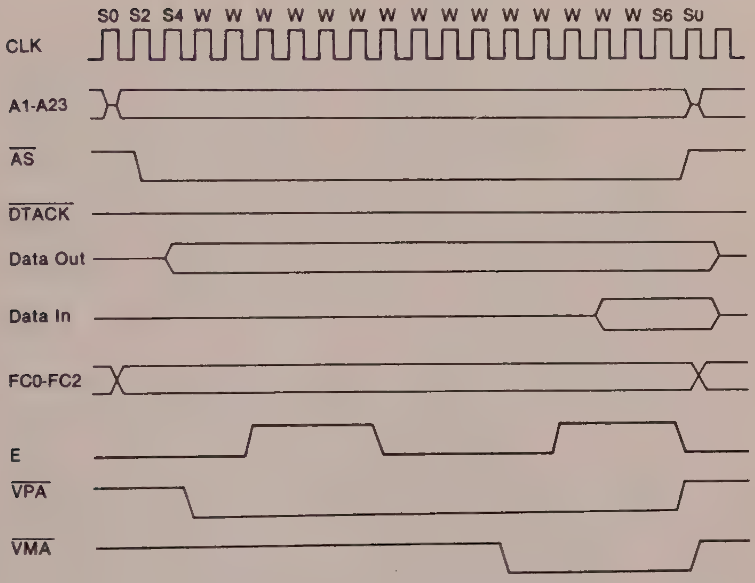


Figure 3.17 "Worst" timing exchange with a synchronous peripheral (6800 family)

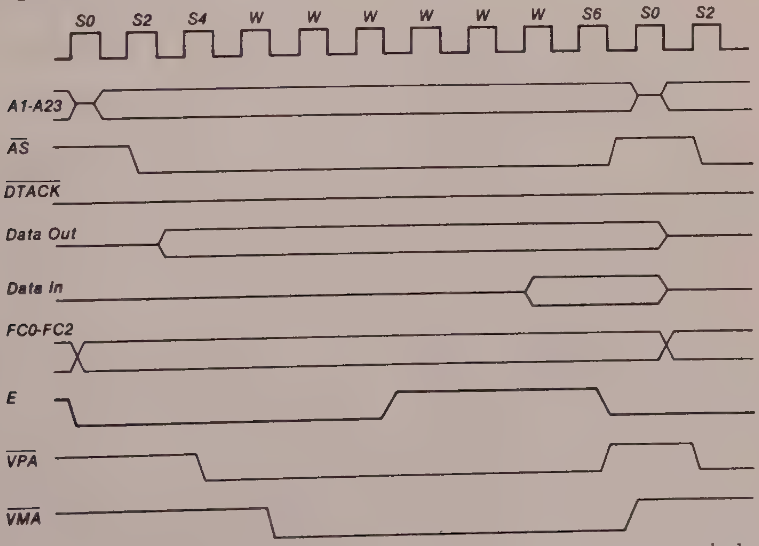


Figure 3.18 "Best" timing with a synchronous peripheral (6800 family)

4 Exception Procedures

EXCEPTIONS

1 Overview

The name trap or exception is given to a change in routeing of the program which is generally the result of internal (software) or external (hardware) events.

Each trap or exception has a byte associated with it that represents a vector number which, when multiplied by four, gives the offset of the corresponding vector.

The 68000 microprocessor contains 255 vectors in memory, arranged in an exception table 512 words in length (1024 bytes), from address $000000 to address $0003FF. See table 4.1.

Each exception vector is 32 bits long, except for the initialisation vector which is coded in 64 bits.

It should also be noted that all the vectors in the exception table are located in the data supervisor memory area, except for the initialisation vector which resides in the program supervisor memory area, thus providing greater security. Table 4.1 shows how the exception table is organised.

Before embarking on a detailed study of the different types of exception, it is valuable to have a knowledge of the general procedure followed by the processor when handling an exception. This may be summarised as follows

1. A temporary copy of the status register is made in an internal register of the 68000.
2. Bit S (S = 1) is asserted, thus placing the processor in supervisor mode. All exceptions will therefore be handled in supervisor mode.
3. Trace bit (T = 0) of the status register is disabled.
4. The vector number is obtained.
5. The program counter and the previously copied status register are saved to the supervisor stack. (Additional information is stored in the case of address error or bus error exceptions.)
6. The table is consulted for the start address of the exception program.

We shall see that only exceptions caused by an external event do not more or less exactly follow this procedure.

Table 4.1 Exception Vector Assignment

Vector Nos.	Addresses Dec	Hex	Space	Assignment
0	0	000	SP	Reset supervisor stack
-	4	004	SP	Reset program counter
2	8	008	SD	Bus error
3	12	00C	SD	Address error
4	16	010	SD	Illegal instructions
5	20	014	SD	Division by zero
6	24	018	SD	CHK instruction
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	48	030	SD	Unassigned, reserved
13	52	034	SD	Unassigned, reserved
14	56	038	SD	Unassigned, reserved
15	60	03C	SD	Uninitialised interrupt vector
16-23	64	04C	SD	Unassigned, reserved
-	95	05F		Unassigned, reserved
24	96	060	SD	Spurious interrupt
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
	191	0BF	SD	TRAP instruction vectors
48-63	192	0C0	SD	Unassigned, reserved
	255	0FF		Unassigned, reserved
64-255	256	100	SD	User interrupt vectors
	1023	3FF		

Note: SP = Supervisor Program (FC2 = FC1 = 1, FC0 = 0)
SD = Supervisor Data (FC2 = FC0 = 1, FC1 = 0)

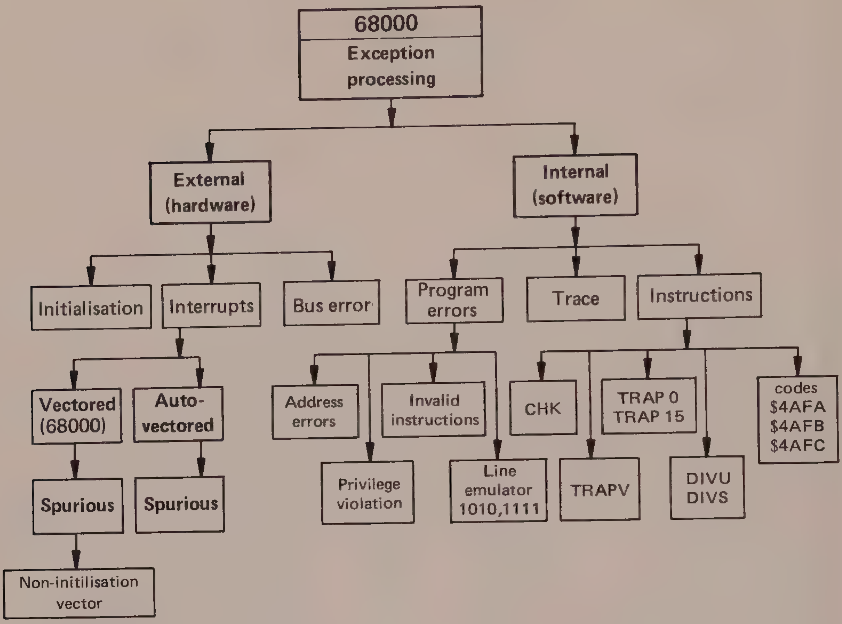


Figure 4.1

2 Internal Exceptions

Internal exceptions or traps occur in the following instances

1. when certain instructions are being carried out
2. when there is a programming error
3. when the trace bit of the status register is asserted.

This new concept confirms the software bias of the MC 68000 and the considerable importance that the manufacturer has attached to ensuring the security of the system.

Handling Certain Instructions

a) Division by zero : Vector number 5

Division by zero is a typical example of the trap set by the MC 68000 for the programmer when executing the instructions DIVS and DIVU where the divisor is zero.

When this occurs, the processor generates internally the vector number 5 (corresponding to trap division by zero) which when multiplied by four gives the address of the vector associated with the trap. The address, which is found in the exception table (5 x 4 = $14), will need to be loaded by the programmer with the start address of the division by zero program.

Of course, in practice it is up to the programmer to write a subroutine, for example to recalculate the divisor more accurately, send an error message or simply halt execution of the program.

b) TRAP CHK : Vector number 6
The CHK instruction allows one to test whether the operand contained in a data register lies between two values. If this is the case, the program continues: if not, the program is rerouted in the exception table to address $18 (6 x 4 = 24 = $18), in order to recover the start address of the exception program CHK.

c) TRAPV : Vector number 7
If the condition V = 1 is satisfied on execution of the instruction TRAPV, the program is branched to the associated vector to find the address of the trap exception procedure in the event of overflow.

This "exceptional" instruction is completely tailored to ensure that the 68000 has satisfactorily carried out the division required by the DIVU or DIVS instructions (see chapter 6). In fact, if the processor suspects that overflow will occur, it does not carry out the division but sets V to 1.

d) TRAPs available to the user: Vector numbers 32 to 47
There are 16 TRAP instructions, called TRAP #0, TRAP #1 to TRAP #15, which provide the programmer with the facility to reroute at will a program sequence to a resource systems type of exception procedure (for example, breakpoints).

Program Error
There are three types of exception or trap resulting from a programming error.

a) Address error : Vector number 3

Unfortunately, when the 68000 processor detects an address anomaly (for example, writing a word to an odd address), it is too late to halt execution of the instruction, as the 68010 would. Figure 4.2 shows that the signals UDS and LDS are asserted by the 68000 (not by the 68010), which will execute the procedure before being rerouted to the trap, address error, but after executing the following microprogram

1. copy the status register (SR) in an internal register of the 68000
2. enable the supervisor mode by S = 1
3. disable the trace mode by T = 0.

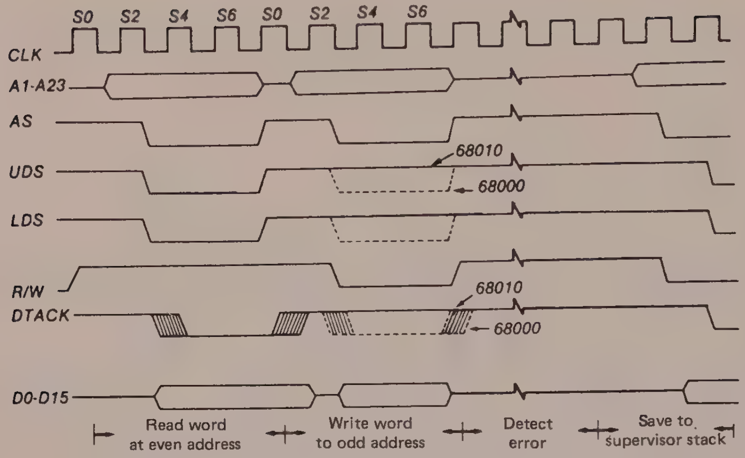


Figure 4.2 Timing of an address error

Since the 68000 processor has not been able to execute the current instruction, the contents of the program counter (PC) are incremented by between two and ten bytes in relation to the address of the first word of the instruction, before being saved to the supervisor stack. This particular feature may cause problems for the programmer when it comes to determining in the exception program what the main program return address should be (see figure 6.6).

The following have to be saved

1. the status register previously copied in the internal register of the 68000 - to the supervisor stack.

2. the instruction register containing the op-code of the aborted instruction.

3. the current value of the address transmitted on lines A1 to A23 - also saved to the supervisor stack.

Finally, the manufacturer offers the user a super status word consisting of information that indicates

1. the status of the processor at the moment of the error by means of lines FC0 to FC2;

2. whether the processor was at read or write;

3. whether the processor was handling an exception or not.

Although inadequate, compared with the 26 words stored by the 68010 in the same circumstances, the above information does allow the situation to be analysed quite precisely.

b) Privilege violation : Vector number 8

If the processor tries to execute a supervisor resource in user mode, the program is rerouted to a trap called "privilege violation".

In fact, as we have already noted, all functions are accessible to the programmer in supervisor mode, whereas in user mode the resources capable of modifying the system state are said to be privileged.

This concept of hierarchical levels, which exists already on some minicomputers, provides the system with a degree of security never achieved with 8-bit processors. For example, resource systems are protected in a multi-user configuration, such as the OS9 68K from Microware and Motorola.

The privileged instructions are

```
STOP                   AND.W # data,SR
RESET                  EOR.W # data,SR
RTE                    OR.W # data,SR
MOVE.W Source,SR       MOVE.L An,USP
                       MOVE.L USP,An
```

c) Illegal instructions : Vector number 4
Unimplemented instructions

```
code 1010   vector number 10
code 1111   vector number 11
```

An instruction is said to be invalid if the 4-bit MSB of the instruction word is not recognised by the processor. If such a combination is decoded by the instruction decoder register of the CPU, an invalid instruction trap is produced.

Three op-codes, $4AFA, $4AFB and $4AFC, lead to an illegal procedure. The first two ($4AFA and $4AFB) are reserved by Motorola for the system; the third $4AFC can be used by the programmer to force the 68000 into an illegal instruction trap.

Instructions whose binary combination of bits 15 to 12 corresponds to code $A(1010) and $F(1111) are not implemented on the 68000, 68008 and 68010. However, they are assigned an emulation vector which allows the operating system to detect certain program errors or to emulate instructions unimplemented by the manufacturer but developed by the user.

Note that the new 16/32-bit MC 68020 microprocessor recognises the binary combination 1111(F), which allows it to use the MC 68881 floating point co-processor. The MC 68020 has additional instructions (such as MOVE F, MOVE SB, PACK and UNPK), some of which manipulate strings of ASCII characters.

Assertion of Trace Bit (T = 1) : Vector number 9
After executing each instruction the processor tests internally the logical state of the status register

trace bit T. If this is asserted (T = 1), the processor obtains the start address of the trace program from the exception table. If on the other hand bit T is negated (T = 0), the processor moves on to execute the next instruction (see programming example in chapter 7).

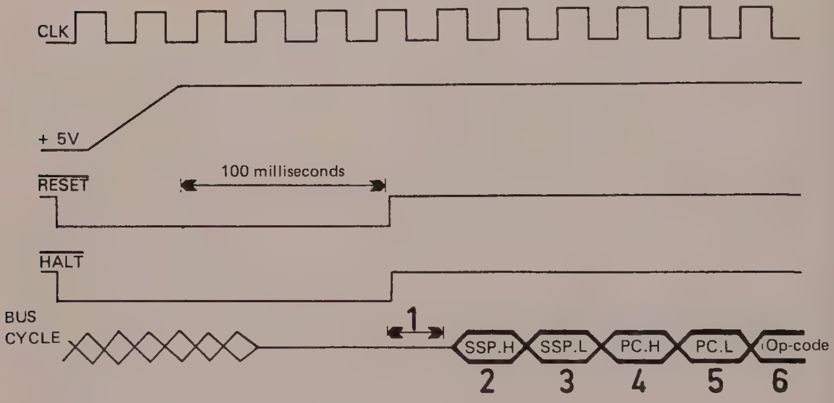
4 External Exceptions

External exceptions can be generated by the following.

1. On start, keeping both the RESET and HALT lines low for 100 ms, this being the initialisation phase: or again low for 10 clock cycles in order to exit the 68000 from the HALT state following a double bus error.

2. A bus error detected by an external device or by a MC 68451 MMU circuit, which asserts the input BERR of the 68000 processor low, following a hardware anomaly during execution of an instruction (protected segment on write).

3. An interrupt request made to the processor by means of lines IPL0, IPL1 and IPL2.



1. Internal start up time.
2. Load 16 high order bits of the supervisor stack pointer.
3. Load 16 low order bits of the supervisor stack pointer with the contents of addresses $000002 and $000003.
4. Load 16 high order bits of program counter with the contents of addresses $000004 and $000005.
5. Load 16 low order bits of program counter with the contents of addresses $000006 and $000007.
6. Fetch first instruction.

Figure 4.3 Sequence of operations on RESET (Courtesy of Motorola)

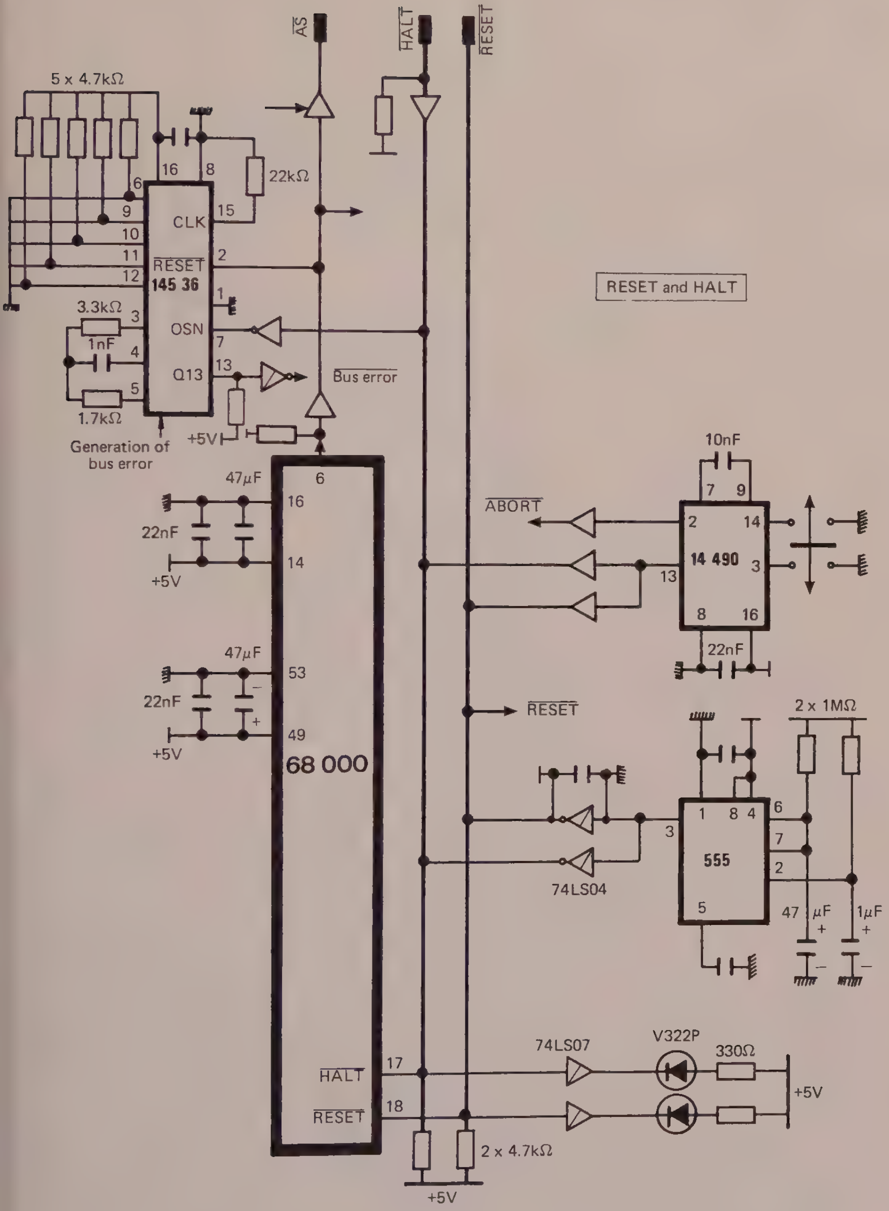


Figure 4.4 $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ circuit.
(Copyright Microprocess)

Initialisation exception or reset : Vector number 0

When requested by an external source, initialisation of the processor is carried out by holding low for 100 ms the bidirectional lines RESET and HALT that are fixed on entry (see figure 4.3).

Initialisation consists of the following sequences

a) Bit S of the status register is asserted (S = 1), thus placing the processor in supervisor mode.

b) Bit T (trace) of the status register is negated (T = 0), disabling the trace function.

c) The interrupt mask (I2, I1, I0) is set at level 7, that is, I2 = 1, I1 = 1 and I0 = 1.

d) Since the vector number is generated internally, the processor loads the supervisor stack pointer with the contents of addresses $000000 and $0000003 and the program counter with the contents of addresses $0000004 and $0000007. We now understand why the initialisation vector is coded in 64 bits. The processor then executes the instructions located at the address initially loaded in the PC.

If an anomaly occurs during recovery of the initialisation (reset) vector, the processor reacts as though there is a catastrophic error, called a double bus error, by placing itself at the halt state, so inhibiting all program execution.

Control of the system can only be regained by action on the RESET and HALT line lasting 10 clock cycles.

Bus error : Vector number 2

A bus error may occur for example when the processor tries to access a nonexistent or protected work area, such as missing or faulty peripheral devices or protected RAM memories.

When an external electronic circuit recognises a bus error, it pulls the BERR line low, thus alerting the processor that there is/ a hardware anomaly during execution of the instruction. The timing diagram of figure 4.5 shows the different stages that lead the 68000 to handle the bus error exception. As in the case of the address error exception, this operation is preceded by saving on the supervisor stack any information that may help the user to identify the hardware error. Figures 4.6 and 4.7 show how the bus error is handled by the processor and how the information is stored in the supervisor stack.

If an anomaly occurs during handling of the bus error exception, the processor goes to halt, which stops the exception handling. This is known as a double bus error.

Once again, system control can only be regained by action on the reset and halt lines.

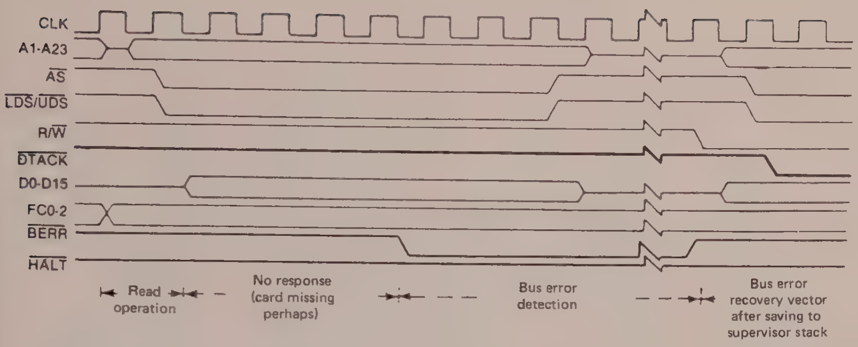

CLK
A1-A23
AS
LDS/UDS
R/W
DTACK
D0-D15
FC0-2
BERR
HALT
Read operation
No response (card missing perhaps)
Bus error detection
Bus error recovery vector after saving to supervisor stack

Figure 4.5 Timing of a bus error
(Courtesy of Motorola)

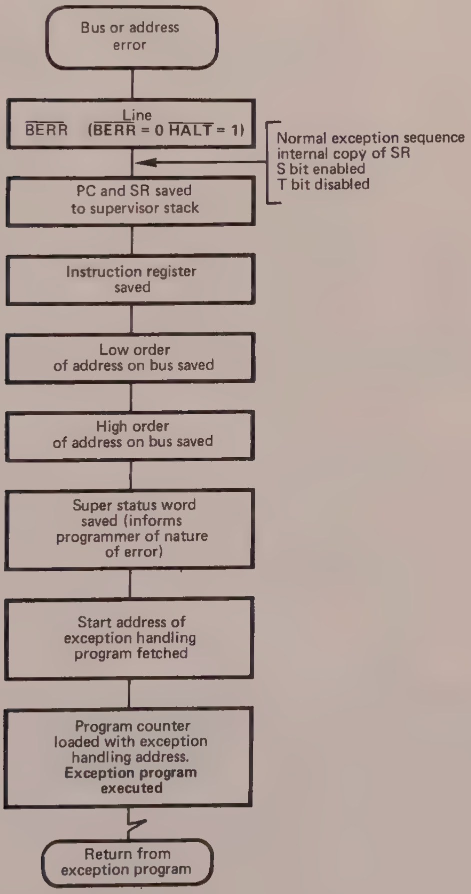

Bus or address error
BERR Line (BERR = 0 HALT = 1)
Normal exception sequence
internal copy of SR
S bit enabled
T bit disabled
PC and SR saved to supervisor stack
Instruction register saved
Low order of address on bus saved
High order of address on bus saved
Super status word saved (informs programmer of nature of error)
Start address of exception handling program fetched
Program counter loaded with exception handling address. Exception program executed
Return from exception program

Figure 4.6 Bus error handling

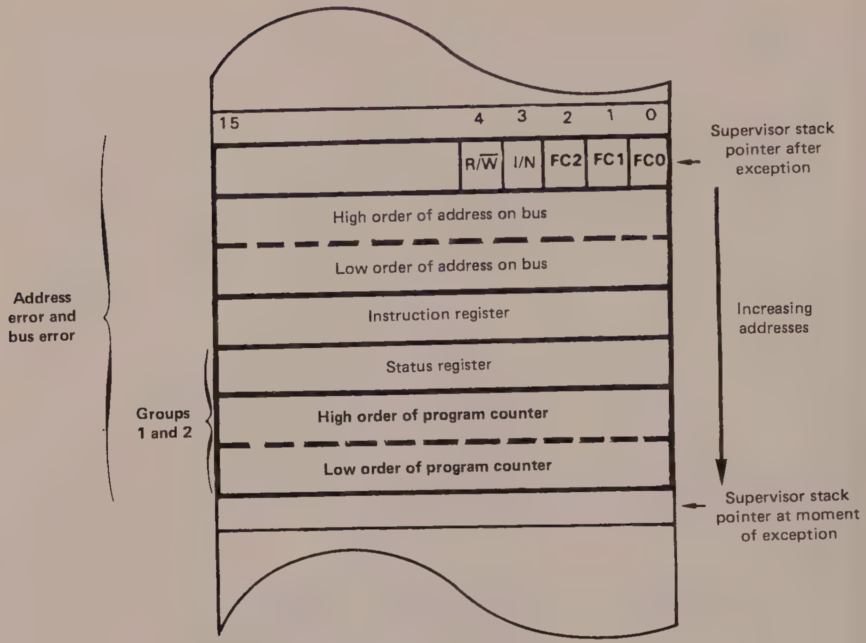

15
4
3
2
1
0
R/W̄
I/N
FC2
FC1
FC0
Supervisor stack pointer after exception
High order of address on bus
Low order of address on bus
Instruction register
Status register
High order of program counter
Low order of program counter
Increasing addresses
Supervisor stack pointer at moment of exception
Address error and bus error
Groups 1 and 2

Figure 4.7 Information saved to the supervisor stack when a bus error or an address error occurs

Definition of Status Super Word
Function code : logical status of lines FC2, FC1, FC0 (see figure 1.2)

I/N	: 0	instruction in course of execution
	: 1	exception in course of execution (this was not an instruction)

$R/\overline{W}$	: 0	the processor was at write
	: 1	the processor was at read

Interrupts
The 68000 microprocessor possesses 192 usable vectors for peripherals that can provide a vector number (for example, MFP 68901, PI/T 68230, etc) and 7 autovectors allocated to 6800 family circuits (ACIA 6850, Timer 6840 and PIA 6821) that do not generate a vector number.

Seven levels of priority, fixed by the programming of the interrupt mask (see status register), can be assigned to these 199 vectors, as shown in the following table.

Level	I2	I1	I0	
7	1	1	1	Highest priority
6	1	1	0	
5	1	0	1	
4	1	0	0	
3	0	1	1	
2	0	1	0	
1	0	0	1	Lowest priority
0	0	0	0	No priority (no interrupt request)

The levels of interrupt are numbered 1 to 7 (level 0 indicating that there is no interrupt request), with level 1 being the lowest priority and level 7 the highest.

Interrupt recognition
When an interrupt request reaches the processor, it is first made to wait, before being interpreted by the processor at the end of the instruction cycle. (See figure 4.7 for timing diagram.)
If the interrupt level present on lines $\overline{\text{IPL0}}$, $\overline{\text{IPL1}}$ and $\overline{\text{IPL2}}$ is less than or equal to the interrupt mask, the processor executes the next instruction and ignores the request. However, if the request level is greater than that of the mask, the processor proceeds to the interrupt recognition described below. Note that level 7 priority is a special case; it cannot be inhibited by the interrupt mask. Level 7 interrupt thus provides a non-maskable interrupt capability.

1. The contents of the status register (SR) are copied into an internal (non-programmable) register of the 68000.
2. Bit S of the status register is asserted (S = 1), thus placing the 68000 in supervisor mode.
3. Bit T of the SR is negated (T = 0), disabling the trace function.
4. The interrupt level present on lines $\overline{\text{IPL0}}$ to $\overline{\text{IPL2}}$ is recopied into the status register. Because of this, the processor will only be able to be interrupted by an interrupt level greater than the one currently being handled.
5. Address lines A1 to A3 reflect the status of lines $\overline{\text{IPL0}}$ to $\overline{\text{IPL2}}$, so that an external decodirg logic can determine which interrupt request is currently being recognised.

The remaining lines of the address bus (A4-A23) and the outputs FC0, FC1 and FC2 are set to 1.

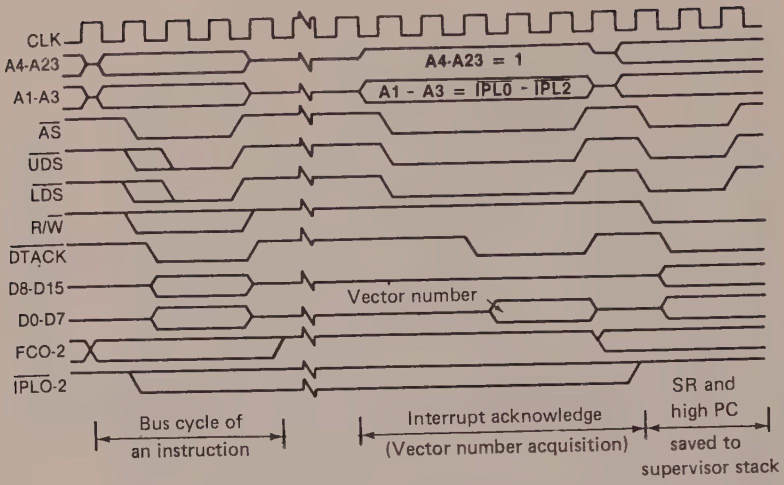

CLK
A4-A23
A1-A3
AS
UDS
LDS
R/W
DTACK
D8-D15
D0-D7
FC0-2
IPL0-2
A4-A23 = 1
A1 - A3 = IPL0 - IPL2
Vector number
Bus cycle of an instruction
Interrupt acknowledge (Vector number acquisition)
SR and high PC saved to supervisor stack

Figure 4.8 Interrupt from 68000 peripherals
(Courtesy of Motorola)

Problem

Does the interrupt recognised by the processor come from a 6800 peripheral (figure 4.8) or from a 68000 peripheral (figure 4.9)?

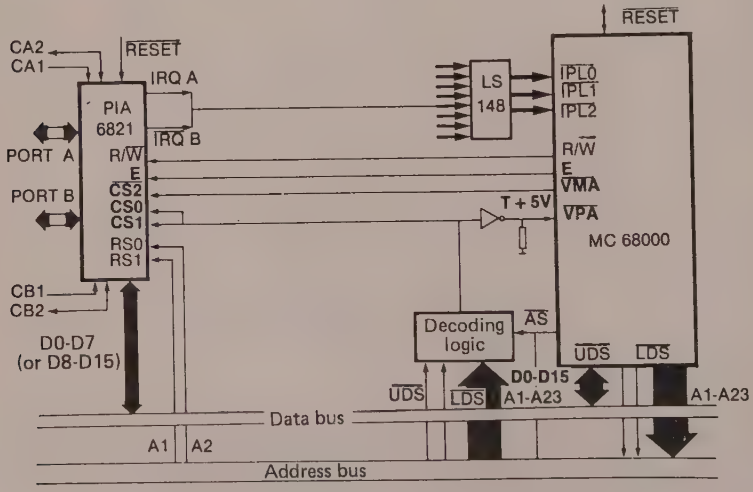


Figure 4.9

6a. 6800 peripheral
The VPA line (valid peripheral address) at low alerts the 68000 processor that a 6800 peripheral (PIA, ACIA) requires its attention and that, in order to work with it, it must synchronise itself with the clock

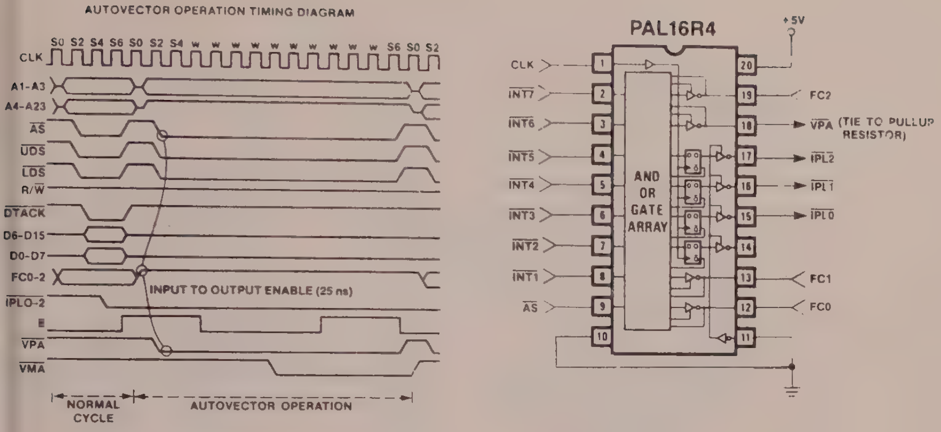


Figure 4.10 Autovectorisation using programmable array logic
(Courtesy of Monolithic Memories)

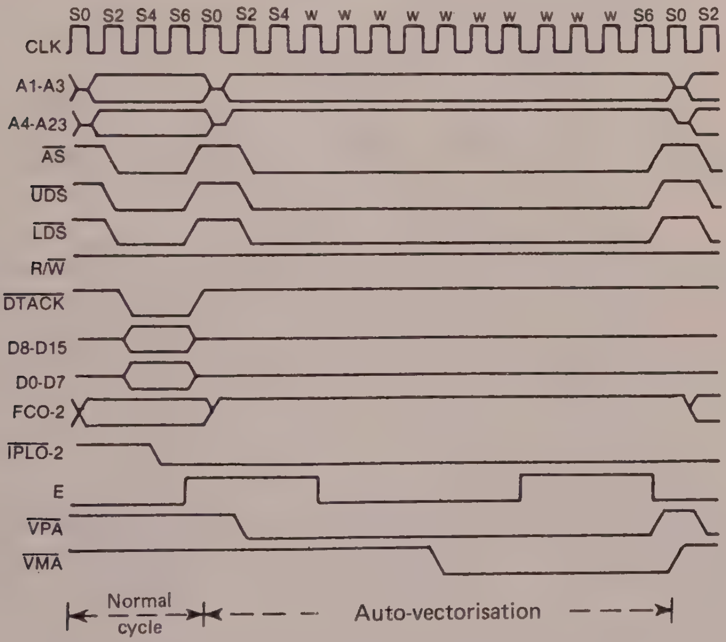


Figure 4.11 Timing of interrupts from 6800 peripherals
(Courtesy of Motorola)

signal E, which is equivalent to the 02 of the 6800. On enabling VMA (valid memory address), the 68000 addresses the peripheral and indicates that it is ready to interact in synchronisation with clock E (the 68000 is synchronised).

6800 peripherals do not generate vector numbers. The

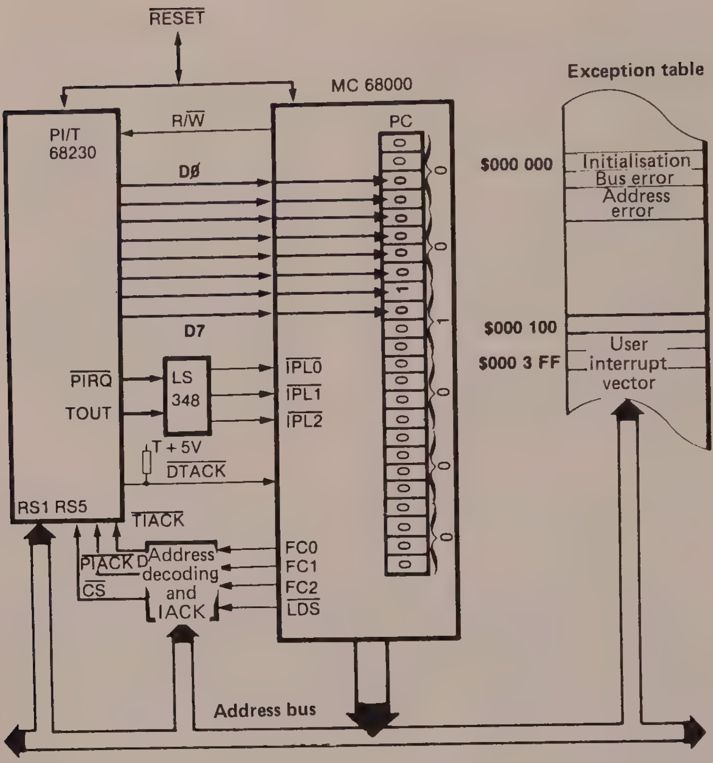


Figure 4.12

68000 therefore uses the autovectorisation procedure that allows it to access the 7 autovectors of the exception table (numbers 25_{10} to 31).

The vector number is determined from the priority level established by lines $\overline{\text{IPL2}}$ to $\overline{\text{IPL0}}$, remembering that these lines are enabled at low state.

Example
Interrupt mask: I2 = 1; I1 = 0; I0 = 1
Interrupt request: $\overline{\text{IPL2}}$ = 1; $\overline{\text{IPL1}}$ = 1; $\overline{\text{IPL0}}$ = 0
The 68000 "internally" supplies the vector number 30 that corresponds to level 6 (vector address = 30_{10} x 4 = 120_{10} = \$78).

6b. 68000 peripheral

When the decoding logic recognises the interrupting circuit, it places a vector number on lines D0-D7 of the data bus ($\overline{\text{LDS}}$ = $\overline{\text{UDS}}$ = 0) and sends the $\overline{\text{DTACK}}$ signal to confirm the transfer of a data item.

When the 68000 recognises $\overline{\text{DTACK}}$ ($\overline{\text{DTACK}}$ = 0) during a read cycle (R/$\overline{\text{W}}$ = 1), the data are latched and the read bus cycle is terminated, changing UDS and LDS to high, and disabling DTACK.

Interrupt Controller Logic Diagram
External Vector Generation

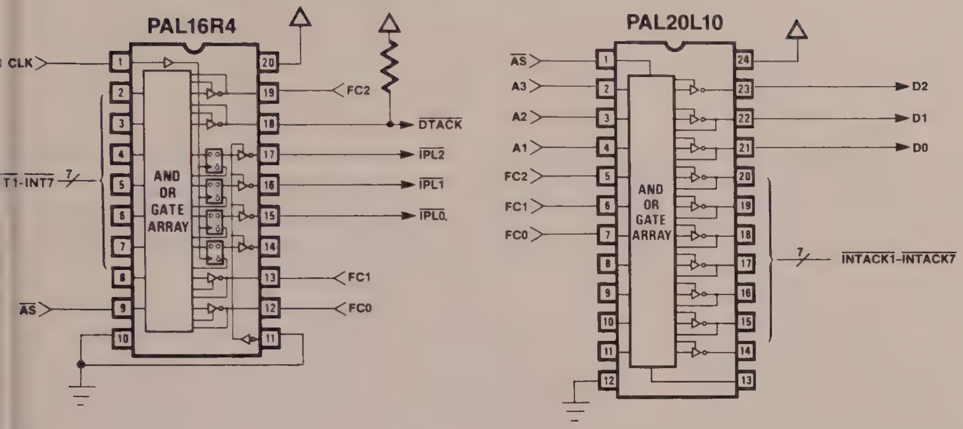
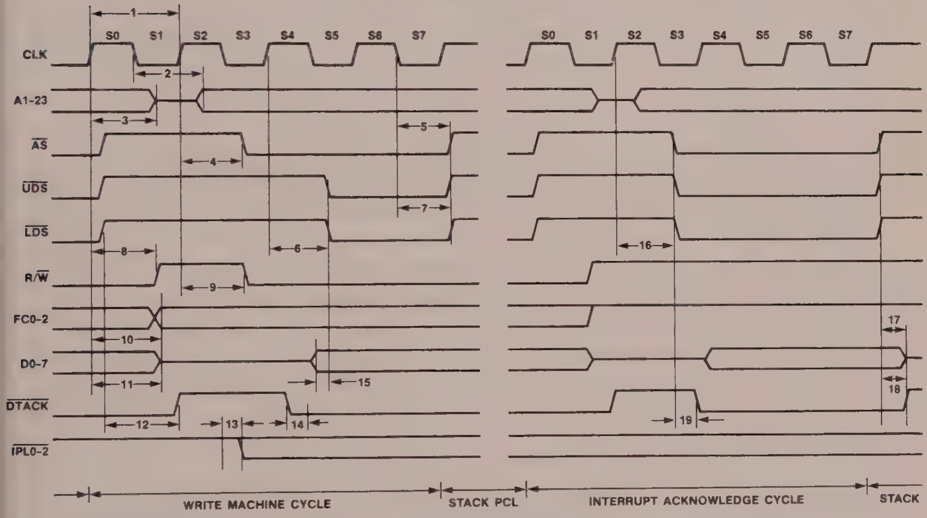


Figure 3



1 CLOCK PERIOD (80 ns)
2 CLOCK LOW TO ADDRESS (55 ns)
3 CLOCK HIGH TO ADDRESS HI-Z (60 ns)
4 CLOCK HIGH TO AS LOW (55 ns)
5 CLOCK LOW TO AS HIGH (50 ns)
6 CLOCK HIGH TO DS LOW (55 ns)
7 CLOCK LOW TO DS HIGH (50 ns)
8 CLOCK HIGH TO R/W HIGH (60 ns)
9 CLOCK HIGH TO R/W LOW (60 ns)
10 CLOCK HIGH TO FC VALID (55 ns)
11 CLOCK HIGH TO DATA HI-Z (60 ns)
12 AS HIGH TO DTACK HIGH (70 ns)
13 PAL's CLOCK TO OUT (25 ns)
14 MINIMUM SETUP TIME (20 ns)
15 DATA VALID TO DS LOW (15 ns)
16 CLOCK HIGH TO AS, DS LOW (55 ns)
17 PAL's INPUT TO HI-Z (25 ns)
18 PAL's INPUT TO HI-Z (25 ns)
19 PAL's INPUT TO HI-Z (25 ns)

Figure 4.13 Interrupt vectorisation using programmable array logic (Courtesy of Monolithic Memories)

Example
Suppose that the peripheral circuit PI/T 68230 (see section 4.2) positions the vector number 64 in base 10 (equivalent to 40 in base 16). Figure 4.11 shows that the address pointed to, equal to \$40 x 4 = \$100, is indeed the vector corresponding to the number 64_{10}.

Problem
What happens if during interrupt recognition no peripheral (whether 6800 or 68000) replies by maintaining at low the signals VPA (for the 6800 family) and DTACK (for the 68000)?

Answer
The explanations given so far would lead one to assume that the BERR line (bus error exception) would terminate the acquisition of the vector number. Now, the the program is rerouted by the 68000 towards the spurious interrupt vector number 24_{10}.

7. Saving the program counter to the supervisor stack.
8. Saving the status register, previously copied in an internal 68000 register, to the supervisor stack.
9. Retrieval of the begin address of the interrupt program in the specified vector.
10. Execution of the interrupt program which, like every exception program, will have to terminate with the instruction RTE (Return from Exception).

Uninitialised Interrupt Vectors : Vector number 15_{10}
Some 68000 peripherals have both vectored and autovectored interrupts; an example is the PI/T MC 68230 circuit (see section 4.2).

This circuit has two interrupt vector registers; one is assigned to the ports (PIVR) and the other to the timer (TIVR).

On interrupt vectorisation, the vector number placed on lines D0-D7 by the PI/T 68230 is the operand previously loaded into these registers by the programmer.

Example
Suppose that the PI/T 68230 circuit generates a vectored interrupt after time out (that is, at the timer level), and that the timer interrupt vector register (TIVR) has not been initialised. In this case, and in common with most of the 68000 peripherals that have interrupt vector registers, the MC 68230 peripheral dispatches the uninitialised interrupt vector or vector number 15_{10}($0F).

In fact, this number is automatically loaded into the interrupt vector registers (PIVR and TIVR) when there is a reset on the peripheral circuit. As a result, it is possible to recover in a uniform way from a programming error (by having the same number for all circuits).

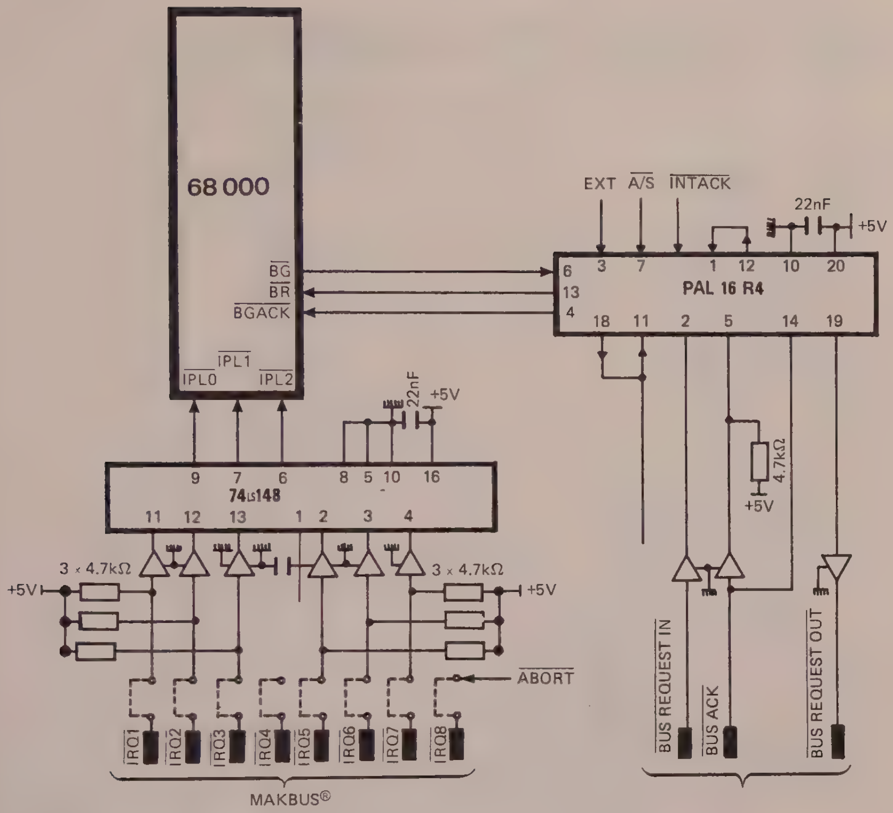


Figure 4.14 Shows interrupt priority control and bus arbitration in a multiprocessor environment. (Copyright Microprocess)

5 Recognition and Exception Priority

Overview

Exceptions are classified by groups according to the following two criteria (see table 4.2).

1. Recognition of the exception by the 68000 processor.

2. The consequential effects of the exception on the program or the system.

Thus a bus error type of exception recognised during a clock cycle (1) with consequences that we know, is not of the same importance for the system as a division by zero type of exception recognised during an instruction cycle (3). This concept inevitably leads one to think in terms of group priority and of hierarchy within the group.

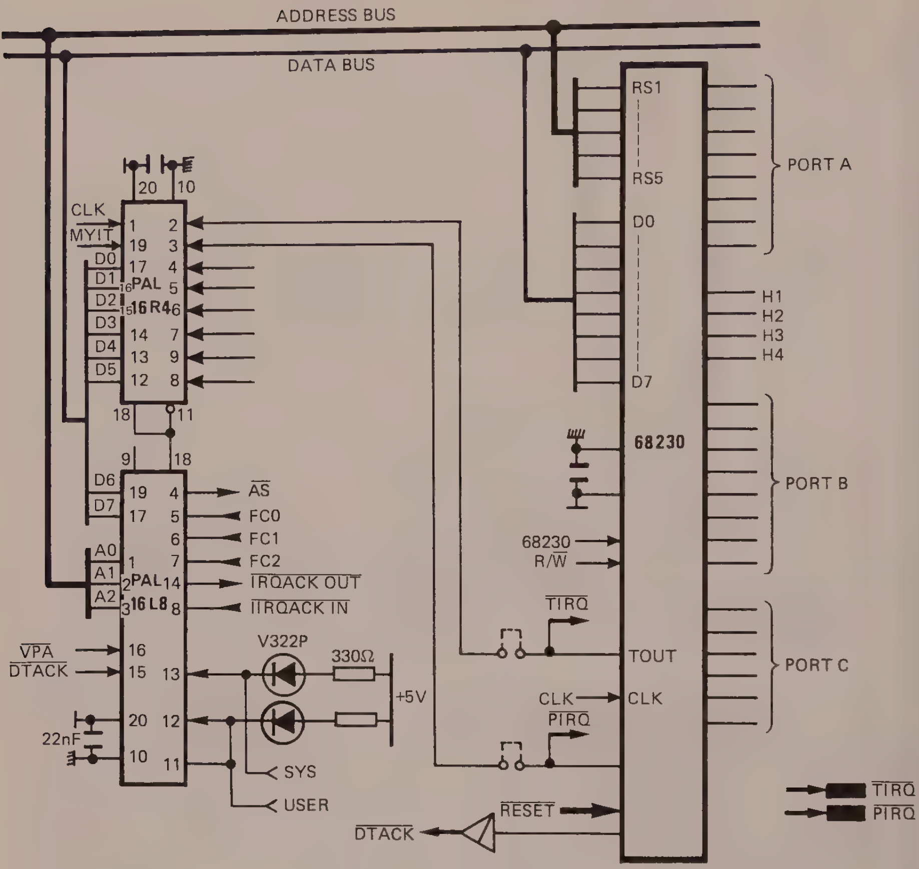


Figure 4.15 Interrupt device interface. The PAL16R4 allows priority encoding on 7 input signals I1 to I7 (I1 highest, I7 lowest). The PAL16L8 decodes lines FC0, FC1, FC2 and A0, A1, A2 so that vectors can be generated in phase with the interrupt acknowledgement. (Copyright Microprocess)

Group 0

Group 0 consists of three exceptions - initialisation, bus error and address error - which are recognised by the 68000 at the end of a clock cycle. These occupy the position of highest priority group. In addition, if during the handling of an address error exception the BERR (bus error) line is set to low, the processor abandons handling the address error exception in order to execute the bus error exception. Thus, even at the heart of group 0 a hierarchy of exceptions is established. As a result, the initialisation exception possesses the highest priority, while the lowest priority is assigned to the address error exception.

Table 4.2 Priorities and Exception Groups

Group	Group	Exceptions	Hierarchy	Recognition
↑	0	Reset Bus error Address error	↑	End of clock cycle
	1	Trace interrupts Illegal instructions Privilege violation	↑	End of instruction cycle
	2	TRAP # 0 to TRAP # 15 TRAPV CHK DIVU,DIVS (if zero div.)	I	During an instruction cycle

↑ Increasing priority
I No priority

Group 1
Of lesser priority than group 0, the exceptions that make up group 1 include those recognised at the end of an instruction cycle (3), like tracing and interrupts, together with those recognised at the end of a bus cycle (2), namely illegal instructions and privilege violations.

The different nature of the exceptions that go to make up group 1 explains the natural hierarchy between them, as can be seen from the following.

Tracing ⟶ Highest priority
Interrupts
Illegal instructions

(including invalid instructions - not implemented - as well as codes $4AFA, $4AFB and $4AFC)
Privilege violation ⟶ Lowest priority

Group 2
Group 2 has the lowest group priority. It consists of instructions that eventually lead to a trap (exception) like CHK and TRAPV.

Given that a single instruction cannot be executed at once, and that recognition takes place during an instruction cycle (3), no hierarchy is established between the instructions of group 2. Table 4.2 summarises the three groups.

Definitions

(1) Clock cycle
Clock period fed to the CLK input of the MC 68000. (T = 125 ns for a frequency of 8 MHz and 100 ns for a frequency of 10 MHz.)

(2) Bus cycle
Time sequence required to complete the following cycles

byte read or write
word read or write
read, modify or write (TAS instruction)

(3) Instruction cycle
Time sequence necessary to execute a 68000 instruction.

Exception	Periods
Address error	50 (4/7)
Bus error	50 (4/7)
Interrupt	44 (5/3)*
Illegal instructions	34 (4/3)
Privileged instructions	34 (4/3)
Trace	34 (4/3)

*It is accepted that the interrupt recognition cycle lasts four external clock periods. The numbers in brackets after the period numbers are the numbers of read and write bus cycles used in order to execute the exception.

Example

Bus error	50 periods made up of 4 read bus cycles and 7 write bus cycles.

TECHNICAL FILE

Parallel Interface/Timer MC 68230
The PI/T MC 68230 peripheral circuit is constructed using HMOS technology and comes in a 48-pin plastic or ceramic package. It is compatible with the MC 68000, MC 6809E and MC 6809.

A Motorola design, this circuit allows one to expand considerably on both the hardware (number of circuits, PIA, timer) and the software of applications as varied as interfacing parallel printers, time measurement, burglar alarms, and so on.

Its asynchronous data bus D0-D7 allows it to operate at speeds (clock frequencies) greater or less than its own, using bus master circuits such as the MC 68450 direct memory access controller.

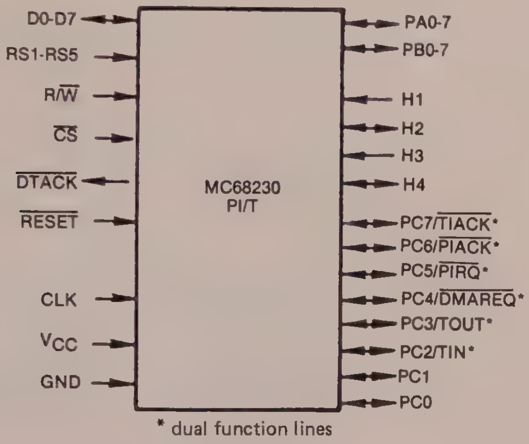

D0-D7
RS1-RS5
R/$\overline{\text{W}}$
$\overline{\text{CS}}$
$\overline{\text{DTACK}}$
$\overline{\text{RESET}}$
CLK
VCC
GND
MC68230
PI/T
PA0-7
PB0-7
H1
H2
H3
H4
PC7/$\overline{\text{TIACK}}$*
PC6/$\overline{\text{PIACK}}$*
PC5/$\overline{\text{PIRQ}}$*
PC4/$\overline{\text{DMAREQ}}$*
PC3/TOUT*
PC2/TIN*
PC1
PC0
* dual function lines

Figure 4.16 MC 68230 pin assignment

Pin Assignment
The MC 68230 has an asynchronous data bus (D0-D7), a read/write line (R/$\overline{\text{W}}$)and a transfer recognition output (DTACK). There are five register selection lines (RS1-RS5) and one circuit selection line (CS). There are two concatenable ports (PA0-PA7 and PB0-PB7), a dual function port (PC0-PC7) consisting of the timer, input, output and interrupt lines (TIN, TOUT and TIACK) and the port interrupt lines (PIACK and PIRQ). Finally, there is the direct memory access (DMAREQ) and initialisation (RESET) line.

The PI/T MC 68230 has three 8-bit ports (A,B,C), four handshake lines (H1-H4), a 24-bit timer and vectored or autovectored interrupts. Note that the port functions are independent of the timer functions.

The 25 registers that are programmable by the data bus fix four operation modes at the port level.

8-bit unidirectional operation
16-bit unidirectional operation (ports A, PA0-PA7, and B, PB0-PB7, can be concatenated to make a 16-bit port)
8-bit bidirectional operation
16-bit bidirectional operation

Each port a has double buffer, consisting of two 8-bit static latches which, for example, allow the throughput to be increased for applications such as the output of characters to a printer.

The direction of the port lines is fixed a) by program in unidirectional mode (8 and 16), and b) by pin H1 in bidirectional mode (8 and 16).

Some of the lines of port C (PC2 to PC7) can be individually programmed to fulfil the following roles

Timer	input timer (PC2/TIN) output timer (PC3/TOUT) interrupt recognition timer (PC7/$\overline{\text{TIACK}}$)
Ports	autovectored interrupt request (PC5/$\overline{\text{PIRQ}}$) port interrupt recognition (PC6/$\overline{\text{PIACK}}$) direct memory acces request (PC4/$\overline{\text{DMAREQ}}$)

The PI/T 68230 timer, depending on how its control register is programmed, allows a) interrupts to be generated periodically, b) an interrupt to be generated after a programmable delay, c) time elapsed to be measured, and so on. To achieve this it has three 8-bit counter registers, each also assigned to three 8-bit preloading registers.

The timer uses as its time reference the PI/T clock or an external clock fed to the timer input (TIN) with the possibility of dividing the latter by 32 (five prescale bits).

Table 4.3 68000/68008/68010/68020 Peripherals (Motorola, Mostek, Philips, Signetics, Hitachi, Rockwell)

IPC	68120/1	DPLL	68459
BIM	68153	MPCC-2	68561
VME	68172	DUSCC	68562
VMS	68173	SIO	68564
VME	68174	LANCE	68590
PI/T	68230	MPCC	68652
DDMA	68440	PGC1	68653
DMAC	68450	EPC1	68661
MMU	68451	DUART	68681
BAM	68452	FPCP	68881
IMDC	68454	MFP	68901

5 Addressing Modes of MC 68000, MC 68008 and MC 68010

TYPES OF ADDRESS

The 68000 has 14 addressing modes grouped into the following six categories.

- Address Register Direct
- Address Register Indirect
- Absolute Address
- Immediate Address
- Relative Program Counter Address
- Implicit Address

DEFINITION

Addressing mode is the name given to the different ways that an instruction can access an address in order to carry out an operation on it.

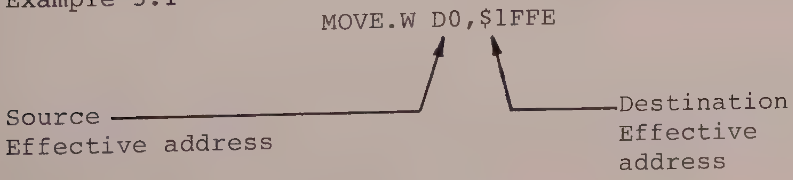
1 Address Register Direct

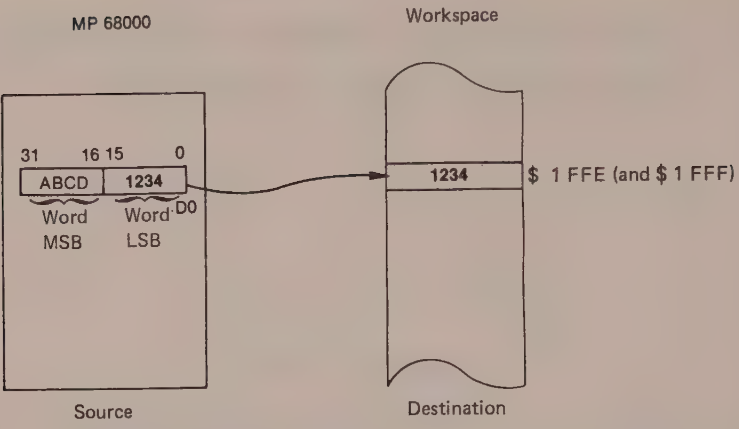
The direct addressing mode comprises two types

direct data register address (EA = Effective Address)	: EA = Dn
direct address register address	: EA = An

Data Register Direct Address : EA = Dn

Example 5.1





Role of the instruction
The instruction MOVE.W D0,$1FFE (or MOVE D0,$1FFE) gives the order to the processor to transfer the 16 low order bits of the data register D0 to the destination location $1FFE (and $1FFF).

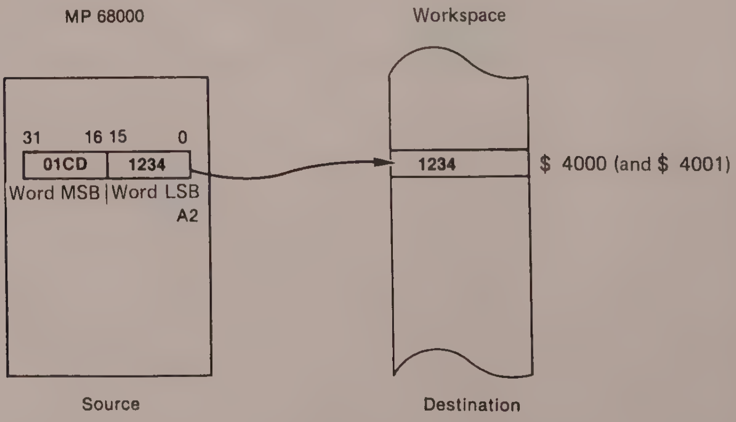
Details
The size specified by the instruction can be

8-bit byte	(MOVE.B D0,$1FFE)
16-bit word	(MOVE.W D0,$1FFE)
32-bit long word	(MOVE.L D0,$1FFE)

If however the size is a word or a long word, the destination address can only be even. Only a byte size allows the programmer to choose an even or an odd address.

Address Register Direct Address : EA = An
a) The source is specified by the address register
Example 5.2

```
MOVE.W A2,$4000
```



Role of the instruction
The instruction MOVE.W A2,$4000 directs the CPU to transfer the LSB word of the address register A2 (bits 0 to 15) to the destination location $4000.

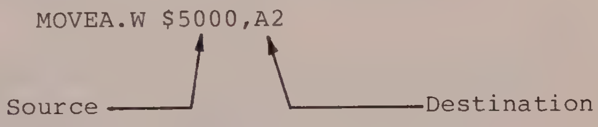
Details
The size specified by the instruction can be

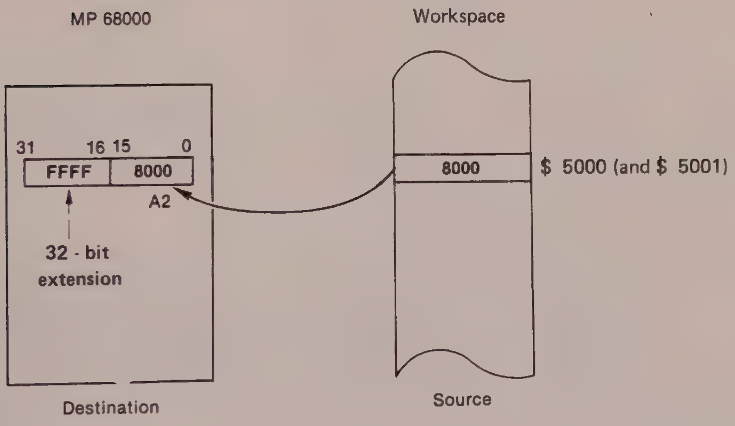
8-bit byte	(MOVE.B A2,$4000)
16-bit word	(MOVE.W A2,$4000)
32-bit long word	(MOVE.L A2,$4000)

The removal of the contents of an address register does not affect the condition code register (CCR).

b) The address register specifies the destination

Example 5.3





Role of the instruction
The instruction MOVEA.W$5000,A2 (or MOVEA $5000,A2) transfers the contents of the source address $5000 to the destination location, that is, address register A2.

Details
The size specified by the instruction can be word or long word.
When a register An is used as destination, the transfer of an operand to it leads systematically to a 32-bit sign extension.

Demonstration

```
   *
>  *                          MOVE.W $5000,A2
>  2000   0022-3478  0A08-5000  0120-4E71 }  loading of program
>  2000   3478-   5000-   4E71-          }
>  *
>  * BEFORE EXECUTION:
>  * ========================
>  :R   PC= 000400 # 0800    S=0 S 000    C=    .....    SP= 00000600
        D0= 00000000   D1= 00000000   D2= 00000000   D3= 00000000
        D4= 00000000   D5= 00000000   D6= 00000000   D7= 00000000
        A0= 00000000   A1= 00000000  [A2= 00000000]  A3= 00000000
        A4= 00000000   A5= 00000000   A6= 00000000   A7= 00000600

>  5000   0000-[4000]   storing data at address $5000
>  5000   4000-
>  .A2 = 00000000 --FFFFFFFF
>  :R   PC= 000400 # 0800    S=0 S 000    C=    .....    SP= 00000600
        D0= 00000000   D1= 00000000   D2= 00000000   D3= 00000000
        D4= 00000000   D5= 00000000   D6= 00000000   D7= 00000000
        A0= 00000000   A1= 00000000  [A2= FFFFFFFF]  A3= 00000000
        A4= 00000000   A5= 00000000   A6= 00000000   A7= 00000600

>  2000;T  >1
*  TRAC   PC= 002004 # 4E71    S=0 S 000    C=    .....    SP= 00000600
        D0= 00000000   D1= 00000000   D2= 00000000   D3= 00000000
        D4= 00000000   D5= 00000000   D6= 00000000   D7= 00000000
        A0= 00000000   A1= 00000000  [A2= 00004000]  A3= 00000000
        A4= 00000000   A5= 00000000   A6= 00000000   A7= 00000600

>  *
>  * AFTER EXECUTION:
>  * ======================
>  *
>  5000   4000-8000 }
>  5000   8000-     }         storing data at address $5000
>  .A2 = 00004000 --          contents of A2 before program execution
>  *
>  * BEFORE EXECUTION:
>  * ========================
>  *
>  2000;T  >1
*  TRAC   PC= 002004 # 4E71    S=0 S 000    C=    .....    SP= 00000600
        D0= 00000000   D1= 00000000   D2= 00000000   D3= 00000000
        D4= 00000000   D5= 00000000   D6= 00000000   D7= 00000000
        A0= 00000000   A1= 00000000  [A2= FFFF8000]  A3= 00000000
        A4= 00000000   A5= 00000000   A6= 00000000   A7= 00000600

>  *
>  * AFTER EXECUTION:
>  * ======================
>  *
>
```

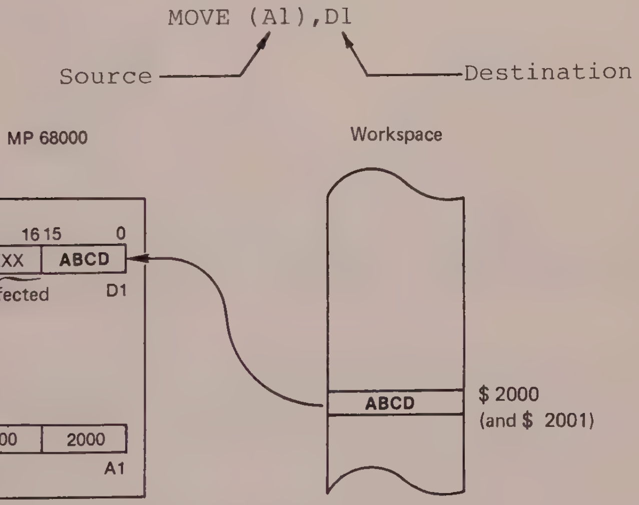
2 Address Register Indirect

There are six types of indirect address mode, as set out below.

Address Register Indirect
Address Register Indirect with Postincrement
Address Register Indirect with Predecrement
Address Register Indirect with Displacement
Address Register Indirect with Displacement and Short Index
Address Register Indirect with Displacement and Long Index

Address Register Indirect
EA = (An)

Example 5.4



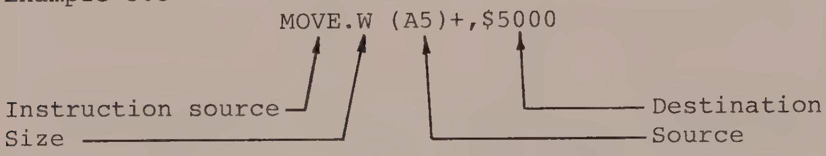
Role of the instruction
The instruction MOVE (A1),D1 or MOVE.W (A1),D1 informs the CPU that the operand to be transferred to the destination register D1 is located at the address pointed to by the source register A1.

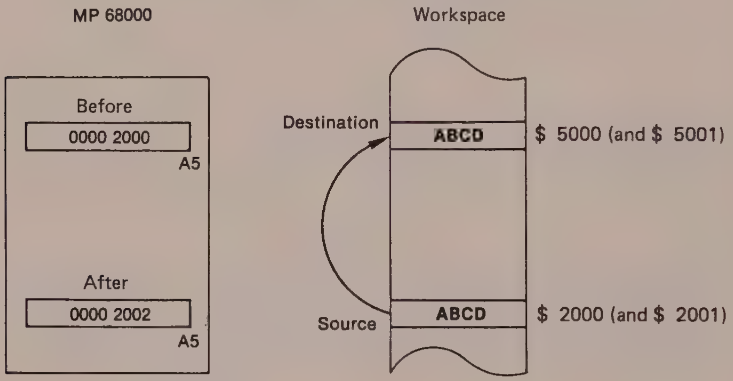
Details
The size specified by the instruction can be byte, word or long word. However, if the size is word or long word, the address register indirect must point to an exclusively even address. If this rule is not respected, an exception rerouteing called an illegal address will occur.

Generally speaking, these details will apply whatever the type of indirect address.

Address Register Indirect with Postincrement
EA = (An)+

Example 5.5





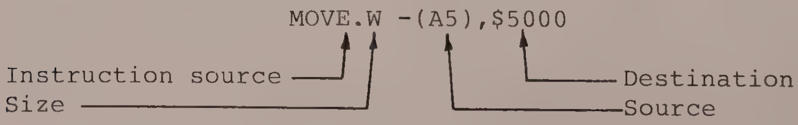
Role of the instruction
The instruction MOVE.W (A5)+,$5000 (or MOVE (A5)+, $5000) transfers the contents of the address specified by the address register A5 to the destination location $5000. Address register A5 is then incremented by two.

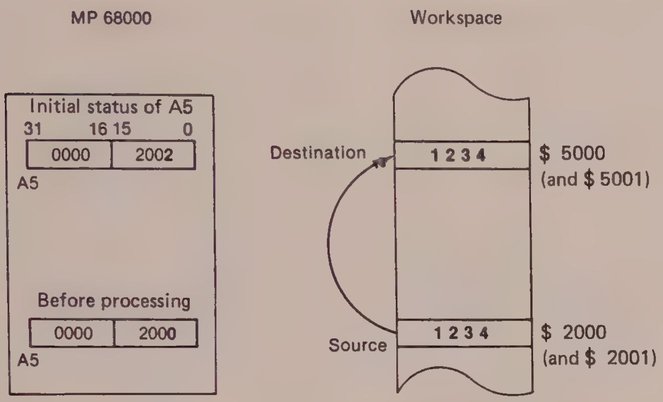
Details
The address register used as indirection must without fail point to an even address, when the size specified by the instruction is word or long word. On the other hand, if the size is byte, the address pointed to can be even or odd.

When requested with postincrement, the content of the address register is incremented by one, two or four, depending on whether the size specified is byte, word or long word. If the address register is the stack pointer and the size is byte, the increment of the stack pointer is two and not one.

Address Register Indirect with Predecrement
EA = -(An)

Example 5.6





Role of the instruction
Once the effective address is obtained, and after the address register A5 is decremented by two, the contents of the address pointed to by A5 is transferred to the destination location.

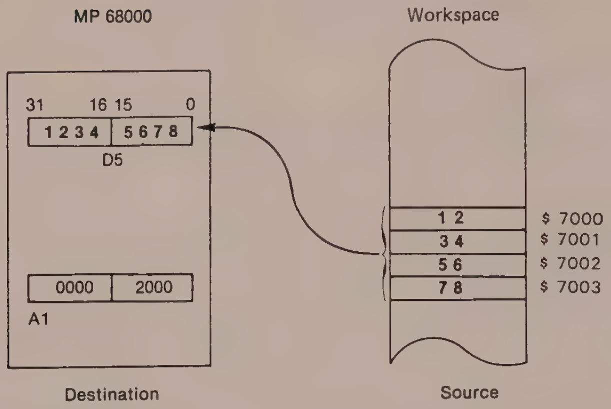
Details
The size specified by the instruction can be byte, word or long word. If the address register used as indirection is the stack pointer, and if the size is byte, the register An is always decremented by two in order to keep the size to word.

Address Register Indirect with Displacement
EA = (An) + d16

Example 5.7

```
                      MOVE.L $5000(A1),D5
                                ↑      ↑   ↑
         ⎧ 16-bit signed ───────┘      │   │
         ⎪ displacement                │   │
Source   ⎨                             │   │
         ⎪ Register (An) ──────────────┘   │
         ⎩                                 │
           Destination ────────────────────┘
```

Role of the instruction
The instruction MOVE.L $5000(A1),D5 loads the 32 bits of destination register D5 with the contents of the effective address, whose value is equal to the sum of the contents of address register A1 and the signed 16-bit displacement $5000.



Calculation of effective address (EA)

```
          A1                  ———→  0000 2000
                                    +

          16-bit signed       ———→  0000 5000
          displacement
          with 32-bit
          extension
                                    _________
                                    0000 7000
```

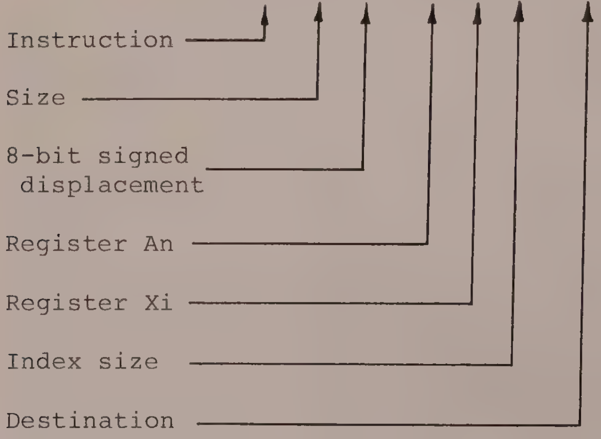
Details
For a word or long word operation, the effective address must be even. However, for a byte size the effective address can be even or odd.

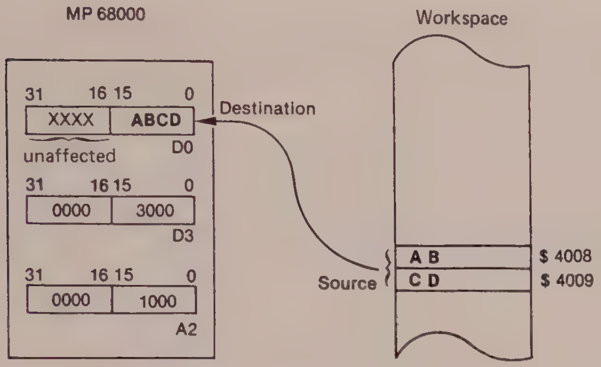
Address Register Indirect with 8-bit Signed Displacement and Short Index

EA = (An) + (Xi.W) + d8

Example 5.8

MOVE.W $08 (A2,D3.W), D0





```
  *
> * INSTRUCTION: MOVE.W $08(A2,D3.W),D0 :OP CODE :$3032 3008
> * ========================
> *
> 2000   3478--3032  5000--3008   4E71--  } loading of program
> 2000   3032--    3008--    4E71--
> *
> * BEFORE EXECUTION:
> * ============================
> :R    PC= 002004 # 4E71     S=0 S 000     C=     .....    SP= 00000600
        D0= 00000000  D1= 00000000  D2= 00000000  D3= 00000000
        D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
        A0= 00000000  A1= 00000000  A2= FFFF8000  A3= 00000000
        A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600

> *
> .D3 = 00000000 --3000<----contents of D3
> .A2 = FFFF8000 --1000<----contents of A2
> *
> :R    PC= 002004 # 4E71     S=0 S 000     C=     .....    SP= 00000600
        D0= 00000000  D1= 00000000  D2= 00000000  D3= 00003000
        D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
        A0= 00000000  A1= 00000000  A2= 00001000  A3= 00000000
        A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600

> *
> 4008  002A--ABCD<-----------contents of source address
> *
> 2000:T >1
* TRAC   PC= 002004 # 4E71     S=0 S 000     C=     .N...    SP= 00000600
      ->D0= 0000ABCD  D1= 00000000  D2= 00000000  D3= 00003000
     |  D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
     |  A0= 00000000  A1= 00000000  A2= 00001000  A3= 00000000
     |  A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600
     |
      -------------------Destination
> *
> *
> * AFTER EXECUTION:
> * ============================
  *
>
```

Role of the instruction
The instruction MOVE.W $08(A2,D3),D0 instructs the processor to transfer the contents of the effective address (source) to the destination register D0.

Calculation of effective address

```
          EA = (An) + (Xi.W) + 8-bit sign-extended
                                  displacement
```

where
An represents an address register
Xi determines the index register which can be a data register or an address register
Xi.W means that when the size if the index register is word, there is 32-bit signed extension when EA is calculated.

The source effective address is therefore

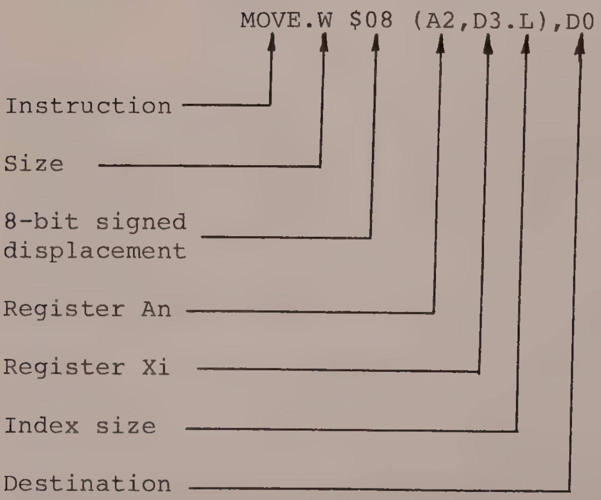
```
          Address register        (An)             1000
                                          +
          Index register          (Xi.W)      0000 3000
          with extension
                                          +
          8-bit sign-extended     (d8)        0000 0008
          displacement
                                              _________
                                   EA         0000 4008
```

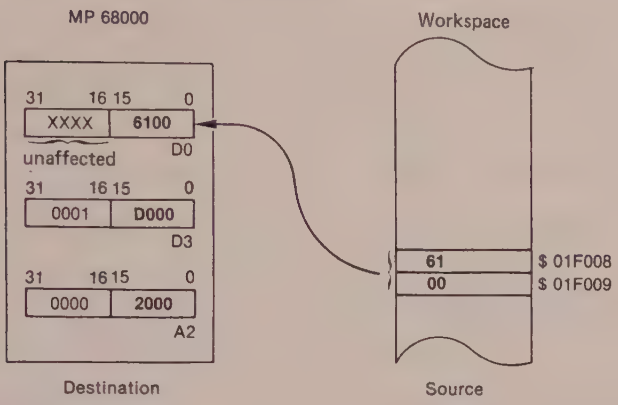
Details
See address register indirect with displacement.

Address Register Indirect with 8-bit Signed Displacement and Long Index

$$EA = (An) + (Xi.L) + d8$$

Example 5.9





```
> *
> * INSTRUCTION: MOVE.W #08(A2,D3.L),D0
> * ================
> *
> 2000   04A0-3032   0A08-3808   0120-4E71 }  loading of program
› 2000   3032-   3808-   4E71-
> *
> * BEFORE EXECUTION
> * =========================
> *
> ;R   PC= 000400 # 4800    S=0 S 000    C=    .....    SP= 00000600
      D0= 00000000   D1= 00000000   D2= 00000000   D3= 00000000
      D4= 00000000   D5= 00000000   D6= 00000000   D7= 00000000
      A0= 00000000   A1= 00000000   A2= 00000000   A3= 00000000
      A4= 00000000   A5= 00000000   A6= 00000000   A7= 00000600

> *
> .D3 = 00000000 --1D000 <----loading of D3
> .A2 = 00000000 --02000 <----loading of A2
> *
> 1F008   6100-                   contents of source address
> ;R   PC= 000400 # 4800    S=0 S 000    C=    .....    SP= 00000600
      D0= 00000000   D1= 00000000   D2= 00000000   D3= 0001D000
      D4= 00000000   D5= 00000000   D6= 00000000   D7= 00000000
      A0= 00000000   A1= 00000000   A2= 00002000   A3= 00000000
      A4= 00000000   A5= 00000000   A6= 00000000   A7= 00000600

> 2000;T >1
* TRAC   PC= 002004 # 4E71    S=0 S 000    C=    .....    SP= 00000600
      D0= 00006100   D1= 00000000   D2= 00000000   D3= 0001D000
      D4= 00000000   D5= 00000000   D6= 00000000   D7= 00000000
      A0= 00000000   A1= 00000000   A2= 00002000   A3= 00000000
      A4= 00000000   A5= 00000000   A6= 00000000   A7= 00000600

  the contents of D0.W have been loaded with the contents of addresses
> *                          $1F008 and $1F009
> * AFTER EXECUTION
> * ========================
> *
```

Role of the instruction
The instruction MOVE.W $08(A2,D3.L),D0 instructs the CPU to transfer the contents of the effective address (source) to the destination register D0.

Calculation of effective address

EA = (An) + (Xi.L) + 8-bit sign-extended displacement
which gives

```
        Address register    (An)     xxxx 2000
                                   +
        Index register      (Xi.L)   0001 D000
                                   +
        8-bit sign-extended          0000 0008
        displacement                 _________
                             EA =    0001 F008
```

Note that this addressing mode is equivalent to the record type in Pascal, since it allows addressing by base limit.

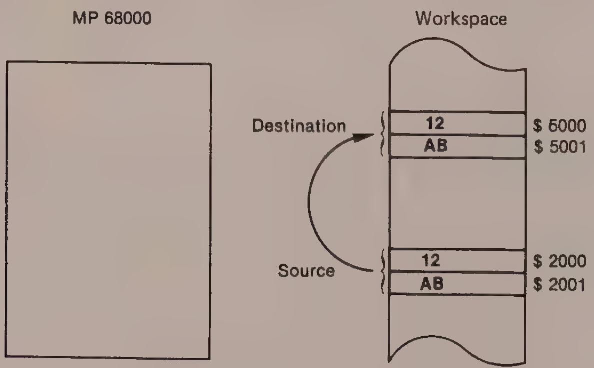
Absolute Address
There are two types of absolute address

```
     absolute short address : EA = 16-bit address
      absolute long address : EA = 24-bit address
```

Absolute Short Address: EA = 16-bit address

Example 5.10

```
                    MOVE.W $2000,$5000
or
                     MOVE   $2000,$5000
                              ↑     ↑
Source ───────────────────────┘     └──────────Destination
```



Role of the instruction
The instruction MOVE.W $2000,$5000 tells the CPU to transfer the contents of the source address ($2000 and $2001) to the destination address ($5000 and $5001).

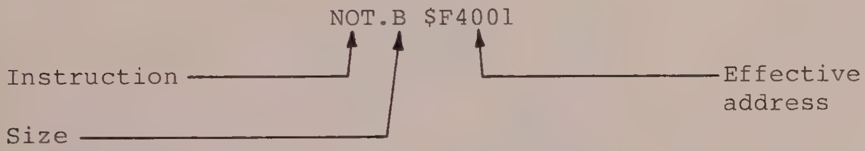
This instruction is especially interesting since it allows a movement from memory to memory without passing through a working register of the control unit.

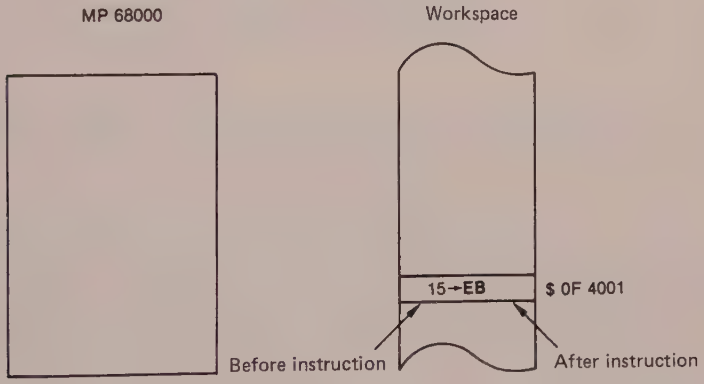
Details
The size specified by the instruction can be

8-bit byte with EA = even or odd
16-bit word with EA = exclusively even
32-bit long word with EA = exclusively even

Absolute Long Address : EA = 24-bit address

Example 5.11





Role of the instruction

The instruction NOT.B $0F4001 carries out a one´s complement on the contents of the effective address $0F4001.
This type of address allows access to a memory position beyond 64K.

Details
The size specified by the instruction must be

byte for an even or odd effective address
word for an even effective address
long word for an even effective address.

4 Immediate Address

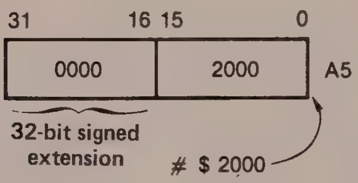
The immediate address mode allows an operand (data item), whether 8, 16 or 32 bits, to be sent to one of the following

data register
address register
memory location

Immediate Address with Address Register Destination

Example 5.12

```
                    MOVEA.W #$2000,A5
or
                     MOVEA #2000,A5
```



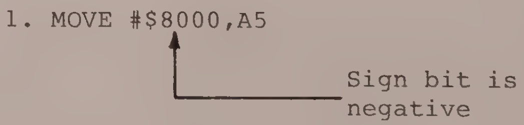
The above instruction loads address register A5 with the immediate value #$2000.

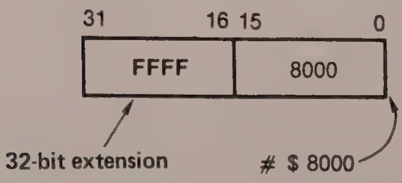
Details

1. There is always a 32-bit extension when the destination is an address register (for a 16-bit source or data item).

2. The size specified by the instruction can only be word or long word. Byte is forbidden.

Example 5.13

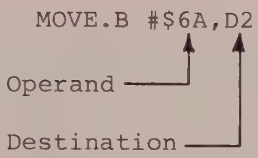


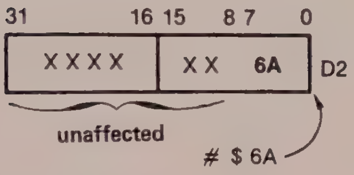


2.MOVE.B #$80,A5   This is forbidden.

Immediate Address with Data Register Destination

Example 5.14





The instruction MOVE #$6A,D2 loads data register D2 with operand #$6A. (Note that the # indicates the mode.)

Details
1. The size specified by the instruction can be

Byte	MOVE.B #$80,D2
Word	MOVE #$8000,D2
Long word	MOVE.L #$FFFFFFFF,D2

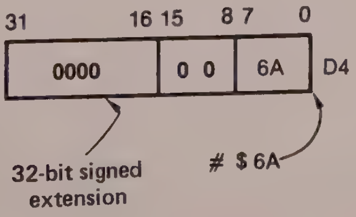
2. There is no bit extension when the size stipulates a byte or a word.

Immediate Rapid Address with Data Register Destination (MOVEQ instruction only)

Example 5.15

```
MOVEQ #$6A,D4
```

This 16-bit instruction loads destination register D4 with operand #$6A.



Details of MOVEQ
1. The destination is always a data register.
2. The size can only be byte.
3. There is 32-bit sign extension (the only occurrence of sign extension on a data register).

Special note
The instructions ADDQ and SUBQ specify a 3-bit operand whose different combinations code values lying between 1 and 8 inclusive.
  000 represents value 8; 001 value 1; 010 value 2, and so on, with 111 representing value 7.

Simulation

Case 1 : Positive operand

```
786A MOVEQ #$6A,D4
```

```
> *
> * INSTRUCTION: MOVEQ #$6A,D4
> * ====================
> *
> 2000  3032-786A  }        loading of program
> 2000  786A-      }
> *
> * BEFORE EXECUTION
> * ==============================
> *
> ;R  PC= 002004 # 4E71   S=0 S 000   C=   .....   SP= 00000600
      D0= 00006100  D1= 00000000  D2= 00000000  D3= 0001D000
      D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
      A0= 00000000  A1= 00000000  A2= 00002000  A3= 00000000
      A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600

>
> 2000;T >1
* TRAC  PC= 002002 # 4E71   S=0 S 000   C=   .....   SP= 00000600
      D0= 00006100  D1= 00000000  D2= 00000000  D3= 0001D000
      D4= 0000006A  D5= 00000000  D6= 00000000  D7= 00000000
      A0= 00000000  A1= 00000000  A2= 00002000  A3= 00000000
      A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600

> *
> * AFTER EXECUTION
> * ==============================
> *
>
```

Case 2 : Negative operand

```
7880 MOVEQ #$80,D4
```

```
  *
> * INSTRUCTION: MOVEQ #$80,D4
> * =================
> *
> 2000  786A-7880  4E71- }  loading of program
> *
> * BEFORE EXECUTION
> * ========================
> *
> ;R   PC= 002002 # 4E71    S=0 S 000    C=    .....   SP= 00000600
       D0= 00006100  D1= 00000000  D2= 00000000  D3= 0001D000
       D4= 0000006A  D5= 00000000  D6= 00000000  D7= 00000000
       A0= 00000000  A1= 00000000  A2= 00002000  A3= 00000000
       A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600

> 2000;T >1
* TRAC  PC= 002002 # 4E71    S=0 S 000    C=    .N...   SP= 00000600
       D0= 00006100  D1= 00000000  D2= 00000000  D3= 0001D000
       D4= FFFFFF80  D5= 00000000  D6= 00000000  D7= 00000000
       A0= 00000000  A1= 00000000  A2= 00002000  A3= 00000000
       A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600

> *
> * AFTER EXECUTION
> * =======================
> *
>
```

Immediate Address with Memory Location Destination

Example

ADDI.B #$80,$4000

The above instruction tells the processor to add operand #$80 to the contents of destination address $4000, then store the result in destination address $4000.

Example 5.16

$\#\$80 + \boxed{2\ 0}\ \$4000 \rightarrow \boxed{A\ 0}\ \$4000$

5 Address Relative to Program Counter
This address mode consists of two types

program counter with displacement
program counter with index

Basis
The value contained in the program counter (PC) is used to calculate the effective address for instruction handling, with the knowledge that the PC value is equal to the current instruction address +2.

All instructions using relative program counter addressing must be written in a section of program called RORG. This special feature allows position independent programs to be written.

Program Counter Address with Displacement

Example
In the following we shall examine some program instructions defined in a RORG section.

```
           00002000                     RORG     $2000

           0001E178        OUTMES       EQU      $1E178
           00000018        VECTCHK      EQU      $18
           0001E2FE        MONIT        EQU      $1E2FE
           00000005        NUMBER       EQU      5

    002000 4FFA00DA                     LEA      STACK,SP
    002004 41FA001E                     LEA      TRAPCHK,A0
    002008 21C80018                     MOVEA.L  A0,VECTCHK
    00200C 43FA00CE                     LEA      TABLE,A1
    002010 7205                         MOVEQ    #NUMBER,D1
    002012 0C59000F        LOOP         CMPI.W   #$000F,(A1)+
    002016 57C9FFFA                     DBEQ     D1,LOOP
    00201A 43BC0005                     CHK      #NUMBER,D1
    00201E 4EF90001E2FE                 JMP      MONIT
                           *
           00002024        TRAPCHK      EQU      *
    002024 48E74040                     MOVEM.L  D1/A1,-(SP)
    002028 41FA000E                     LEA      MESSAGE,A0
    00202C 4EB90001E178                 JSR      OUTMES
    002032 4CDF0202                     MOVEM.L  (SP)+,D1/A1
    002036 4E73                         RTE      Return
                           *
    002038 20              MESSAGE      DC.B     '  VALUE     NO
    002062 04                           DC.B     04
    002064 00000078                     DS.L     30
           000020DC        STACK        EQU      *
    0020DC 0000000A        TABLE        DS.W     5
                                        END
```

1. The instruction LEA TRAPCHK,A0 located at address $2004, gives the order to the processor to load register A0 with the effective address specified by the label TRAPCHK which is positioned at address $2024 (LEA = Load Effective Address).

Calculation of effective address

```
Formula    EA =      (PC)          +        d 16
            |          |                     |
giving      v          v                     v
           2024 =    (2004 + 2) +          00 1 E
           /             |                     \
          v              v                      v
TRAPCHK              Program                  Relative
Label address      counter +2              displacement
                                         shown by op-code
                                  (2024 - 20006 = 00 1 E)
```

Table 5.1 Summary of Addressing Modes

Mode	Notation	Operation
Data register direct	Dn	EA = Dn
Address register direct	An	EA = An
Address register indirect	(An)	EA = (An)
Address register indirect with postincrement of 1,2 or 4	(An)+	EA = (An) then An := An + 1,2 or 4 depending on size
Address register indirect with predecrement of 1,2 or 4	-(An)	An := An - 1,2 or 4 depending on size, then EA = (An)
Address register indirect with displacement	(An) + d16	EA = (An) + 16-bit sign extended displacement
Address register indirect with displacement/index	(An,Xi) + d8	EA =(An) + (Xi.w) with 8-bit sign extended displacement
Address register indirect with displacement and long index	(An,Xi.L) d8	EA = (An) + (Xi.L) with 8-bit sign extended displacement
Absolute short address	Addr 16	EA = (addr 16)
Absolute long address	Addr 24	EA = (addr 24)
Immediate address	# data	data item
Relative program counter address with displacement	raddr 16	EA = (PC) + d 16
Relative program counter address with displacement and index	raddr 8 (Xi)	EA = (PC) + (Xi) + d 8
Implicit address		EA = SR,USP,SP,PC

2. The instruction LEA TABLE, A1 located at address $200C suggests the same handling as before, that is, loading address register A1 with the effective address defined by the label TABLE.

Calculation of the effective address

$$\begin{tikzcd}
\text{Formula} & \text{EA} = \arrow[d] & (\text{PC}) \arrow[d] & + & \text{d 16} \arrow[d] \\
& 20\text{DC} = \arrow[dl] & (200\text{C} + 2) + \arrow[d] & & 00\ \text{CE} \arrow[d] \\
\text{TABLE Label address} & & \text{Program counter} +2 & & \text{Relative displacement shown by op-code } (20\ \text{DC} - 200\ \text{E})
\end{tikzcd}$$

Program Counter Address with Index
The basis of operation is the same as for program counter with displacement except that for calculation of the effective address an index register has to be taken into account.

Formula

$$\text{EA} = (\text{PC}) + (\text{Xi.W}) + \text{d8}$$

or

$$\begin{tikzcd}
& \text{EA} = (\text{PC}) & + (\text{Xi.L}) & + \text{d8} & \\
\text{Current value of PC + 2} \arrow[ur] & & \text{Index register, a 16 or 32-bit data or address register} \arrow[u] & & \text{8-bit signed displacement} \arrow[ul]
\end{tikzcd}$$

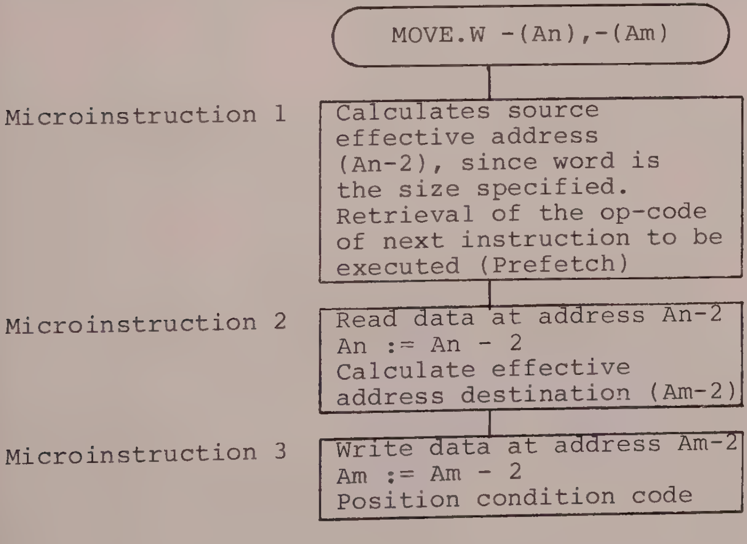
Implicit Address
Table 5.2 lists the instructions that make implicit reference to the following

program counter (JMP, BRA)
user stack pointer (MOVE USP)
supervisor stack pointer (TRAP, DIV)
status register (RTE, RTR).

Table 5.2

Instruction	Implied registers
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 division (DIVS)	SSP, SR
Unsigned division (DIVU)	SSP, SR
Jump (JMP)	PC
Jump to subroutine (JSR)	PC, SP
Link and allocate (LINK)	SP
Move condition codes (MOVE CCR)	SR
Move status register (MOVE SR)	SR
Move user stack pointer (MOVE USP)	USP
Push effective address (PEA)	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

Each 68000 instruction can be broken down into several microinstructions. An example is given below.



MOVE.W -(An),-(Am)

Microinstruction 1: Calculates source effective address (An-2), since word is the size specified. Retrieval of the op-code of next instruction to be executed (Prefetch)

Microinstruction 2: Read data at address An-2
An := An - 2
Calculate effective address destination (Am-2)

Microinstruction 3: Write data at address Am-2
Am := Am - 2
Position condition code

6 68000 Instruction Set

DETAILS AND PROGRAMMING

Motorola deliberately restricted the instruction set of the 16-bit MC 68000 (and 68008) microprocessors to 56 instructions that offer extreme flexibility. (Note that the newer 16/32-bit MC 68010 has 57 types of instruction.)

Even if this number appears small (although programmers will probably not complain), some instructions offer several thousand combinations because of the different address modes available and the type of data that they can manipulate.

The five basic data types are

- bit
- BCD digit (4 bits)
- byte (8 bits)
- word (16 bits)
- long word (32 bits)

The 68000 instructions operate on byte, word and/or long word, which in assembly language need to be specified by .B, .W and .L. If the size is word, the suffix .W is assumed by default.

Example

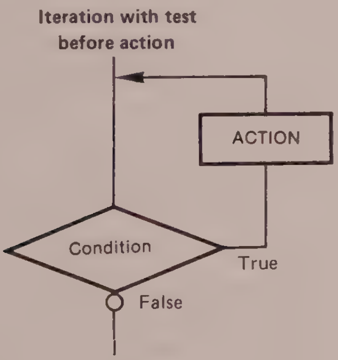
```
         MOVE.B  Source, destination
         MOVE.W  Source, destination
      or MOVE    Source, destination
         MOVE.L  Source, destination
```

The operation code (op-code) of all the 68000 instructions is fixed in 16 bits (word), but an extension is required when the specified address mode uses constants, absolute addresses or displacements.

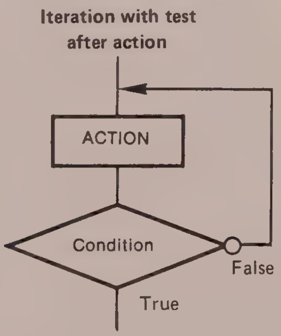
As a consequence, a 68000 instruction can be coded in from one to five words (2 to 10 bytes).

We have arbitrarily classified the 56 instructions into three categories: memory reference and special, arithmetical and logical, and program control instructions (see also appendixes 1 to 8).

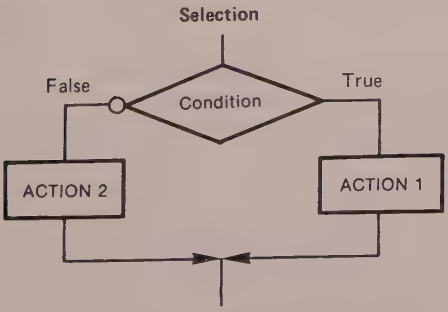
In what follows we shall not be studying in detail all 56 of the 68000 instructions; we shall confine our examination to those that do not have an approximate 8-bit equivalent. The reader is advised to study the three types of flowchart shown in figure 6.1, as well as the 68000 assembler directives listed in tables 6.1 and 6.2, in order to be able to follow the discussion of this chapter without difficulty.



```
In pseudo-code
WHILE CONDITION TRUE
  | ACTION
  |
END
```



```
In pseudo-code
REPEAT
  | ACTION
UNTIL condition TRUE
```



```
In pseudo-code
IF condition TRUE THEN
  | ACTION1
ELSE
  | ACTION2
END
```

Figure 6.1

Table 6.1 68000 Assembler Directives

Directives	Examples	Role
ORG	ORG $4000	Origin in absolute short of program
ORG.L	ORG.L $40000	Origin in absolute long of program
RORG	RORG 0 RORG $1000	Relative origin of program
EQU	BASE EQU $1000 PIACA EQU BASE + 1 LF EQU $0A	Symbol equivalence up to 32 bits maximum
SET	BASE SET BASE-1	Temporarily fixes the value of a symbol
DC.B	TEXT DC.B "HELLO" LFCR DC.B $0A,$0D	Byte constants stored at successive addresses
DC.W	WRITE DC.W $F000,TAB DISPLAY DC.W-3000,SET	Word constants stored at successive addresses
DC.L	VAUE DC.L $ABCDFFFF LINE DC.L VALUE +3	Long word constants stored at successive addresses
DS.B	STACK DS.B 20	Reserve memory (bytes)
DS.W	BUFFER DS.W 2	Reserve memory (words)
DS.L	TEXT DS.L 3	Reserve memory (long words)
END	END	End of assembly directive
MACRO	CALCULATE MACRO CAL1,CAL2	Definition of Macro instruction
ENDM	ENDM	End of Macro

Table 6.2 68000 Assembler Directives

Directives	Examples	Role
LLEN	LLEN 120	Fixes number of characters per line (here 120)
PLEN	PLEN 40	Fixes number of lines per page (here 40)
NOOBJ	NOOBJ	No object output
IFEQ	MULT1 EQU0 IFEQ MULT1	Conditional assembly Assembles if equal to 0

IFNE	MULT2 EQU1 IFNE MULT2	Conditional assembly Assembles if different
ENDC	ENDC	End of conditional assembly
SPC	SPC 6	Line space
TTL	TTL MP 68000	Program title. Source with 60 characters max.

Memory Reference and Special Instructions
Table 6.3 lists these instructions.

Loading Instructions
LEA and PEA cause a pointer to be initialised (LEA Src,An means An := effective address) and saved to the stack (PEA Src means Src = effective address -> -(SP)).
These two instructions, which share the same addressing modes, are to some extent complementary.

Table 6.3 Memory Reference and Special Instructions

Mnemonic	Operand	Size	Notes	Description
LEA	Src,An	L	1	Load An with Src
PEA	Src	L	-	Save Src -> -(SP)
MOVE	Src,Dst	L,W,B	-	Copy Src -> Dst
MOVEP	Dn,Src	L,W	2	Store Sn -> Dst
MOVEP	Dst,Dn	L,W	2	Load Dn with Dst
MOVEM	Regs,Dst	L,W	3,4	Store Regs -> Dst
MOVEM	Src,Regs	L,W	1,3,5	Load Src -> Regs
BSET	Numb,Dst	B,L	7,8	Tests Numb and sets
BCLR	Numb,Dst	B,L	7,8	Tests Numb and clears
BCHG	Numb,Dst	B,L	7,8	Tests Numb and changes
BTST	Numb,Dst	B,L	7,8	Tests Numb in Dst
CMPM	Src,Dst	L,W,B	9	Dst-Src; CCR set
CMP	Src,Dst	L,W,B	10	Dst-Src; CCRset
CMPI	Src,Dst	L,W,B	10	Dst-Src; CCR set
CHK	Src,Dn	W	7	If Dn < 0 or Dn > Src -> TRAP
TAS	Dst	B	7	If Dst MSB = 0 then MSB = 1
CLR	Dst	L,W,B	11	Clears Dst
TST	Dst	L,W,B	11	Z and N set according to Dst

Special instructions				
SWAP	Dn	L	-	Exchanges word MSB with word LSB
EXT	Dn	L,W	-	Extends sign
EXG	Xn,Xm	L	-	Exchanges Xn with Xm

Src	= Source	W	= Word
Dst	= Destination	B	= Byte
Numb	= Bit number	Dn	= Data registers
Regs	= Registers	Xn	= Data or address registers
L	= Long word	Xm	= Data or address registers

(1) Src can only use addressing modes
(An),d(An),d(An,Xi),ABS.W,ABS.L,d(PC),d(PC,Xi)
(2) Src and Dst can only use addressing mode d(An)
(3) A0-A5/D1-D5 is equivalent to
A0/A1/A2/A3/A4/A5/D1/D2/D3/D4/D5
A0/A3/A6/D1-D3 is equivalent to A0/A3/A6/D1/D2/D3
(4) Dst can use the following addressing modes
(An),-(An),d(An),d(An,Xi),ABS.W,ABS.L
(5) Src can use the following addressing modes
(An),(An)+,d(An),d(An,Xi),ABS.W,ABS.L,d(PC),d(PC,Xi)
(6) Src and Dst use addressing modes Dn and -(An)
(7) Src and Dst cannot be An registers
(8) Numb can be an operand between 1 and 8 or the contents of Dn register in 32 bits
(9) Src and Dst can only be (An)+
(10) For CMP, Dst can only be a Dn register. Memory/memory comparison only possible with (9). For CMPI, Src can only be an operand
(11) Src and Dst cannot be An registers.

The MOVE instruction, which is the most flexible of the 56, can be selected in 12 288 different ways. It would be wrong to conclude from this that the structure of the 68000 is general purpose, but it should be noted that the following movements are possible with this instruction: memory to memory, memory to register, register to memory, operand to memory.

The MOVEP instruction is particularly suitable for programming peripheral circuits that occupy an alternate memory field, or are in other words addressable either via the lower line (odd address) or via the upper line (even address).

The two versions of the MOVEM instruction (load and store) are designed to move, in a predetermined order, a list of address or data registers from or to a block of memory.

Bit Manipulation Instructions

The second group of table 6.3 comprises four bit manipulation instructions. For example, a bit can be tested, then set to 1 (BSET) or to 0 (BCLR); it can also be changed (BCHG), or simply tested to establish its state (BTST). The bit number can be specified as static and immediate (modulo 8), or dynamically by the contents of a Dn register (modulo 32).

Comparison Instructions

The first two instructions of the third group are concerned with comparisons. This is one area where the instructions set could be criticised. In fact, the memory to memory comparison (CMPM) is only possible with a source and destination having the addressing mode (An)+. As for the CMP instruction, the source can only be a Dn register. Finally, CMPI compares the destination with the source specified as immediate. (CMPI # Immediate, Dst.)

The CHK instruction compares the word LSB of a register Dn with a bounded value, where the lower bound is 0 and the upper bound is a 16-bit signed operand. If the word LSB does not belong to the interval, the processor is rerouted to the exception procedure TRAPCHK whose vector number is $6.

The TAS (test and set) instruction allows management of a resource that can be shared by several processors, since during a single bus cycle it executes the reading, testing and finally modifying of a destination byte (memory or register) called a semaphore.

The next two instructions, CLR and TST, present little difficulty, save for the fact that the destination cannot be an An register.

Special Instructions

The fourth and last group in this category only affect An and Dn registers. The instruction SWAP exchanges bits 0-15 of a Dn register with bits 16-31. EXT carries out a signed 16-bit or 32-bit extension in a Dn register.

The last of the special instructions, EXG, instructs the 68000 to exchange the 32 bits of a source register with the 32 bits of a destination register (register = An and/or Dn).

Arithmetic and Logical Instructions

There are 34 arithmetic and logical instructions. They can be divided into four types: addition, subtraction and complementation; multiplication and division; logical instructions; shifts and rotations. Table 6.4 lists the various instructions in this category.

Table 6.4 Arithmetic and Logical Instructions, Shifts and Rotations

Mnemon	Operand	Size	Notes	Description
ADDI	Imm,Dst	B,W,L	1	Dest + operand -> Dst
SUBI	Imm,Dst	B,W,L	1	Dest - operand -> Dst
ADDQ	Imm,Dst	B,W,L	1	Dest + operand -> Dst
SUBQ	Imm,Dst	B,W,L	1	Dest - operand -> Dst
ADD	Src,Dst	B,W,L	1	Dest + source -> Dst
SUB	Src,Dst	B,W,L	1	Dest - source -> Dst
ADDA	An,Src	W,L	1	An + source -> An
SUBA	An,Src	W,L	1	An - source -> An
ADDX	Src,Dst	B,W,L	2	Dest + srce + X -> Dst
SUBX	Src,Dst	B,W,L	2	Dest - srce - X -> Dst
ABCD	Src,Dst	B	2	Dst(10) + srce(10) + X -> Dst
SBCD	Src,Dst	B	2	Dst(10) - srce(10) - X -> Dst
NBCD	Dst	B	3	0-Dst(10) - X -> Dst
NEG	Dst	B,W,L	3	Two´s complement of Dst
NEGX	Dst	B,W,L	3	Two´s complement with X of Dst
NOT	Dst	B,W,L	3	One´s complement of Dst
MULU	Src,Dn	W	3	Dn * source -> Dn
MULS	Src,Dn	W	3	Dn * source -> Dn
DIVU	Src,Dn	W	3	32 bits Dn/16 Src -> DnCRST:QUT]
DIVS	Src,Dn	W	3	32 bits DN/16 Src -> DnCRST:QUT]
		Logical Instructions		
AND	Src,Dst	B,W,L	1	Dst . Src -> Dst
ANDI	Imm,Dst	B,W,L	1	Dst . Imm -> Dst
OR	Src,Dst	B,W,L	1	Dst + Src -> Dst
ORI	Imm,Dst	B,W,L	1	Dst + Imm -> Dst
EOR	Src,Dst	B,W,L	3	Dst + Src -> Dst
EORI	Imm,Dst	B,W,L	3	Dst + Imm -> Dst
		Shifts and Rotations		
ASL	CntDst	B,W,L	-	Arithmetic shift left
ASR	CntDst	B,W,L	-	Arithmetic shift right
LSL	CntDst	B,W,L	-	Logical shift left
LSR	CntDst	B,W,L	-	Logical shift right
ROL	CntDst	B,W,L	-	Rotate left
ROR	CntDst	B,W,L	-	Rotate right
ROXL	CntDst	B,W,L	-	Rotate left with extend
ROXR	CntDst	B,W,L	-	Rotate right with extend

Src = source	An = address register
Dst = destination	Dn = data register
Imm = immediate	

CntDst = counter (including destination when specified)
(1) Memory to memory operations are not possible
(2) Src can only use addressing modes Dn,-(An)
(3) Src and Dst cannot be an An register.

Addition, Subtraction and Complement
The first four instructions, ADDI, SUBI, ADDQ, SUBQ, carry out the addition (or more correctly, incrementation, as such instructions do not exist), and the subtraction (decrementation) between the destination and the source, coded as immediate. If the immediate operand lies between 1 and 8 inclusive, it will be preferable to choose the instructions ADDQ and SUBQ which are more efficient in code and execution time, as can be seen from the following

ADDI.B #3,$4000 is coded in 6 bytes and requires 21 clock cycles;

ADDQ.B #3,$4000 is coded in 4 bytes, with an execution time of 17 cycles.

Note that a good assembler will, if you are not too rigorous in your requirements, look after coding the instruction as it should. The ADD and SUB instructions add and subtract in binary the source and destination before storing the result in the destination. When the destination is a Dn register, we have available in the source all the addressing modes. On the other hand, when the source is a Dn register, the addressing modes making reference to the program counter are not allowed. The next two instructions, ADDA and SUBA, only concern the An registers as destination.

The ADDX and SUBX instructions allow multiple-precision calculations to be carried out, where source and destination can only use the addressing modes Dn and -(An); this appears quite logical, in view of the method of calculation (low bit to high bit, or predecrement mode).

It is also possible to write directly in BCD (which eliminates the well-known DAA of the 6800 and 6809). Here the 68000 has three instructions available: ABCD (addition in BCD); SBCD (subtraction in BCD); and NBCD (complement in BCD).

The remaining instructions in group (1) instruct the 68000 to two´s complement (NEG), and with bit extension (NEGX), or to one´s complement (NOT).

Multiplication and Division

The four instructions of group (2) multiply (MULU and MULS) and divide (DIVU and DIVS) unsigned and signed binary numbers.

MULU and MULS multiply the 16 bits of a Dn register (bits 0-15) by the 16 bits of the source that may use all addressing modes except An. The 32 bits of the result are saved in Dn.

DIVU and DIVS divide the 32 bits of a Dn register by the 16 bits of the source, with again all addressing modes except An available, before saving the 16 bits of the remainder in the word MSB of Dn (bits 31-16) and the 16 bits of the quotient in the word LSB of Dn (bits 15-0).

Two conditions may prevent the 68000 from carrying out division.

a) if the divisor is zero, the processor is rerouted to the exception procedure "zero division", the vector number of which is $5.

b) if the result exceeds the 32 bits of Dn (remainder and quotient in 16 bits), causing overflow (V := 1). In such a case no internal provision has been made to cope with the situation (as has been done for division by zero). It is therefore up to the programmer to verify if overflow has occurred, by testing flag V, and to take any necessary action.

Note that neither source nor destination operands are altered if such anomalies occur.

Logical Instructions

The next group (3), the logical instructions, will be well known to programmers. It includes AND and ANDI which carry out a logical AND of the source and the destination, with the result being stored in the destination.

We have however noted that memory to memory operations are not possible and that, for EOR and EORI (exclusive OR), the source can only be a Dn register or an immediate operand.

Shift and Rotate Instructions

These form the last group (4) of the arithmetic and logical instructions. Here the operand CntDst can be written #Cnt,Dst, where #Cnt codes as immediate an operand between 1 and 8. This operand specifies the number of shifts or rotations to be made in the destination (which can only be a Dn register).

CntDst can also be written Dm,Dst, where Dm is a Dn register that codes the number of shifts or rotations to be made in Dst. Up to 63 operations (modulo 64) are possible.

CntDst can also be written as Dst, where Dst represents a memory address. The number of shifts and rotations is always 1, so it is superfluous to specify the source. Note that the memory size can only be word.

Program Control

The most significant advances have been made within the category of the program control instructions. Table 6.5 lists the various instructions involved.

Table 6.5 Program Control Instructions

Mnemon	Operand	Size	Notes	Description
				Unconditional sequence break and no operation
NOP	/	/	1	No operation
JMP	Address	/	1	Unconditional jump to address
BRA	Displ.	/	3	Branch always
				Call and return subroutines and block memory allocation
JSR	Address	/	1	Jump to a subroutine
BSR	Displ.	/	3	Branch to a subroutine
RTS	/	/	/	Return from subroutine
RTR	/		/	Return with CCR restored
LINK	An,imm,dpl	/	2	Link with the stack
UNLK	An	/	/	Unlink from stack
				Condition operation
Bcc	Displ.	/	3,4	Branch conditionally
DBcc	Dn,dpl	/	3,4	Test, decrement, branch
Scc	Dst	B	4,5	If cc is true then $FF -> else 0 -> Dst
				Handling on CCR and SR registers
MOVE	Src,CCR	W	6,7	Copy Src in CCR
OR	Src,CCR	B	8	Inclusive OR between CCR and Src
ORI	Imm,CCR	B	8	Inclusive OR between CCR and operand
AND	Src,CCR	B	8	Logical AND between CCR and Src
ANDI	Imm,CCR	B	8	Logical AND between CCR and operand
EORI	Imm,CCR	B	8	Exclusive OR between CCR and operand
MOVE	SR,Dst	W	5	Copy SR in Dst

		Privileged instructions		
OR	Src,SR	W	8,9	Inclusive OR between SR and Src
ORI	Imm,SR	W	8,9	Inclusive OR between SR and operand
AND	Src,SR	W	8,9	Logical AND between SR and Src
ANDI	Imm,SR	W	8,9	Logical AND between SR and operand
EORI	Imm,SR	W	8,9	Exclusive OR between SR and operand
MOVE	An,USP	L	9	Copy An in USP
MOVE	USP,An	L	9	Copy USP in An
RTE	/	/	9	Return from exception
RESET	/	/	9	Set Reset line low
STOP	Imm	W	9	Load SR with Imm, then stop
		Logical traps		
TRAP	Vectnum	/	19	Logical exception
TRAPV	/		/	Exception if V = 1

Notes
(1) The address is specified in absolute: all addressing modes allowed except Immediate,Dn,An,(An)+,-(An).
(2) Displacement lies between \$8000 (-32 768) and \$7FFF (+32 767).
(3) Displacement is 16 bits signed (32K).
(4) See table 6.5 for condition codes.
(5) Destination can use all addressing modes except Immediate,An,d(PC),d(PC,XI).
(6) Source may use all addressing modes except An.
(7) Only word size is allowed even if destination is byte (in which case CCR is loaded with the source LSB).
(8) The data on immediate is a function of the instruction specified size.
(9) Privileged instructions can only be handled in supervisor mode.
(10) The vector lies between 0 and 15.

The first group (1) is typical of the instructions sets of the majority of microprocessors (6800/6809, 8080/8085, 6502).

The first two instructions of the next group (2) allow calls to be made to a subroutine - absolute with JSR and relative with BSR.

The instructions RTS and RTR instruct the 68000 to return to the calling program. The difference between these two instructions lies in the execution. In the case of RTS, restoration of the return address, previously stored in the stack, is made when the subroutine is called. With RTR, the CCR is restored first, then the return address to the calling program. Note that in the case of RTR only the return address is saved on the stack when the subroutine is called.

The LINK instruction automatically allocates a working area (block of RAM) to the calling program, to be used for example for passing parameters or storing local data. As for UNLK, its role is to free this working area.

These two high-level instructions are completely suited to the writing of reentrant programs, that is, those programs or subroutines that work on a block of memory belonging to the calling program.

The instructions in the third group (3) concern conditional operations, where the logical condition is specified by the instruction mnemonic in the cc (condition code) form.

The Bcc type of instruction will already be familiar to assembly language programmers of Motorola microprocessors. Note that with Bcc displacement the 68000 executes a conditional branch if the cc condition is true. The relative displacement is coded in one or two bytes.

The DBcc type of instruction has no equivalent in 8-bit microprocessors, at least in the Motorola range. This conditional branch instruction, also called a looping primitive, operates in relation to the following three parameters

the conditional branch condition specified by cc (for example, DBEQ, DBNE, DBMI)

the loop counter Dn

the relative signed 16-bit displacement.

The Scc type instructs the CPU to set the destination byte (Dn or memory registers) at $FF if the cc condition is true and at $00 if it is false.

The fourth group of instructions concerns those that have the condition code register CCR as destination, that is, the user byte of the status register SR.

The fifth group is very important, as the following discussion will reveal. In fact, all the instructions in this group are called privileged instructions; that is, they can only be executed in supervisor mode (S := 1). If one of these instructions is executed is user mode (S := 0), a privilege violation occurs: the 68000 is rerouted in the exception table to address $20 (vector number 8*4 = 32 or $20) in order to recover the start address of the exception program.

Note that all the instructions capable of modifying the status of the processor (loading SR) are privileged.

The RTE instruction allows return from an exception procedure to a normal procedure.

The RESET instruction, which is also privileged, instructs the 68000 to hold the RESET line (here an output) at low for 124 clock cycles, this being for example the time required to initialise a peripheral circuit (PIA 6821, PTM 6840, PI/T 68230).

The 68000 instructions set has 16 logical traps (the 6809 has 3: SWI, SWI2 and SWI3) which, when one of them is executed, cause the 68000 processor to branch to an exception procedure. Each trap has its own vector number and therefore its own exception vector.

Examples

TRAP #0 has the number 32_{10} and the address $32_{10}*4 = 128_{10} = \$80$.

TRAP #15 has the number 47_{10} and the address $47_{10}*4 = 191_{10} = \$BF$.

On the other hand, the TRAPV instruction is the logical trap that is available to the programmer if the status flag V is at 1 when this instruction is executed. This instruction is especially useful, for example, in the case of DIVS and DIVU. In fact, if the 68000 finds that it cannot carry out such an operation because of lack of space, it sets the status flag to 1 to inform the programmer that overflow has occurred.

Example

```
DIVU D1,D0 := $FFFFFFFF
TRAPV D1 = $XXXX0002
BRA*
```

The 68000 carries out the signed or unsigned division of the 32 bits of the destination (D0 in the above example) by the 16 low order bits of the source (D1).

The 32 bits of the result available in the destination are distributed as follows: the remainder in 16 bits (bits 16-31 of the destination); the quotient in 16 bits (bits 0-15 of the destination).

If the result exceeds this format, the 68000 does not carry out the operation, but sets V to 1, without altering the destination and source registers.

The same principle applies with the TRAPV instruction, but in this case the program is rerouted to the exception table at address $1C which corresponds to vector number 7.

DBcc INSTRUCTIONS

Role of DBcc

Instruction DBcc Dn,d16 is a looping primitive with three parameters: the condition specified by cc, the loop counter represented by a data register Dn, and the relative 16-bit displacement.

Execution of this instruction by the CPU causes the following sequence of events

1. Condition cc is tested (cc can be one of the 16 conditions listed in table 6.6). If the condition is true, instruction DBcc is terminated and the processor executes the rest of the program.

Table 6.6 Conditions (cc) Used with Instructions DBcc, Scc, Bcc

Mnemonic	cc conditions	Logical equation
T (not Bcc)	True	1
F (not Bcc)	False	0
HI	High	$C + Z = 0$
LS	Low or same	$C + Z = 1$
CC	Carry clear	$C = 0$
CS	Carry set	$C = 1$
NE	Not equal	$Z = 0$
EQ	Equal	$Z = 1$
VC*	Overflow clear	$V = 0$
VS*	Overflow set	$V = 1$
PL	Plus	$N = 0$
MI	Minus	$N = 1$
GE*	Greater or equal	$N \oplus V = 0$
LT*	Less than	$N \oplus V = 1$
GT*	Greater than	$Z \oplus (N \oplus V) = 0$
LE*	Less or equal	$Z \oplus (N \oplus V) = 1$

*Used in two´s complement mode

2. If condition cc is false, the LSB word of register Dn is decremented by one.

3. If the decrementation of register Dn has brought about the result -1, instruction DBcc is terminated and the processor executes the remainder of the program.

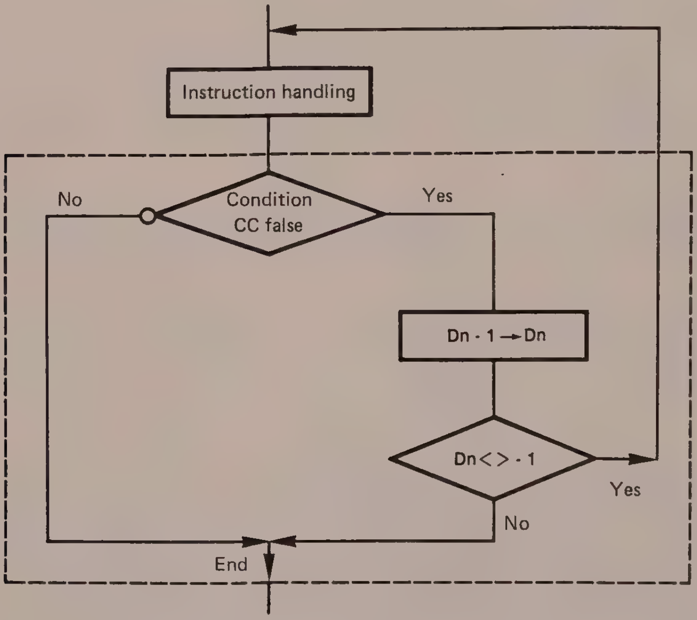
If the result is the opposite, the contents of the PC are added to the relative 16-bit displacement so that the branch address can be determined.

Expression of Instruction DBcc
In assembler

```
                    DBcc Dn,d16

DBcc  : instruction mnemonic (conditional branch)
Dn    : register Dn, loop counter
d 16  : 16-bit signed displacement
```

Flowchart and Pseudo-code



In pseudo-code

```
IF cc FALSE THEN
    Dn := Dn -1
    IF Dn<> - 1 THEN
      PC := PC + 16-bit displacement
    END
ELSE
  NOP

END
```

Figure 6.2

Examples

1. The assembler programs in listings 6.1a and 6.1b instruct the MC 68000 to print out n times the message "The 68000 microprocessor is wonderful".

   The value n stored in byte $2054 before execution of the program represents the contents of the loop counter register of the primitive DBLT D2,LOOP.

   The different simulations prove that the test on the loop counter is carried out on value -1.

Listing 6.1a

```
                          ***************************************************************
                          *                                                             *
                          *      THREE  PARAMETER  LOOPING  PRIMITIVE                   *
                          *       DBCC  INSTRUCTION                                     *
                          *                                                             *
                          ***************************************************************

       0001F9E9           ACIA     EQU      $1F9E9             ; ADDRESS ACIA 6850 EUROMAK 68000

       00002000                    RORG     $2000              ;

002000 4FFA00AE                    LEA.L    STACK,SP           ; INITIALISE SSP
002004 41F90001F9E9                LEA.L    ACIA,A0            ; INITIALISE A0 ADDRESS ACIA
                          *
00200A 143A0048                    MOVE.B   COUNTER,D2         ; LOAD COUNTER LOOPING
                          *
00200E 610A               LOOP     BSR.S    PCRLF              ;
002010 6128                        BSR.S    TEXT               ;
002012 5DCAFFFA                    DBLT     D2,LOOP            ;
002016 4E41                        TRAP     #1                 ; RETURN TO EUROMAK 68000 MONITOR
002018 0000                        DC.W     0                  ;
                          *
00201A 2F00               PCRLF    MOVE.L   D0,-(SP)           ; SAVE ALTERED REGISTER
00201C 700A                        MOVEQ    #$0A,D0            ; D0.B := ASCII LINE FEED
00201E 6108                        BSR.S    OUT                ; OUTPUT LF
002020 700D                        MOVEQ    #$0D,D0            ; D0.B := ASCII CARRIAGE RETURN
002022 6104                        BSR.S    OUT                ; OUTPUT CR
002024 201F                        MOVE.L   (SP)+,D0           ; RESTORE REGISTER
002026 4E75                        RTS
                          *
002028 2F01               OUT      MOVE.L   D1,-(SP)           ; SAVE ALTERED REGISTER
00202A 1210               OUT1     MOVE.B   (A0),D1            ; ACIA READY ?
00202C 02010002                    ANDI.B   #2,D1              ;
002030 67F8                        BEQ.S    OUT1               ; NO WAIT
002032 11400002                    MOVE.B   D0,2(A0)           ; OK TRANSMIT
002036 221F                        MOVE.L   (SP)+,D1           ; RESTORE REGISTER
002038 4E75                        RTS
                          *
00203A 48E78040           TEXT     MOVEM.L  D0/A1,-(SP)        ; SAVE ALTERED REGISTERS
00203E 43FA0015                    LEA.L    TEXT1,A1           ; A1:= @ START ADDRESS TEXT
002042 1019               LOOP2    MOVE.B   (A1)+,D0           ; DO  WHILE D0 <> EOT
002044 0C000004                    CMPI.B   #EOT,D0            ; !
002048 6704                        BEQ.S    ENDTRANS           ; ! /* TRANSMIT CHAR */
00204A 61DC                        BSR.S    OUT                ; !
```

```
00204C 60F4                BRA.S   LOOP2                ; ENDDO
00204E 4CDF0201  ENDTRANS  MOVEM.L (SP)+,D0/A1          ; RESTORE REGISTERS
002052 4E75                RTS
                 *
002054 00000001  COUNTER   DS.B    1
002055 20        TEXT1     DC.B    '         THE  68000  MICROPROCESSOR IS  WONDERFUL  '
002087 04                  DC.B    EOT
       00000004  EOT       EQU     4
002088 00000028            DS.L    10
       000020B0  STACK     EQU     *
                           END
```

```
> *
> * SIMULATION PROGRAM
> * ==============================
> *
> 2054.B
* 002054 > 02
* 002055 >
> 2000;G
           THE   68000  MICROPROCESSOR IS   WONDERFUL
           THE   68000  MICROPROCESSOR IS   WONDERFUL
           THE   68000  MICROPROCESSOR IS   WONDERFUL
> *
> * WHY ??
> * ------
> 2054.B
* 002054 > 03
* 002055 >
> 2000;G
           THE   68000  MICROPROCESSOR IS   WONDERFUL
           THE   68000  MICROPROCESSOR IS   WONDERFUL
           THE   68000  MICROPROCESSOR IS   WONDERFUL
           THE   68000  MICROPROCESSOR IS   WONDERFUL
>
```

```
                 *************************************************************************
                 *                                                                       *
                 *    THREE  PARAMETER  LOOPING  PRIMITIVE   (EXERCISE 2 )               *
                 *                                                                       *
                 *    TWO STRINGS ARE COMPARED TO SEE IF THEY ARE THE SAME               *
                 *                                                                       *
                 *************************************************************************

                 *
                 * A1 = POINTS TO THE BEGINNING OF THE FIRST STRING
                 * A2 = POINTS TO THE BEGINNING OF THE SECOND STRING
                 * D2 = NUMBER OF CHARCTERS
       0001E178  OUTMES    EQU     $1E178               ; PRINT TEXT STRING OF CHARS

       00002000            ORG     $2000

002000 4FF820DC            LEA.L   STACK,SP             ; INITIALISE STACK POINTER SSP
002004 43F8202A            LEA.L   TEXT1,A1             ; INITIALISE POINTER STRING1
002008 45F8205E            LEA.L   TEXT2,A2             ; INITIALISE POINTER STRING2
                 *
00200C 2209                MOVE.L  A1,D1                ; D1:=A1
00200E 240A                MOVE.L  A2,D2                ; D2:=A2
002010 9481                SUB.L   D1,D2                ; D2:=D2-D1 /* COUNTER CHARS */
                 *
```

```
002012 B509          LOOP     CMPM.B  (A1)+,(A2)+           ; DO WHILE MEM[A2]=MEM[A1]
002014 56CAFFFC               DBNE    D2,LOOP               ; ! /* CORRECT */
                     *                                      ; ENDDO
002018 6604                   BNE.S   ERROR                 ; IF Z=0 THEN ERROR
00201A 4E41          RETURN   TRAP    #1                    ; ENDIF
00201C 0000                   DC.W    0

00201E 41F82092      ERROR    LEA.L   TEXTERROR,A0            ; A0:= @ TEXT STRING
002022 4EB90001E178           JSR     OUTMES                  ; PRINT TEXT STRING
002028 60F0                   BRA.S   RETURN                  ;


00202A 20            TEXT1    DC.B    '        THE  68000  MICROPROCESSEUR  IS  WONDERFUL '
00205D 04                     DC.B    4
00205E 20            TEXT2    DC.B    '        THE  68000  MICROPROCESSEUR  IS  WONDERFUL '
002091 04                     DC.B    4
002092 20            TEXTERROR DC.B   ' STRINGS  CHARS NO EQUAL , ERROR '
0020B3 04                     DC.B    4
0020B4 00000028               DS.L    10
       000020DC      STACK    EQU     *
                              END

****** TOTAL ERRORS   0--   0


SYMBOL TABLE - APPROXIMATELY  504 SYMBOL ENTRIES LEFT

ERROR      00201E LOOP      002012 OUTMES     01E178 RETURN     00201A
STACK      0020DC TEXT1     00202A TEXT2      00205E TEXTERRO   002092
```

2. The program example shown in listing 6.2 is not directed exclusively at the loop primitive, but also gives a brief reminder of address register indirect with index.

Listing 6.2

```
                     **********************************************************************
                     *                                                                    *
                     *  EXAMPLE OF USE OF "INDIRECT ADRESSING"  WITH  INDEX               *
                     *                  AE = d+An+Xn                                      *
                     *  FINDING  THE  LARGEST OPERAND IN THE TABLE (size Long Word)       *
                     *    Program Written in Position Independent Code                    *
                     *                                                                    *
                     **********************************************************************

       00002000               ORG     $2000

002000 43F83000               LEA.L   $3000,A1              ;INITIALISE POINTER I
002004 2049                   MOVE.L  A1,A0                 ;INITIALISE POINTER J
002006 2610                   MOVE.L  (A0),D3               ; D3:= MEM[J]
002008 7009                   MOVEQ   #10-1,D0              ; D0:=9 /* LOOP COUNTER */
00200A 4281                   CLR.L   D1                    ; D1.L:= 0
00200C 5881          LOOP     ADDQ.L  #4,D1                 ; REPEAT
00200E B6811800               CMP.L   0(A1,D1.L),D3         ; ! D1.L:=D1.L+4
002012 6C04                   BGE.S   GREAT                 ; ! IF D3 >= MEM[A1+D1.L] THEN
```

```
002014 26311800             MOVE.L  D(A1,D1.L),D3          ; ! !          //
002018 51C8FFF2     GREAT   DBRA    D0,LOOP                ; ! ELSE
00201C 4E41                 TRAP    #1                     ; ! !  D3:= MEM[A1,D1.L]
00201E 0000                 DC.W    0                      ; ! ENDIF
                            END     ;                        UNTIL D0=-1

****** TOTAL ERRORS   0--   0


SYMBOL TABLE - APPROXIMATELY  510 SYMBOL ENTRIES LEFT

GREAT      002018 LOOP       00200C
```

Simulation

```
   *
 > * SIMULATION PROGRAM:
 > * ==================================
 > *
 > 3000.L
 * 003000 > 00000000
 * 003004 > 88888888
 * 003008 > FFFFFFFF
 * 00300C > 55555555
 * 003010 > EEEEEEEE
 * 003014 > 77777777
 * 003018 > 66666666
 * 00301C > 12345678
 * 003020 > 99999999
 * 003024 > 74444444
 * 003028 >
 > *
 > 2010:V
 > 2000:G
 * VSTP   PC= 00201C # 4E41    S=0 S 000    C=    .....   SP= 00000600

 > ;R   PC= 00201C # 4E41    S=0 S 000    C=    .....   SP= 00000600
        D0= 0000FFFF   D1= 00000028   D2= 00000000   D3= 77777777
        D4= 00000000   D5= 00000000   D6= 00000000   D7= 00000000
        A0= 00003000   A1= 00003000   A2= 00000000   A3= 00000000
        A4= 00000000   A5= 00000000   A6= 00000000   A7= 00000600

 > *
 > *
 > 3000.L
 * 003000 > 44444444
 * 003004 > 66666666
 * 003008 > 77777777
 * 00300C > 00000000
 * 003010 > AAAAAAAA
 * 003014 > 66666666
 * 003018 > 77777700
 * 00301C > 55555555
 * 003020 > EEEEEEEE
 * 003024 > 7FFFFFFF
 * 003028 >
 > *
 > 2010:V
 > 2000:G
 * VSTP   PC= 00201C # 4E41    S=0 S 000    C=    .....   SP= 00000600
        D0= 0000FFFF   D1= 00000028   D2= 00000000   D3= 7FFFFFFF
        D4= 00000000   D5= 00000000   D6= 00000000   D7= 00000000
        A0= 00003000   A1= 00003000   A2= 00000000   A3= 00000000
        A4= 00000000   A5= 00000000   A6= 00000000   A7= 00000600
```

Scc INSTRUCTION

1 Role of Scc

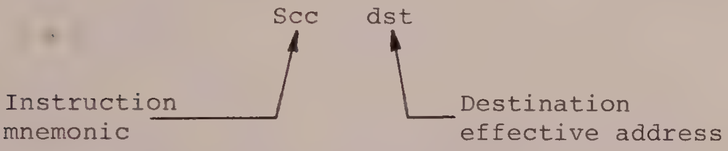
Instruction Scc dst tests one of the 16 cc conditions summarised in table 6.6.

The destination byte is positioned at $FF if the condition specified by cc is true, and at 00 if cc is false.

This instruction is generally used to position a boolean variable after evaluating an expression, where the true variable is coded $FF and the false variable is coded 00 (see the example in listing 6.3).

2 Syntax of Scc

In assembler



Flowchart and Pseudo-code

```
IF cc TRUE THEN
  | destination := $FF
ELSE
  | destination := 00
ENDIF
```

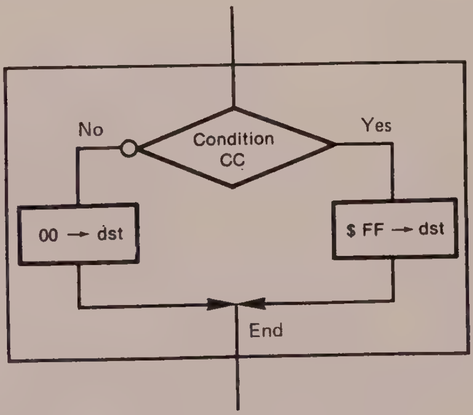


Figure 6.3

Listing 6.3

```
                     ***********************************************************************
                     *                                                                     *
                     *    U S E         Scc      I N S T R U C T I O N                     *
                     *                                                                     *
                     *    PROGRAM  TO  DETECT  16  BIT  PALINDROMES                        *
                     *                                                                     *
                     *              (USING MACROS)                                         *
                     *                                                                     *
                     ***********************************************************************

                     *
                     ** ALGORITHM :
                     *  ---------
                     *
                     *
                     *  BTST D1,(A0) !                              ! BTST D0,(A0)
                     *               !                              !
                     *               V                              V
                     *                + + + + + + + + + + + + + + + +
                     *                ^                             ^
                     *          15 -!                               !- 0
                     *

                     *
                     ** MACROS
                     *  ------

                     TEXT     MACRO
                     \@       MOVEM.L D0-D7/A0-A6,-(A7)
                              LEA     STRING\1,A0
                              JSR     OUTMES
                              MOVEM.L (A7)+,D0-D7/A0-A6
                              RTS
                              ENDM

       0001E178      OUTMES   EQU     $1E178       ; OUTPUT STRING OF CHARS (EUROMAK 68000)

       00002000               ORG     $2000

002000 4FF82194               LEA.L   STACK,A7     ; INITIALISE STACK POINTER SSP
002004 41F82074               LEA.L   TABLE,A0     ; INITIALISE ADDRESS PALINDROMES
002008 6110                   BSR.S   SEARCH       ;
                     *
00200A 4A382194               TST.B   FLAG
00200E 6706                   BEQ.S   CORRT        ; IF FLAG <>0 THEN ERROR
002010 613A                   BSR.S   @001         ;
002012 4E41          ENDPROG  TRAP    #1           ; RETURN EUROMAK 68000
002014 0000                   DC.W    0
                     *
002016 6148          CORRT    BSR.S   @002
002018 60F8                   BRA.S   ENDPROG      ;
```

```
550 00201A 48E7F000      SEARCH   MOVEM.L D0-D3,-(A7)
560 00201E 51F82194               SF      FLAG                   ; FLAG := FALSE
570 002022 4202                   CLR.B   D2                     ;
580 002024 4203                   CLR.B . D3                     ;
590 002026 4200                   CLR.B   D0                     ; INITIAL RIGHT HAND BIT NUMBER
600 002028 720F                   MOVEQ   #15,D1                 ; INITIAL LEFT  HAND BIT NUMBER
610 00202A 0110          LOOP     BTST    D0,(A0)                ; TEST RIGHT HAND BIT
620 00202C 57C2                   SEQ     D2                     ;
630 00202E 0310                   BTST    D1,(A0)                ; TEST LEFT  HAND BIT
640 002030 57C3                   SEQ     D3                     ;
650 002032 B602                   CMP.B   D2,D3                  ; IF BITS ARE EQUAL THEN
660 002034 660C                   BNE.S   EXIT                   ; ! /* MOVE TO NEXT BIT */
670 002036 5240                   ADDQ    #1,D0                  ; ELSE
680 002038 5341                   SUBQ    #1,D1                  ; ! /* EXIT */
690 00203A 0C000008               CMPI.B  #8,D0                  ; ENDIF
700 00203E 66EA                   BNE.S   LOOP                   ;
710 002040 6004                   BRA.S   EXIT1                  ; IF END TEST THEN EXIT1
720 002042 50F82194      EXIT     ST      FLAG                   ; FLAG := TRUE
730 002046 4CDF000F      EXIT1    MOVEM.L (A7)+,D0-D3
740 00204A 4E75                   RTS
750                      *
760                      *
770                               TEXT    1
770 00204C 48E7FFFE      @001     MOVEM.L D0-D7/A0-A6,-(A7)
770 002050 41F82076               LEA     STRING1,A0
770 002054 4EB90001E178           JSR     OUTMES
770 00205A 4CDF7FFF               MOVEM.L (A7)+,D0-D7/A0-A6
770 00205E 4E75                   RTS
780                      *
790                      *
800                               TEXT    2
800 002060 48E7FFFE      @002     MOVEM.L D0-D7/A0-A6,-(A7)
800 002064 41F820A3               LEA     STRING2,A0
800 002068 4EB90001E178           JSR     OUTMES
800 00206E 4CDF7FFF               MOVEM.L (A7)+,D0-D7/A0-A6
800 002072 4E75                   RTS

820 002074 00000002      TABLE    DS.W    1


840 002076 0A            STRING1  DC.B    $A,$D
850 002078 20                     DC.B    '          THE  WORD  IS NOT A PALINDROME  '
860 0020A0 0A                     DC.B    $A,$D,4
870 0020A3 0A            STRING2  DC.B    $A,$D
880 0020A5 20                     DC.B    '          THE  WORD  IS A PALINDROME '
890 0020C8 0A                     DC.B    $A,$D,4

910 0020CC 000000C8               DS.L    50
920        00002194      STACK    EQU     *
930 002194 00000001      FLAG     DS.B    1
940                               END

****** TOTAL ERRORS   0--   0


SYMBOL TABLE - APPROXIMATELY  493 SYMBOL ENTRIES LEFT
```

Simulation

```
   *
 > *
 > **   S I M U L A T I O N     P R O G R A M
 > *    =====================================================
 > *
 > *
 > 2074.W
 * 002074 > AFF5
 * 002076 >
 > 2000;G
          THE  WORD  IS NOT A PALINDROME

 > 2074.W
 * 002074 > FFFF
 * 002076 >
 > 2000;G
          THE  WORD  IS A PALINDROME

 > 2074.W
 * 002074 > A55A
 * 002076 > 2
 > 2000;G
          THE  WORD  IS A PALINDROME

 > 2074.W
 * 002074 > A5A5
 * 002076 >
 > 2000;G
          THE  WORD  IS A PALINDROME

 >
```

BSET, BCLR, BCHG, BTST INSTRUCTIONS

The MC 68000 has four test instructions that allow it to work at the bit level.

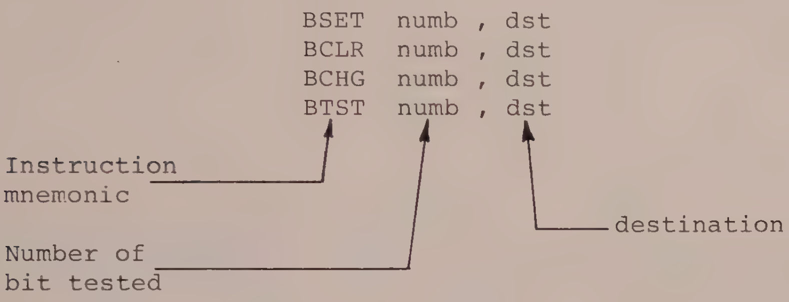
These instructions operate on long words in the data registers, but on bytes in the addresses. Unfortunately, there were not sufficient op-code combinations available to provide other options.

The bit concerned in the test, which we shall call numb, can be specified statically as immediate, or dynamically by data registers whose contents are the bit to be tested.

If the source or the destination is a data register, the size is a long word and the bit tested lies between 0 and 31 (modulo 32).

If the size is byte, the destination can only be a memory location and the bit tested lies between 0 and 7 (modulo 8).

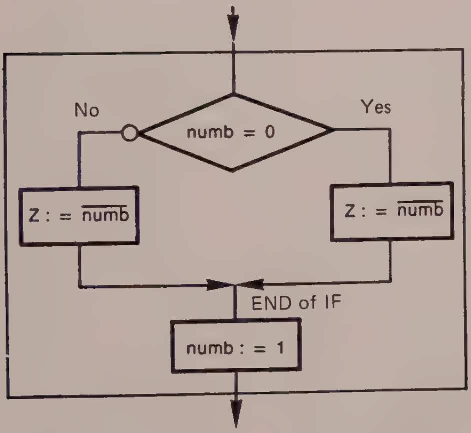
1 Syntax of the Instructions
In assembler



Flowchart and Pseudo-code

BSET numb, dst
Test a bit and set

Example : Test a bit at 0

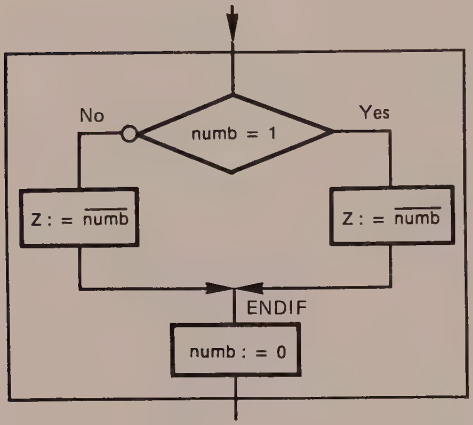


```
IF numb = 0 THEN
|     Z := 1
ELSE
|     Z := 0
ENDIF
   numb := 1
```

Figure 6.4

BCLR numb, dst
Test a bit and clear

Example : Test a bit at 1

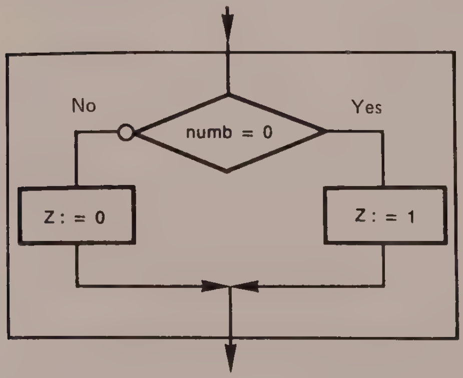

No
numb = 1
Yes
Z : = $\overline{\text{numb}}$
Z : = $\overline{\text{numb}}$
ENDIF
numb : = 0

```
IF numb  = 1 THEN
|      Z := 0
ELSE
|      Z := 1
ENDIF
   numb := 0
```

Figure 6.5

BTST numb, dst
Test a bit

Example : Test a bit at 0

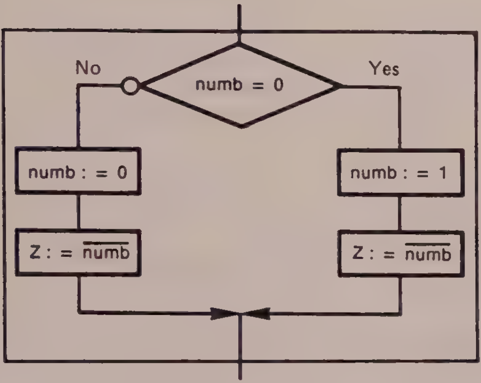


```
IF numb = 0 THEN
|      Z := 1
ELSE
|      Z := 0
ENDIF
```

Figure 6.6

BCHG numb, dst
Test a bit and change

Example : Test a bit at 0


No
numb = 0
Yes
numb : = 0
Z : = $\overline{\text{numb}}$
numb : = 1
Z : = $\overline{\text{numb}}$

```
IF numb = 0 THEN
|  numb := 1
|     Z := 1
ELSE
|  numb := 0
|     Z := 0
ENDIF
```

Figure 6.7

2 Program Examples
The program shown in listing 6.4 causes the number of 1s and 0s contained in a long word to be displayed.

Listing 6.4

```
                        ***********************************************************************
                        *                                                                     *
                        *     PROGRAM  TO  DISPLAY  A  LONG  WORD  IN  BINARY                 *
                        *                                                                     *
                        ***********************************************************************

         0001F9E9       ACIA     EQU      $1F9E9              ; ADDRESS ACIA EUROMAK 68000 SYSTEM

         00002000                ORG      $2000
  002000 4FF82068                LEA.L    STACK,SP            ; INITIALISE STACK POINTER SYST.
  002004 4DF90001F9E9            LEA.L    ACIA,A6             ; INITIALISE A6 ADDRESS ACIA 6850
                        *
  00200A 7C0A                    MOVEQ    #$0A,D6             ; LINE FEED
  00200C 6136                    BSR.S    TSTACIA             ; OUTPUT LINE FEED
  00200E 7C0D                    MOVEQ    #$0D,D6             ; CARRIAGE RETURN
  002010 6132                    BSR.S    TSTACIA             ; OUTPUT CARRIAGE RETURN
                        *
  002012 20382050                MOVE.L   (LWORD),D0          ; LOAD LONG WORD
  002016 6102                    BSR.S    COUNTBIT            ;
  002018 4E41                    TRAP     #1                  ; RETURN TO EUROMAK 68000 SYSTEM
                        *
```

```
00201A 2F02      COUNTBIT MOVE.L  D2,-(SP)        ; SAVE ALTERED REGISTER
00201C 741F               MOVEQ   #31,D2          ; D2:=31 /* COUNTER BIT */
00201E 0500      TEST     BTST.L  D2,D0           ; REPEAT
002020 6704               BEQ.S   BIT0            ; ! IF BIT D2 <> 0 THEN
002022 610C               BSR.S   DISPLAY1        ; ! ! /* DISPLAYED 1 */
002024 6002               BRA.S   SUIT            ; ! ELSE
002026 6112      BIT0     BSR.S   DISPLAY0        ; ! ! /* DISPLAYED 0 */
002028 51CAFFF4  SUIT     DBRA    D2,TEST         ; ! ENDIF
00202C 241F               MOVE.L  (SP)+,D2        ; UNTIL D2=-1
00202E 4E75               RTS     ;               RESTORE REGISTER
                 *
002030 2F06      DISPLAY1 MOVE.L  D6,-(SP)        ; SAVE ALTERED REGISTER
002032 7C31               MOVEQ   #$31,D6         ; D6.B:= '1'
002034 610E               BSR.S   TSTACIA         ;
002036 2C1F               MOVE.L  (SP)+,D6        ; RESTORE REGISTER
002038 4E75               RTS
                 *
00203A 2F06      DISPLAY0 MOVE.L  D6,-(SP)        ; SAVE ALTERED REGISTER
00203C 7C30               MOVEQ   #$30,D6         ; D6.B:= '0'
00203E 6104               BSR.S   TSTACIA         ;
002040 2C1F               MOVE.L  (SP)+,D6        ; RESTORE REGISTER
002042 4E75               RTS     ;               RETURN
                 *
002044 08160001  TSTACIA  BTST.B  #1,(A6)         ; ACIA READY ?
002048 67FA               BEQ.S   TSTACIA         ; NO BRANCH TSTACIA
00204A 1D460002           MOVE.B  D6,2(A6)        ; YES TRANSMIT CHAR
00204E 4E75               RTS     ;

002050 00000004  LWORD    DS.L    1               ; RESERVE 1 LONG WORD
002054 00000014           DS.L    5               ; RESERVE 5 LONG WORD FOR THE STACK

       00002068  STACK    EQU     *
                          END

****** TOTAL ERRORS  0--   0
```

Simulation

```
       *
     > * SIMULATION PROGRAM
     > * ==================================
     > *
     > *
     > * 2050.L= LONG WORD
     > * ---------------------------------
     > 2050.L
     * 002050 >  A0A0A0A0
     * 002054 >
     > 2000;G
    10100000101000001010000010100000
     > *
     > *
     > 2050.L
     * 002050 >  FFFFFFFF
     * 002054 >
     > *
     > 2000;G
```

```
11111111111111111111111111111111
 > *
 > *
 > 2050.L
 * 002050 > 00000000
 * 002054 >
 > *
 > 2000;G
00000000000000000000000000000000
 >
```

The program shown in listing 6.5 counts the number of 1s in a long word.

Listing 6.5

```
                     ***********************************************************************
                     *                                                                     *
                     *   PROGRAM  TO  COUNT  THE  NUMBER OF " 1 " IN A LONG  WORD          *
                     *                                                                     *
                     ***********************************************************************

                     *
                     * FUNCTION REGISTERS
                     * -----------------
                     * D0.B = COUNT BIT
                     * D1.B = COUNT LOOP
                     * D2.L = LONG WORD
                     *
        00002000              ORG     $2000

 002000 4FF82048              LEA.L   STACK,SP          ; INITIALISE STACK POINTER SSP
 002004 2438202E              MOVE.L  (LWORD),D2        ; D2:= (LWORD)
 002008 6104                  BSR.S   TSTO              ;
 00200A 4E41                  TRAP    #1                ; RETURN TO EUROMAK 68000 SYSTEM
 00200C 0000                  DC.W    0

 00200E 48E7C000     TSTO     MOVEM.L D0-D1,-(SP)       ; SAVE ALTERED REGISTERS
 002012 4200                  CLR.B   D0                ; D0.B := 0  /* COUNT BIT "1" */
 002014 721F                  MOVEQ   #31,D1            ; D1.B :=31
 002016 4A82         LOOP1    TST.L   D2                ; DO  WHILE  D2 <> 0
 002018 670A                  BEQ.S   ENDPROGRAM        ; !
 00201A 0382                  BCLR.L  D1,D2             ; !  IF NUMB BIT <> 0 THEN
 00201C 6702                  BEQ.S   LOOP              ; !  ! Z:=0 ; NUMB BIT:=0; D0:=D0+1
 00201E 5200                  ADDQ.B  #1,D0             ; !  ELSE
 002020 51C9FFF4     LOOP     DBRA    D1,LOOP1          ; !  ! Z:=1 ; NUMB BIT:=0
 002024 11C02032     ENDPROGRAM MOVE.B D0,NUMB          ; !  ENDIF
 002028 4CDF0003              MOVEM.L (SP)+,D0-D1       ; ENDDO
 00202C 4E75                  RTS

 00202E 00000004     LWORD    DS.L    1
 002032 00000002     NUMB     DS.W    1
 002034 00000014              DS.L    5
        00002048     STACK    EQU     *
                              END
```

```
****** TOTAL ERRORS   0--   0


SYMBOL TABLE - APPROXIMATELY  505 SYMBOL ENTRIES LEFT

ENDPROGR   002024 LOOP     002020 LOOP1    002016 LWORD    00202E
NUMB       002032 STACK    002048 TST0     00200E
```

```
  *
> * SIMULATION PROGRAM:
> * ================================*
> *
> 202E.L
* 00202E > 00000007
* 002032 >
> *
> * JUST BEFORE PROGRAM EXECUTION
> * -----------------------------------------
> *
> 2032  0300-
> 2032  0300-0000
> 2000;G
> *
> * AFTER EXECUTION
> * -----------------------
> *
> 2032  0300-
> *
> * JUST BEFORE PROGRAM EXECUTION
> * -----------------------------------------
> *
> 202E.L
* 00202E > FFFFFFFF
* 002032 >
> *
> 2032  0300-
> 2000;G
> *
> * AFTER EXECUTION
> * -----------------------
> 2032  2000-
> *
> 202E.L
* 00202E > FFFFFFFE
* 002032 >
> 2000;G
> 2032  1F00-
> *
> *
> 202E  FFFF-0000  FFFE-0000
> *
> 2032  1F00-
> 2000;G
> *
> * AFTER EXECUTION
> * -----------------------
> 2032  0000-
>
```

LSL, LSR, ROL, ROR, ROXL, ROXR, ASL, ASR INSTRUCTIONS

1 Role of the Instructions

Rotate and shift operations are carried out on byte, word or long word from bits 1 to 8 in static or from bits 1 to 63 in dynamic (when the destination is a Dn register). On the other hand, if the destination location is a memory position, operations on it can only be carried out on one bit.

2 Syntax of the Instructions
Static : #Cnt,Dn

```
ASR.W #cnt,Dn
```

ASR.W	:	instruction mnemonic, Arithmetic Shift Right
#cnt	:	shift count (1 to 8 maximum)
Dn	:	destination register

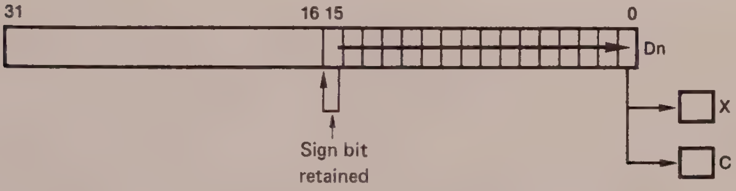


Figure 6.8

Example

```
ROL.W #8,Dn
```

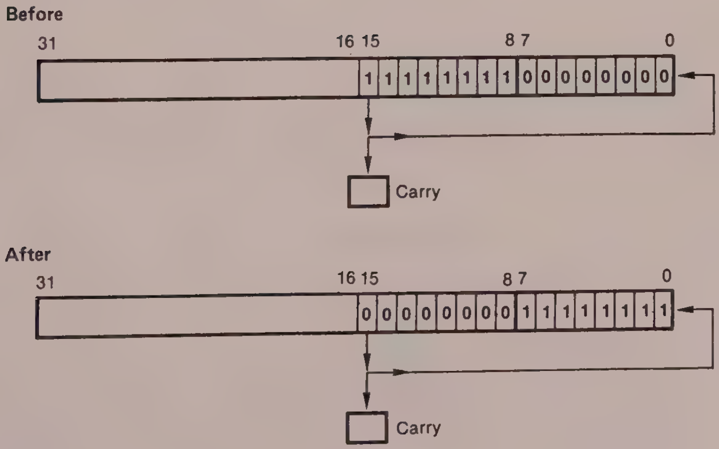


Figure 6.9

Dynamic : Dm,Dn
The total number of rotations or elementary shifts of the content of Dn is specified by the 6 low order bits of register Dm (modulo 64).

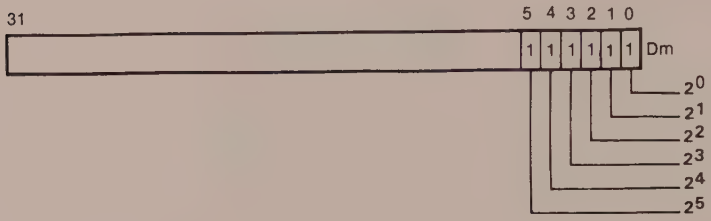


Figure 6.10

Example
The maximum number of rotations or shifts is equal to

N = 1 + 2 + 4 + 8 + 16 + 32 = 63

Programming

```
     MOVEQ #$08,Dm    load number of operations
                      to be carried out
                      (for our example 8 - 2 )

     LSR.W Dm,Dn      8 logical shifts right in Dn
```

Memory Position
If the destination location (dst) is an address, all memory addressing modes are authorised, except for immediate (# Op, dst), relative and relative indexed.
However, the word is the only authorised size.

Example

ROXR.W dst

ROXR.W : instruction mnemonic, rotate operand extended right
dst : destination effective address

The instruction ROXR.W dst causes a rotation of one bit to the right.

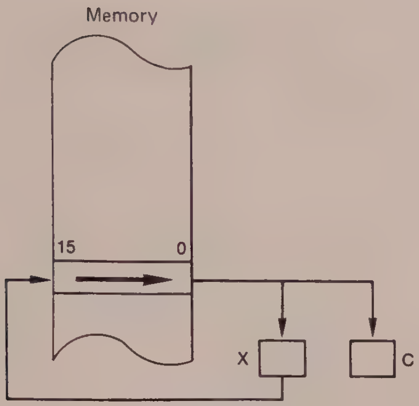


Figure 6.11

MULU AND MULS INSTRUCTIONS

Program Example
The program shown in listing 6.6 carries out a BCD -> binary conversion and displays the result (see simulation).

Listing 6.6

```
                        ************************************************************************
                        *                                                                      *
                        * U S I N G  "M U L U   and M U L S"  I N S T R U C T I O N S         *
                        *                                                                      *
                        * WRITTEN BY: PATRICK JAULENT                                          *
                        * MACMILLAN EDITION                                                    *
                        *                                                                      *
                        ************************************************************************
                        *
                        ** BCD --> BINARY   CONVERSION
                        * ===========================
                        * RESULT DISPLAY IN BINARY

         0001F9E9       ADDRACIA EQU    $1F9E9              ;ACIA 6850 SYSTEM EUROMAK 68000

         00002000                ORG    $2000

002000 4FF82156                  LEA.L   STACK,SP           ;INITIALISE STACK SSP
002004 4BF90001F9E9              LEA.L   ADDRACIA,A5        ; A5.L:= ADDRESS ACIA 6850
00200A 3F38208C                  MOVE.W  NUMBERBCD,-(SP)    ;
00200E 610A                      BSR.S   CONVERSION         ;
002010 321F                      MOVE.W  (SP)+,D1           ; RESTORE RESULT
002012 615E                      BSR.S   PCRLF              ;
002014 6134                      BSR.S   DISPLAYED          ;
002016 4E41                      TRAP    #1                 ;
002018 0000                      DC.W    0
```

```
                   *
                   ** BCD --> BINARY CONVERSION
                   * -------------------------
                   * ALGORITHM :
                   * =========
                   *
                   *   RESULT= Di1*10^0 + Di2*10^1 + Di3*10^2 + Di4*10^3
                   *

00201A 48E77800    CONVERSION MOVEM.L D1-D4,-(SP)      ;SAVE ALTERED REGISTERS
00201E 322F0014               MOVE.W  20(SP),D1        ;LOAD NUMBER BCD
002022 4283                   CLR.L   D3               ;D3.L:=0
002024 4284                   CLR.L   D4               ;D4.L:=0
002026 7401                   MOVEQ   #1,D2            ;INITIALIZATION MULTIPLICAND
002028 0C422710    LOOP       CMPI.W  #10000,D2        ; DO  WHILE D2 < #$1000
00202C 6C12                   BGE.S   EXIT             ; !
00202E 3801                   MOVE.W  D1,D4            ; ! D4.W:=D1
002030 0244000F               ANDI.W  #$F,D4           ; ! /* MASQ DIGIT LOW */
002034 C8C2                   MULU    D2,D4            ; ! D4.W*D2.W:=D4.L
002036 D644                   ADD.W   D4,D3            ; ! D3.W+D4.W:=D3
002038 E849                   LSR.W   #4,D1            ; ! /* SHIFT NYBBLE TO HIGH */
00203A C4FC000A               MULU    #10,D2           ; ! D2.W*10:=D2.L
00203E 60E8                   BRA.S   LOOP             ; ENDDO
002040 3F430014    EXIT       MOVE.W  D3,20(SP)        ; STORE RESULT
002044 4CDF001E               MOVEM.L (SP)+,D1-D4      ; RESTORE REGISTERS
002048 4E75                   RTS
                   *
                   ** OUTPUT BINARY RESULT
                   * ===================

00204A 48E7F000    DISPLAYED MOVEM.L D0-D3,-(SP)       ; SAVE ALTERED REGISTERS
00204E 7403                   MOVEQ   #4-1,D2          ; NYBBLE NUMBER
002050 7603        RBIN1      MOVEQ   #4-1,D3          ; BITS BUMBER
002052 7020                   MOVEQ   #$20,D0          ; ASCII SPACE
002054 612A                   BSR.S   OUTCH1           ; OUTPUT CHAR.
002056 103C0030    RBIN2      MOVE.B  #'0',D0          ; ASCII 0
00205A E349                   LSL.W   #1,D1            ;
00205C 6404                   BCC.S   RBIN3            ; IF BIT =0 THEN
00205E 103C0031               MOVE.B  #'1',D0          ; ! /* 0 ,DISPLAYED */
002062 611C        RBIN3      BSR.S   OUTCH1           ; ELSE
002064 51CBFFF0               DBRA    D3,RBIN2         ; ! /* 1 ,DISPLAYED */
002068 51CAFFE6               DBRA    D2,RBIN1         ; ENDIF
00206C 4CDF000F               MOVEM.L (SP)+,D0-D3      ; RESTORE REGISTERS
002070 4E75                   RTS


                   *
                   ** OUTPUT LINE FEED & CARRIAGE RETURN
                   * ==================================

002072 2F00        PCRLF      MOVE.L  D0,-(SP)
002074 700A                   MOVEQ   #$0A,D0
002076 6108                   BSR.S   OUTCH1
002078 700D                   MOVEQ   #$0D,D0
00207A 6104                   BSR.S   OUTCH1
00207C 201F                   MOVE.L  (SP)+,D0
00207E 4E75                   RTS
```

```
                          *
                          ** SUBROUTINE TRANSMIT CHARACTER
                          *  ===========================

    002080 08150001      OUTCH1   BTST.B  #1,(A5)         ; TEST ACIA TRANSMIT READY ?
    002084 67FA                   BEQ.S   OUTCH1          ;
    002086 18400002               MOVE.B  D0,2(A5)        ; OK . TRANSMIT CHAR.
    00208A 4E75                   RTS

    00208C 00000002      NUMBERBCD DS.W   1
    00208E 000000C8               DS.L    50
           00002156      STACK    EQU     *
                                  END

****** TOTAL ERRORS   0--   0


SYMBOL TABLE - APPROXIMATELY  500 SYMBOL ENTRIES LEFT

ADDRACIA   01F9E9 CONVERSI   00201A DISPLAYE   00204A EXIT      002040
LOOP       002028 NUMBERBC   00208C OUTCH1     002080 PCRLF     002072
RBIN1      002050 RBIN2      002056 RBIN3      002062 STACK     002156
```

```
   *
 > * SIMULATION PROGRAM
 > * ==================
 > *
 > * 208E.W = NUMBER BCD
 > * -------------------
 > *
 > 208E.W
 * 00208E > 0010
 * 002090 >
 > 2000;G
0000 0000 0000 1010
 > *
 > *
 > 208E.W
 * 00208E > 0100
 * 002090 >
 > 2000;G
0000 0000 0110 0100
 > *
 > *
 > 208E.W
 * 00208E > 1000
 * 002090 >
 > 2000;G
0000 0011 1110 1000
 > *
 > *
 >
```

ABCD AND SBCD INSTRUCTIONS

1 Role of ABCD and SBCD

The instructions ABCD and SBCD carry out the addition and the subtraction of operands using binary coded decimal arithmetic. Both operations are byte only.

As a consequence, the MC 68000 has no need of the well known decimal adjustment instruction DAA that 8-bit microprocessors like the MC 6800 and MC 6809 possess.

These instructions (ABCD and SBCD) add and subtract the source operand to/from the destination operand, taking account of the extend bit, and store the result in the destination location.

2 Syntax of ABCD and SBCD

The ABCD and SBCD instructions use only two addressing modes, as follows.

Data Register Direct, where the source operand and the destination operand are respectively contained in a register Dn.

Example

```
                     ABCD Dn, Dnl
                     SBCD Dn, Dnl

Source                     Source                     Destination
```

Predecrement Register Indirect addressing where the source and destination operands are stored in the the addresses to by the registers An.

Example

```
                  ABCD - (An), -(Anl)
                  SBCD - (An), -(Anl)

Source                                        Destination
```

Choice of Address Mode Explained

Motorola´s choice of the addressing mode with predecrementation is justified by the type of calculation involved.

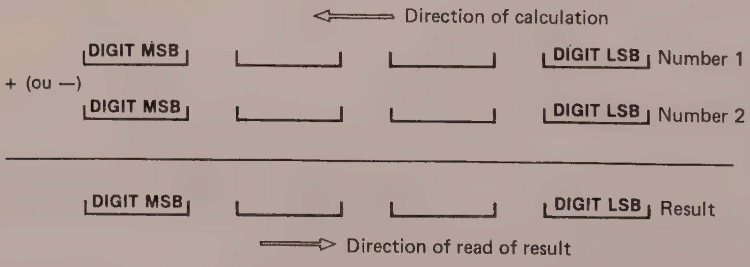


Figure 6.12

In fact, decimal arithmetic calculation requires numbers (operands) to be handled as shown in the figure below; that is, from the least significant digit (LSB) towards the most significant digit (MSB).
The predecrement address mode carries out the calculation of the LSB digits (stored in the MSB addresses) towards the MSB digits (stored in the LSB addresses), thus facilitating the reading of the result into memory.

(Destination) + (Source) + (X) -> Destination

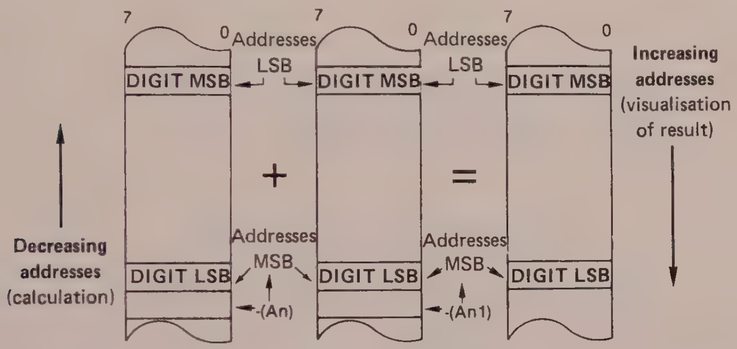


Figure 6.13 The instruction ABCD -(An), -(An1).
How it functions.

Program Example
The program of listing 6.7 demonstrates the use of the ABCD instruction.

Listing 6.7

```
                        *************************************************************
                        *                                                           *
                        *     U S E   " A B C D "   I N S T R U C T I O N           *
                        *                                                           *
                        *  WRITTEN BY: PATRICK JAULENT                              *
                        *  EDITION   : MACMILLAN                                    *
                        *************************************************************

                        *
                        * ADD THE SOURCE OPERAND TO THE DESTINATION OPERAND ALONG WITH THE
                        * EXTEND BIT (X), AND STORE RESULT IN THE DESTINATION .
                        *
        00002000                ORG     $2000

    002000 41F82024             LEA.L   DESTINATION,A0          ; INITIALISE POINTER DEST.
    002004 43F8201E             LEA.L   SOURCE,A1               ; INITIALISE SOURCE POINTER.
```

```
                       *
    002008 7204                MOVEQ   #5-1,D1             ; NUMBER OPERATION
    00200A 023C000F            ANDI.B  #$0F,CCR             ; X:=0
    00200E C109        LOOP    ABCD    -(A1),-(A0)         ;
    002010 51C9FFFC            DBRA    D1,LOOP             ;
    002014 4E41                TRAP    #1
    002016 0000                DC.W    0

    002018 00000006            DS.B    6
           0000201E    SOURCE  EQU     *
    00201E 00000006            DS.B    6
           00002024    DESTINATION EQU *
                               END

****** TOTAL ERRORS   0--   0


SYMBOL TABLE - APPROXIMATELY  509 SYMBOL ENTRIES LEFT

DESTINAT   002024 LOOP       00200E SOURCE     00201E
```

Simulation

```
       *
     > * SIMULATION PROGRAM:
     > * ==================
     > *
     > * DESTINATION:
     > * ------------
     > *
     > 201F.B
     * 00201F > 98
     * 002020 > 10
     * 002021 > 15
     * 002022 > 90
     * 002023 > 10
     * 002024 >
     > *
     > * SOURCE:
     > * -------
     > *
     > 2019.B
     * 002019 > 02
     * 00201A > 30
     * 00201B > 60
     * 00201C > 10
     * 00201D > 80
     * 00201E >
     > *
     >
     > 2000;G
     > *
     > * RESULT --> DESTINATION
     > * ----------------------
     > *
     > 201F  > 00201E  9000-  4076-  0090-
     >
     > *
     > * DESINATION:
     > * ===========
     > *
     > 201F.B
     * 00201F > 00
```

```
* 002020 > 99
* 002021 > 99
* 002022 > 99
* 002023 > 99
* 002024 >
> *
> * SORCE
> * SOURCE:
> * ----------
> *
> 2019.B
* 002019 > 00
* 00201A > 99
* 00201B > 99
* 00201C > 99
* 00201D > 99
* 00201E >
> *
> *
> 2000;G
> *
> * RESULT ----> DESTINATION
> * ----------------------------------------
> 201F   > 00201E   9001-   9999-   9998-
```

DIVU and DIVS INSTRUCTIONS

1 Role of DIVU and DIVS

The DIVU and DIVS instructions tell the MC 68000 to divide the 32 bits of a data register (destination) by the 16 bits of one of the following

an operand	DIVU #16,d Dn DIVS #16,d Dn
a memory position	DIVU saddr,d Dn DIVS saddr,d Dn
the low order bits of a register Dn	DIVU sDn,d Dn DIVS sDn,d Dn

2 How the Instruction is Executed

(Destination) : (Source) -> Destination

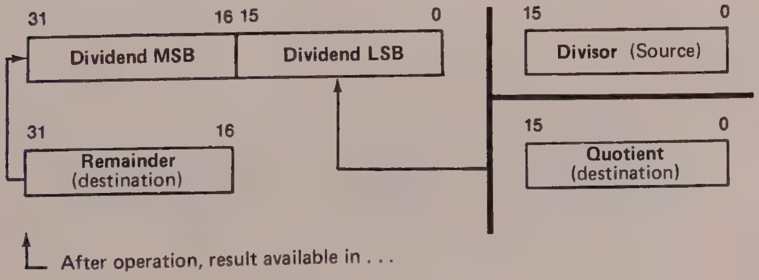


Figure 6.14

See also the program in listing 6.8.

Details
Both source and destination cannot be an address register.
Any zero division causes a trap.
If overflow occurs the 68000 does not carry out the division (the registers are not modified) but sets V to 1.

Listing 6.8

```
*
 > ****************************************************************
 > *                                                              *
 > *    " D I V U    AND    D I V S "    I N S T R U C  T I O N S *
 > *                                                              *
 > ****************************************************************
 > *
 > *  PROGRAM:
 > *  ========
 > *
 > *         $8001         DIVU  D1,D0
 > *         $4E76         TRAPV
 > *         $4E71         NOP
 > *         $60FE         BRA   *
 > *
 > *
 > 2000   0022--8001
 > 002002 06D2--4E76          loading of program
 > 2004   008E--4E71
 > 2006   0911--60FE
 > *
 > *
 > * FIRST SIMULATION:
 > * ------------------
 > *
 > *
 > *
 > *
 > .D0 = 00000000 --0000FFFF      Dividend
 > .D1 = 00000000 --00000002      Divisor
 > ;R  PC= 000400 # 0050    S=0 S 000    C=    .....   SP= 00000600
        D0= 0000FFFF  D1= 00000002  D2= 00000000  D3= 00000000
        D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
        A0= 00000000  A1= 00000000  A2= 00000000  A3= 00000000
        A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600

 > *
 > * BEFORE EXECUTION
 > * ------------------
 > 2000;G                  Quotient
 * ABRT  PC= 002006 # 60FE    S=0 S 000    C=    .....   SP= 00000600
        D0= 00017FFF  D1= 00000002  D2= 00000000  D3= 00000000
        D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
        A0= 00000000  A1= 00000000  A2= 00000000  A3= 00000000
        A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600

          Remainder
 > *
 > * AFTER EXECUTION
 > * ------------------
 >
 *
> * SECOND SIMULATION:
> * --------------------------------
> *
> .D1 = 00000002 -00000000      Divisor
> .D0 = 00017FFF -00002222      Dividend
> ;R  PC= 002006 # 60FE    S=0 S 000    C=     .....   SP= 00000600
        D0= 00002222  D1= 00000000  D2= 00000000  D3= 00000000
        D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
        A0= 00000000  A1= 00000000  A2= 00000000  A3= 00000000
        A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600

> *
> * BEFORE EXECUTION:
> * --------------------------------
> *
> 2000;G
* DIV- ERR! 0000 00000000 0000
 REG:   PC= 002002 # 4E76    S=0 S 000    C=    ..Z..   SP= 00000600
        D0= 00002222  D1= 00000000  D2= 00000000  D3= 00000000
        D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
        A0= 00000000  A1= 00000000  A2= 00000000  A3= 00000000
        A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600

> *
> * DIVISION BY ZERO !!!
> * ------------------------------
> *
> * THIRD SIMULATION:
> * --------------------------------
> *
> .D1 = 00000000 -00000004      Divisor
> .D0 = 00002222 -FFFFFFFF      Dividend
> *
> 2000;G                                               V := 1
* TRV- ERR! 0000 00000000 0000
 REG:   PC= 002004 # 4E71    S=0 S 000    C=    .N.V.   SP= 00000600
        D0= FFFFFFFF  D1= 00000004  D2= 00000000  D3= 00000000
        D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
        A0= 00000000  A1= 00000000  A2= 00000000  A3= 00000000
        A4= 00000000  A5= 00000000  A6= 00000000  A7= 00000600

> *
> * " V:=1 IF OVERFLOW
> * ------------------------------------
> *
>
```

CHK INSTRUCTION

1 Role of CHK

The CHK instruction compares the 16 low order bits of a data register with a bounded value.

By definition, the lower bound is zero and the upper bound is a 16-bit number.

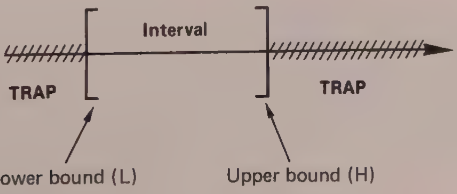


Figure 6.15

The trap operates when the value that is examined, which is contained in a register Dn, does not belong to the interval.

It should be noted that for the CHK instruction the inequalities are always strict.

2 Syntax of CHK

In assembler

```
                   CHK <ea>, Dn
                         ^      ^
Effective address -------'      '-----Data register
```

In pseudo-code

```
IF Dn < L THEN
|  TRAP CHK
ELSE
|  IF Dn > H THEN
|  |  TRAP CHK
|  ELSE
|  |  Execution of next instruction
|  ENDIF
ENDIF
```

Application Exercises

1. Write in 68000 assembly language a program to search for a value called entry in a table. If this entry ($000F for our example) is not found in the table - arbitrarily fixed at five words - the program will need to be rerouted into the TRAP CHK, with the aim of displaying the message "Value not found in table". On the other hand, if the entry is present, the system returns to the control of the monitor.

The algorithm used to search for the entry will need to be established in pseudo-code and the program written in position independent code.

Suggested solution

*Pseudo-code

```
/*STACK INITIALISATION*/
/*VECTOR CHK INITIALISATION*/
/*TABLE ADDRESS INITIALISATION*/
  NUMBER : = 5
  READ 1 VALUE IN TABLE
       IF  VALUE <> ENTRY THEN
       |   NUMBER : = NUMBER - 1
       |   IF  NUMBER <> -1 THEN
       |   |   REPEAT
       |   ELSE
```

```
    |    |   DISPLAY "VALUE NOT FOUND IN TABLE"
    |   ENDIF
  ELSE
    |  /*RETURN MONITOR*/
  ENDIF
```

Listing 6.9

```
                  **************************************************************
                  *                                                            *
                  *  SEARCH A TABLE OF 5 WORDS FOR SPECIFIC OPERAND (#$000F)   *
                  *  IF NOT FOUND THEN "TRAP CHK"                              *
                  *  THE PROGRAM IS WRITTEN IN   A "RORG" SECTION              *
                  *                                                            *
                  **************************************************************

       0001E178   OUTMES   EQU     $1E178              ; SUBROUTINE PRINT STRING OF CHARS
       00000018   VECTORCHK EQU    6*4                 ; ADDRESS VECTOR TRAP CHK
       00000005   NUMBER   EQU     5                   ; SIZE TABLE

       00002000            RORG    $2000

002000 4FFA013E            LEA.L   STACK,SP            ; INITIALISE POINTER SSP
002004 41FA001C            LEA.L   TRAPCHK,A0          ; A0:= ADDRESS PROGRAM EXCEPTION CHK
002008 21C80018            MOVE.L  A0,VECTORCHK        ; INITIALISE EXCEPTION VECTOR CHK
00200C 43FA0028            LEA.L   TABLE,A1            ; INITIALISE ADDRESS TABLE
002010 7204                MOVEQ   #NUMBER-1,D1        ; INITIALISE SIZE TABLE
002012 0C59000F   LOOP     CMPI.W  #$000F,(A1)+        ; DO WHILE MEM[A1] <> #$F AND D1 <-1
002016 57C9FFFA            DBEQ    D1,LOOP             ; ! /READ MEM[A1]; A1:=A1+2
00201A 43BC0005            CHK     #NUMBER,D1          ; ENDDO
00201E 4E41                TRAP    #1                  ; IF D1 <0 THEN "TRAP CHK"
002020 0000                DC.W    0

       00002022   TRAPCHK  EQU     *
002022 48E7FFFE            MOVEM.L D0-D7/A0-A6,-(SP)
002026 41FA0018            LEA     TEXT,A0             ; INITIALISE A0 BEGIN STRING OF CHARS
00202A 4EB90001E178        JSR     OUTMES              ;
002030 4CDF7FFF            MOVEM.L (SP)+,D0-D7/A0-A6
002034 4E73                RTE


002036 0000000A   TABLE    DS.W    5                   ; STORAGE BLOCK FOR 5 WORDS
002040 20         TEXT     DC.B    '        THE  OPERAND  IS  NOT  FOUND  IN  TABLE !!! '
002074 0A                  DC.B    $0A,$0D,4
002078 000000C8            DS.L    50
       00002140   STACK    EQU     *
                           END

****** TOTAL ERRORS   0--    0


SYMBOL TABLE - APPROXIMATELY  504 SYMBOL ENTRIES LEFT

LOOP       002012 NUMBER     000005 OUTMES     01E178 STACK      002140
TABLE      002036 TEXT       002040 TRAPCHK    002022 VECTORCH   000018
```

Simulation

```
  *
> ** S I M U L A T I O N    P R O G R A M :
> *    ===========================================================
> *
> * 2036.W  IN  2040.W  = STORAGE  BLOCK  FOR 5 WORDS
> * ----------------------------------------------------------------
> *
> 2036.W
* 002036 > 0000
* 002038 > 6666
* 00203A > 8888
* 00203C > DDDD
* 00203E > 9999
* 002040 > 000F
* 002042 >
> 2000;G      THE  OPERAND  IS  NOT  FOUND  IN  TABLE !!!

> *
> *
> 2036.W
* 002036 > 0000
* 002038 > 6666
* 00203A > 8888
* 00203C > EEEE
* 00203E > 000F
* 002040 >
> 2000;G
> *
> * CORRECT
> * -------
>
```

2. The program that follows is especially interesting since it allows the following expression to be tested

$$-|N| \leqslant \text{number} \leqslant +|N|$$

Listing 6.10

```
                    ***********************************************************************
                    *                                                                     *
                    *            EXCEPTION :  CHK  INSTRUCTION                            *
                    *                                                                     *
                    *              -|N|  =< NUMBER =<  +|N|                               *
                    *                                                                     *
                    ***********************************************************************

        0001E178    OUTMES    EQU     $1E178              ; MONITOR SUBROUTINE PRINT CHAR
        00000018    VECTORCHK EQU     6*4                 ; VECTOR TRAP CHK

        00002000              ORG     $2000

 002000 4FF820E6              LEA.L   STACK,SP            ; INITIALIZE STACK POINTER SSP
 002004 21FC00002034
            0018              MOVE.L  #TRAPCHK,VECTORCHK  ; INITIALIZE CHK INSTRUCTION
                    *
 00200C 31FC10002048          MOVE.W  #$1000,H            ; LIMIT H
```

```
002012 31FCE000204A          MOVE.W  #-$2000,L          ; LIMIT L
002018 3038204C              MOVE.W  NUMBER,D0          ; LOAD NUMBER FOR CHECK
                     *
00201C 9078204A              SUB.W   L,D0               ; D0:=D0-L
002020 3238204A              MOVE.W  L,D1               ; D1:=L
002024 93782048              SUB.W   D1,H               ; H:=H-D1
                     *
                     *    ----------[--------]----------
                     *    L=-$2000   NUMBER   H=$1000
                     *
002028 41B82048              CHK     H,D0               ; IF  D0 < -$2000   THEN
00202C D078204A              ADD.W   L,D0               ; ! /* TRAP CHK */
002030 4E41                  TRAP    #1                 ; ELSE IF D0> $1000 THEN
002032 0000                  DC.W    0                  ; !    ! /* TRAP CHK */
                     *                                  ; !    ENDIF
                     *                                  ; ENDIF
                     *
       00002034      TRAPCHK EQU     *
002034 48E7FFFE              MOVEM.L D0-D7/A0-A6,-(SP)      ; SAVE ALTERED REGISTERS
002038 41F8204E              LEA.L   TEXT,A0                ; A0:= @ TEXT
00203C 4EB90001E178          JSR     OUTMES                 ;
002042 4CDF7FFF              MOVEM.L (SP)+,D0-D7/A0-A6      ; RESTORE REGISTERS
002046 4E73                  RTE
                     *
                     * RESERVE
                     * -------
002048 00000002      H       DS.W    1                      ; ADDRESS LIMIT H
00204A 00000002      L       DS.W    1                      ; ADDRESS LIMIT L
00204C 00000002      NUMBER  DS.W    1                      ; NUMBER FOR CONTROL

00204E 0A            TEXT    DC.B    $0A,$0D
002050 20                    DC.B    '          TRAP   CHK          '
00206A 0A                    DC.B    $0A,$0D,04
00206E 00000078              DS.L    30
       000020E6      STACK   EQU     *
                             END

****** TOTAL ERRORS   0--   0
```

Simulation

```
> * SIMULATION PROGRAM:
> * ==========================
> *
> * 204C.W = NUMBER FOR CONTROL
> *
> 204C.W
* 00204C > 1000
* 00204E >
> 2000;G
> *
> *
> 204C.W
* 00204C > 1001
* 00204E >
> 2000;G
          TRAP    CHK
```

```
> *
> *
> 204C.W
* 00204C > E000
* 00204E >
> *
> 2000;G
> *
> 204C.W
* 00204C > DFFF
* 00204E >
> *
> 2000;G
        TRAP   CHK

>
> *
> *
> 204C.W
* 00204C > FFFF
* 00204E >
> 2000;G
>
```

MOVEM INSTRUCTION

1 Role of MOVEM

The different versions of the MOVEM instruction transfer, according to a predetermined order, a list of address and/or data registers to or from a block of memory.

The second word of the instruction (the first always being the op-code) is established from a 16-bit order table, in which each register occupies one bit, so that any combination of the 16 registers can be specified by the instruction.

2 Syntax of MOVEM

Whatever type of transfer is involved, whether registers to memory or memory to registers, the size specified by the instruction is word, which is the default size, or long word.

1. We take as a first example the transfer of a list of registers to a block of memory (see figure 6.16).

```
MOVEM.W A1/A5/D2/D4,$1000
```

MOVEM.W	: instruction mnemonic and size (word)
A1/A5/D2/D4	: list of registers whose 16 LSB bits are transferred to the memory block.
$1000	: destination effective address automatically incremented by two if the size is word (or by 4 if it is long word)

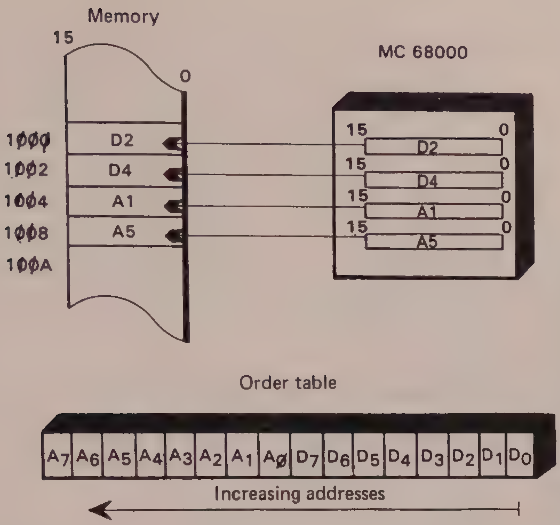


Figure 6.16 MOVEM.W A1/A5/D2/D4,$1000

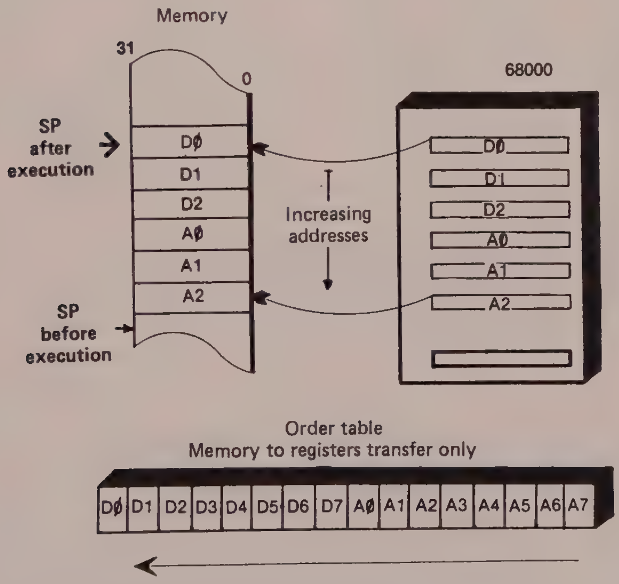


Figure 6.17 MOVEM.L A0-A2/D0-D2,-(SP)

2. Our second example is the transfer of a list of registers to a memory block (stack) in predecrement mode (see figure 6.17).

MOVEM.L	:	instruction mnemonic and size (long word)
A0-A2/D0-D2	:	the 32 bits of each register in the list are transferred to the stack.
-(SP)	:	stack pointer requested in predecrement mode

Listing 6.11

```
                    *************************************************************************
                    *                                                                       *
                    *      U S E    O F   "M O V E M"    I N S T R U C T I O N              *
                    *                                                                       *
                    *************************************************************************

                    *
                    * COPY N BYTES FROM ONE LOCATION TO ANOTHER LET L (LENGTH) BE
                    * 2048 (2 K BYTES)
                    * EXEMPLE: 2048/32=64 LOOPS
                    * -------
                    *
                    * A0 =POINTS TO THE SOURCE BLOCK
                    * A1 =POINTS TO THE DESTINATION BLOCK
                    * D0 =NUMBER OF 32 BYTES TO MOVE
                    *
       00002000              RORG    $2000

002000 41FAFFFE              LEA.L   SOURCE-32,A0              ;POINTER TO SOURCE
002004 43FA101A              LEA.L   DESTINATION,A1            ;POINTER TO DESTINATION
002008 303C0800              MOVE.W  #64*32,D0                 ;COUNT BYTES TO XFER
                    *
00200C 4CF004FE0000 LOOP     MOVEM.L 0(A0,D0.W),D1-D7/A2       ;REPEAT
002012 48E17F20              MOVEM.L D1-D7/A2,-(A1)            ;! /* MOVE DATA  IN */
002016 04400020              SUBI.W  #32,D0                    ;! /* SEND DATA OUT */
00201A 66F0                  BNE.S   LOOP                      ;! D0:=D0-32
00201C 4E41                  TRAP    #1                        ;UNTIL D0=0
00201E 0000                  DC.W    0

       00002020     SOURCE   EQU     *
002020 00000800              DS.B    64*32
002820 00000800              DS.B    64*32
       00003020     DESTINATION EQU  *
                             END

****** TOTAL ERRORS   0--   0


SYMBOL TABLE - APPROXIMATELY  509 SYMBOL ENTRIES LEFT

DESTINAT   003020 LOOP      00200C SOURCE     002020
```

3. Our third example is the transfer of a memory block to a list of registers in postincrement mode (see figure 6.18).

```
MOVEM.L (SP)+,D0-D7/A0-A6
```

MOVEM.L   : instruction mnemonic and size (long word)
(SP)+   : stack pointer used in postincrement mode
D0-D7/A0-A6: list of registers loaded with the contents of addresses pointed to by SP*.

*Note that the intervals of a list of registers are indicated by dashes and the different register names by the slash symbol. For example,
A0-A3/D0-D4 means A0, A1, A2, A3 and D0, D1, D2, D3, D4
A0/A4/D0/D6 means A0, A4 and D0, D6.

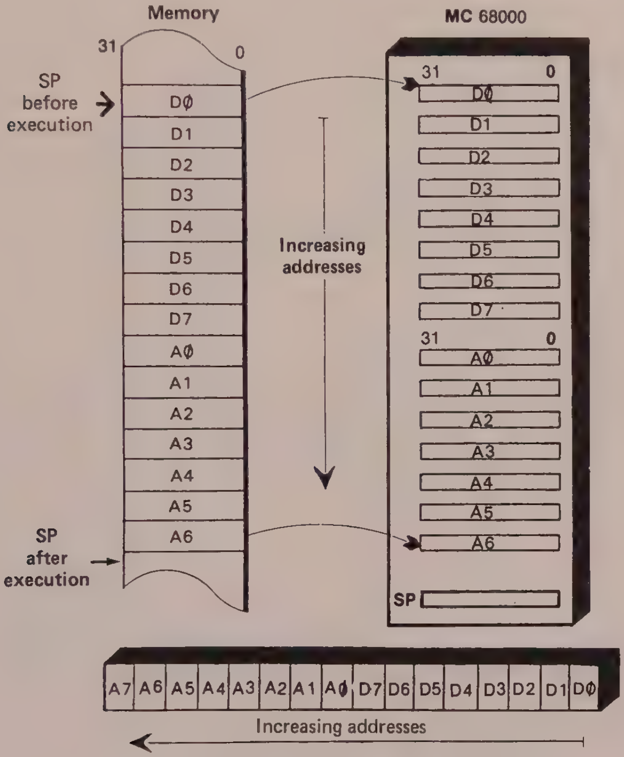


Figure 6.18 MOVEM.L(SP)+,D0-D7/A0-A6

MOVEP INSTRUCTION

The MOVEP instruction facilitates access to the peripheral devices that occupy alternate bytes in a memory area.

1 Some Reminders about Addressing

When the MC 68000 processor wishes to interact with an 8-bit memory location, it uses the internal bit A0 to select the even address (A0 = 0) or odd address (A0 = 1) and the outputs LDS and UDS to enable the lines, whether lower D0-D7 (LDS = 0) or upper D8-D15 (UDS = 0).

2 Role of MOVEP
The MOVEP instruction allows one to program the 8-bit peripheral circuits of the MC 6800 (PIA 6821, PTM 6840) or MC 68000 (PI/T 68230) via the lower line (D0-D7) if the peripheral address is odd, or via the upper line (D8-D15) if the address is even.

3 Syntax of MOVEP
In assembler

```
MOVEP.W d16(An),Dn       Read from
MOVEP.L d16(An),Dn       peripheral

MOVEP.W Dn,d16(An)       Write to
MOVEP.L Dn,d16(An)       peripheral
```

4 Programming Examples

Without using MOVEP
Let us assume that we wish to program the PIA 6821 circuit according to the model shown in figure 6.19, assuming that this circuit occupies odd addresses (4) that are programmable via the lower line.

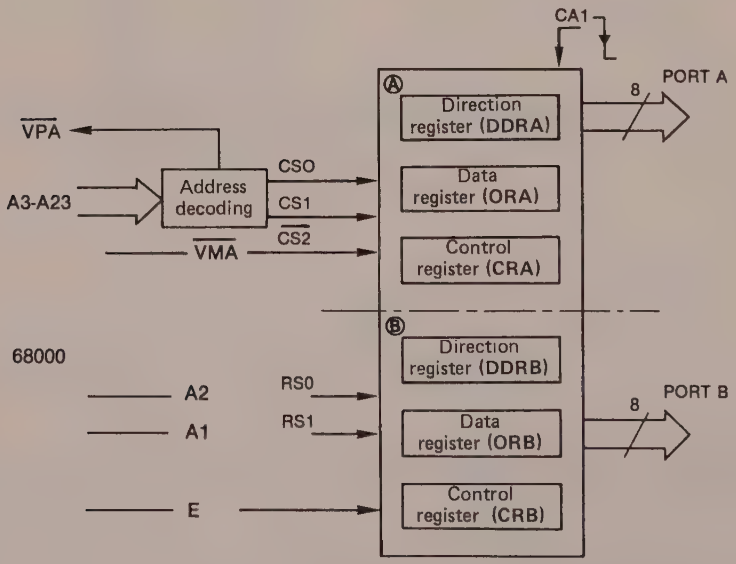


Figure 6.19 Programming: ports A and B on output; interrupt on CA1 authorised; access to ORA and ORB.

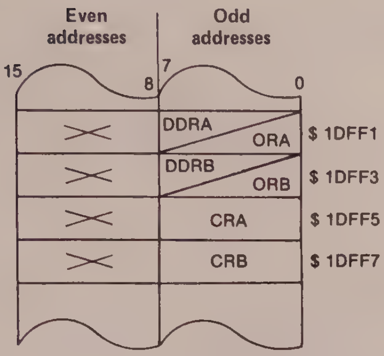


Figure 6.20 Memory positioning of peripheral

Program *Equivalences*

```
PIADOA   EQU $1DFF1          Direction/data registers
                             of port A
PIADOB   EQU PIADOA +2       Direction/data registers
                             of port B
PIACRA   EQU PIADOA +4       Control register of port A
PIACRB   EQU PIADOA +6       Control register of port B

         *Program*

MOVE.B #$FF,PIADOA           Port A on output
MOVE.B #$FF,PIADOB           Port B on output
MOVE.B #$05,PIACRA           Interrupt authorised
                             on CA1, access to ORA
MOVE.B #$04,PIACRB           Access to ORB
```

The above is exactly the same method of programming as with the MC 6800.

Using MOVEP

Program *Equivalence*

```
PIADOA EQU $1DFF1            Base address of peripheral
                             circuit
MOVE.L # PIADOA,A6           Base initialisation
                             address
MOVE.L #$FFFF0504,D1         Loading of command long
                             word of PIA
MOVEP.L D1,0(A6)             Transfer of contents of D1
                             to alternate byte memory
```

After each byte transfer, the address specified by A6 is incremented by two (register A6 is not modified).

Programming Exercise

Let us suppose that we want to write in 68000 source language the initialisation program of N peripheral PIA 6821 circuits. The operands (32 bits) belonging to each peripheral circuit are stored in a table, whose begin address is MEMTABLE.

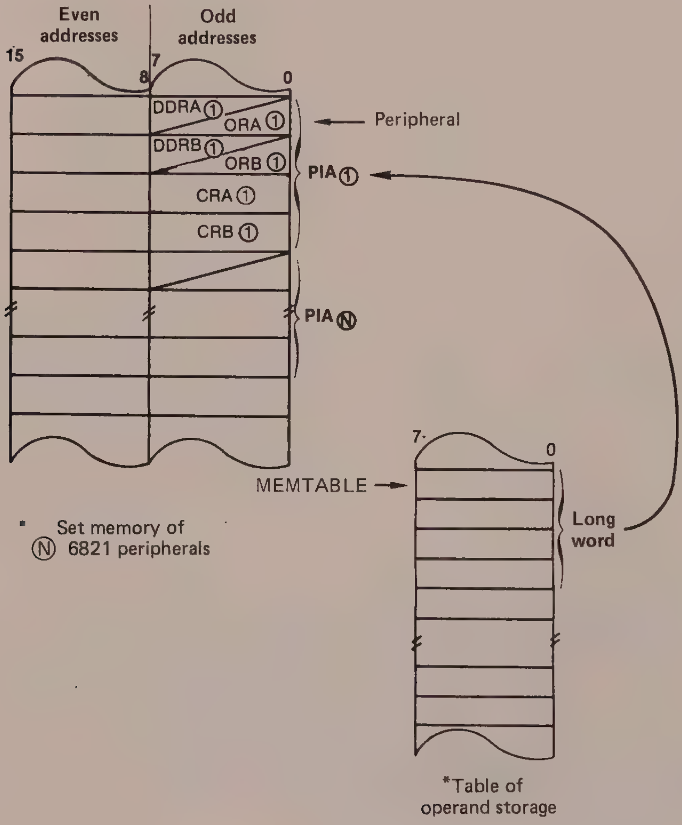


Figure 6.21

Program

```
     MOVEQ # N,D0          Initialisation of number
                           of peripherals
     LEA MEMTABLE,A0       Loading A0 with
                           MEMTABLE address
     LEA PERIPHERAL,A1     Loading A1 with
                           peripheral address
     BRA IN                Adjustment of
                           primitive DBF
```

```
LOOP MOVE.L (A0)+,D1     Loading of programming
                         long word (A0) + 4 -> A0
     MOVEP.L D1,0(A1)    Transfer of operand to
                         8-bit peripheral
     ADDQ.L #8,A1        Incrementation of
                         table pointer
IN   DBF D0,LOOP
     END
```

LINK AND UNLK INSTRUCTIONS

1 Role of LINK and UNLK

These high level instructions allow automatic allocation of an area of memory used for the storage of local variables or for the passing of parameters between two programs.

These instructions simplify the writing of reentrant subroutines that work on areas of memory that belong to the calling program, or shareable subroutines that can be shared by several users.

2 Syntax of LINK and UNLK

LINK (with stack)
Assembler notation

LINK An, # displacement

LINK	: instruction mnemonic
An	: represents the block pointer (frame pointer FP). Only address registers A0-A6 can be used as FP, as A7 represents the stack pointer.
# displacement	: signed 16-bit displacement ($-32768 \leqslant d \leqslant +32767$)

The value of the displacement included in the LINK instruction is used to decrement the stack pointer so that it frees a memory location when it is required for storing local variables or parameters. These variables need to stored in relation to the frame pointer.

Method of Operation

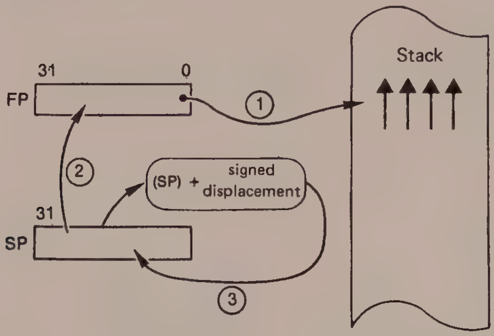


Figure 6.22

1. Saving of the contents of FP to the stack.
2. Creation of a new FP with the contents of SP.
3. Reserving of a work area fixed by the addition of SP with displacement.

UNL(IN)K (disconnection of stack)
Assembler notation

UNLK An

UNLK	:	instruction mnemonic
An	:	represents the frame pointer (block pointer)

Method of Operation

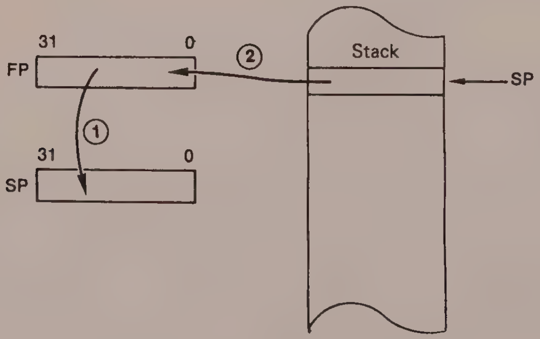


Figure 6.23

1. The stack pointer (SP) is loaded with the contents of the frame pointer (FP).
2. Register FP is loaded with the contents of the address pointed to by SP.

Example
Reserving an area of memory between the main program and the stack pointer

```
LEA $ 2000,SP        Initialisation of stack pointer
LEA $ 2002,A6        Initialisation of frame pointer

LINK A6,#-4          Reserves two words in common zone
NOP
NOP
JSR INPUTBCD         Call subroutine
                     INPUTBCD (1st number)
JSR INPUTBCD         Call subroutine
                     INPUTBCD (2nd number)
MOVE.W (A6)+,D5      (A6) -> D5 ; (A6) + 2 -> A6
MULU   (A6)+,D5      (A6) x (D5) -> ; (A6) + 2 -> A6
UNLK A6
  :
  :
Rest of main program

INPUTBCD  *Capture of ASCII character from keyboard
          *Conformity test
          *Conversion of ASCII character to binary
           (D5 contains the binary number)
           MOVE.W D5,-(A6)
           RTS
```

We shall now examine the above program in detail.
The two instructions LEA $2000,SP and LEA $2002,A6 load the stack pointer SP with $00002000 and the frame pointer FP with $00002002.
The instruction LINK A6,#-4 causes the following operations to occur.
1. The "old" FP value (the 32 bits of A6) are saved to the stack.

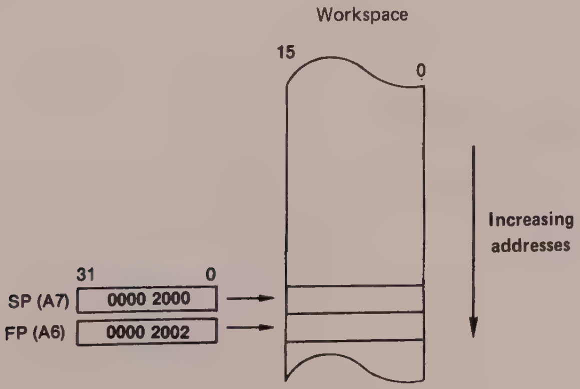


Figure 6.24 (Part)

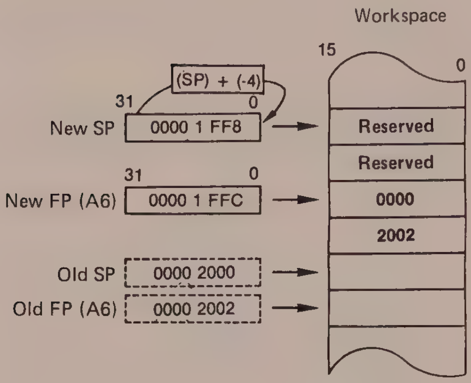


Figure 6.24 (Continued)

2. A new FP is created with the contents of the stack pointer SP (remember that SP has been decremented by 4 during the previous operation).

3. A location is reserved in the stack to store two words (4 bytes).

The NOP (no operation) instructions are there to represent the programming of the peripheral circuit (PI/T 68230 or ACIA 6850) connected to a keyboard which, for the purposes of this example, we have not considered it necessary to write.

JSR INPUTBCD, after the return address has been saved to the stack, causes the CPU to jump to the subroutine INPUTBCD.

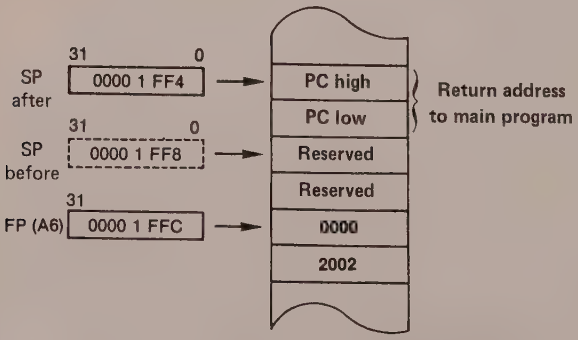


Figure 6.25

The subroutine INPUTBCD, whose different sequences have been omitted intentionally, instructs the MC 68000 to

a) fetch an ASCII character from the keyboard and carry out a conformity test on the character obtained;

b) convert this character into binary (D5 contains the binary number);

c) store the first number fetched in the reserved area of memory, this operation being carried by the instruction MOVE D5, -(A6), before returning to the calling program via RTS.

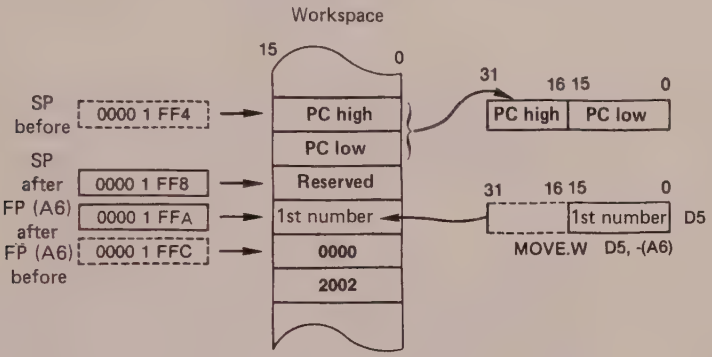


Figure 6.26

Instruction JSR INPUTBCD causes the processor to make a new call to the subroutine INPUTBCD, with the aim of storing the second number in reserve.

Instruction RTS returns control to the calling program (SP after RTS points to the second number fetched).

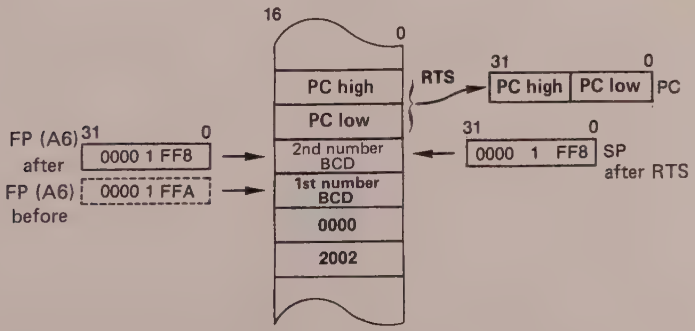


Figure 6.27

Instruction MOVE.W (A6)+,D5 loads the LSB word of the data register D5 with the contents of the address pointed to by register A6(A6 = FP); A6 is then incremented by two in order to complete the addressing with postincrement.

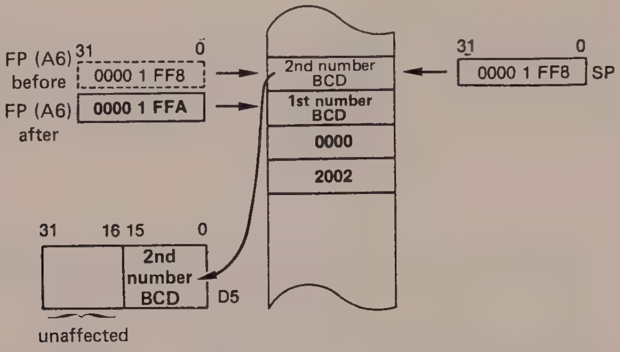


Figure 6.28

The next instruction MULU (A6) +,D5 carries out the unsigned multiplication of the 16 bits pointed to by A6 with the 16 low order bits of register D5. The 32-bit result is available in D5. The content of A6 is then incremented by two (postincrement), which positions FP at address $00001FFC.

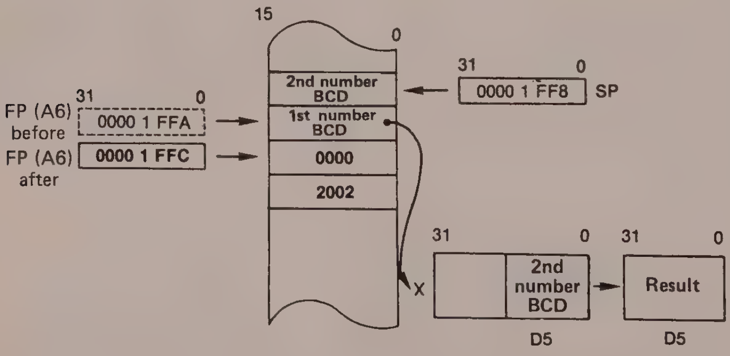


Figure 6.29

Instruction UNLK A6 causes the following
1. Loading of SP with the contents of FP.
2. Loading of FP with the contents of the address pointed to by SP (SP is incremented by 4 after this operation).

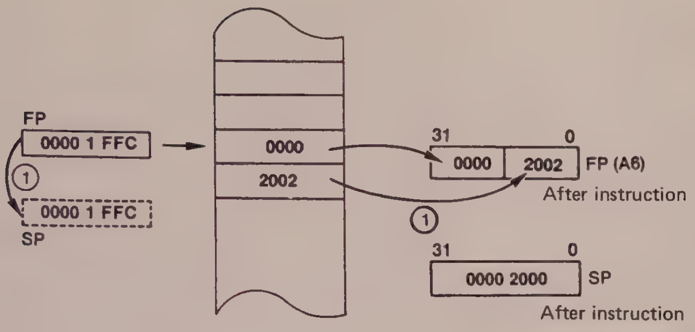


Figure 6.30

The following example illustrates the value of the LINK and UNLK instructions in the writing of a subroutine that can be called by several users.

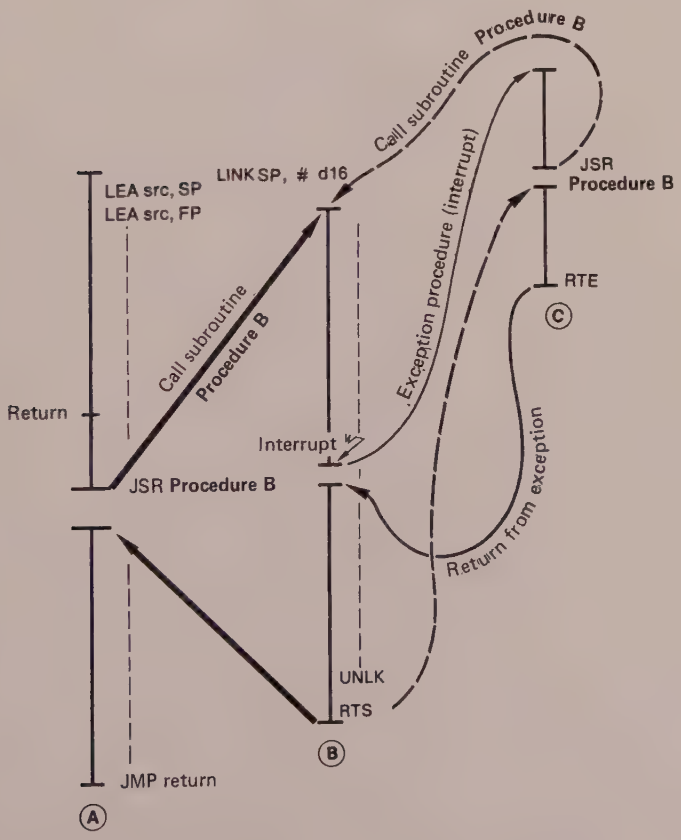


Figure 6.31

Simulation (see figure 6.32)

Program (A) calls the interruptable procedure (B) by means of instruction JSR PROCEDURE B.

The first instruction of procedure (B), namely LINK FP,#d16, reserves for calling program (A) its own working storage area.

While executing procedure (B), the processor is interrupted by an external device (hard or soft processor). After recognising the interrupt, the 68000 saves the program counter and the status register to the supervisor stack, before executing the exception program (C) which itself calls procedure (B) (JSR PROCEDUREB).

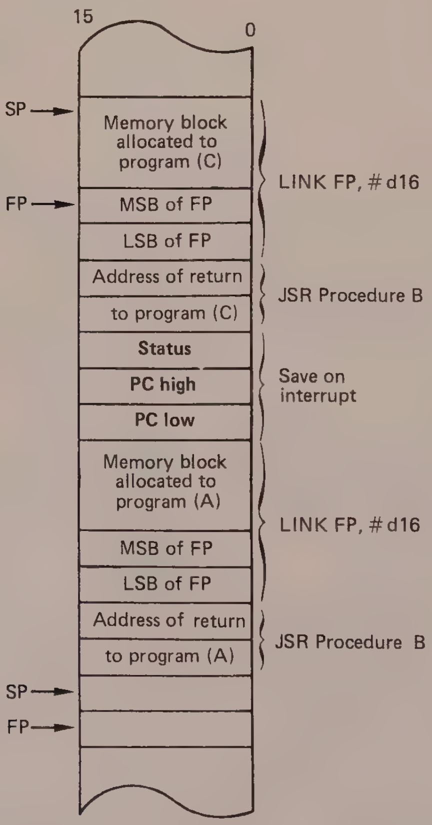


Figure 6.32

In a manner similar to the first call of procedure (B), the calling program (C) is provided with its own workspace.

After it has executed procedure (B), the MC 68000 repositions the frame pointer at the state prior to the interrupt by executing the instruction UNLK FP.

Instruction RTS orders the processor to return to the exception program which has to terminate with the instruction RTE, in order to ensure return to procedure (B) for the program to continue. Of course, the workspace currently defined by the frame pointer belongs to program (A).

Instruction UNLK FP resets register FP to its initialisation value, before executing RTS which allows it to return to program (A).

Example of the use of LINK and UNLK

Listing 6.12

```
***********************************************************************
*                                                                     *
*    U S E   O F   "L I N K   and  U N L K "  I N S T R U C T I O N S *
*                                                                     *
***********************************************************************
* UNSIGNED DIVISION:
* ------------------
* THE SIMPLEST BINARY DIVISION ALGORITHM IS ALSO BASED ON THE TECHNIQUE
* WE LEARNED IN GRAMMAR SCHOOL.
*
*          DIVISION:   64/32 = 32 BITS REMAINDER
*          --------          = 32 BITS QUOTIENT
* ALGORITHM "PSEUDOCODE"
* ====================
*
* PROCEDURE DIVISION 64 BITS
* !
* !  STATUS :=0
* !  D0.L   :=MSB DIVIDEND
* !  D1.L   :=LSB DIVIDEND
* !  IF  D0.L >= DIVISOR THEN
* !  !
* !  !   /* WRITE : " DIVIDE OVERFLOW " */
* !  !   /* WRITE : " DIVIDE BY 0 */
* !  !   STATUS :=1
* !  !
* !  ELSE
* !  !  COUNTER :=32
* !  !  DO  WHILE  COUNT <> 0
* !  !  !
* !  !  !  D1.L:= D1.L*2
* !  !  !  D0.L:= D0.L*2  WITH X
* !  !  !
* !  !  !  IF  C=1 THEN
* !  !  !  !
* !  !  !  !   D1.L:=D1.L+1
* !  !  !  !   D0.L:=D0.L-DIVISOR
* !  !  !  ELSE
```

```
                        * !   !   !   !
                        * !   !   !   !   IF D0.L >= DIVISOR THEN
                        * !   !   !   !   !
                        * !   !   !   !   !  D1.L:=D1.L+1
                        * !   !   !   !   !  D0.L:=D0.L-DIVISOR
                        * !   !   !   !   !
                        * !   !   !   !   ENDIF
                        * !   !   !  ENDIF
                        * !   !   !
                        * !   !   !  COUNTER:=COUNTER-1
                        * !   !  ENDDO
                        * !   !
                        * !   !  REMAINDER :=D0.L
                        * !   !  QUOTIENT  :=D1.L
                        * !   !
                        * !  ENDIF
                        * ENDPROCEDURE
                        *


                        *
                        * MACRO
                        * -----
                        SHIFT64  MACRO
                        \@       LSL.L    #1,\1
                                 ROXL.L   #1,\2
                                 ENDM
                        *
                        * INDEX TABLE
                        * -----------
                        *
       00000000         OLDFP    EQU      0                   ;OLD FRAME POINTER
       FFFFFFFE         COUNTER  EQU      OLDFP-2             ;COUNTER
       00000008         DVSR     EQU      OLDFP+8             ;DIVISOR (INPUT)
       0000000C         MSBDVD   EQU      OLDFP+12            ;MSB DIVIDEND (INPUT)
       00000010         LSBDVD   EQU      OLDFP+16            ;LSB DIVIDEND (INPUT)
       00000018         STATUS   EQU      OLDFP+24            ;STATUS
       0000001A         REMD     EQU      OLDFP+26            ;REMAINDER (OUTPUT)
       0000001E         QUOT     EQU      OLDFP+30            ;QUOTIENT  (OUTPUT)
       0001E178         OUTMES   EQU      $1E178              ; OUTPUT STRING OF CHARS

                        *
                        * D1.L = LSB DIVIDEND
                        * D0.L = MSB DIVIDEND
                        * A6.L = FRAME POINTER
                        * A7.L = STACK POINTER
                        *

       00002000                  RORG     $2000

002000 4FFA0196                  LEA.L    STACK,A7            ; INITIALISE POINTER SSP
002004 0FFCFFFFFFF2              ADD.L    #-14,A7             ; RESERVE SPACE FOR OUTPUT PARAMT
                        *
00200A 2F3A0082                  MOVE.L   LSBDIVIDEND,-(A7)   ;
00200E 2F3A007A                  MOVE.L   MSBDIVIDEND,-(A7)   ;
002012 2F3A007E                  MOVE.L   DIVISOR,-(A7)       ;
002016 6122                      BSR.S    DIVISION            ; PROCEDURE DIVISION
                        *
```

```
002018 4A5F                       TST.W    (A7)+                ; IF STATUS <> 0 THEN
00201A 6610                       BNE.S    ERROR                ; ! /* WRITE ERROR */
00201C 41FA007C                   LEA.L    MEMREMAINDER,A0      ;
002020 209F                       MOVE.L   (A7)+,(A0)           ;
002022 41FA0072                   LEA.L    MEMQUOTIENT,A0       ;
002026 209F                       MOVE.L   (A7)+,(A0)           ;
002028 4E41             RETURN    TRAP     #1                   ; RETURN EUROMAK 68000
00202A 0000                       DC.W     0
                        *
00202C 41FA0070         ERROR     LEA.L    STRINGERROR,A0
002030 4EB90001E178               JSR      OUTMES
002036 508F                       ADDQ.L   #8,A7
002038 60EE                       BRA.S    RETURN
                        *
                        * PROCEDURE DIVISION
                        * ==================

00203A 4E56FFFE         DIVISION LINK      A6,#-2               ;
00203E 426E0018                   CLR.W    STATUS(A6)           ;
                        *
002042 202E000C                   MOVE.L   MSBDVD(A6),D0        ; D0.L:=MSB DIVIDEND
002046 222E0010                   MOVE.L   LSBDVD(A6),D1        ; D1.L:=LSB DIVIDEND
00204A B0AE0008                   CMP.L    DVSR(A6),D0          ;
00204E 6506                       BCS.S    OK                   ;
002050 526E0018                   ADDQ.W   #1,STATUS(A6)        ; STATUS:=1
002054 6026                       BRA.S    EXIT                 ;
                        *
002056 3D7C0020FFFE     OK        MOVE.W   #32,COUNTER(A6)      ;

                                  SHIFT64  D1,D0
00205C E389             @001      LSL.L    #1,D1
00205E E390                       ROXL.L   #1,D0

002060 6506                       BCS.S    INCR1                ;
002062 B0AE0008                   CMP.L    DVSR(A6),D0          ;
002066 6506                       BCS.S    LOOP1                ;
002068 5201             INCR1     ADDQ.B   #1,D1                ;
00206A 90AE0008                   SUB.L    DVSR(A6),D0          ;
00206E 536EFFFE         LOOP1     SUBQ.W   #1,COUNTER(A6)       ;
002072 6EE8                       BGT.S    @001

                        *
002074 2D40001A                   MOVE.L   D0,REMD(A6)          ;
002078 2D41001E                   MOVE.L   D1,QUOT(A6)          ;
00207C 4E5E             EXIT      UNLK     A6
                        *
00207E 2F570010                   MOVE.L   (A7),16(A7)          ;
002082 DFFC00000010               ADD.L    #16,A7               ;
002088 4E75                       RTS


00208A 00000004         MSBDIVIDEND DS.L 1
00208E 00000004         LSBDIVIDEND DS.L 1
002092 00000004         DIVISOR   DS.L     1
002096 00000004         MEMQUOTIENT DS.L 1
00209A 00000004         MEMREMAINDER DS.L 1
00209E 0A               STRINGERROR DC.B $0A,$00
0020A0 20                         DC.B     '        DIVISION BY 0 OR QUOTIENT TOO BIG !! '
```

```
1540 0020CD 0A                    DC.B    $0A,$00,4
1550 0020D0 000000C8              DS.L    50
1560        00002198     STACK    EQU     *
1570                              END

****** TOTAL ERRORS   0--   0


SYMBOL TABLE - APPROXIMATELY  486 SYMBOL ENTRIES LEFT
```

Simulation

```
             *
           > * SIMULATION DIVISION
           > * ========================
           > *
           > * 208A.L= MSB DIVIDEND
           > * 208E.L= LSB DIVIDEND
           > * 2092.L= DIVISOR
           > * 2096.L= QUOTIENT
           > * 209A.L= REMAINDER
           > *
           > *
           > 208A.L
           * 00208A > 00000000
           * 00208E > FFFFFFFF
           * 002092 > 00000002
           * 002096 > 00000000
           * 00209A > 00000000
           * 00209E >
           > 2000;G
           > *
           > * DISPLAY RESULT
           > * ----------------
           > *
           > PRNT.L BEG>208A  END>209A
           * 002080 0010DFFC 00000010 4E750000 0000FFFF
           * 002090 FFFF0000 00027FFF FFFF0000 00010A0D
           > *
           >
           >
           > * DIVISION BY 0 !!
           > * ===================
           > *
           > 208A.L
           * 00208A > 00000000
           * 00208E > AAAAAAAA
           * 002092 > 00000000
           * 002096 >
           > 2000;G
                 DIVISION BY 0 OR QUOTIENT TOO BIG !!

           > *
           > *
           > * QUOTIENT TOO BIG !!
           > * ===================
           > *
           > 208A.L
           * 00208A > 10000000
           * 00208E > 00000000
           * 002092 > 00000002
           * 002096 >
           > 2000;G
                 DIVISION BY 0 OR QUOTIENT TOO BIG !!
```

```
> *
> *
> 208A.L
* 00208A > 00000001
* 00208E > 00000000
* 002092 > 00000002
* 002096 >
> 2000;G
> PRNT.L BEG>208A  END>209A
* 002080 0010DFFC 00000010 4E750000 00010000
* 002090 00000000 00028000 00000000 00000A0D
> *
```

TAS (Test and Set) INSTRUCTION

1 General Aspects of Semaphores

Definition

When several processors (hard or soft) use one and the same resource (printer, working memory, etc), we shall speak of a shareable resource. It is vital, in order to ensure synchronisation between the processors and the resource, to assign a semaphore register (register or memory byte) to the resource. (For example, the peripheral circuit 6809/68000 IPC MC 68121 has six semaphore registers.)

A semaphore register is made up of one or more flags allocated to the same resource which inform the programmer about the availability of the resource (SEM bit) and about the arbitration between several processors (bit ownership in the case of the MC 68121).

Generally, when the SEM bit is at 1 it indicates that the resource assigned to this semaphore is occupied, while SEM bit at 0 shows that it is available.

Method of Operation

The method of operation is very simple. If in a multiprocessor environment a particular processor wishes to use a shareable resource, it must first establish that it is free, by reading the semaphore register; it then modifies the SEM bit to reserve the resource for itself (that is, if it is available) and writes the SEM bit in the semaphore register.

When the processor has finished using the resource, it makes it available again by resetting the SEM bit to 0.

Example

Two processors, P1 and P2, share the same resource (a peripheral connected to a printer) within a multiprocessor system.

Let us assume that processor P1 wishes to access the shareable resource first;in order to do this it reads the semaphore register allocated to this resource and tests the SEM bit.

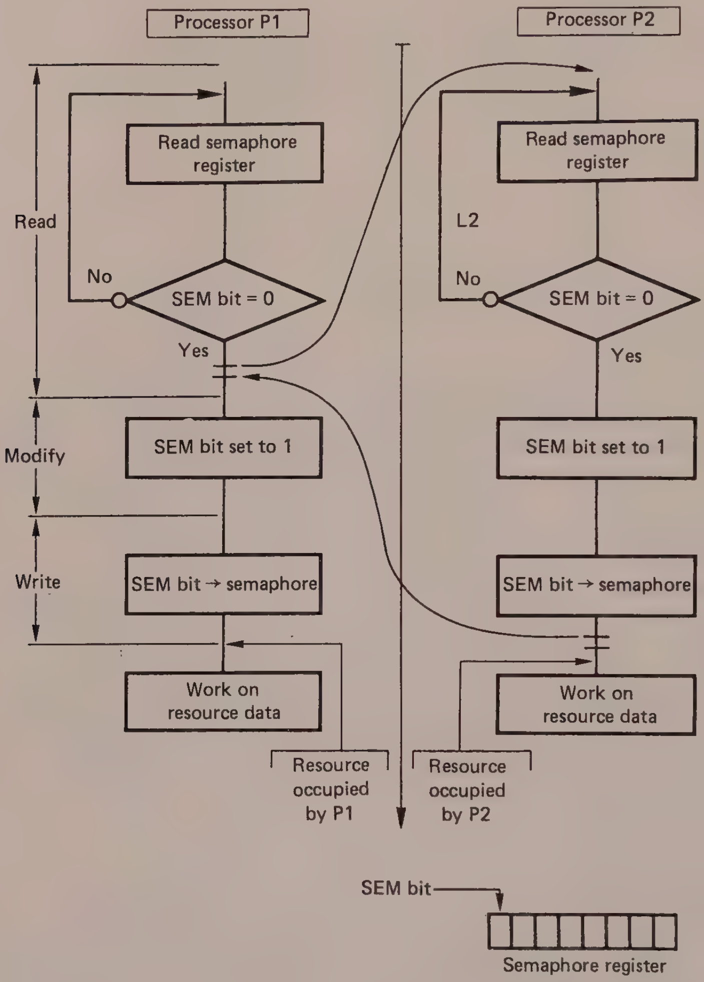


Figure 6.33

(Note that in the case of the TAS instruction branch L1, L2 does not exist. If the resource is occupied (SEM = 1) the processor continues with the program.)

Bit SEM set to 0 indicates that the the resource is available, as processor P1 finds.

Processor P2 can only assume control of the bus after it has set SEM to 1 (SEM = 1 indicating that the resource is occupied).

When processor P2 also wishes to access the resource, it reads the semaphore register and, after a test, confirms that the resource is available. Next, processor P2 sets the SEM bit to 1 in the semaphore register, thus surrendering the resource that it was occupying.

However, before using the resource, processor P1 reassumes control of the bus. Processor P1 continues execution of its program by setting SEM to 1 (we should not forget that the bus transfer took place before SEM was set to 1), before storing it in the semaphore register. The resource has therefore also been reserved by P1.

What would happen if the resource was a peripheral like the MC 68230 connected to a printer?

One can envisage catastrophic results. To conclude, this method can only be used if the read/modify/write cycle cannot be interrupted, or in other words it is indivisible.

2 Definition of TAS (Test and Set)

The TAS carries out the following during a single bus cycle

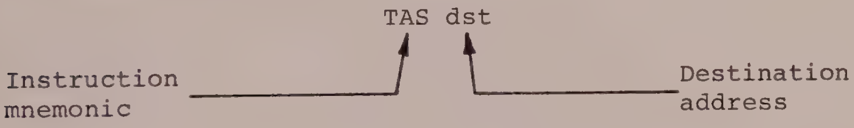
reads the destination byte

modifies the condition codes (Z and N)

writes a 1 in bit 7 of the destination byte (the other bits are unaffected).

3 Syntax of TAS

In assembler



Flowchart and Pseudo-code

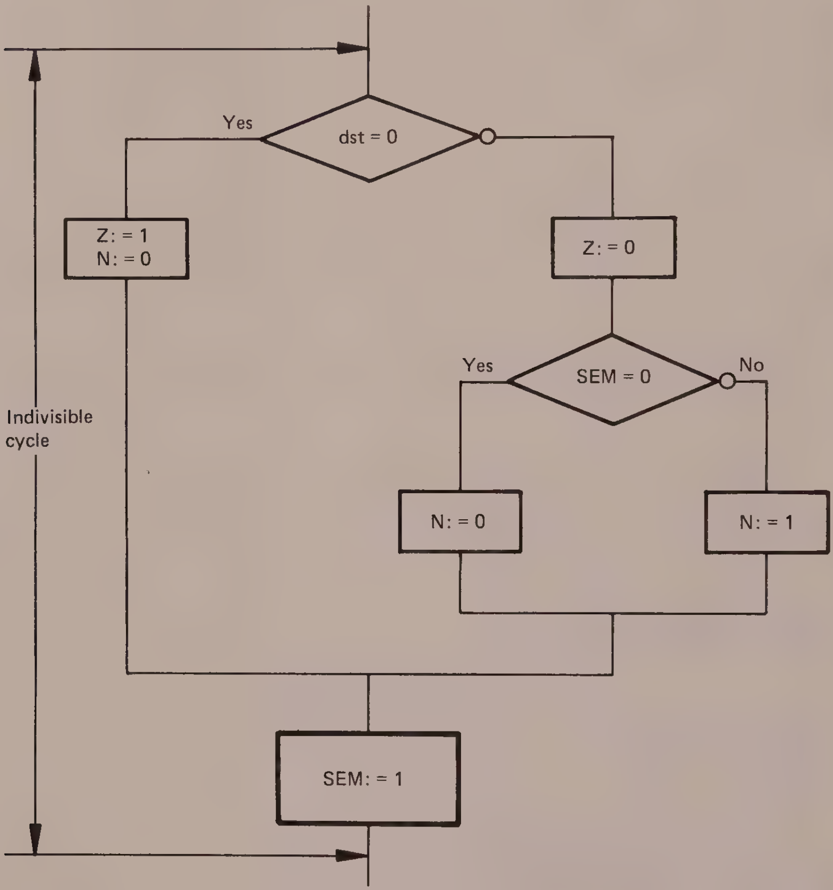
Question

What is the difference between the instruction BSET #7,dst and the instruction TAS dst?

Answer

These two instructions require the CPU to read the destination (memory byte), modify its value and write a 1 in bit 7 of the destination.

However, instruction TAS dst must be used if two processors share the same resource (dst represents the semaphore of the resource). During execution of instruction TAS dst by the CPU, the processor inhibits the data bus so that another processor cannot access dst before the two read and write bus cycles have been completed (see timing diagram of figure 6.35). This prevents two processors from reading dst simultaneously and finding the destination MSB at zero, before one of the processors had set it to 1.



```
IF      dst = 0 THEN
         Z := 1
         N := 0
ELSE
        IF SEM = 0
            N := 0
        ELSE
            N := 1
        ENDIF
ENDIF
```

Figure 6.34

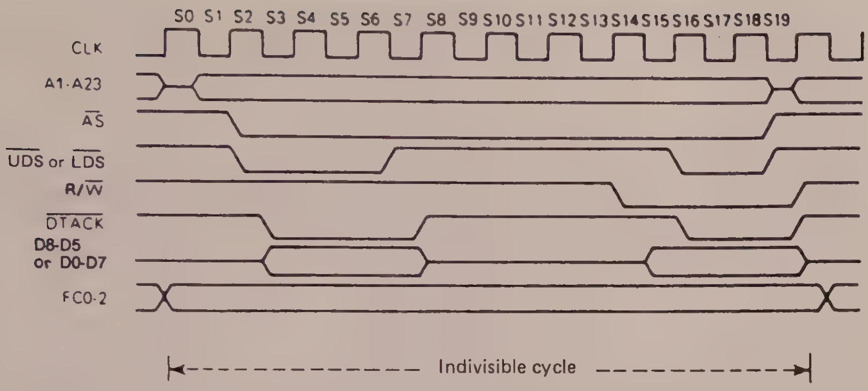
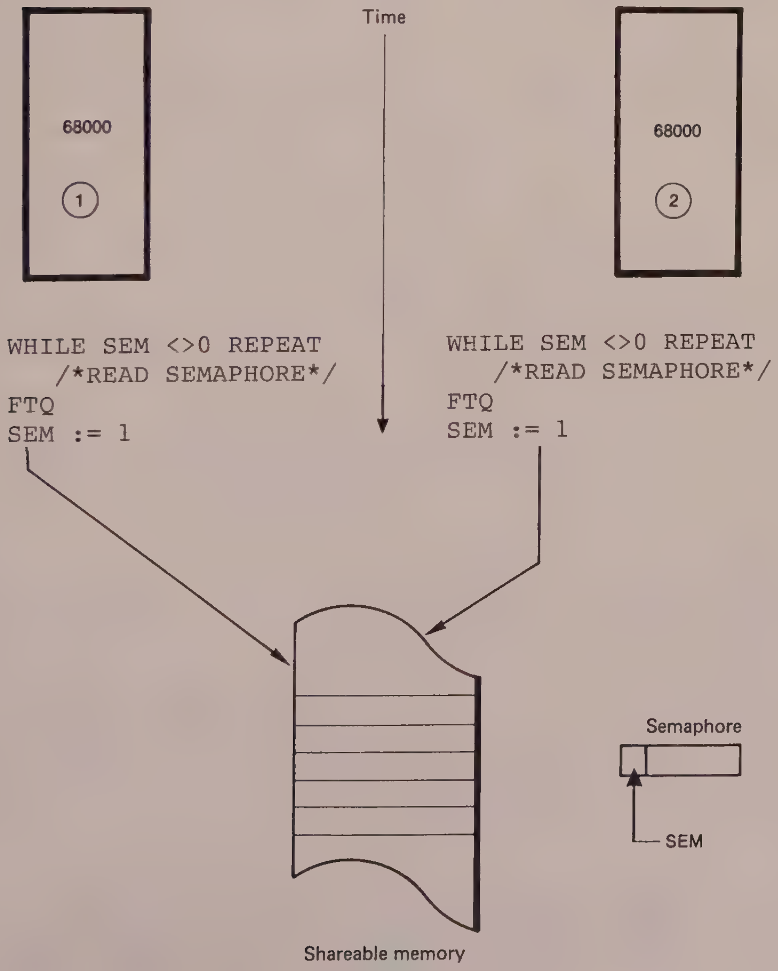

S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19
CLK
A1-A23
$\overline{AS}$
$\overline{UDS}$ or $\overline{LDS}$
R/$\overline{W}$
$\overline{DTACK}$
D8-D5 or D0-D7
FC0-2
Indivisible cycle

Figure 6.35 Read/modify/write timing diagram (Courtesy of Motorola)



```
WHILE SEM <>0 REPEAT
   /*READ SEMAPHORE*/
FTQ
SEM := 1
```

```
WHILE SEM <>0 REPEAT
   /*READ SEMAPHORE*/
FTQ
SEM := 1
```

Figure 6.36 Synchronisation of two 68000 processors sharing one memory

7 Programming Exercises

This final chapter contains two longer programs that show how to obtain the best performance from the 68000. The first program deals with exceptions; the second is a dynamic memory test.

1 EXCEPTIONS

The program that follows makes use of different hardware exceptions, such as interrupts generated from a peripheral circuit of the 6800 family (PIA 6821), a bus error caused on purpose in order to examine the different sequences of event, the trace mode and if the conditions are satisfied the trapping of a spurious interrupt.

Although this exercise is educational, the programmer should find that it causes him to think about the group priorities of the different exceptions (see the program tests).

HARDWARE USED

1. A 68000 Euromak development system
2. A hard copy printer
3. A simulator connected on a PIA interface card, with call switches.

STUDY OF THE LISTING

```
                    **************************************************************************
                    *                                                                        *
                    *          E X C E P T I O N S                                           *
                    *          -------------------                                           *
                    *  - BUS ERROR                                                           *
                    *  - TRACE MODE                                                          *
                    *  - INTERRUPT AUTOVECTOR                                                *
                    *  - SPURIOUS  INTERRUPT                                                 *
                    *                                                                        *
                    *                   (SYSTEM 68000 EUROMAK MICROPROCESS                   *
                    *                                                                        *
                    **************************************************************************
                    *
                    ** EXCEPTION VECTOR
                    *


          00000008  BUSERROR EQU   2*4                       ;BUS ERROR VECTOR
          00000024  TRACEMOD EQU   9*4                       ;TRACE MODE
          00000060  SPURIOUS EQU   24*4                      ;SPURIOUS INTERRUPT
          00000068  AUTOLEV2 EQU   26*4                      ;LEVEL 2 INTERRUPT AUTO-VECTOR
```

```
       0000007C     AUTOLEV7 EQU     31*4             ;LEVEL 7 INTERRUPT AUTO-VECTOR
       0001DE01     ADDRPIA  EQU     $1DE01           ;PIA 6821 ADDRESS
       0001E178     OUTMESS  EQU     $1E178           ;OUTPUT STRING OF CHARCS


       00002000              RORG    $2000            ;POSITION INDEPENDENT CODE

002000 4FFA025C     RETURN   LEA.L   STACK,SP         ;INITIALISE STACK POINTER
                    *
002004 4DFA0054              LEA.L   INITPIA,A6        ;
002008 21CE0068              MOVE.L  A6,AUTOLEV2      ;INITIALISE LEVEL 2 AUTO-VECTOR
                    *
00200C 4DFA0066              LEA.L   ABORT,A6         ;
002010 21CE007C              MOVE.L  A6,AUTOLEV7      ;INITIALISE LEVEL 7 AUTO-VECTOR
                    *
002014 4DFA009A              LEA.L   HARDERROR,A6     ;
002018 21CE0008              MOVE.L  A6,BUSERROR      ;INITIALISE BUS ERROR VECTOR
                    *
00201C 4DFA006A              LEA.L   NONVPA,A6        ;
002020 21CE0060              MOVE.L  A6,SPURIOUS      ;INITIALISE SPURIOUS VECTOR
                    *
002024 4DFA0076              LEA.L   TRACING,A6       ;
002028 21CE0024              MOVE.L  A6,TRACEMOD      ;INITIALISE TRACE MODE
                    *
                    ** ADDRESSING PIA 6821
                    *  ------------------
                    *
                    *  *****************************************
                    *  *          !          !         !         !
                    *  *  $1DE01  !  $1DE03  !  $1DE05 !  $1DE07 !
                    *  *          !          !         !         !
                    *  *========================================
                    *  *          !          !         !         !
                    *  *  DDRA    !  DDRB    !   CRA   !   CRB   !
                    *  *   OR     !   OR     !         !         !
                    *  *  ORA     !  ORB     !         !         !
                    *  *          !          !         !         !
                    *  *****************************************
                    *
                    ** INITIALISE PIA 6821
                    *  ------------------
                    *
00202C 4DF90001DE01          LEA.L   ADDRPIA,A6           ;INITIALISE ADDRESS PIA
002032 223CFFFF0504          MOVE.L  #$FFFF0504,D1        ;A & B SIDE ALL OUTPUTS
002038 03CE0000              MOVEP.L D1,0(A6)             ;INTERRUPT CAUSED BY CA1
00203C 46FC2100              MOVE.W  #$2100,SR            ;ENABLE INTERRUPTS
                    *
002040 323C8000     LOOP     MOVE.W  #$8000,D1            ;
002044 038E0000     LOOP1    MOVEP.W D1,0(A6)             ;WRITE ORA and ORB
002048 61000088              BSR     DELAY                ;
00204C E259                  ROR.W   #1,D1                ;ROTATE RIGHT
00204E 64F4                  BCC.S   LOOP1                ;

                    *
                    ** ENABLE TRACE MODE (T:=1) ,LOAD STATUS REGISTER AND STOP
                    *  -------------------------------------------------------
                    * A TRACE EXCEPTION WILL OCCUR IF THE TRACE STATE IS ON WHEN THE STOP
                    * INSTRUCTION IS EXECUTED.
```

```
                          * IF AN INTERRUPT REQUEST ARRIVES WHOSE PRIORITY IS HIGHER THAN THE
                          * CURRENT PROCESSOR PRIORITY ,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.
                          *
                          *

002050 46FCA100                    MOVE.W  #$A100,SR              ;T:=1,S:=1,MASK LEVEL 1
002054 4E722000                    STOP    #$2000                ;
002058 60E6                        BRA.S   LOOP                   ;

                          *
                          ** PROGRAM EXCEPTION CAUSED BY LEVEL 2 INTERRUPT (CA1 OF PIA 6821)
                          * ==============================================================

00205A 48E7FFFE           INITPIA  MOVEM.L D0-D7/A0-A6,-(SP)       ;SAVE REGISTERS
00205E 41FA0080                    LEA.L   STRING1,A0             ;INITIALISE POINTER A0
002062 4EB90001E178                JSR     OUTMESS                ;SUBROUTINE MONITOR
002068 12390001DE01                MOVE.B  ADDRPIA,D1             ;DISABLE INTERRUPT PIA
00206E 4CDF7FFF                    MOVEM.L (SP)+,D0-D7/A0-A6      ;RESTORE REGISTERS
002072 4E73                        RTE
                          *
                          ** PROGRAM EXCEPTION CAUSED BY LEVEL 7 INTERRUPT
                          * ============================================


002074 48E7FFFE           ABORT    MOVEM.L D0-D7/A0-A6,-(SP)
002078 41FA008F                    LEA.L   STRING2,A0
00207C 4EB90001E178                JSR     OUTMESS
002082 4CDF7FFF                    MOVEM.L (SP)+,D0-D7/A0-A6
002086 4E73                        RTE


                          *
                          ** PROGRAM EXCEPTION CAUSED BY SPURIOUS INTERRUPT
                          * ============================================
                          * IF DURING THE INTERRUPT ACKNOWLEDGE CYCLE ,THE PIA NO DEVICES
                          * RESPOND BY ASSERTING VPA,THE PROCESSOR 68000 FETCHES THE SPURIOUS
                          * INTERRUPT VECTOR.


002088 48E7FFFE           NONVPA   MOVEM.L D0-D7/A0-A6,-(SP)
00208C 41FA009A                    LEA.L   STRING3,A0
002090 4EB90001E178                JSR     OUTMESS
002096 4CDF7FFF                    MOVEM.L (SP)+,D0-D7/A0-A6
00209A 4E73                        RTE


                          *
                          ** TRACE MODE CAUSED IF THE T BIT IS ASSERTED AT THE BEGINNING OF THE
                          * ================================================================
                          * EXECUTION OF AN INSTRUCTION.
                          * ==========================
00209C 48E7FFFE     TRACING  MOVEM.L DO-D7/AO-A6,-(SP)
0020A0 41FA00A6              LEA.L   STRING4,A0
0020A4 4EB90001E178          JSR     OUTMESS
0020AA 4CDF7FFF              MOVEM.L (SP)+,DO-D7/AO-A6
0020AE 4E73                  RTE


                    *
                    ** BUS ERROR CAUSED BY "HARDWARE PROBLEMS"
                    *  =====================================



0020B0 48E7FFFE     HARDERROR MOVEM.L DO-D7/AO-A6,-(SP)
0020B4 41FA00B8              LEA.L   STRING5,A0
0020B8 4EB90001E178          JSR     OUTMESS
0020BE 4CDF7FFF              MOVEM.L (SP)+,DO-D7/AO-A6
                    *
0020C2 47FAFF3C              LEA.L   RETURN,A3                 ;A3:= @ RETURN
0020C6 4FEF000E              LEA.L   $E(SP),SP                 ;
                    *
                    ** PEA INSTRUCTION
                    *  ===============
                    * THE EFFECTIVE ADDRESS IS COMPUTED AND PUSHED ONTO THE STACK
                    * A LOND WORD ADDRESS IS PUSHED ONTO THE STACK.

0020CA 4853                  PEA     (A3)
0020CC 4FEFFFFE              LEA.L   -2(SP),SP
0020D0 4E73                  RTE

                    *
                    ** DELAY
                    *  =====

0020D2 3F04         DELAY    MOVE.W  D4,-(SP)
0020D4 383C61A8              MOVE.W  #25000,D4
0020D8 51CCFFFE     DELAY1   DBRA    D4,DELAY1
0020DC 381F                  MOVE.W  (SP)+,D4
0020DE 4E75                  RTS



0020E0 0A           STRING1  DC.B    $A,$D
0020E2 20                    DC.B    '         LEVEL 1 INTERRUPT (PIA CA1) '
002106 0A                    DC.B    $A,$D,4
002109 0A           STRING2  DC.B    $A,$D
00210B 20                    DC.B    '         LEVEL 7 INTERRUPT '
002125 0A                    DC.B    $A,$D,4
002128 0A           STRING3  DC.B    $A,$D
00212A 20                    DC.B    '         SPURIOUS INTERRUPT '
002145 0A                    DC.B    $A,$D,4
002148 0A           STRING4  DC.B    $A,$D
00214A 20                    DC.B    '         TRACE MODE..TRACE MODE.. '
00216B 0A                    DC.B    $A,$D,4
00216E 0A           STRING5  DC.B    $A,$D
002170 20                    DC.B    '         BUS ERROR....BUS ERROR....  '
002193 0A                    DC.B    $A,$D,4
```

```
002196 000000C8          DS.L    50
       0000225E    STACK EQU     *
                         END

****** TOTAL ERRORS   0--   0


SYMBOL TABLE - APPROXIMATELY  489 SYMBOL ENTRIES LEFT

ABORT     002074 ADDRPIA   01DED1 AUTOLEV2  000068 AUTOLEV7   00007C
BUSERROR  000008 DELAY     002002 DELAY1    002008 HARDERRO   0020B0
INITPIA   00205A LOOP      002040 LOOP1     002044 NONVPA     002088
OUTMESS   01E178 RETURN    002000 SPURIOUS  000060 STACK      00225E
STRING1   0020E0 STRING2   002109 STRING3   002128 STRING4    002148
STRING5   00216E TRACEMOD  000024 TRACING   00209C
```

1 Stack and Exception Table Initialisation Module

The instruction LEA.L STACK,SP loads the effective address identified by the stack label into the SP register.

The following instructions all have the same function, namely to store the start address of the exception program in the corresponding vector.

Example
LEA.L HARDERROR,A6 loads register A6 with the address indicated by HARDERROR ($20B0) while the storing of the contents of A6 in the BUSERROR address is ensured by the instruction MOVE.L A6,BUSERROR. The label BUSERROR has been previously defined in the list of equivalences, being the address resulting from the following multiplication

$$2*4 = \$8$$

Number in base 10 of bus error vector

Address of bus error vector

2 Initialisation of PIA 6821 Circuit Module

The reader will certainly have noticed the address table of the 6821 which shows the memory allocation of the PIA.

Instruction LEA.L INITPIA,A6 load the effective address $1DFF1 into register A6.

The next two instructions, MOVE.L #$FFFF0504,D1 and MOVEP.L D1,0(A6) (already studied in the section on MOVEP) tell the processor to store, in alternate bytes and starting from address $1DFF1, the long word contained in register D1.

The PIA 6821 is therefore programmed as follows

ports A and B on output (bit 2 of CRA and CRB is at 0, after a RESET)
interrupts are authorised on CA1
access to the data registers of the PIA (ORA and ORB) via bit 2 of CRA and CRB at 1.

The next instructions present no difficulty and these will be translated by an algorithm that will need to take account of the delay subroutine.

```
Statement of algorithm
BEGIN
|   FOR D1 altering from $8000 to $0000 by right shift 1
|   |
|   |   PIA := D1
|   |   /*Under / DELAY PROGRAM*/
|   |   D4  := 25000
|   |   REPEAT
|   |   |  D4 := D4 - 1
|   |   UNTIL D4 = 0
|   ENDFOR
END
```

3 Enable Trace Mode and Halt Program Module

The privileged instruction MOVE.W #$A100,SR sets the trace T and supervisor S bits of the SR to 1; it also positions the interrupt mask at level 1.

Instruction STOP #$2000, which is also privileged (this explains why bit S is confirmed to be at 1 by the previous instruction), transfers the operand $2000 to the status register before halting execution of the program at the next instruction BRA LOOP.

Execution of the program can only be resumed after one of the following three interrupts has been handled: reinitialisation, interrupt or trace.

A trace exception occurs if bit T is asserted before execution of the STOP instruction by the processor, which is the case here (MOVE.W #$A100,SR).

Note too that it is possible to modify the mask level by the STOP instruction (see the fourth program test).

We shall not examine each exception program in detail, but will conclude with a study of the bus error module.

```
First program run
> 2000;G
       TRACE MODE..TRACE MODE..  Execution of trace procedure

       TRACE MODE..TRACE MODE..

       LEVEL 1 INTERRUPT (PIA CA1)  PIA interrupt (level 2)

       LEVEL 1 INTERRUPT (PIA CA1)
       LEVEL 1 INTERRUPT (PIA CA1)
TRACE MODE..TRACE MODE..

       LEVEL 1 INTERRUPT (PIA CA1)

       LEVEL 1 INTERRUPT (PIA CA1)

       LEVEL 1 INTERRUPT (PIA CA1)

       LEVEL 1 INTERRUPT (PIA CA1)   Level 2 interrupt interrupted
       LEVEL 7 INTERRUPT             by level 7 interrupt


       TRACE MODE..TRACE MODE..
       LEVEL 7 INTERRUPT


       LEVEL 7 INTERRUPT

       LEVEL 1 INTERRUPT (PIA CA1)

       LEVEL 7 INTERRUPT

       LEVEL 1 INTERRUPT (PIA CA1)

       LEVEL 7 INTERRUPT

       LEVEL 1 INTERRUPT (PIA CA1)

       LEVEL 7 INTERRUPT

       TRACE
       LEVEL 1 INTERRUPT (PIA CA1)

       LEVEL 1 INTERRUPT (PIA CA1)
MODE..TRACE MODE
       LEVEL 7 INTERRUPT


       LEVEL 7 INTERRUPT

       LEVEL 7 INTERRUPT
       LEVEL 1 INTERRUPT (PIA CA1)

       LEVEL 1 INTERRUPT (PIA CA1)

       LEVEL 1 INTERRUPT (PIA CA1)

       LEVEL
       LEVEL 7 INTERRUPT
 1 INTERRUPT (PIA CA1)

       LEVEL 7 INTERRUPT
```

```
Second program run
> 2000;G                                 Simulation of a crash ( the PIA
         TRACE MODE..TRACE MODE..        card will be deselected during
                                         program execution, then reselected)
         TRACE MODE..TRACE MODE..

         TRACE MODE..TRACE MODE..

         LEVEL 1 INTERRUPT (PIA CA1)  PIA interrupt level 2

         TRACE MODE..TRACE MODE..

         LEVEL 7 INTERRUPT              Abort interrupt level 7

         TRACE MODE..TRACE MODE..

         TRACE MODE..TRACE MODE..

         TRACE MODE..TRACE MODE..

         BUS ERROR....BUS ERROR....     PIA card deselected

         BUS ERROR....BUS ERROR....
                                        Crash
         BUS ERROR....BUS ERROR....

         BUS ERROR....BUS ERROR....

         BUS ERROR....BUS ERROR....

         BUS ERROR....BUS ERROR....

         BUS ERROR....BUS ERROR....

         TRACE MODE..TRACE MODE..       PIA card reselected    Restore

         TRACE MODE..TRACE MODE..

         LEVEL 1 INTERRUPT (PIA CA1)

         TRACE MODE..TRACE MODE..

         LEVEL 1 INTERRUPT (PIA CA1)

         TRACE MODE..TRACE MODE..

         LEVEL 7 INTERRUPT

         TRACE MODE..TRACE MODE..

         LEVEL 7 INTERRUPT

         LEVEL 7 INTERRUPT

         TRACE MODE..TRACE MODE..

         LEVEL 7 INTERRUPT

         LEVEL 7 INTERRUPT

         LEVEL 7 INTERRUPT

         LEVEL 7 INTERRUPT
```

```
> Third program run
> 2000;G                                                Card is not selected
        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....
        LEVEL 7 INTERRUPT


        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....

        BUS ERROR....BUS ERROR....




> Fourth program run
> 2000;G                                             Interrupt from PIA, but
        TRACE MODE..TRACE MODE..                     the 68000 does not
                                                     recognise VPA at low
        TRACE MODE..TRACE MODE..                     during the interrupt
                                                     recognition phase
        TRACE MODE..TRACE MODE..

        TRACE MODE..TRACE MODE..

        TRACE MODE..TRACE MODE..

        SPURIOUS INTERRUPT

        SPURIOUS INTERRUPT

        SPURIOUS INTERRUPT

        SPURIOUS INTERRUPT

        SPURIOUS INTERRUPT
SPURIOUS INTERRUPT

SPURIOUS INTERRUPT

SPURIOUS INTERRUPT

SPURIOUS INTERRUPT

SPURIOUS INTERRUPT

SPURIOUS INTERRUPT

SPURIOUS INTERRUPT

SPURIOUS INTERRUPT

SPURIOUS INTER
```

4 Bus Error Module

The contents of the supervisor stack after the bus error and before the first instruction of the exception program are shown in figure 7.1 (see also demonstration listing).

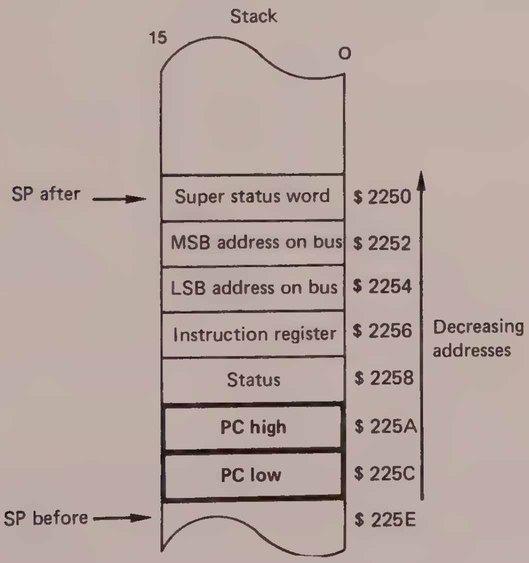


Figure 7.1

The reader will recall that, as in the case of an address error (see chapter 4 on exceptions), when a bus error occurs the contents of the PC saved on the stack may have been incremented by 2 to 10 bytes in relation to the address where the error has occurred (the above is no longer true with the MC 68020 and MC 68010).

The instruction MOVEM.L D0-D7/A0-A6, -(SP) saves the context to the supervisor stack (remembering that all exceptions are handled in supervisor mode).

LEA.L STRING5,A0 loads register A0 with the start address of the characters string

"BUS ERROR ... BUS ERROR"

Instruction JSR OUTMES calls the monitor subroutine, characters output.

We then come to instruction MOVEM.L (SP)+, D0-D7/A0-A6 which restores the context previously saved to the stack. After this instruction the stack pointer SP points to the super status word (see figure 7.1).

What happens if after instruction MOVEM.L (SP)+, D0-D7/A0-A6 the 68000 processor is ordered by RTE to return to the main program?

Certainly, if one were to believe figure 7.1 the super status word would be loaded into register SR, and the address sent along the bus at the moment of the error to the PC. The reader can imagine the problems that this would cause.

We shall leave it to the reader to continue study of the bus error exception program which, to judge from the foregoing, must be very carefully examined. In the following demonstration the program has been rerun from address $2000 in order facilitate comparison with the listing.

```
 >
 > 20B0;V     Stop point
 > 2000;G     Run command
        TRACE MODE..TRACE MODE..

        TRACE MODE..TRACE MODE..

        LEVEL 1 INTERRUPT (PIA CA1)

        LEVEL 1 INTERRUPT (PIA CA1)

        TRACE MODE..TRA
        LEVEL 1 INTERRUPT (PIA CA1)
CE MODE..

        LEVEL 7 INTERRUPT

        TRACE MODE..TRACE MODE..

        TRACE MODE..TRACE MODE..
                    Deselection of PIA card
 * VSTP  PC= 0020B0 # 48E7   S=0 S 000   C=   .....  SP= 00002250

 > ;R  PC= 0020B0 # 48E7   S=0 S 000   C=   .....  SP= 00002250
        D0= 00000000  D1= FFFF0080  D2= 00000000  D3= 00000000
        D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
        A0= 00000000  A1= 00000000  A2= 00000000  A3= 00000000
        A4= 00000000  A5= 00000000  A6= 0001DE01  A7= 00000600
```

```
> *
> ■
> 2250   0385-           Super status word
> 002252 0001--          address on bus
> 002254 DE01--
> 002256 038E--          instruction code
> 002258 2000--          status register (SR)
> 00225A 0000--
> 00225C 2046-           contents of program counter (PC)
> 00225E 4100--
> /
* TRAC   PC= 0020B4 # 41FA    S=0 S 000    C=    .....   SP= 00002214
       D0= 00000000  D1= FFFF0080  D2= 00000000  D3= 00000000
       D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
       A0= 00000000  A1= 00000000  A2= 00000000  A3= 00000000
       A4= 00000000  A5= 00000000  A6= 0001DE01  A7= 00000600

> *
> * AFTER INSTRUCTION: MOVEM.L D0--D7/A0--A6,-(SP)
> * ================================
> *
> 20C2:V
> ;P
       BUS ERROR....BUS ERROR....

* VSTP   PC= 0020C2 # 47FA    S=0 S 000    C=    ..Z..   SP= 00002250
       D0= 00000000  D1= FFFF0080  D2= 00000000  D3= 00000000
       D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
       A0= 00000000  A1= 00000000  A2= 00000000  A3= 00000000
       A4= 00000000  A5= 00000000  A6= 0001DE01  A7= 00000600

> *
> * AFTER INSTRUCTION: MOVEM.L (SP)+,D0--D7/A0--A6
> * ================================
> *
> *
> /
* TRAC   PC= 0020C6 # 4FEF    S=0 S 000    C=    ..Z..   SP= 00002250
       D0= 00000000  D1= FFFF0080  D2= 00000000  D3= 00000000
       D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
       A0= 00000000  A1= 00000000  A2= 00000000  A3= 00002000
       A4= 00000000  A5= 00000000  A6= 0001DE01  A7= 00000600

> *
> ** AFTER INSTRUCTION : LEA.L RETURN,A3
> *   ================================
> *
> /
* TRAC   PC= 0020CA # 4853    S=0 S 000    C=    ..Z..   SP= 0000225E
       D0= 00000000  D1= FFFF0080  D2= 00000000  D3= 00000000
       D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
       A0= 00000000  A1= 00000000  A2= 00000000  A3= 00002000
       A4= 00000000  A5= 00000000  A6= 0001DE01  A7= 00000600

> *
> * AFTER INSTRUCTION : LEA.L $E(SP),SP
> * ================================
> *
> /
* TRAC   PC= 0020CC # 4FEF    S=0 S 000    C=    ..Z..   SP= 0000225A
       D0= 00000000  D1= FFFF0080  D2= 00000000  D3= 00000000
       D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
       A0= 00000000  A1= 00000000  A2= 00000000  A3= 00002000
       A4= 00000000  A5= 00000000  A6= 0001DE01  A7= 00000600
```

```
> *
> * AFTER INSTRUCTION: PEA (A3)
> * ==============================
> *
> 225A  0000-   2000-
> *
> *
> /
* TRAC  PC= 0020D0 # 4E73   S=0 S 000   C=   ..Z..   SP= 00002258
      D0= 00000000  D1= FFFF0080  D2= 00000000  D3= 00000000
      D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
      A0= 00000000  A1= 00000000  A2= 00000000  A3= 00002000
      A4= 00000000  A5= 00000000  A6= 0001DE01  A7= 00000600

> *
> * AFTER INSTRUCTION: LEA.L -2(SP),SP
> * ==============================
> *
> /                    Start of program
* TRAC  PC= 002000 # 46FC   S=0 S 000   C=   .NZ..   SP= 0000225E
      D0= 00000000  D1= FFFF0080  D2= 00000000  D3= 00000000
      D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
      A0= 00000000  A1= 00000000  A2= 00000000  A3= 00002000
      A4= 00000000  A5= 00000000  A6= 0001DE01  A7= 00000600

> *
> * AFTER INSTRUCTION: RTE
> * ==============================
> /
* TRAC  PC= 002004 # 4FFA   S=0 S 111   C=   .....   SP= 0000225E
      D0= 00000000  D1= FFFF0080  D2= 00000000  D3= 00000000
      D4= 00000000  D5= 00000000  D6= 00000000  D7= 00000000
      A0= 00000000  A1= 00000000  A2= 00000000  A3= 00002000
      A4= 00000000  A5= 00000000  A6= 0001DE01  A7= 00000600

>
```

2 DYNAMIC MEMORY TEST

```
*********************************************************************
*                                                                   *
*                M E M O R Y     T E S T                            *
*                =========================                          *
*                                                                   *
*  WRITTEN BY PATRICK JAULENT : MAY 1984                            *
*  COPYRIGHT: MICROPROCESS                                          *
*                                                                   *
*********************************************************************
*
* ALGORITHM: THE PROGRAM IS COMPOSED OF 5 MODULES .THERE ARE:
* =========
*
* -1- TEST : WRITING AT THE SELECTED ADDRESS : THE OPERAND $00000000
*            AND RE-READING. IF THE RE-READING DOES NOT CORRESPOND
*            TO THE WRITING THEN "ERROR 1".
*
* -2- TEST : WALKING 1: WRITING AT THE SELECTED ADDRESS  THE OPERAND
*            $80 TO THE OPERAND $00 BY ROTATING ON PLACE TO THE RIGHT.
*             IF THERE'S A FAILURE THEN " ERROR 2 ".
*
```

```
                * -3- TEST : WRITING AT THE SELECTED ADDRESS  THE OPERAND $FFFFFFFF
                *             THERE ARE 2 SEQUENCES:
                *             ----------------------
                *                   -1- VERIFICATION IF THE READING CORRESPONDS TO
                *                       THE WRITING,IF THERE IS A FAILURE,THEN "ERROR 3".
                *
                *                   -2- VERIFICATION IF THE REST OF THE FIELD MEMORY IS
                *                       UNCHANGED IF NOT SO,THERE ARE "PROBLEMS" OF
                *                       REDUNDANCY THE "ERROR 4".
                *
                * -4- TEST : WALKING 0: WRITING AT THE SELECTED ADDRESS  THE OPERAND
                *            $7F TO THE OPERAND $00 BY ROTATING ON PLACE THE RIGHT.
                *             IF THERE IS A FAILURE THEN " ERROR 5 ".
                *
                * -5- TEST : THE LAST TEST USES  THE SOFTWARE FEATURES  OF THE
                *             "TAS" INSTRUCTION.
                *             DESCRIPTION:
                *             -----------
                *                        TEST AND SET THE BYTE OPERAND ADDRESSED BY THE
                *                        EFFECTIVE ADDRESS FIELD.THE OPERATION IS
                *                        INDIVISIBLE.
                *

                ** FUNCTION REGISTERS
                *  ==================
                *  A0 = POINTS TO STRING OF THE CHARACTERS
                *  A1 = START ADDRESS
                *  A2 = END    ADDRESS
                *  A3 = USE FOR RETURN
                *  A4 = USE IN THE REDUNDANCY TEST
                *  A5 = ADDRESS ACIA 6850
                *  A7 = STACK POINTER
                *
                *  D0 = USE SUBROUTINE TO TRANSMIT CHARACTERS
                *  D1 = COUNTER LOOPING
                *  D2 = USE SUBROUTINE "WALKING"
                *  D3 = USE SUBROUTINE OUTHEX AND ADDRTST
                *  D5 = USE SUBROUTINE DELAY
                *  D6 = MODULE IDENTIFIER
                *  D7 = USE SUBROUTINE DELAY
                *


                *
                ** EQUATES
                *  =======
      00000004  START     EQU     1*4             ;ADDRESS PROGRAM COUNTER
      00000008  BUSERROR  EQU     2*4             ;BUS ERROR VECTOR
      0000000C  ADDERROR  EQU     3*4             ;ADDRESS ERROR VECTOR
      00000018  TRAPCHEK  EQU     6*4             ;CHK INSTRUCTION
      00000060  SPURIOUS  EQU     24*4            ;SPURIOUS INTERRUPT
      00000064  IRQACIA   EQU     25*4            ;LEVEL 1 INTERRUPT AUTOVECTOR
      0000007C  ABORTIT   EQU     31*4            ;LEVEL 7 INTERRUPT AUTOVECTOR
      0001F9E9  ADDRACIA  EQU     $1F9E9          ;ADDRESS ACIA 6850

      0000000D  CR        EQU     $0D             ;CARRIAGE RETURN
      0000000A  LF        EQU     $0A             ;LINE FEED
      00000004  EOT       EQU     $4              ;END OF TRANSMIT
```

```
       00000020      SPC     EQU     $20              ;SPACE
       00000008      BS      EQU     $8               ;
       00000007      BEL     EQU     $7               ;CONTROL-G (BELL)


                     *
                     ** THE SYSTEM IS IN SUPERVISOR MODE
                     ** IF THE SYSTEM IS IN USER MODE THEN WRITE:
                     **
                     **            LEA.L BEGIN,A0
                     **            MOVE.L A0,$80    (TRAP #0)
                     **            TRAP #0
                     **            DC.W 0
                     *

                     *
                     ** INITIALISE STACK POINTER (SSP) AND EXCEPTION VECTOR
                     * =====================================================
       00002000              RORG    $2000            ;P.I.C PROGRAM

002000 4FFA08A6      BEGIN   LEA.L   STACK,A7         ;INITIALIZE SSP
002004 46FC2700              MOVE.W  #$2700,SR        ;MASQ INTERRUPT LEVEL 7
                     *
                     ** IF THERE ARE "ILLEGAL INSTRUCTIONS" THE 68000 MICROPROCESSOR  STOPS
                     * ------------------------------------------------------------------

002008 97CB                  SUB.L   A3,A3            ;A3.L:=0
00200A 47EB0010              LEA     16(A3),A3        ;A3.L:=$10
00200E 45FA02DC              LEA.L   STOPPING,A2       ;A2.L:= @ STOPPING
002012 26CA          LOOP    MOVE.L  A2,(A3)+         ;MEM[A3]:=A2;A3.L:=A3.L+4
002014 B7FC00000020          CMPA.L  #$20,A3          ;A3.L=$20 ?
00201A 66F6                  BNE.S   LOOP             ;
                     *
                     *
00201C 45FA0352              LEA.L   ERRORBUS,A2
002020 21CA0008              MOVE.L  A2,BUSERROR        ;INITIALISE BUS ERROR VECTOR
002024 21CA000C              MOVE.L  A2,ADDERROR        ;INITIALISE ADDRESS ERROR VECTOR
                     *
002028 45FA020E              LEA.L   CHECK,A2           ;
00202C 21CA0018              MOVE.L  A2,TRAPCHEK        ;INITIALISE CHK INSTRUCTION
                     *
002030 45FA0318              LEA.L   IRQ6850,A2         ;
002034 21CA0064              MOVE.L  A2,IRQACIA         ;INITIALISE LEVEL 1 AUTOVECTOR
                     *
002038 45FA0322              LEA.L   SPURIOUSIRQ,A2     ;
00203C 21CA0060              MOVE.L  A2,SPURIOUS        ;INITIALISE SPURIOUS INTERRUPT
                     *
002040 45FA0324              LEA.L   LEVEL7IRQ,A2       ;
002044 21CA007C              MOVE.L  A2,ABORTIT         ;INITIALISE LEVEL 7 AUTOVECTOR
                     *
                     ** CALLING THE SUBROUTINES (MODULES)
                     * =================================

002048 6158                  BSR.S   INITACIA           ;
00204A 6174                  BSR.S   INPUTHEX           ;
                     *
00204C 6100009E              BSR     TEST1              ;
002050 41FA067F              LEA.L   STRING1,A0         ;
```

```
002054 6100023A               BSR     PDATA            ; PRINT STRING1
002058 6100025A               BSR     INPUT1           ; INPUT CHARACTER
00205C 6100024A               BSR     OUTCH1           ; ECHO ON THE CONSOLE
002060 0C00004E               CMPI.B  #'N',D0          ; IF INPUT CHAR.= 'N' THEN
002064 6736                   BEQ.S   ENDTEST          ; ! /* END TESTS */
002066 6152                   BSR.S   INITIRQ          ; ELSE
002068 46FC2000               MOVE.W  #$2000,SR        ; ! /* ENABLE INTERRUPT */
00206C 610000A0               BSR     TEST2            ; ! /* TESTS */
002070 41FA06C9               LEA.L   STRING2,A0       ; !
002074 6100021A               BSR     PDATA            ; ENDIF
                    *
002078 610000AA               BSR     TEST3            ;
00207C 41FA06F4               LEA.L   STRING3,A0       ;
002080 6100020E               BSR     PDATA            ;
                    *
002084 610000DC               BSR     TEST4            ;
002088 41FA071F               LEA.L   STRING4,A0       ;
00208C 61000202               BSR     PDATA            ;
                    *
002090 610000E4               BSR     TEST5            ;
002094 41FA0560               LEA.L   STRING5,A0       ;
002098 610001F6               BSR     PDATA            ;
                    *
00209C 45F80004     ENDTEST   LEA.L   START,A2         ;
0020A0 4ED2                   JMP     (A2)             ;
                    *
                    ** ACIA 6850 INITIALISATION
                    *  ========================
                    *  CONTROL REGISTER = 03   /* MASTER RESET */
                    *  CONTROL REGISTER = 15  /*  8 BITS,1 STOP,DIVISION 16 */
                    *

0020A2 4BF90001F9E9 INITACIA  LEA.L   ADDRACIA,A5        ;
0020A8 1ABC0003               MOVE.B  #3,(A5)            ; MASTER RESET
0020AC 3E3C1000               MOVE.W  #$1000,D7          ;
0020B0 51CFFFFE     LOOP1     DBRA    D7,LOOP1           ; DELAY
0020B4 1ABC0015               MOVE.B  #$15,(A5)          ; 8 BITS ,1 STOP ,DIVISION 16
0020B8 4E75                   RTS     ;

                    *
                    *   CONTROL REGISTER = 95  /* ENABLE INTERRUPT GET CHARACTER */
                    *
0020BA 1ABC0095     INITIRQ   MOVE.B  #$95,(A5)          ;
0020BE 4E75                   RTS     ;

                    *
                    ** GET START AND END ADDRESSES KEYBOARD
                    *  ===================================


0020C0 41FA0312     INPUTHEX  LEA.L   ADDRBEGIN,A0       ;
0020C4 611C                   BSR.S   ADDRF              ;
0020C6 0283FFFFFFFE           ANDI.L  #$FFFFFFFE,D3      ; ODD ADDRESS SVP
0020CC 2243                   MOVE.L  D3,A1              ; A1:= START ADDRESS
                    *
0020CE 41FA037F               LEA.L   ADDREND,A0         ;
0020D2 610E                   BSR.S   ADDRF              ;
0020D4 0283FFFFFFFE           ANDI.L  #$FFFFFFFE,D3      ; ODD ADDRESS SVP
```

```
00200A 2443                 MOVE.L  D3,A2               ; A2:= END ADDRESS
                   *
00200C B5C9                 CMPA.L  A1,A2               ; IF A2.L =< A1.L THEN
00200E 6FE0                 BLE.S   INPUTHEX            ; ! /* RETURN INPUTHEX */
0020E0 4E75                 RTS     ;                     ENDIF
                   *
0020E2 610001AC    ADDRF    BSR     PDATA               ; PRINT TEXT STRING
0020E6 6100010A             BSR     INADD               ; GETS ADDRESSES
0020EA 4E75                 RTS


                   *
                   ** TEST 1 : WRITE $00000000 AND READ IF ADDRESS EQUALS $00000000
                   *  ========================================================
                   *
                   * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
                   * ------------------D
                   *                   A * *
                   *
0020EC 48E700C0    TEST1    MOVEM.L A0-A1,-(A7)         ;SAVE ALTERED REGISTERS
0020F0 7C01                 MOVEQ   #1,D6               ;IDENTIFIER MODULE 1
0020F2 B3CA        LOOPT1   CMPA.L  A2,A1               ;DO  WHILE A1 =< A2
0020F4 6212                 BHI.S   ENDT1               ;!
0020F6 4291                 CLR.L   (A1)                ;! MEM[A1]:=$00000000
0020F8 4A91                 TST.L   (A1)                ;! IF MEM[A1] <> $00000000 THEN
0020FA 6708                 BEQ.S   OKTST1              ;! ! /* A0.L:= @ TEXT ERROR1 */
0020FC 41FA0371             LEA.L   ERROR1,A0           ;! ! /* PRINT STRING1 */
002100 61000144             BSR     FAILURE             ;! !
002104 5889        OKTST1   ADDQ.L  #4,A1               ;! ENDIF
002106 60EA                 BRA.S   LOOPT1              ;!
002108 4CDF0300    ENDT1    MOVEM.L (A7)+,A0-A1         ;ENDDO
00210C 4E75                 RTS

                   *
                   ** TEST 2 :WALKING 1
                   *  =================
                   * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
                   * ------------------D    *
                   *                   A *
                   * A0.L:= ADDRESS TEXT STRING2 INPUT PARAMETER PASSED
                   * D2.B:=$80 INPUT PARAMETER PASSED
                   *

00210E 48E72080    TEST2    MOVEM.L D2/A0,-(A7)         ;SAVE ALTERED REGISTERS
002112 E30E                 LSL.B   #1,D6               ;IDENTIFIER MODULE 2
002114 41FA0397             LEA.L   ERROR2,A0           ;A0.L:= ADDRESS TEXT ERROR 2
002118 7480                 MOVEQ   #$80,D2             ;D2.B:= 10000000
00211A 61000082             BSR     WALK                ;CALLING WALK
00211E 4CDF0104             MOVEM.L (A7)+,D2/A0         ;RESTORE REGISTERS
002122 4E75                 RTS

                   *
                   ** TEST 3 :WRITE $FFFFFFFF AND READ
                   *  ================================
                   * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
                   * ------------------D
                   *                   A * *     *
                   *
```

```
002124 48E700C8          TEST3    MOVEM.L AO-A1/A4,-(A7)      ;SAVE ALTERED REGISTERS
002128 E30E                       LSL.B   #1,D6               ;IDENTIFIER MODULE 3
00212A B3CA              LOOPT3   CMPA.L  A2,A1               ;DO  WHILE A2=<A1
00212C 622E                       BHI.S   END31               ;!
00212E 2849                       MOVE.L  A1,A4               ;!
002130 588C                       ADDQ.L  #4,A4               ;! A4.L:=A1.L+4
002132 22BCFFFFFFFF               MOVE.L  #$FFFFFFFF,(A1)     ;! MEM[A1]:=$FFFFFFFF
002138 0C91FFFFFFFF               CMPI.L  #$FFFFFFFF,(A1)     ;! IF MEM[A1] <> $FFFFFFFF THEN
00213E 6708                       BEQ.S   END3                ;! ! AO.L:= ADDRESS TEXT ERROR 3
002140 41FA03AA                   LEA.L   ERROR30,AO          ;! !
002144 61000100                   BSR     FAILURE             ;! ! /* PRINT TEXT ERROR 3 */
                         *                                    ;! ENDIF
002148 B9CA              END3     CMPA.L  A2,A4               ;! DO WHILE A4=<A2
00214A 620C                       BHI.S   END32               ;! !
00214C 4A94                       TST.L   (A4)                ;! !  IF MEM[A4] <> 0 THEN
00214E 6704                       BEQ.S   OKT3                ;! !  !
002150 610000FE                   BSR     TEST31              ;! !  ! /* ADDRESS ERROR */
                         *                                    ;! !  ENDIF
002154 588C              OKT3     ADDQ.L  #4,A4               ;! !  A4.L:=A4.L+4
002156 60F0                       BRA.S   END3                ;! ENDDO
002158 5889              END32    ADDQ.L  #4,A1               ;! A1.L:=A1.L+4
00215A 60CE                       BRA.S   LOOPT3              ;ENDDO
                         *
00215C 4CDF1300          END31    MOVEM.L (A7)+,AO-A1/A4      ;RESTORE REGISTERS
002160 4E75                       RTS

                         *
                         ** TEST 4 : WALKING 0
                         *  =================
                         *
                         * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
                         * ---------------- D     *
                         *                  A *
                         *
                         * AO.L:= ADDRESS TEXT STRING4,INPUT PARAMETER PASSED
                         * D2.B:= $7F INPUT PARAMETER PASSED
                         *

002162 48E72080          TEST4    MOVEM.L D2/AO,-(A7)         ;SAVE ALTERED REGISTERS
002166 E30E                       LSL.B   #1,D6               ;INDENTIFIER MODULE 4
002168 41FA03E2                   LEA.L   ERROR4,AO           ;AO.L:= ADDRESS TEXT ERROR 4
00216C 747F                       MOVEQ   #$7F,D2             ;D2.B:=01111111
00216E 612E                       BSR.S   WALK                ;CALLING SUBROUTINE WALK
002170 4CDF0104                   MOVEM.L (A7)+,D2/AO         ;RESTORE REGISTERS
002174 4E75                       RTS
                         *
                         ** TEST 5 : READ/WRITE (QUICK)
                         *  ========================
                         *
                         * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
                         * ----------------D
                         *                 A * *
                         *

002176 48E700C0          TEST5    MOVEM.L AO-A1,-(A7)         ;SAVE ALTERED REGISTERS
00217A E30E                       LSL.B   #1,D6               ;IDENTIFIER MODULE 5
00217C B3CA              LOOPT5   CMPA.L  A2,A1               ;DO  WHILE A1=<A2
00217E 6218                       BHI.S   END5                ;!
```

```
002180 4211                 CLR.B    (A1)                   ;! MEM[A1]:=0
002182 4AD1                 TAS.B    (A1)                   ;! R/M/W ;MEM[A1]:=$80
002184 08510007             BCHG.B   #7,(A1)                ;! MEM[A1]:=$00
002188 4A11                 TST.B    (A1)                   ;! IF MEM[A1] <> $00 THEN
00218A 6708                 BEQ.S    OKT5                   ;! ! /* ERROR 5 */
00218C 41FA03FF             LEA.L    ERROR5,A0              ;! ! A0.L:= ADDRESS TEXT ER5
002190 610000B4             BSR      FAILURE                ;! ! /* PRINT TEXT ERROR 5 */
                   *                                        ;! ENDIF
002194 5289        OKT5     ADDQ.L   #1,A1                  ;! A1.L:=A1.L+1
002196 60E4                 BRA.S    LOOPT5                 ;ENDDO
002198 4CDF0300    END5     MOVEM.L  (A7)+,A0-A1            ;RESTORE REGISTERS
00219C 4E75                 RTS
                   *
                   ** WALKING PROCEDURE
                   *  =================
                   *
                   * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
                   * ----------------- D * * *   *
                   *                   A * *
                   *
                   * D2.B:=$80 ,A0.L:= ADDRESS TEXT ERROR2
                   * OR
                   * D2.B:=$7F ,A0.L:= ADDRESS TEXT ERROR4
                   *

00219E 48E7E8C0    WALK     MOVEM.L  D0-D2/D4/A0-A1,-(A7)
0021A2 B3CA        LWALK    CMPA.L   A2,A1                  ;DO WHILE A1=<A2
0021A4 622E                 BHI.S    ENDW                   ;!
0021A6 1811                 MOVE.B   (A1),D4                ;! SAVE MEM[A1]
0021A8 7207                 MOVEQ    #8-1,D1                ;! D1.B:=07
0021AA 1282        LWALK1   MOVE.B   D2,(A1)                ;! FOR D2=N UNTIL 0 DO
0021AC B411                 CMP.B    (A1),D2                ;! ! MEM[A1]:=D2.B
0021AE 671A                 BEQ.S    OKTW                   ;! ! IF MEM[A1] <> D2 THEN
0021B0 2F08                 MOVE.L   A0,-(A7)               ;! ! ! SAVE A0.L
0021B2 610000DC             BSR      PDATA                  ;! ! ! /* PRINT TEXT ERROR */
0021B6 61000140             BSR      ADDRTEST               ;! ! ! /* ADDR.TEST ODD/EVEN *
0021BA 204E                 MOVE.L   A6,A0                  ;! ! ! A0.L:= ADDR ODD/EVEN
0021BC 610000D2             BSR      PDATA                  ;! ! ! /* PRINT ADDRESS */
0021C0 2009                 MOVE.L   A1,D0                  ;! ! ! D0:=D0-D7/D8-D15
0021C2 61000100             BSR      OUTHEX                 ;! ! ! /* PRINT D0.L */
0021C6 6112                 BSR.S    DELAY                  ;! ! ! /* DISPLAY DELAY */
0021C8 205F                 MOVE.L   (A7)+,A0               ;! ! ! RESTORE A0.L
                   *                                        ;! ! ENDIF
0021CA E21A        OKTW     ROR.B    #1,D2                  ;! !
0021CC 51C9FFDC             DBRA     D1,LWALK1              ;! ENDFOR
0021D0 12C4                 MOVE.B   D4,(A1)+               ;! RESTORE MEM[A1];A1.L:=+1
0021D2 60CE                 BRA.S    LWALK                  ;ENDDO
0021D4 4CDF0317    ENDW     MOVEM.L  (A7)+,D0-D2/D4/A0-A1
0021D8 4E75                 RTS

                   *
                   ** DELAY FOR DISPLAYING
                   *  ==================
                   *
                   * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
                   * ------------------D           *   *
                   *                   A
                   *
```

```
0021DA 48E70500     DELAY    MOVEM.L D5/D7,-(A7)              ;SAVE ALTERED REGISTERS
0021DE 7A04                  MOVEQ   #5-1,D5                  ;
0021E0 3E3CFFFF     LPT2     MOVE.W  #$FFFF,D7                ;
0021E4 51CFFFFE     LPT1     DBRA    D7,LPT1                  ;
0021E8 51CDFFF6              DBRA    D5,LPT2                  ;
0021EC 4CDF00A0              MOVEM.L (A7)+,D5/D7              ;RESTORE REGISTERS
0021F0 4E75                  RTS
                    *
                    ** GET ADDRESS OF KEYBOARD
                    * ====================
                    * D2.B:= ERROR FLAG
                    * D1.B:= COUNTER CHAR
                    * D3.L:= ADDRESS
                    * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
                    * -----------------
                    *                     D   * * *
                    *                     A

0021F2 48E76000     INADD    MOVEM.L D1-D2,-(A7)              ;SAVE REGISTERS
0021F6 4283                  CLR.L   D3                       ;D3.L:=0
0021F8 7205                  MOVEQ   #6-1,D1                  ;INITIALISE COUNTER
0021FA 6112         LOOPAD   BSR.S   INMEX                    ;
0021FC 4A02                  TST.B   D2
0021FE 6B08                  BMI.S   ENDAD                    ;
002200 E98B                  LSL.L   #4,D3                    ;
002202 D680                  ADD.L   D0,D3                    ;
002204 51C9FFF4              DBRA    D1,LOOPAD                ;
002208 4CDF0006     ENDAD    MOVEM.L (A7)+,D1-D2              ;RESTORE REGISTERS
00220C 4E75                  RTS

                    *
                    **  GET ONE HEX CHARACTER
                    * ========================
                    *  IF 0-9 OR A-F THEN CONVERTED BCD OTHERWISE D2.B:=$80 AND RETURN
                    *

00220E 610000A4     INMEX    BSR     INPUT1                   ;GET ON HEX
002212 61000094              BSR     OUTCH1                   ;ECHO
002216 0C00000D              CMPI.B  #CR,D0                   ;IF CHAR=CR THEN
00221A 6606                  BNE.S   SUITEMEX                 ;!
00221C 08C20007              BSET.B  #7,D2                    ;! D2.B:=$80
002220 6014                  BRA.S   ENDMEX                   ;! RETURN TO INADD
                    *                                         ;ELSE
002222 04000030     SUITEMEX SUBI.B  #'0',D0                  ;! ASCII->BCD CONVERSION
002226 47FAFFE6              LEA.L   INMEX,A3                 ;! A3:= à RETURN IF TRAP CHK
00222A 41BC0016              CHK     #$16,D0                  ;! IF CHAR <0 OR >9 THEN
                    *                                         ;! ! /* TRAP CHK */
                    *                                         ;! ENDIF
00222E 0C000009              CMPI.B  #9,D0                    ;! IF CHAR > 9 THEN
002232 6F02                  BLE.S   ENDMEX                   ;! ! /* AJUST */
002234 5F00                  SUBQ.B  #7,D0                    ;! ENDIF
002236 4E75         ENDMEX   RTS     ;

                    *
                    ** TRAP CHK
                    * ========

002238 7007         CHECK    MOVEQ   #BEL,D0                  ; BIP !!
```

```
00223A 616C                 BSR.S   OUTCH1                  ; OUTPUT CHAR
00223C 7008                 MOVEQ   #BS,D0                  ; B.SPACE
00223E 6168                 BSR.S   OUTCH1                  ;
002240 2F4B0002             MOVE.L  A3,2(A7)                ; STORE RETURN ADDRESS
002244 4E73                 RTE
                   *
                   ** PRINT TEXT STRING PROCEDURE
                   *  ==========================
                   *

002246 6148        FAILURE  BSR.S   PDATA                   ; OUTPUT TEXT
002248 2009                 MOVE.L  A1,D0                     ; D0.L:= ADDRESS ERROR
00224A 6178                 BSR.S   OUTHEX                    ; OUTPUT ADDRESS ERROR
00224C 618C                 BSR.S   DELAY                     ; DISPLAYED DELAY
00224E 4E75                 RTS


                   *
                   ** PRINT THE "LINE" ADDRESS ERROR
                   *  ==============================
                   * IF THERE IS A FAILURE ,THE SUBROUTINE EXECUTES AN "EXCLUSIVE OR" BETWEEN
                   * THE TWO POINTERS (A4.L & A1.L).THE RESULT WILL BE A LOGICAL 1 IN EACH
                   * BIT POSITION WHERE THERE WAS A FAILURE.
                   *
                   * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
                   * ------------------D * * * *   *
                   *                   A *
                   *


002250 48E74480    TEST31   MOVEM.L D1/D5/A0,-(A7)          ;SAVE ALTERED REGISTERS
002254 2209                 MOVE.L  A1,D1                     ;D1.L:= ADDRESS ERROR
002256 2A0C                 MOVE.L  A4,D5                     ;D5.L:= ADDRESS ERROR
002258 B385                 EOR.L   D1,D5                     ;D5.L⊕D1.L :=D5.L
00225A 41FA02CE             LEA.L   ERROR31,A0                ;A0.L:= ADDRESS TEXT ERROR
00225E 6130                 BSR.S   PDATA                     ;PRINT TEXT
002260 6106                 BSR.S   BINARYADDR                ;CALLING OUTPUT BINARY ADR
002262 4CDF0122             MOVEM.L (A7)+,A0/D1/D5            ;RESTORE REGISTERS
002266 4E75                 RTS
                   *
002268 48E7B400    BINARYADDR MOVEM.L D0/D2-D3/D5,-(A7)       ;SAVE ALTERED REGISTERS
00226C 7407                 MOVEQ   #8-1,D2                   ;NUMBER NYBBLE
00226E 7603        RBIN1    MOVEQ   #4-1,D3                   ;NUMBER BIT
002270 610000CC             BSR     SPACE                     ;
002274 103C0058    RBIN2    MOVE.B  #'X',D0                   ;D0.B:=$58
002278 E38D                 LSL.L   #1,D5                     ;
00227A 6404                 BCC.S   RBIN3                     ;IF BIT=0 THEN
00227C 103C0031             MOVE.B  #'1',D0                   ; ! /* 'X' DISPLAYED */
002280 6126        RBIN3    BSR.S   OUTCH1                    ; ELSE
002282 51CBFFF0             DBRA    D3,RBIN2                  ; ! /* '1' DISPLAYED *
002286 51CAFFE6             DBRA    D2,RBIN1                  ; ENDIF
00228A 4CDF002D             MOVEM.L (A7)+,D0/D2-D3/D5
00228E 4E75                 RTS
                   *
                   ** PRINT TEXT STRING OF  CHARACTERS
                   *  ==================================
                   *
                   * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
```

```
                  * ------------------D *
                  *                   A *

002290 48E78080   PDATA    MOVEM.L A0/D0,-(A7)          ;SAVE ALTERED REGISTERS
002294 1018       PDATA1   MOVE.B  (A0)+,D0             ; DO WHILE D0 <> EOT
002296 0C000004            CMPI.B  #EOT,D0              ; !
00229A 6704                BEQ.S   ENDPD                ; !
00229C 610A                BSR.S   OUTCH1               ; ! /* OUTPUT CHAR */
00229E 60F4                BRA.S   PDATA1               ; !
0022A0 2C48       ENDPD    MOVE.L  A0,A6                ; ENDDO
0022A2 4CDF0101            MOVEM.L (A7)+,A0/D0
0022A6 4E75                RTS
                  *
                  ** TEST ACIA 6850 READY FOR TRANSMIT
                  *  ================================
                  *
0022A8 08150001   OUTCH1   BTST.B  #1,(A5)              ; DO WHILE TDRE =0
0022AC 67FA                BEQ.S   OUTCH1               ; ! /* READ STATUS */
0022AE 1B400002            MOVE.B  D0,2(A5)             ; ENDDO
0022B2 4E75                RTS
                  *
                  ** INCHNP GETS D0 CHAR (NO PARITY)
                  *  ==================================
                  *
0022B4 08150000   INPUT1   BTST.B  #0,(A5)              ; DO WHILE RDRF =0
0022B8 67FA                BEQ.S   INPUT1               ; ! /* READ STATUS */
0022BA 102D0002            MOVE.B  2(A5),D0             ; ENDO
0022BE 0200007F            ANDI.B  #$7F,D0              ; READ CHAR AND MASQ
0022C2 4E75                RTS
                  *
                  ** OUTPUT "FAILED ADDRESS" ON THE CONSOLE
                  *  ======================================
                  * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
                  * ------------------D * *   *
                  *                   A
                  *
0022C4 48E7D000   OUTHEX   MOVEM.L D0-D1/D3,-(A7)       ;SAVE ALTERED REGISTER
0022C8 7207                MOVEQ   #8-1,D1              ;COUNTER DIGIT=7
0022CA 2600                MOVE.L  D0,D3                ;D3.L:= ADDRESS ERROR
0022CC E99B       LOOPOUT  ROL.L   #4,D3                ; REPEAT
0022CE 2003                MOVE.L  D3,D0                ; !
0022D0 0200000F            ANDI.B  #$F,D0               ; ! MASQ
0022D4 06000030            ADDI.B  #'0',D0              ; ! BCD -> ASCII CONVER.
0022D8 0C000039            CMPI.B  #'9',D0              ; ! CHECK FOR A-F
0022DC 6302                BLS.S   ASCIICH              ; !
0022DE 5E00                ADDQ.B  #7,D0                ; ! AJST. FOR HEX A-F
0022E0 61C6       ASCIICH  BSR.S   OUTCH1               ; ! OUTPUT CHAR.
0022E2 51C9FFE8            DBRA    D1,LOOPOUT           ; UNTIL D1=-1
0022E6 4CDF000B            MOVEM.L (A7)+,D0-D1/D3       ;RESTORE REGISTERS
0022EA 4E75                RTS
                  *
                  ** STOPPING 68000 MICROPROCESSOR
                  *  ============================
                  *

0022EC 41FA02E0   STOPPING LEA.L   TEXTRST,A0
0022F0 619E                BSR.S   PDATA
0022F2 4E722700   STOPPED1 STOP    #$2700
```

```
0022F6 60FA                 BRA.S   STOPPED1

                   *
                   ** TEST : ADDRESS ERROR
                   *  ====================
                   *
                   * IF ADDRESS ODD (UDS) THEN DATA BUS IS D8-D15
                   * IF ADDRESS EVEN(LDS) THEN DATA BUS IS D0-D7
                   *
                   * MODIFIES REGISTERS: 0 1 2 3 4 5 6 7
                   * -----------------D * *   *
                   *                  A
                   *

0022F8 48E7D000    ADDRTEST MOVEM.L D0-D1/D3,-(A7)         ;SAVE ALTERED REGISTERS
0022FC 1001                 MOVE.B  D1,D0                  ;SAVE ERROR POSITION
0022FE 2609                 MOVE.L  A1,D3                  ;D3.L:= ADDRESS ON ERROR
002300 08030000             BTST.B  #0,D3                  ;IF ADDRESS EVEN THEN
002304 6628                 BNE.S   ODD                    ;!
002306 5040                 ADDQ    #8,D0                  ;! /* ADJUSTMENT D8-D15 */
002308 0C000009             CMPI.B  #9,D0                  ;! IF DIGIT >=9 THEN
00230C 6F20                 BLE.S   ODD                    ;! !
00230E 0400FFFA             SUBI.B  #-6,D0                 ;! ! /* DECIMAL AJUST */
002312 1600                 MOVE.B  D0,D3                  ;! ! D3.B:=D0.B
002314 7201                 MOVEQ   #2-1,D1                ;! ! D1.B:=1
002316 E91B        EVEN     ROL.B   #4,D3                  ;! ! REPEAT
002318 2003                 MOVE.L  D3,D0                  ;! ! ! D0.L:=D3.L
00231A 0200000F             ANDI.B  #$F,D0                 ;! ! ! MASQ MSB DIGIT
00231E 06000030             ADDI.B  #'0',D0                ;! ! ! BCD->ASCII CONVERSION
002322 6184                 BSR.S   OUTCH1                 ;! ! ! OUTPUT DIGIT
002324 51C9FFF0             DBRA    D1,EVEN                ;! ! UNTIL D1=-1
002328 4CDF000B    EXITADR  MOVEM.L (A7)+,D0-D1/D3         ;! ENDIF
00232C 4E75                 RTS     ;!
00232E 610E        ODD      BSR.S   SPACE                  ;ELSE
002330 0200000F             ANDI.B  #$F,D0                 ;! /* OUTPUT SPACE */
002334 06000030             ADDI.B  #'0',D0                ;! /* MASQ MSB DIGIT */
002338 6100FF6E             BSR     OUTCH1                 ;! /* BCD=>ASCII CONV */
00233C 60EA                 BRA.S   EXITADR                ;! /* OUTPUT DIGIT *
                   *                                       ;ENDIF
                   *
                   ** PRINT SPACE
                   *  ===========
                   * MODIFY REGISTER: D0
                   *

00233E 2F00        SPACE    MOVE.L  D0,-(A7)               ;SAVE D0
002340 7020                 MOVEQ   #SPC,D0                ;
002342 6100FF64             BSR     OUTCH1                 ;
002346 201F                 MOVE.L  (A7)+,D0               ;RESTORE D0
002348 4E75                 RTS

                   *
                   ** ACIA 6850 INTERRUPT
                   *  ===================
                   *
00234A 102D0002    IRQ6850  MOVE.B  2(A5),D0               ;
00234E 41FA0303             LEA.L   TEXTIRQ,A0               ;
002352 6100FF3C             BSR     PDATA                    ;
```

```
002356 4E722100                STOP    #$2100                      ; LEVEL 1 ACIA MASQ
00235A 4E73                    RTE     ;

                     *
                     ** SPURIOUS INTERRUPT
                     *  ================
                     *
00235C 41FA0320      SPURIOUSIRQ LEA.L TEXTSPURIOUS,A0             ;
002360 6100FF2E                BSR     PDATA                       ;
002364 4E73                    RTE     ;

                     *
                     ** LEVEL 7 INTERRUPT
                     *  ===============
                     *

002366 41FA0344      LEVEL7IRQ LEA.L  TEXTLEVEL7,A0                ;
00236A 6100FF24                BSR     PDATA
00236E 4E73                    RTE

                     *
                     ** BUS ERROR AND ADDRESS ERROR PROCEDURE
                     *  ====================================
                     *
                     * CASE
                     * !
                     * ! IF BUS ERROR OR ADDRESS ERROR DURING TEST1 THEN EXECUTE TEST2
                     * ! IF BUS ERROR OR ADDRESS ERROR DURING TEST2 THEN EXECUTE TEST3
                     * ! IF BUS ERROR OR ADDRESS ERROR DURING TEST3 THEN EXECUTE TEST4
                     * ! IF BUS ERROR OR ADDRESS ERROR DURING TEST4 THEN EXECUTE TEST5
                     * ! IF BUS ERROR OR ADDRESS ERROR DURING TEST5 THEN END
                     * !
                     * OTHERWISE
                     * ! /* ERROR STOPPING 68000 MICROPROCESSOR */
                     * !
                     * ENDCASE
                     *
                     *
002370 41FA02B4      ERRORBUS LEA.L   TEXTBUS,A0                  ;
002374 6100FF1A               BSR     PDATA                       ;
002378 202F0002               MOVE.L  2(A7),D0                    ;RESTORE ADDRESS ERROR
00237C 6100FF46               BSR     OUTHEX                      ;PRINT ADDRESS ERROR
002380 6100FE58               BSR     DELAY                       ;DISPLAYED DELAY
002384 1C06                   MOVE.B  D6,D6                       ;REGISTER CCR POSITIONED
002386 6742                   BEQ.S   OTHERWISE                   ; ERROR STOPPED 68000
002388 44C6                   MOVE.B  D6,CCR                      ;CCR:=D6.B
00238A 650C                   BCS.S   MODULE1                     ;ERROR DURING TEST1
00238C 6914                   BVS.S   MODULE2                     ;ERROR DURING TEST2
00238E 671C                   BEQ.S   MODULE3                     ;ERROR DURING TEST3
002390 6B24                   BMI.S   MODULE4                     ;ERROR DURING TEST4
002392 602C                   BRA.S   MODULE5                     ;ERROR DURING TEST5
002394 508F          EXITBA   ADD.L   #8,A7                       ;NO GOOD BUT !!!
002396 4E73                   RTE     ;
                     *
002398 47FAFD74      MODULE1  LEA.L   TEST2,A3                    ;
00239C 2F4B000A               MOVE.L  A3,10(A7)                   ;RETURN EXECUTE TEST 2
0023A0 60F2                   BRA.S   EXITBA                      ;
                     *
```

```
0023A2 47FAFD80  MODULE2   LEA.L   TEST3,A3                    ;
0023A6 2F4B000A            MOVE.L  A3,10(A7)                   ;RETURN EXECUTE TEST 3
0023AA 60E8                BRA.S   EXITBA                      ;
                 *
0023AC 47FAFDB4  MODULE3   LEA.L   TEST4,A3                    ;
0023B0 2F4B000A            MOVE.L  A3,10(A7)                   ;RETURN EXECUTE TEST 4
0023B4 60DE                BRA.S   EXITBA                      ;
                 *
0023B6 47FAFDBE  MODULE4   LEA.L   TEST5,A3                    ;
0023BA 2F4B000A            MOVE.L  A3,10(A7)                   ;RETURN EXECUTE TEST 5
0023BE 60D4                BRA.S   EXITBA
                 *
0023C0 47FAFCDA  MODULE5   LEA.L   ENDTEST,A3                  ;
0023C4 2F4B000A            MOVE.L  A3,10(A7)                   ;END TEST
0023C8 60CA                BRA.S   EXITBA
                 *
0023CA 47FAFF20  OTHERWISE LEA.L   STOPPING,A3                 ;
0023CE 2F4B000A            MOVE.L  A3,10(A7)                   ;ERROR STOPPING 68000
0023D2 60C0                BRA.S   EXITBA                      ;


                 *
                 * TEXTS
                 * ====
                 *
0023D4 1B        ADDRBEGIN DC.B    $1B,$45,$1B,$68
0023D8 4D                  DC.B    'MEMORY  TESTS :(C) 1984 BY MICROPROCESS ,INC '
002405 1B                  DC.B    $1B,$69,LF,LF,LF,LF,CR
00240C 50                  DC.B    'PRESS ANY KEY TO   STOP  PROGRAM '
00242D 0A                  DC.B    LF,LF,LF,CR
002431 20                  DC.B    '         BEG    ADDRESS ---> '
00244E 04                  DC.B    EOT
00244F 0A        ADDREND   DC.B    LF,CR
002451 20                  DC.B    '         END    ADDRESS ---> '
00246E 04                  DC.B    EOT
00246F 0A        ERROR1    DC.B    LF,LF,CR
002472 20                  DC.B    '         IT''S IMPOSSIBLE TO WRITE "0"   '
002498 20                  DC.B    '           IN ====>  '
0024AC 04                  DC.B    EOT
0024AD 0A        ERROR2    DC.B    LF,CR
0024AF 20                  DC.B    '         IT''S IMPOSSIBLE TO CARRY "1" IN D'
0024D8 04                  DC.B    EOT
0024D9 20                  DC.B    '         IN ====>  '
0024EB 04                  DC.B    EOT
0024EC 0A        ERROR30   DC.B    LF,CR
0024EE 20                  DC.B    '         IT''S IMPOSSIBLE TO WRITE "1"      '
002517 20                  DC.B    '         IN ====> '
002529 04                  DC.B    EOT
00252A 0A        ERROR31   DC.B    LF,CR
00252C 20                  DC.B    '         ADDRESS''  ERRORS ====> '
002548 04                  DC.B    EOT
00254C 0A        ERROR4    DC.B    LF,CR
00254E 20                  DC.B    '         IT''S IMPOSSIBLE TO CARRY "0" IN D'
002577 04                  DC.B    EOT
002578 20                  DC.B    '           IN ====>  '
00258C 04                  DC.B    EOT
00258D 0A        ERROR5    DC.B    LF,CR
00258F 20                  DC.B    '         TIMING''  PROBLEMS             '
```

```
002586 20                    DC.B   '           IN ====>  '
0025CD 04                    DC.B   EOT
0025CE 0A          TEXTRST   DC.B   LF,CR
0025D0 20                    DC.B   '         PRESS ON RESET  TO QUIT      '
0025F5 04                    DC.B   EOT
0025F6 0A          STRING5   DC.B   LF,LF,LF,LF,CR
0025FB 20                    DC.B   '         END  MEMORY  TESTS             '
002622 0A                    DC.B   LF,LF,CR,EOT
002626 0A          TEXTBUS   DC.B   LF,CR
002628 20                    DC.B   '         BUS  ERROR  OR  ADDRESS  ERROR    '
002652 04                    DC.B   EOT
002653 0A          TEXTIRQ   DC.B   LF,LF,CR
002656 20                    DC.B   '         PRESS  ON  ABORT  TO  CONTINUE '
00267D 04                    DC.B   EOT
00267E 0A          TEXTSPURIOUS DC.B LF,CR
002680 20                    DC.B   '         SPURIOUS  INTERRUPT !!!         '
0026A9 0A                    DC.B   LF,CR,EOT
0026AC 0A          TEXTLEVEL7 DC.B  LF,CR
0026AE 20                    DC.B   '         TESTS  PROGRAM  RUNNING  '
0026CF 0A                    DC.B   LF,EOT
0026D1 0A          STRING1   DC.B   LF,LF,CR,SP,SP,SP,SP,SP,SP,SP,SP
0026DC 20                    DC.B   '         E N D    O F    T E S T  1        '
002705 0A                    DC.B   LF,LF,CR
002708 20                    DC.B   '         DO  YOU  WANT  CONTINUE  THE TEST ? (Y/N) '
00273A 04                    DC.B   EOT
00273B 0A          STRING2   DC.B   LF,LF,CR,SP,SP,SP,SP,SP,SP,SP,SP
002746 20                    DC.B   '         E N D    O F    T E S T  2        '
00276F 0A                    DC.B   LF,LF,EOT
002772 0A          STRING3   DC.B   LF,LF,CR,SP,SP,SP,SP,SP,SP,SP,SP
00277D 20                    DC.B   '         E N D    O F    T E S T  3        '
0027A6 0A                    DC.B   LF,LF,EOT
0027A9 0A          STRING4   DC.B   LF,LF,CR,SP,SP,SP,SP,SP,SP,SP,SP
0027B4 20                    DC.B   '         E N D    O F    T E S T  4        '
0027DD 0A                    DC.B   LF,LF,EOT
0027E0 000000C8              DS.L   50
       000028A8    STACK     EQU    *
                             END

****** TOTAL ERRORS   0--   0


SYMBOL TABLE - APPROXIMATELY  415 SYMBOL ENTRIES LEFT

ABORTIT    00007C ADDERROR   00000C ADDRACIA   01F9E9 ADDRBEGI   0023D4
ADDREND    00244F ADDRF      0020E2 ADDRTEST   0022F8 ASCIICH    0022E0
BEGIN      002000 BEL        000007 BINARYAD   002268 BS         000008
BUSERROR   000008 CHECK      002238 CR         00000D DELAY      0021DA
END3       002148 END31      00215C END32      002158 END5       002198
ENDAD      002208 ENDMEX     002236 ENDPD      0022A0 ENDT1      002108
ENDTEST    00209C ENDW       002104 EOT        000004 ERROR1     00246F
ERROR2     0024AD ERROR30    0024EC ERROR31    00252A ERROR4     00254C
ERROR5     00258D ERRORBUS   002370 EVEN       002316 EXITADR    002328
EXITBA     002394 FAILURE    002246 INADD      0021F2 INITACIA   0020A2
INITIRQ    0020BA INMEX      00220E INPUT1     0022B4 INPUTHEX   0020C0
IRQ6850    00234A IRQACIA    000064 LEVEL7IR   002366 LF         00000A
LOOP       002012 LOOP1      0020B0 LOOPAD     0021FA LOOPOUT    0022CC
LOOPT1     0020F2 LOOPT3     00212A LOOPT5     00217C LPT1       0021E4
LPT2       0021E0 LWALK      0021A2 LWALK1     0021AA MODULE1    002398
```

```
MODULE2   0023A2 MODULE3   0023AC MODULE4   0023B6 MODULE5   0023C0
ODD       00232E OKT3      002154 OKT5      002194 OKTST1    002104
OKTW      0021CA OTHERWIS  0023CA OUTCH1    0022A8 OUTHEX    0022C4
PDATA     002290 PDATA1    002294 RBIN1     00226E RBIN2     002274
RBIN3     002280 SPACE     00233E SPC       000020 SPURIOUS  000060
SPURIOUS  00235C STACK     0028A8 START     000004 STOPPED1  0022F2
STOPPING  0022EC STRING1   0026D1 STRING2   00273B STRING3   002772
STRING4   0027A9 STRING5   0025F6 SUITEMEX  002222 TEST1     0020EC
TEST2     00210E TEST3     002124 TEST31    002250 TEST4     002162
TEST5     002176 TEXTBUS   002626 TEXTIRQ   002653 TEXTLEVE  0026AC
TEXTRST   0025CE TEXTSPUR  00267E TRAPCHEK  000018 WALK      00219E
```

```
  MEMORY  TESTS :(C) 1984 BY MICROPROCESS ,INC



Press any key to stop program
BEGINNING  ADDRESS ----> 7800
ENDING     ADDRESS ----> 7804

E N D    O F    T E S T  1

DO  YOU  WANT  CONTINUE  THE  TEST ? (Y/N) Y
IT'S IMPOSSIBLE TO CARRY "1" IN D 7           IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 6           IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 5           IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 4           IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 2  3?       IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 1           IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 0           IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 7   odd     IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 6 addresses IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 5           IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 4           IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 2  3?       IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 1           IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 0           IN ====>  00007803

E N D    O F    T E S T  2



E N D    O F    T E S T  3



IT'S IMPOSSIBLE TO CARRY "0" IN D 3              IN =====>  00007801
IT'S IMPOSSIBLE TO CARRY "0" IN D 3  line        IN =====>  00007803
                                     fault
E N D    O F    T E S T  4




END  MEMORY  TESTS


PRESS ON RESET  TO QUIT
```

Crash occurs because a data bus line is down (odd address $\overline{LDS} = 0$; $\overline{UDS} = 1$).

```
  MEMORY  TESTS :(C) 1984 BY MICROPROCESS ,INC



 Press any key to stop program
BEGINNING  ADDRESS -----> 7800
ENDING     ADDRESS -----> 7802

E N D   O F   T E S T  1

DO  YOU  WANT  CONTINUE  THE  TEST ? (Y/N) Y
IT'S IMPOSSIBLE TO CARRY "1" IN D11          IN =====>  00007800
IT'S IMPOSSIBLE TO CARRY "1" IN D11          IN =====>  00007802
                                    line fault
E N D   O F   T E S T  2


IT'S IMPOSSIBLE TO WRITE "1"               IN =====>  00007800

PRESS  ON  ABORT  TO  CONTINUE   program stopped via keyboard
TESTS  PROGRAM  RUNNING          level 7 interrupt


E N D   O F   T E S T  3


IT'S IMPOSSIBLE TO CARRY "0" IN D15             IN =====>  00007800
IT'S IMPOSSIBLE TO CARRY "0" IN D14             IN =====>  00007800
IT'S IMPOSSIBLE TO CARRY "0" IN D13             IN =====>  00007800
IT'S IMPOSSIBLE TO CARRY "0" IN D12             IN =====>  00007800
IT'S IMPOSSIBLE TO CARRY "0" IN D10    11?      IN =====>  00007800
IT'S IMPOSSIBLE TO CARRY "0" IN D 9             IN =====>  00007800
IT'S IMPOSSIBLE TO CARRY "0" IN D 8    even     IN =====>  00007800
IT'S IMPOSSIBLE TO CARRY "0" IN D15  addresses  IN =====>  00007802
IT'S IMPOSSIBLE TO CARRY "0" IN D14             IN =====>  00007802
IT'S IMPOSSIBLE TO CARRY "0" IN D13             IN =====>  00007802
IT'S IMPOSSIBLE TO CARRY "0" IN D12             IN =====>  00007802
IT'S IMPOSSIBLE TO CARRY "0" IN D10    11?      IN =====>  00007802
IT'S IMPOSSIBLE TO CARRY "0" IN D 9             IN =====>  00007802
IT'S IMPOSSIBLE TO CARRY "0" IN D 8             IN =====>  00007802

E N D   O F   T E S T  4


TIMING   PROBLEMS                               IN =====>  00007800
TIMING   PROBLEMS                               IN =====>  00007802


END  MEMORY  TESTS


PRESS ON RESET  TO QUIT
```

Same program crash as in previous simulation, but this time at even addresses ($\overline{LDS}$ = 1; $\overline{UDS}$ = 0).

```
 MEMORY  TESTS :(C) 1984 BY MICROPROCESS ,INC i




Press any key to stop program
BEGINNING  ADDRESS ------> 7800
ENDING     ADDRESS ------> 7FFF

E N D   O F   T E S T  1

DO  YOU  WANT  CONTINUE  THE TEST ? (Y/N) Y

E N D   O F   T E S T  2


ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 XXXX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 X1XX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 XXXX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 1XXX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 11XX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 XXXX
ADDRESS'S  ERRORS =====>  XXXX XXX

PRESS  ON  ABORT  TO  CONTINUE    interrupt
TESTS  PROGRAM  RUNNING
                              XXXX XXXX XXXX XXXX XXX1 11XX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 1XXX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 X1XX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 XXXX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXXX X1XX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXXX 1XXX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXXX 11XX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXXX 11XX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXXX 1XXX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXXX X1XX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 XXXX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 X1XX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 XXXX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 1XXX
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 11XX


Crash occurs because an address line is down
```

```
 MEMORY  TESTS :(C) 1984 BY MICROPROCESS ,INC
```

```
Press any key to stop program
BEGINNING  ADDRESS -----> 7800
ENDING     ADDRESS -----> 7FFF

E N D   O F   T E S T  1

DO  YOU  WANT  CONTINUE  THE TEST ? (Y/N) Y

E N D   O F   T E S T  2


ADDRESS'S  ERRORS ====>  XXXX XXXX XXXX XXXX XXXX XXXX XXX1 XXXX
ADDRESS'S  ERRORS ====>  XXXX XXXX XXXX XXXX XXXX XXXX XX1X XXXX
ADDRESS'S  ERRORS ====>  XXXX XXXX XXXX XXXX XXXX XXXX XX11 XXXX
ADDRESS'S  ERRORS ====>  XXXX XXXX XXXX XXXX XXXX XXXX X1XX XXXX
ADDRESS'S  ERRORS ====>  XXXX XXXX XXXX XXXX XXXX XXXX X1X1 XXXX
ADDRESS'S  ERRORS ====>  XXXX XXXX XXXX XXXX XXXX XXXX X11X XXXX
ADDRESS'S  ERRORS ====>  XXXX XXXX XXXX XXXX XXXX XXXX X111 XXXX
ADDRESS'S  ERRORS ====>  XXXX XXXX XXXX XXXX XXXX XXXX 1XXX XXXX
ADDRESS'S  ERRORS ====>  XXXX XXXX XX
SPURIOUS  INTERRUPT !!!

SPURIOUS  INTERRUPT !!!

SPURIOUS  INTERRUPT !!!

SPURIOUS  INTERRUPT !!!

SPURIOUS  INTERRUPT !!!

SPURIOUS  INTERRUPT !!!

SPURIOUS  INTERRUPT !!!
```

So long as the $\overline{\text{IRQ}}$ line connected to the inputs $\overline{\text{IPL0}}$ to $\overline{\text{IPL2}}$ via a 74LS148 circuit is low, the 68000 loops in the spurious interrupt procedure.

Several address lines are down

The $\overline{\text{VPA}}$ input of the 68000 is disconnected from the decoding logic of the ACIA 6850 circuit.

Two consequences are: address errors
pressing a key causes an interrupt (IRQ ACIA 6850) leading to a spurious interrupt. In fact, recognition of the absence of the $\overline{\text{VPA}}$ signal on (autovectored) interrupt causes the 68000 to branch to the exception table at address $60 ($24 \times 4 = 96_{10} = \$60$).

```
MEMORY  TESTS :(C) 1984 BY MICROPROCESS ,INC




Press any key to stop program
BEGINNING  ADDRESS -----> 7800
ENDING     ADDRESS -----> 7804

E N D   O F   T E S T  1

DO  YOU  WANT  CONTINUE  THE TEST ? (Y/N) Y
IT'S IMPOSSIBLE TO CARRY "1" IN D 7        IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 6        IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 5        IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 4        IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 3        IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 2        IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 1        IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 0        IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "1" IN D 7        IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 6        IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 5        IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 4        IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 3        IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 2        IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 1        IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "1" IN D 0        IN ====>  00007803

E N D   O F   T E S T  2


IT'S IMPOSSIBLE TO WRITE "1"             IN ====>  00007800
ADDRESS    ERRORS ====>  XXXX XXXX XXXX XXXX XXXX XXXX XXXX X1XX
IT'S IMPOSSIBLE TO WRITE "1"             IN ====>  00007804

E N D   O F   T E S T  3


IT'S IMPOSSIBLE TO CARRY "0" IN D 7          IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "0" IN D 6          IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "0" IN D 5          IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "0" IN D 4          IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "0" IN D 3          IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "0" IN D 2          IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "0" IN D 1          IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "0" IN D 0          IN ====>  00007801
IT'S IMPOSSIBLE TO CARRY "0" IN D 7          IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "0" IN D 6          IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "0" IN D 5          IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "0" IN D 4          IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "0" IN D 3          IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "0" IN D 2          IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "0" IN D 1          IN ====>  00007803
IT'S IMPOSSIBLE TO CARRY "0" IN D 0          IN ====>  00007803

E N D   O F   T E S T  4
```

```
 .MEMORY  TESTS :(C) 1984 BY MICROPROCESS ,INC




Press any key to stop program
BEGINNING  ADDRESS -----> 7804
ENDING     ADDRESS -----> 7808

E N D    O F    T E S T  1

DO  YOU  WANT  CONTINUE  THE TEST ? (Y/N) Y
IT'S IMPOSSIBLE TO CARRY "1" IN D15          IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "1" IN D14          IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "1" IN D13          IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "1" IN D12          IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "1" IN D11          IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "1" IN D 9          IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "1" IN D 8          IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "1" IN D15          IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "1" IN D14          IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "1" IN D13          IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "1" IN D12          IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "1" IN D11          IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "1" IN D 9          IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "1" IN D 8          IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "1" IN D15          IN =====>   00007808
IT'S IMPOSSIBLE TO CARRY "1" IN D14          IN =====>   00007808
IT'S IMPOSSIBLE TO CARRY "1" IN D13          IN =====>   00007808
IT'S IMPOSSIBLE TO CARRY "1" IN D12          IN =====>   00007808
IT'S IMPOSSIBLE TO CARRY "1" IN D11          IN =====>   00007808
IT'S IMPOSSIBLE TO CARRY "1" IN D 9          IN =====>   00007808
IT'S IMPOSSIBLE TO CARRY "1" IN D 8          IN =====>   00007808

E N D    O F    T E S T  2


IT'S IMPOSSIBLE TO WRITE "1"               IN =====>   00007804
ADDRESS'S  ERRORS =====>  XXXX XXXX XXXX XXXX XXXX XXXX XXXX 11XX
IT'S IMPOSSIBLE TO WRITE "1"               IN =====>   00007808

E N D    O F    T E S T  3


IT'S IMPOSSIBLE TO CARRY "0" IN D15             IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "0" IN D14             IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "0" IN D13             IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "0" IN D12             IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "0" IN D10             IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "0" IN D 9             IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "0" IN D 8             IN =====>   00007804
IT'S IMPOSSIBLE TO CARRY "0" IN D15             IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "0" IN D14             IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "0" IN D13             IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "0" IN D12             IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "0" IN D10             IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "0" IN D 9             IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "0" IN D 8             IN =====>   00007806
IT'S IMPOSSIBLE TO CARRY "0" IN D15             IN =====>   00007808
IT'S IMPOSSIBLE TO CARRY "0" IN D14             IN =====>   00007808
IT'S IMPOSSIBLE TO CARRY "0" IN D13             IN =====>   00007808
IT'S IMPOSSIBLE TO CARRY "0" IN D12             IN =====>   00007808
IT'S IMPOSSIBLE TO CARRY "0" IN D10             IN =====>   00007808
```

```
  MEMORY  TESTS :(C) 1984 BY MICROPROCESS ,INC




 Press any key to stop program
BEGINNING  ADDRESS -----> 7FFA
ENDING     ADDRESS -----> 8000
BUS  ERROR  OR  ADDRESS  ERROR    00008000
IT'S IMPOSSIBLE TO CARRY "1" IN D11              IN ====>  00007FFE
BUS  ERROR  OR  ADDRESS  ERROR    00008000
BUS  ERROR  OR  ADDRESS  ERROR    00008000
BUS  ERROR  OR  ADDRESS  ERROR    00008000
BUS  ERROR  OR  ADDRESS  ERROR    00008000
PRESS ON RESET  TO QUIT
```

A data line is down
There is no more RAM available from address \$8000 (no $\overline{\text{DTACK}}$, therefore bus error)

Appendix 1 Memory Reference Instructions

Mnemonic	Function	Operation	Size	Assembler notation	Details
MOVE	Transfer source to destination	Src,dst	B/W/LW	MOVE.Size Src,dst	Insufficient memory to memory instructions, but the memory to register, register to memory and operand to memory modes are sufficient for most applications.
ADD	Add destination operand to source operand. Result in destination	Src,dst	B/W/LW	ADD.Size Src,dst	Impossible to ADD memory to memory.
SUB	Subtract destination operand from source operand. Result in destination	Src,dst	B/W/LW	SUB.Size Src,dst	Impossible to SUB memory to memory.
CMP	Compare destination with source	Src,dst	B/W/LW	CMP.Size Src,dst	Impossible to COMP memory to memory; dst is a Dn register.
AND	AND destination operand to source operand. Result in destination	Src,dst	B/W/LW	AND.Size Src,dst	Impossible to AND memory to memory. Src and dst cannot be an An register. Src can be immediate.
OR	Inclusive OR destination operand to source operand. Result in destination	Src,dst	B/W/LW	OR.Size Src,dst	Impossible to OR memory to memory. Src and dst cannot be an An register. Src can be immediate.
EOR	Exclusive OR destination operand to source operand. Result in destination	Src,dst	B/W/LW	EOR.Size Src,dst	Impossible to EOR memory to memory. Src and dst cannot be an An register. Src can be immediate.
CLR	Clear destination operand	dst	B/W/LW	CLR.Size dst	Destination cannot be an An register.
NEG	Two´s complement of destination operand	dst	B/W/LW	NEG.Size dst	Destination cannot be an An register.
NEGX	Two´s complement with extend bit of the destination operand	dst	B/W/LW	NEGX.Size dst	Destination cannot be an An register.
NOT	One´s complement of destination operand	dst	B/W/LW	NOT.Size dst	Destination cannot be an An register.
TST	Compare operand with zero. CCR is set	dst	B/W/LW	TST.Size dst	Destination cannot be an An register.

Appendix 2 Special Memory Reference Instructions

Mnemonic	Function	Operation	Size	Assembler notation	Details
LEA	Load effective address	Src,An	LW	LEA Src,An	Whole register is affected by the instruction. CCR is not modified
PEA	Save effective address to stack	Src	LW	PEA Src	Src-> (SP)
MOVEP	Transfer from register Dn alternate even or odd addresses to memory block	Dn,dst	W/LW	MOVEP.L Dn,d (dst) or MOVEP	After transfer the data occupies alternate bytes in memory. Special 8-bit peripherals
MOVEP	Load Dn with data from memory block of even or odd addresses	Src,Dn	W/LW	MOVEP.L d(Src),Dn or MOVEP	Load data from alternate memory byte. Special 8-bit peripherals
MOVEM	Transfer multiple registers	regs,dst	W/LW	MOVEM.L regs,dst or MOVEM	List of registers can be written: D0-D5 means that registers D0 to D5 are transferred; D0/D5 that registers D0 and D5 are transferred
MOVEM	Load multiple registers	Src,regs	W/LW	MOVEM.L src,regs or MOVEM	If effective address (Src) is postincrement, only memory to register transfer is allowed
ADDX	Add destination operand with extend bit X to source operand. Result in destination	Src,dst	B/W/LW	ADDX.Size Dn,Dnl or ADDX.Size -(An), -(Anl)	Src and dst use data register or predecrement address modes
SUBX	Subtract destination operand with extend bit X from source operand. Result in destination	Src,dst	B/W/LW	SUBX.Size Dn,Dnl or SUBX.Size -(An), -(Anl)	Src and dst use data register or predecrement address modes
ABCD	Decimal addition with carry (bit X)	Src,dst	B	ABCD Dn,Dnl or ABCD -(An) - (Anl)	Src and dst use data register or predecrement address modes
SBCD	Decimal subtraction with carry (bit X)	Src,dst	B	SBCD Dn,Dnl or SBCD -(An) - (Anl)	Src and dst use data register or predecrement address modes

Appendix 2 Special Memory Reference Instructions (continued)

Mnemonic	Function	Operation	Size	Assembler notation	Details
NBCD	Destination operand and extension bit subtracted from zero	dst	B	NBCD dst	This instruction carries out the 10´s complement if X = 0 or 9´s if X = 1
MULS	Multiply two signed 16-bit operands to give 32-bit result	Src,Dn	W	MULS Src,Dn	Destination is always a Dn register. Src cannot be an address register
MULU	Multiply two unsigned 16-bit operands to give 32-bit result	Src,Dn	W	MULU Src,Dn	Destination is always a Dn register. Src cannot be an address register
DIVS*	Divide signed 32-bit by 16-bit operand to give 32-bit result. Quotient is LSB word; remainder is MSB of result	Src,Dn	W	DIVS Src,Dn	Destination is always a Dn register. Src cannot be an address register
DIVU*	Divide unsigned 32-bit by 16-bit operand to give 32-bit result. Quotient is LSB word; remainder is MSB of result	Src,Dn	W	DIVU Src,Dn	Destination is always a Dn register. Src cannot be an address register
BSET*	Test bit specified by destination operand. After test, bit is set to 1	numb,dst	B/LW	BSET.Size # numb,dst or BSET.Size Dn,dst	numb can be contents of a Dn register or an operand (# numb)
BCLR*	Test bit specified by destination operand. After test, bit is set to 0	numb,dst	B/LW	BCLR.Size # numb,dst or BCLR.Size Dn,dst	numb can be contents of a Dn register or an operand (# numb)
BCHG*	Test bit specified by operand, change its value and write complemented value in bit Z of CCR	numb,dst	B/LW	BCHG.Size # numb,dst or BCHG.Size Dn,dst	numb can be contents of a Dn register or an operand (# numb)
BTST*	Test bit specified by destination operand. After test, bit is unmodified	numb,dst	B/LW	BTST.Size # numb,dst or BTST.Size Dn,dst	numb can be contents of a Dn register or an operand (# numb)

Appendix 2 Special Memory Reference Instructions (continued)

Mnemonic	Function	Operation	Size	Assembler notation	Details
CMPM	Memory comparison by virtual subtraction of source from destination	Src,dst	B/W/LW	CMPM.Size (An)+,(An)+	Src and dst use exclusively post-increment address mode
CHK*	Test contents of a register	Src,Dn	W	CHK Src,Dn	If content of register <0 or greater than upper bound, processor executes trap CHK
TAS*	Test and set an operand	dst	B	TAS.B dst	Tests byte operand designated by dst. If dst = 0, MSB bit of dst is set to 1 (indivisible instruction)
SWAP	Exchanges the high order bits (16-31) with the low order bits (0-15) of a Dn register	Dn	L	SWAP Dn	
EXT	Extends sign bit	Dn	W/LW	EXT Dn	If size is word,bit 7 is copied to bits 8 to 15. If size is long word, bit 15 is copied to bits 16 to 31
EXG	Exchanges contents of two registers	Xn,Xm (for Dn or An register)	LW	EXG Xn, Xm	Exchange between: data registers address registers data register/address register

*See detailed study

Appendix 3 Shift and Rotate Instructions

*See detailed study

Mnemonic	Function	Operation	Size	Assembler notation	Details
ASL*	Arithmetic shift left	Dm,Dn # cnt,Dn dst	B/W/LW	ASL.Size Dm,Dn ASL.Size # cnt,Dn ASL.W dst	Shift count contained in Dm (1 to 63). # cnt indicates total of shifts (1 to 8). Only 1 bit shift in dst
ASR*	Arithmetic shift right	Dm,Dn # cnt,Dn dst	B/W/LW	ASR.Size Dm,Dn ASR.Size # cnt,Dn ASR.W dst	Shift count contained in Dm (1 to 63). # cnt indicates total of shifts (1 to 8). Only 1 bit shift in dst
ROL*	Rotation to left	Dm,Dn # cnt,Dn dst	B/W/LW	ROL.Size Dm,Dn ROL.Size # cnt,Dn ROL.W dst	Rotation number is contained in Dm (1 to 63). # cnt indicates total of rotations (1 to 8). Only 1 bit rotation in dst
ROR*	Rotation to right	Dm,Dn # cnt,Dn dst	B/W/LW	ROR.Size Dm,Dn ROR.Size # cnt,Dn ROR.W dst	Rotation number is contained in Dm (1 to 63). # cnt indicates total of rotations (1 to 8). Only 1 bit rotation in dst
ROXL*	Rotate left with extend	Dm,Dn # cnt,Dn dst	B/W/LW	ROXL.Size Dm,Dn ROXL.Size # cnt,Dn ROXL.W dst	Same as ROL with extend bit included in rotation
ROXR*	Rotate right with extend	Dm,Dn # cnt,Dn dst	B/W/LW	ROXR.Size Dm,Dn ROXR.Size # cnt,Dn ROXR.W dst	Same as ROR with extend bit X included in rotation
LSL*	Logical shift to left	Dm,Dn # cnt,Dn dst	B/W/LW	lSL.Size Dm,Dn LSL.Size # cnt,Dn LSL.W dst	Shift count contained in Dm (1 to 63). # cnt indicates total of shifts. Only 1 bit shift in dst
LSR*	Logical shift to right	Dm,Dn # cnt,Dn dst	B/W/LW	LSR.Size Dm,Dn LSR.Size # cnt,Dn LSR.W dst	Shift count contained in DM (1 to 63). # cnt indicates total of shifts. Only 1 bit shifts in dst

Appendix 4 Program Control Instructions

Mnemonic	Function	Operation	Size	Assembler notation	Details
JMP	Unconditional jump	addr		JMP addr	
JSR	Jump to subroutine	addr		JSR addr	
RTS	Return from subroutine			RTS	
RTR	Return and restore CCR			RTR	CCR register and program counter are recovered from stack
LINK*	Link to stack	An, # displacement		Link An,# displacement	-32768 < displacement < 32767 +32767
UNLK*	Unlink from stack	An		UNLK An	
BRA	Unconditional branch	addr 16	B/W	BRA displacement	If size is byte -128 < displacement < +127 If size is word -32768 < displacement < +32767
BSR	Branch to subroutine	addr 16	B/W	BSR displacement	If size is byte -128 < displacement < +127 If size is word -32768 < displacement < +32767
Bcc	Branch if cc condition is true	addr 16	B/W	Bcc displacement	If cc condition is true PC + displacement -> PC
DBcc*	Loop(s) primitive	Dn,addr 16	W	DBcc Dn,displacement	If cc is false Dn = 1 -> Dn. If Dn<>1, then PC + displacement -> PC Else NOP
Scc*	Set byte according to condition	dst	B	Scc,B dst	If cc condition is true 1´s -> destination Else 0´s -> destination

*See detailed study

Appendix 4 Program Control Instructions (continued)

Mnemonic	Function	Operation	Size	Assembler notation	Details
MOVE	Transfers source to CCR register	Src,CCR	W	MOVE Src,CCR	Src uses all address modes except address register direct
MOVE	Transfers SR register to destination	SR,dst	W	MOVE SR,dst	dst uses all address modes except: address register direct, immediate, relative, relative to PC
OR.B	Inclusive OR between CCR and data specified by instruction	# data,CCR	B	OR.B # data,CCR	
EOR.B	Exclusive OR between CCR and data specified by instruction	# data,CCR	B	EOR.B # data,CCR	
AND.B	Logical AND between CCR and data specified by instruction	# data,CCR	B	AND.B # data,CCR	
MOVE	Transfers source to SR register	Src,SR	W	MOVE Src,SR	Src uses all address modes except address register direct. Privileged instruction
OR	Inclusive OR between SR and data specified by instruction	# data,SR	W	OR # data,SR	Privileged instruction
AND	Logical AND between Sr and data specified by instruction	# data,SR	W	AND # data,SR	Privileged instruction
EOR	Exclusive OR between SR and data specified by instruction	# data,SR	W	EOR # data,SR	Privileged instruction
MOVE	Transfers user stack pointer to An register	USP,An	LW	MOVE.L USP,An	Privileged instruction
MOVE	Transfers An register to user stack pointer	An,USP	LW	MOVE.L An,USP	Privileged instruction

Appendix 4 Program Control Instructions (continued)

Mnemonic	Function	Operation	Size	Assembler notation	Details
RTE	Return from exception			RTE	(SP) + -> SR (SP) + -> PC Privileged instruction
STOP*	Load SR register with data specified by instruction. Then stop processor	# data		STOP # data	# data -> SR, then STOP Privileged instruction
RESET	Output RESET line is set low for 124 clock cycles			RESET	Privileged instruction
NOP	No operation occurs			NOP	
TRAP	Sequence branches to vector number shown by the instruction			TRAP # number	PC -> -(SSP) SR -> -(SSP) (Vector) -> PC
TRAPV	Sequence branches if overflow indicator = 1			TRAPV	If V = 1, then PC -> -(SSP) SR -(SSP), (vector TRAPV) -> PC Else NOP

Appendix 5 PAL Devices

Advantages of Using PALs

Programmable array logic (or logic array) devices have a unique place in the world of logic design. Not only do they offer many advantages over conventional logic, such as TTL, they also provide many features not found anywhere else.

Special features of the PAL family include

Programmable replacement for conventional TTL logic.

Help to reduce IC inventories substantially and simplify their control.

Reduce chip count by at least 4 to 1.

Simplify and speed up prototyping and board layout.

Save space with 20-pin and 24-pin DIP packages.

High speed, 15 ns being a typical propagation delay.

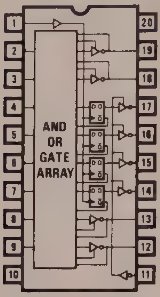
Programmed on standard PROM programmers.

Programmable tristate outputs.

Special feature eliminates the possibility of copying by competitors.

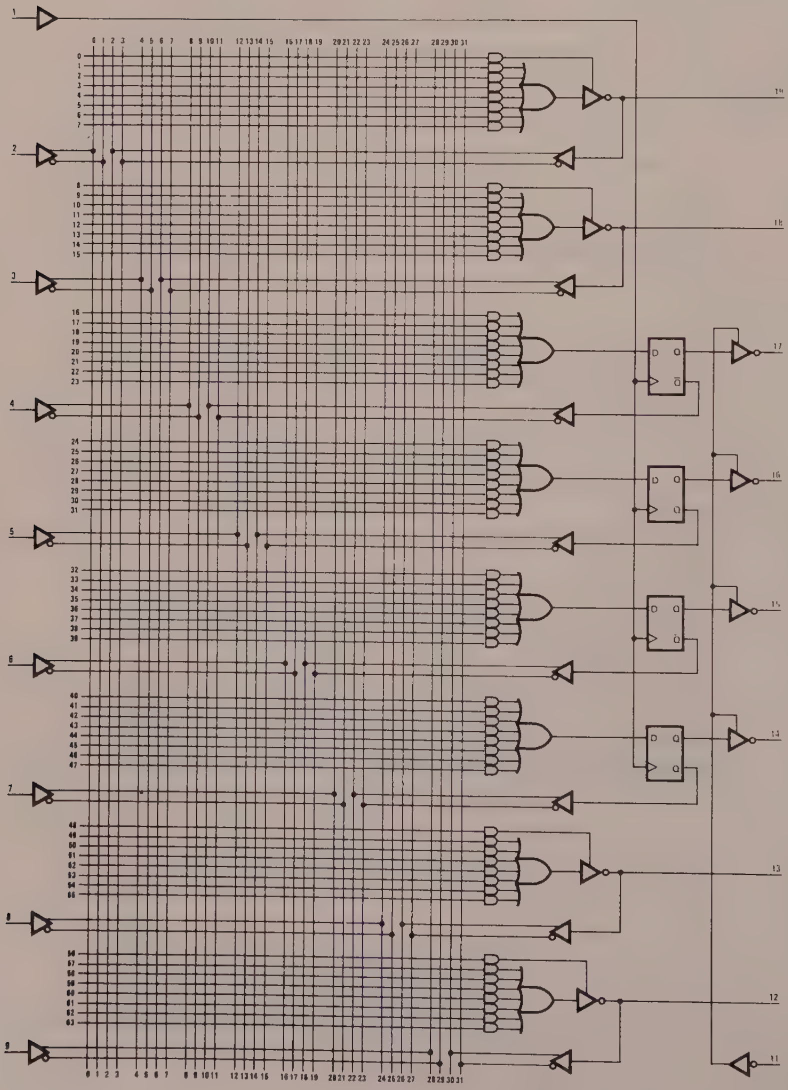
PAL® is a registered trademark of Monolithic Memories Inc.

16R4



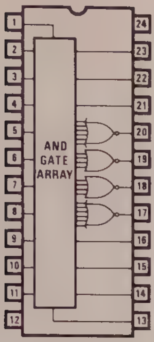
QUAD 16 INPUT REGISTERED AND-OR ARRAY

16R4



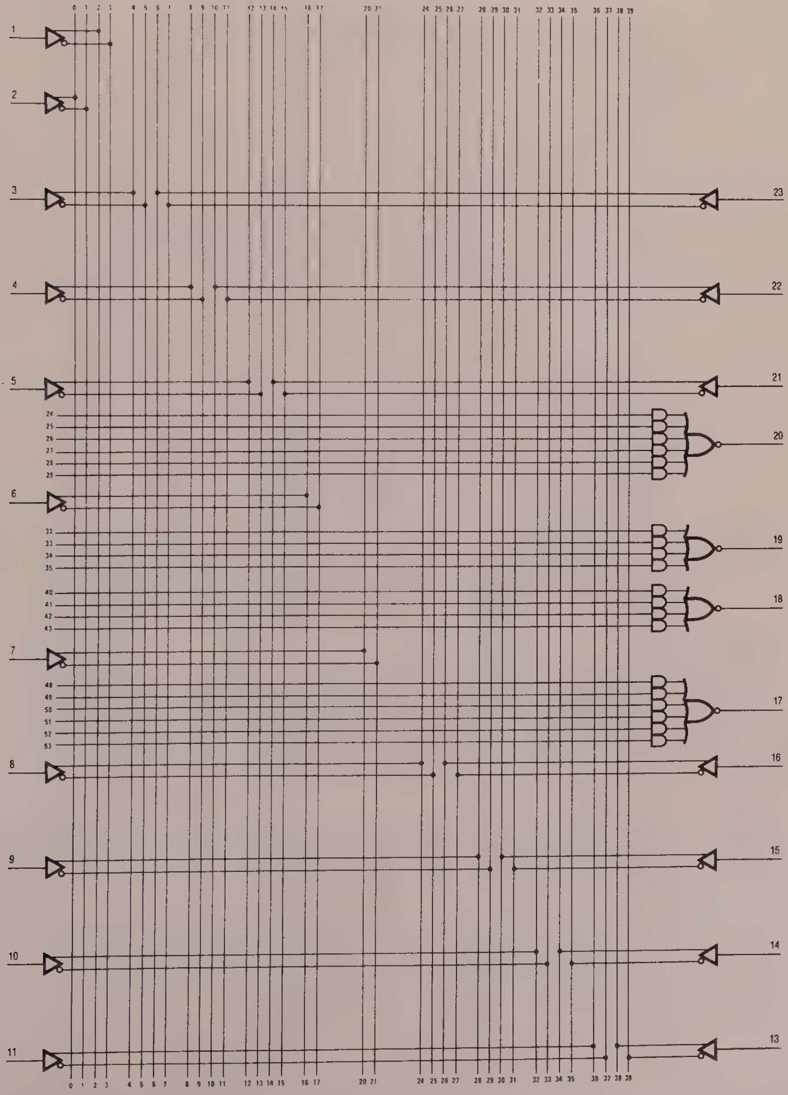
(Courtesy of Monolithic Memories)

18L4

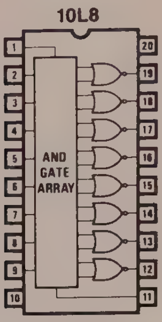


QUAD 18 INPUT
AND-OR-INVERT GATE ARRAY

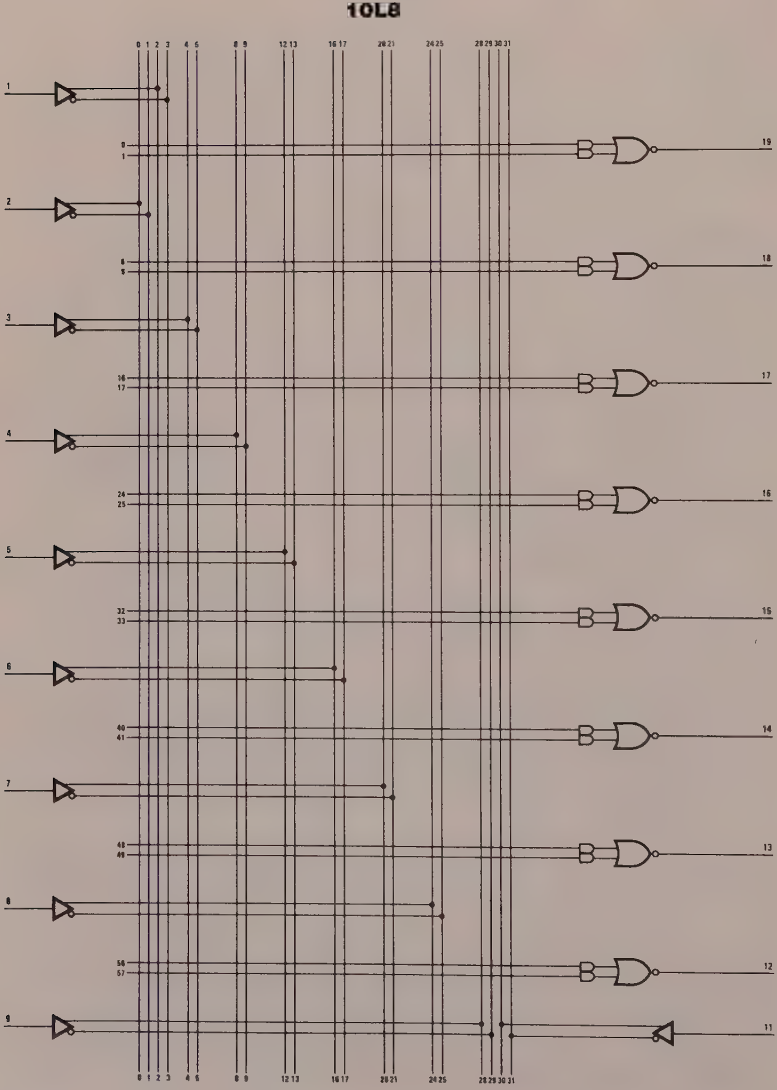
18L4



(Courtesy of Monolithic Memories)



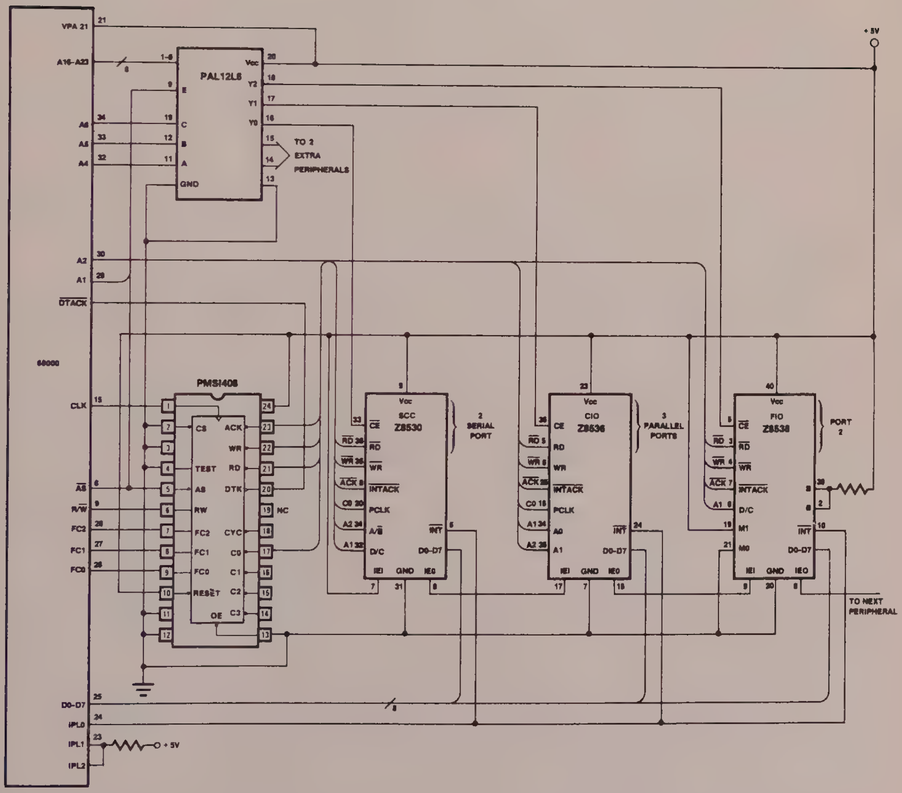
OCTAL 10 INPUT
AND-OR-INVERT GATE ARRAY



(Courtesy of Monolithic Memories)

Application

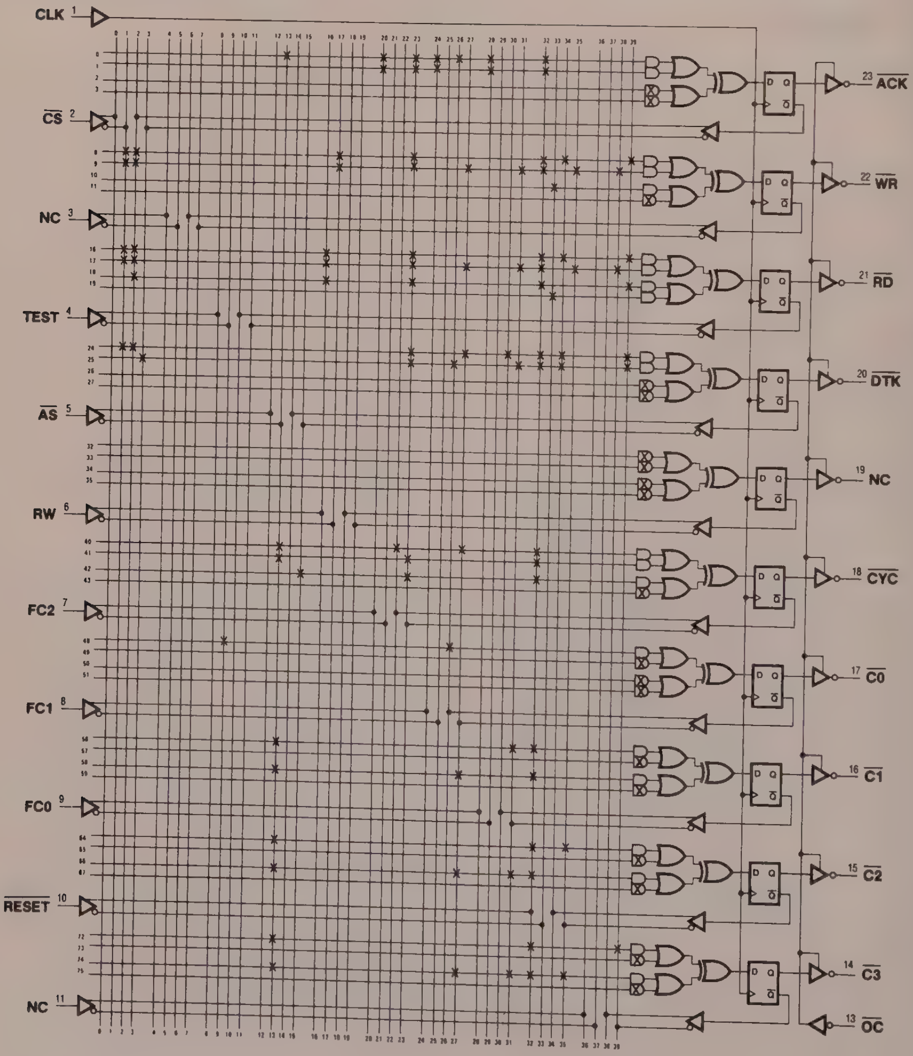
Zilog 8500 Interface with the 68000 Microprocessor



(Courtesy of Monolithic Memories)

Interface Controller for 68000 μP to Zilog 8500 Peripherals

Logic Diagram PAL20X10



(Courtesy of Monolithic Memories)

Macmillan Computer Science Series
Consulting Editor:
Professor F.H. Sumner,
University of Manchester

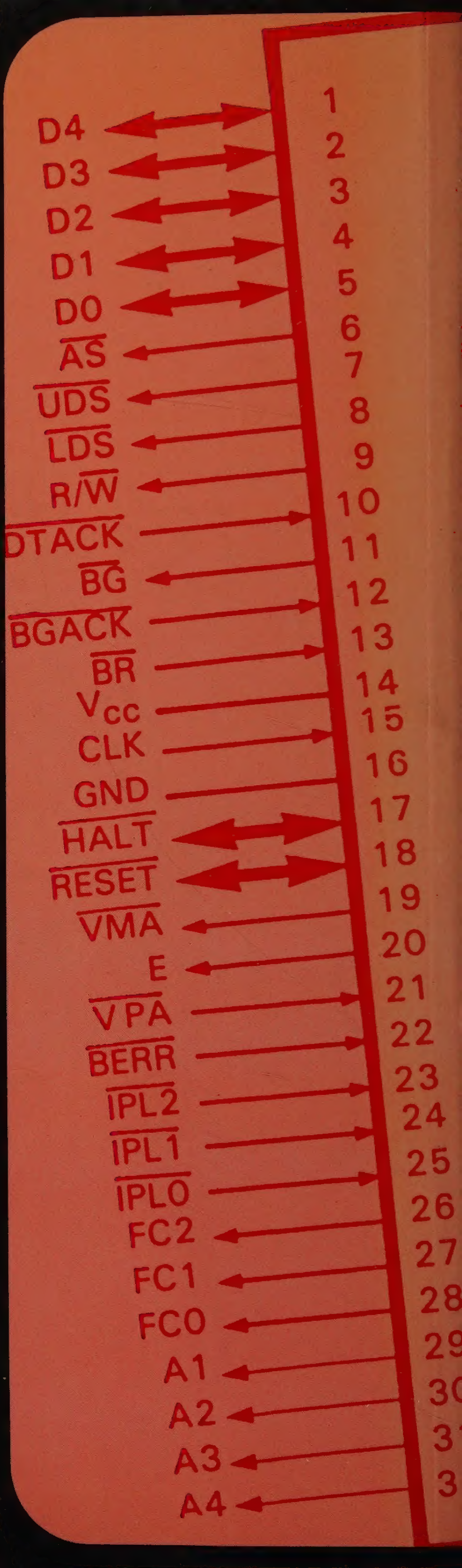
Conceived and produced by Motorola, and now manufactured as second source by several other companies, the 68000 microprocessor has emerged as one of the most useful and versatile 16-bit microprocessors. This book deals with both hardware and software aspects of the 68000 family, and the author, a practising engineer, has included many programming exercises that will appeal both to the student and to the qualified engineer.

In addition a number of examples of circuits are included which illustrate the use of the 68000 with programmable array logic devices.

Apart from the generally comprehensive coverage, the book will be of special interest to the engineer because of its detailed attention to high level instructions such as LINK, UNLK, CHK and TAS.

Patrick Jaulent is head of the microprocessors and systems training department of Microprocess, a French IT company based in Puteaux.

ISBN 0–333–38574–8


D4
D3
D2
D1
D0
AS
UDS
LDS
R/W
DTACK
BG
BGACK
BR
Vcc
CLK
GND
HALT
RESET
VMA
E
VPA
BERR
IPL2
IPL1
IPL0
FC2
FC1
FC0
A1
A2
A3
A4
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28