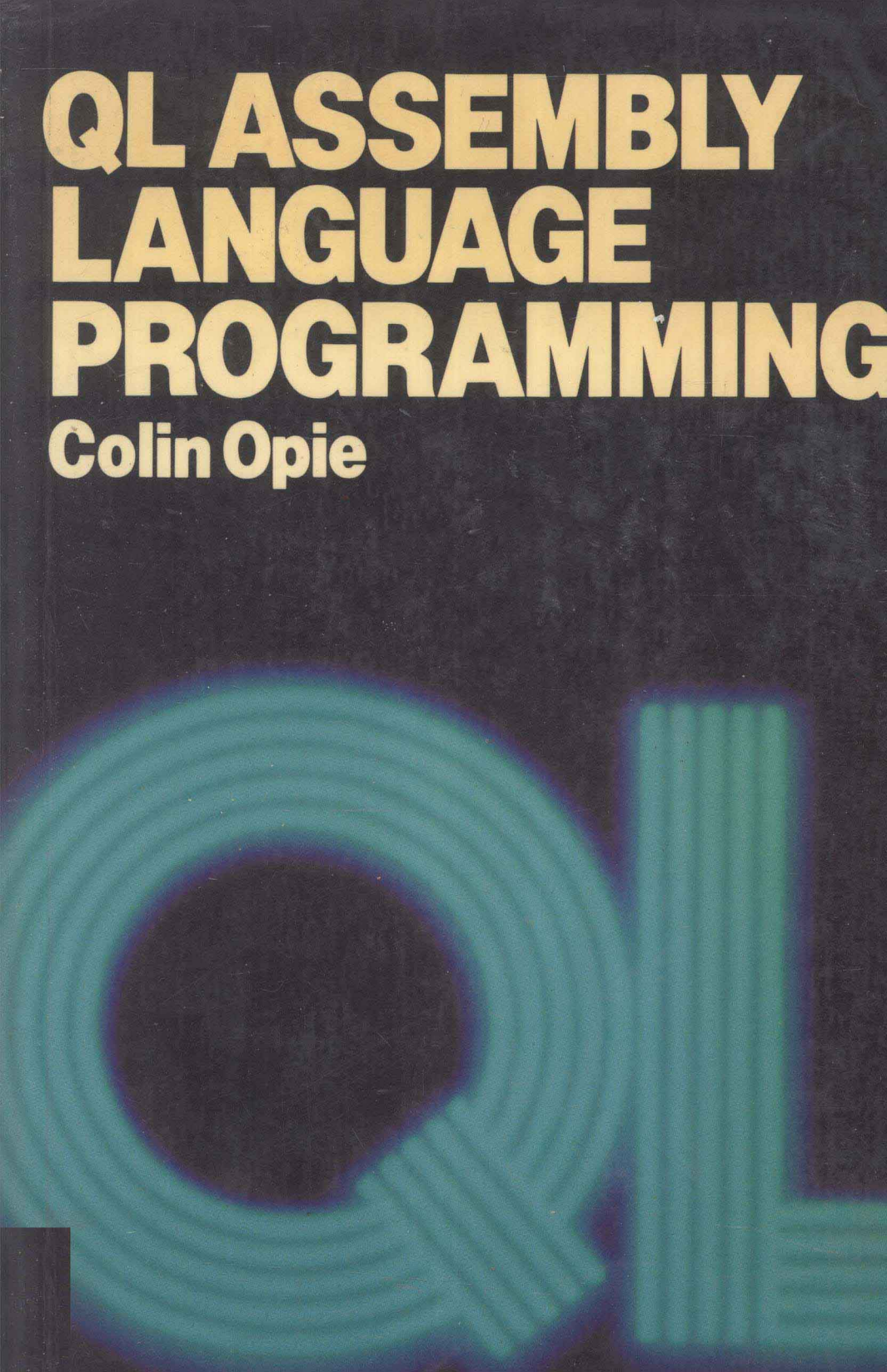
QL ASSEMBLY LANGUAGE PROGRAMMING

Colin Opie



QL Assembly Language Programming

Colin Opie

McGRAW-HILL Book Company (UK) Limited

London · New York · St Louis · San Francisco · Auckland · Bogotá
Guatemala · Hamburg · Johannesburg · Lisbon · Madrid
Mexico · Montreal · New Delhi · Panama · Paris · San Juan
São Paulo · Singapore · Sydney · Tokyo · Toronto

Published by
McGRAW-HILL Book Company (UK) Limited
MAIDENHEAD · BERKSHIRE · ENGLAND

British Library Cataloging in Publication Data

Opie, Colin
QL assembly language programming.
1. Sinclair QL (Computer) – Programming
2. Assembling (Electronic computers)
I. Title
001.64'2 QA76.8.S625

ISBN 0-07-084777-0

Library of Congress Cataloging in Publication Data

Opie, Colin
QL assembly language programming.

Includes index
1. Sinclair QL (Computer) – Programming. 2. Assembler language (Computer program language)
I. Title. II. Title: Q.L. assembly language programming.
QA 76.8.S6216065 1984 001.64'2 84-21796
ISBN 0-07-084777-0

12345 BBP 8654

PREFACE

The microelectronic evolution recently spawned a uniquely cost effective, yet inherently powerful, microcomputer system - the Sinclair QL. This microcomputer is indeed unique, and certainly a 'first' in its breed. Enclosed in the slender black light-weight case is a member of the 68000 family of microprocessors (one of the most advanced processors currently widely available). The QL provides true 32-bit processing, a suite of specially designed state-of-the-art logic arrays, 128K of RAM, a multi-tasking operating system kernel, two Microdrives for backup storage, and a range of I/O facilities including local area networking ports. What makes this system unique, apart from the actual electronics, is the fact that it costs no more, and in many cases much less, than its rival 8-bit microcomputers.

The Sinclair QL comes equipped with a powerful, and truly extensible, 'SuperBASIC'. One important feature of this extensibility is that 68000 machine code routines may be written and merged into SuperBASIC in order to enlarge the variety of commands available. Of course, there is also the option of writing whole application programs in 68000 code, hence obtaining a maximum speed advantage during the running of the package.

This book is about 68000 assembly language programming on the Sinclair QL. There are many good general texts on 68000 programming and there is little point in reproducing such material here. The emphasis is therefore: 'Assuming I have a detailed text on 68000 instructions and their operation, how can I use the Sinclair QL to gain expertise and create useful assembly language programs to run on it?' It is hoped that such an emphasis has given rise to a vital and informative book that is suitable for general programmers, industrial and educational training institutions, and also for OEM design engineers, all of whom may come to use the Sinclair QL. Even though detailed information on each of the 68000 instructions is not included, Chapters 1 and 2, and Appendix A, will go a long way to meeting most needs in this area.

Such a text as this would not be complete unless it could provide sound, practical experience of the theory presented. To this end a full screen-orientated program editor and 68000 assembler/loader package has been developed to complement this book. As will be seen from Part 4 of this book (which describes in detail the operation of the software) the package supports a full 68000 assembler development environment. The actual assembler, for example, provides features normally found only in rather more expensive minicomputer-based versions. The software is available separately on a Microdrive cartridge. Another cartridge is available which holds the source code for all the programs and major subroutines listed or referred to in the text.

My sincere thanks to John Watson, Tom Blackall, Liz Nemecek, and Jenny Wright for their help in the production of this book. Special thanks go to Tony Tebby for his invaluable technical advice and support, without which this book could not have been written. As ever I remain totally grateful to my wife whose patience and support seem ever increasing.

Colin N. Opie
July 1984

ACKNOWLEDGEMENTS

CONTENTS

INTRODUCTION

'One of the pleasantest things in the
world is going a journey; ...'

William Hazlitt

In this book we are going to embark on a journey into the operating environment of the Sinclair QL microcomputer. The excursion will hold many new experiences for most travellers, and there is much to stretch the imagination and inventiveness of everyone. The operating environment of the Sinclair QL is based upon a kernel of procedures collectively known as QDOS. In addition to QDOS there is a set of utility routines which may be entered via well-defined vectors in ROM. These QDOS procedures and general utilities provide the assembly language programmer with a wealth of support ranging from simple character output to floating point arithmetic. The main processor in the Sinclair QL is a Motorola MC68008. This new generation 16-bit processor offers extremely good architectural features. A direct, yet large, memory address space is provided together with a highly consistent instruction set. This book is about the use of this instruction set within the architecture of the Sinclair QL.

The package tour

As this text is specifically aimed at the assembly language programmer it makes sense to look at the general architecture of the 68000 processor, its addressing modes, and the operation of its instructions. These topics will be covered in Part 1. A detailed discussion of each of the 68000 instructions is not given for two reasons. First, such an inclusion would make this text unnecessarily large and expensive. Second, there are a number of suitable texts readily available (e.g., Kane,G., Hawkins,D., and Leventhal,L.: '68000 Assembly Language Programming', Osborne/McGraw-Hill, 1981). The emphasis in the appropriate chapters of this book is to provide a concise 68000 companion.

Part 2, comprising Chapters 3 to 8, describes in detail the QDOS and utility procedures. These procedures are the building blocks for the assembly language programmer's own application programs. Chapter 8 describes the options available when actually loading and running a machine code program, as well as how machine code procedures may be added to SuperBASIC in order to extend the language.

Part 3 contains four chapters of program examples. Chapters 9 and 10 give examples of stand-alone executable programs. Chapter 9 contains a number of utility programs and Chapter 10 deals with graphics. Chapters 11 and 12 give examples of programs which extend the SuperBASIC language. Chapter 11 contains some general utility procedures and

Chapter 12 concentrates on Microdrive file handling. The programs given are full implementations and useful, not simply as utilities, but also as examples of 'how to get the code loaded and executed'.

Part 4 describes the full screen editor and assembler/locator package used to create the assembly language programs given in Part 3. The programs within the package are easy to use and provide a professional approach to assembly language programming on the Sinclair QL. Finally, a number of appendices exist to provide quick reference guides for commonly required information.

Getting started

Exactly how you use this book will depend upon your current expertise in assembly language programming. It is assumed that you have a basic understanding of the techniques of assembly language programming, and are familiar with terms such as registers, addressing modes, stack pointers, and so on. If you already write a fair amount of assembly language code for some other processor (e.g., Z80 or 6502), you will be in a good position to start programming your Sinclair QL very soon. For those of you who already know 68000 assembly language, Part 2 of the book will probably be your starting point.

If you do know the assembly language of some other processor but are unfamiliar with the 68000, Chapters 1 and 2 and Appendix A will give you a good insight into what the 68000 is capable of. As mentioned previously, another suitable text will be required should you desire to look at detailed accounts of each of the 68000 instructions. Once you are happy with the overall design and the capabilities of the 68000, full use can be made of Part 2 in order to actually write, load, and execute an assembly language program.

Whetting the appetite

Assembly language programming on the QL is best performed using a proper assembler package. Programs developed in this way would normally be merged into SuperBASIC as an extension, or run as a separate machine code program by using the EXEC command (see Chapter 8). Very simple machine code programs can be loaded into memory and accessed through the CALL command, this being the approach adopted here simply to whet your appetite a little!

Figure 1 is a listing (output by the McGraw-Hill assembler described in Part 4) of an assembly language version of the SuperBASIC RECOL command. This command accepts a screen channel number followed by eight colour parameters:

```
RECOL #n, c1, c2, c3, c4, c5, c6, c7, c8
```

Each colour parameter defines the new pixel colour for the current respective colour: black, blue, red, magenta, green, cyan, yellow, and white. To rewrite this procedure in assembly language for use with the

CALL statement, it is necessary to know how parameters are passed across. Chapter 8 shows that up to 13 parameters may be passed, and that they will be passed over as long-words in the 68000 registers D1 to D7, and A0 to A5 (in that order). Our example requires nine parameters and they will, therefore, be passed over in registers D1 to D7, and A0 to A1. It is also necessary to know how the respective QDOS routine should be set up. The routine that we are interested in is SD.RECOL (TRAP#3, D0=$26). A full description of this QDOS procedure will be found in Chapter 6. Let us now see how the program in Fig.1 evolved.

```
Demo Program              McGraw-Hill 68000 Ass v1.0A   Page: 0001

                          *H Demo Program
                          ;
  00030000                        org $30000
                          ;
                          ; A short assembly language demonstration
                          ; program. Used in conjunction with SuperBASIC.
                          ; Copyright (c) 1984 McGraw-Hill(UK)
                          ;
                          ;
  00030000 45FA0036       demo:   lea     data(pc),a2     ;Find buffer
  00030004 15420000               move.b  d2,0(a2)        ;Store table
  00030008 15430001               move.b  d3,1(a2)
  0003000C 15440002               move.b  d4,2(a2)
  00030010 15450003               move.b  d5,3(a2)
  00030014 15460004               move.b  d6,4(a2)
  00030018 15470005               move.b  d7,5(a2)
  0003001C 3408                   move.w  a0,d2
  0003001E 15420006               move.b  d2,6(a2)
  00030022 3409                   move.w  a1,d2
  00030024 15420007               move.b  d2,7(a2)
  00030028 224A                   move.l  a2,a1           ;Set data ptr.
  0003002A 363C0000               move.w  #0,d3           ;Timeout=0
  0003002E 2041                   move.l  d1,a0           ;Channel
  00030030 103C0026               move.b  #$26,d0         ;RECOLOUR
  00030034 4E43                   trap    #3
  00030036 4E75                   rts
                          ;
                          ;Workspace
                          ;
  00030038 00             data:   defb    0
                                  defs    20
                          ;
                          end
  Symbols:

  00030038 DATA       00030000 START

0000 error(s)  detected
```

Figure 1 Assembly language version of RECOL

The QDOS procedure requires the eight colour parameters to be set up in a byte table. This means that we have to transfer the contents of the appropriate registers to a small data area. To do this we find out where the data area exists physically for this particular program, and then use byte indexed addressing to perform the transfers. But you say: 'We know where the data area is; it's at $00030038'. In a sense this is true because the program ORG statement has forced this to be the case, but most machine code programs in the QL need to be relocatable. The program shown in Fig.2 is a SuperBASIC program which uses the above machine code routine. When requesting space for the machine code (by using the RESPR command) we do not know, in advance, where SuperBASIC will allocate it. In the example shown we simply asked for 70 bytes to be reserved, and SuperBASIC returned the base address (in variable 'mc') of such an area. Our machine code must work, then, wherever it is put! The LEA instruction found at the beginning of the assembly language program is being used in its 'Program Counter Relative with Displacement' mode, and will store the true absolute position of the beginning of the data area into the designated address register. A detailed discussion of this addressing mode, together with the appropriate assembler syntax (for the McGraw-Hill assembler) will be found in Chapter 2.

Once all the colour parameters have been stored in the data table, it is a simple matter of setting up the appropriate registers for the QDOS call and then executing the TRAP#3 instruction. A final RTS instruction will effect a return to SuperBASIC.

Now let us look at the SuperBASIC program, shown in Fig.2. The program starts by drawing three circles of varying colours. Two of the circles are filled and the third circle is simply an outline. The next operation performed is the storing of the machine code routine into a reserved area of memory. The program is 56 bytes long and therefore a reserved area of 70 bytes is more than sufficient (because the data buffer need only be eight bytes long). The DATA statements hold the denary values corresponding to the hexadecimal instruction opcodes output by the assembler (shown in Fig.1, second column from left). Once this initialization has occurred a small indefinite loop is entered. The loop causes the screen display of circles to invert its colours every 10 seconds.

In SuperBASIC normal program output goes to channel #1. Our machine code subroutine requires the channel 'ID' to be passed over as the first argument in the CALL statement parameter list. You will notice that this parameter, given in lines 170 and 190, is not unity! The SuperBASIC channel numbers have no direct correspondence to the QDOS channel ID values. If no reopening of screen channels is performed the actual correspondence between SuperBASIC screen channels and QDOS channel IDs is as follows:

SuperBASIC #	**QDOS ID**	
	HEX.	**DENARY**
0	**$00000**	**0**
1	**$10001**	**65537**
2	**$20002**	**131074**

It is important to remember that the QDOS IDs will alter if you have reopened a screen channel (e.g., by performing OPEN#1,...). Standard

practice for assembly language programming on the QL would be for particular channel IDs to be determined by use of a suitable algorithm. Chapter 11 contains an appropriate routine.

```
100 REMark Introductory Demonstration Program
120 REMark Copyright (c) 1984 McGraw-Hill(UK)
130 REMark MAIN PROGRAM
140 display_colours
150 mc=RESPR(70): store_mcode
160 PAUSE 200
170 CALL mc,65537,7,6,5,4,3,2,1,0
180 PAUSE 200
190 CALL mc,65537,0,1,2,3,4,5,6,7
200 GO TO 160
210 :
220 REMark ROUTINES
230 DEFine PROCedure display_colours
240 INK 0: FILL 1: CIRCLE 30,60,15
250 INK 4: FILL 0: CIRCLE 60,60,15
260 INK 6: FILL 1: CIRCLE 45,30,15
270 FILL 0
280 END DEFine
290 :
300 DEFine PROCedure store_mcode
310 RESTORE 350
320 FOR c = 0 TO 55: READ n: POKE mc+c,n
330 END DEFine
340 :
350 DATA 69,250,0,54,21,66,0,0,21,67,0,1
360 DATA 21,68,0,2,21,69,0,3,21,70,0,4
370 DATA 21,71,0,5,52,8,21,66,0,6,52,9
380 DATA 21,66,0,7,34,74,54,60,0,0,32,65
390 DATA 16,60,0,38,78,67,78,117
```

Figure 2 Demonstration SuperBASIC program

It is worth stressing that assembly language programming on the QL is best performed using a proper assembler package. Programs should normally be merged into SuperBASIC as extensions to the language, or run as separate machine code programs (jobs) alongside QDOS; either by using the SuperBASIC EXEC command, or by using QDOS job creation/activation procedures. The SuperBASIC CALL command is simple, but very limited, and should only be used for small demonstration/test routines, or for performing the initial linkage of an extended set of SuperBASIC procedures.

PART 1 The 68000 MPU

1 THE 68000 PROCESSOR

At the heart of the Sinclair QL there is a member of the Motorola 68000 family of processors; the Motorola 68008. From a software point of view the 68008 is a full 68000 implementation. Its major difference is that the device package is smaller, and only caters for an 8-bit data bus. An effect of this is that the actual throughput of the processor is reduced, due to overheads in memory addressing. This particular detail should not deter the QL assembly language programmer, who still has at his disposal one of the most powerful state-of-the-art 16/32-bit processors currently available. Also, the 68008 only has 20 of its maximum 32 address lines brought out to its package pins. This means that the addressing range is limited to 1 Megabyte (if you can call 1 Megabyte a 'limitation'!). Before going on to see how this processor may be used within the QL, let us look first at the general features of the 68000.

1.1 Operating modes

Two distinct operating modes are available with the 68000 processor. The two modes are called 'user' mode, and 'supervisor' mode. A flag in the status register will determine which state the processor is in at any one time. Certain instructions (e.g., STOP) cannot be executed while the 68000 is in user mode, and a privilege violation exception process will be initiated by the processor if such an execution is attempted.

When the processor is in user mode, the user stack pointer (USP) will be used by stack related operations. Conversely, the supervisor stack pointer (SSP) will be used when the processor is in supervisor mode.

1.2 68000 registers

The 68000 has eighteen 32-bit registers and one 16-bit status register (see Fig.1.1). The 32-bit register set is divided up into eight data registers, seven address registers, two stack pointers, and a program counter.

DATA REGISTERS

The eight data registers are labelled D0 to D7. Data operations using these registers may be bit, BCD (nibble), byte, word (16-bit), or long-word (32-bit) orientated. Within instructions that permit a data register to be one or more of its operands, any data register may be used. In effect this means that any data register may be used as an accumulator, index register, general purpose register, or loop counter. This is an extremely flexible approach to processor register allocation, and is one of the reasons why the 68000 is so easy to program efficiently.

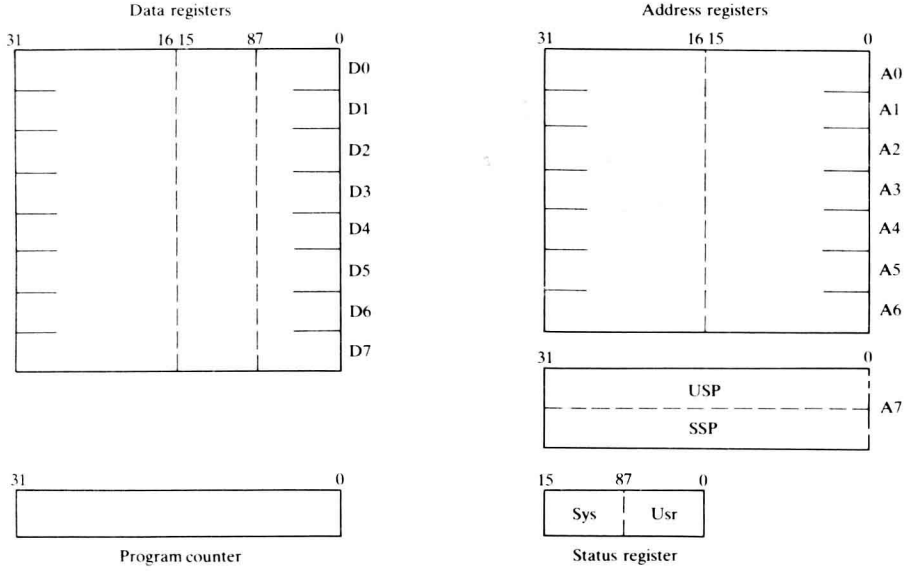


Figure 1.1 68000 internal registers

ADDRESS REGISTERS AND STACK POINTERS

The seven address registers are labelled A0 to A6. The two stack pointers are also treated as address registers and are both labelled A7. Alternative mnemonics for the two stack pointers are USP (user stack pointer) and SSP (supervisor stack pointer). A flag in the status register will determine which state the processor is in (i.e., user or supervisor) and the respective stack pointer will be used accordingly. Operations using the address registers are limited to the types word (16-bit) and long-word (32-bit). In other words, address registers cannot be the source or destination for bit, byte, or logical operations. If an address register is the destination operand, the operation will always be long-word, and the source will be sign extended, if necessary, before use. The address registers are normally used for manipulating and holding addresses rather than data. They may be used also as index registers.

Note that, because the stack pointers are in fact the address register

A7, any legal addressing mode for instructions using address registers will also be legal for the stack pointers. This means that stack pointer register addressing modes for the 68000 are much more versatile than for many other processors.

STATUS REGISTER

The 68000 status register is a 16-bit register split into two distinct bytes. The two bytes correspond to the system status byte (bits 8 to 15) and the user status byte (bits 0 to 7). The system status byte can only be modified when the processor is in supervisor mode. Two mnemonics are allocated to the status register. First, there is CCR, and this refers to the low order user byte of condition codes. Any instruction using this mnemonic will refer to eight bits of data only. Second, there is SR, and this refers to the whole status register. Any instruction using this mnemonic, in order to modify the contents of the status register, will only be executed if the 68000 is in supervisor mode.

Figure 1.2 shows the allocation of flags within the status word. The Carry (C), Zero (Z), Negative (N), and Overflow (V) bits are standard condition flags. There is also an Extend (X) bit flag which is always set to the same state as the Carry flag, if it is affected by any particular instruction. The Extend flag is used for multi-precision arithmetic operations. Chapter 2 describes the relevance of these flags for instructions.

The three least significant bits of the system status byte are used as the interrupt disable mask (IDM) for the 68000. Seven prioritized levels of interrupt are catered for, and each priority interrupt causes execution to be routed through an interrupt vector. The mask in the status register specifies the range of interrupts which are to be ignored. If, for example, the mask is set to binary 011 (3), interrupts 1 to 3 will be ignored by the processor.

Note that the 68008 only has three levels of interrupt (i.e., 2, 5, and 7). On the QL, a level 5 interrupt is transitory and will always generate a level 7 interrupt (non-maskable) within 20 ms.

The Supervisor flag (S) determines whether or not the 68000 is running in supervisor mode. If the bit is set (i.e., 1), the processor will be in supervisor mode. Last, but far from least, is the Trace (T) flag. This flag enables the processor to be run in single-step mode, and permits system debuggers to obtain control over instruction execution.

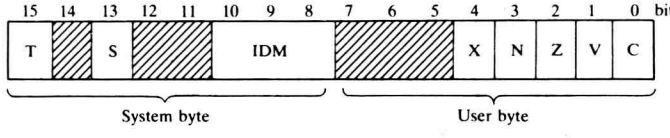


Figure 1.2 68000 status register

1.3 Use of memory

Up to 1 Megabyte of memory may be directly accessed by the 68008. A 20-bit address bus is required to address this amount of memory. Addresses can, therefore, be represented by five digit hexadecimal numbers in the range $00000 to $FFFFF. To access a byte of data in memory, any one of the possible addresses may be used. Accessing a word (i.e., 16 bits) or a long-word (i.e., 32 bits) of memory is a little more restrictive. Words or long-words can only be addressed at even addresses; that is, $00000, $00002, and so on up to $FFFFE.

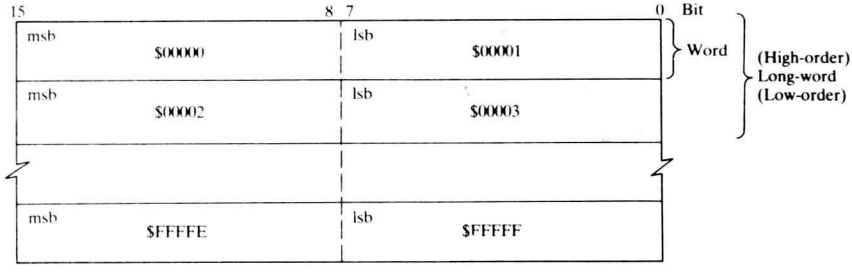


Figure 1.3 Memory usage

In the case of word addressing the most significant byte of the word will be found in the even address, and the least significant byte of the word will be found in the following odd address (see Fig.1.3). Long-word addressing is similar to word addressing in that it involves the equivalent of two word accesses. The high-order word of the long-word will be found first, followed by the low-order word of the long-word.

1.4 Moving between supervisor and user modes

To know that the 68000 has two modes of operation is not particularly useful unless you know how to swap between them. When the 68000 is reset (e.g., at power-on) the bottom eight bytes of memory are loaded into the supervisor stack pointer and the program counter, and instruction execution commences in supervisor mode.

Because we start off in supervisor mode the first thing we need to know how to do is enter user mode. The task is not onerous! Any instruction, which is capable of affecting the state of the S flag in the status register, also has the ability to transfer execution to user mode. An example is the RTE (ReTurn from Exception) instruction. This instruction will load the status register with the word on the stack, and load the program counter with the following two words on the stack. If the word loaded into the status register reset the S flag, user mode will be entered. If the S flag remains set, supervisor mode will continue.

Once you are in user mode, the method of getting back into supervisor mode is to cause some form of exception processing. Exception processing will occur under a number of conditions:

1. Addressing violation. A word or long-word was addressed on an odd byte boundary.
2. Privileged instruction violation. A privileged instruction was executed.
3. Illegal or unimplemented opcode. The instruction executed was not a legal instruction.
4. TRAP instruction execution. All TRAP instructions are treated as internal exceptions.
5. TRAPV, CHK, DIVS, DIVU exception error condition has occurred (e.g., divide by zero).
6. Trace active. If the T flag in the status register is set, exception processing will be performed after each user instruction is executed.
7. External interrupt request. One of seven (n.b., three on the QL) prioritized interrupts has been received.
8. Reset. The 68000 processor has physically been reset.
9. Bus error. An error has occurred on the physical address/data bus of the 68000 processor.

The last two of these exception processes (reset and bus error) are clearly not of much use to the applications programmer! The most common way of entering supervisor mode from a program is through the use of a TRAP instruction.