MC68000 Assembly Language Programming

Brian Bramer

MC68000 Assembly Language Programming

Brian Bramer B.Tech., Ph.D., C.Eng., M.I.E.E., M.I.E.E.E.
Principal Lecturer in Information Technology
School of Mathematics, Computing and Statistics,
Leicester Polytechnic

Edward Arnold

First published 1986
by Edward Arnold (Publishers) Ltd
41 Bedford Square, London WC1B 3DQ

Edward Arnold (Australia) Pty Ltd,
80 Waverley Road, Caulfield East,
Victoria 3145, Australia

Edward Arnold, 3 East Read Street,
Baltimore, Maryland 21202, USA

ISBN: 0 7131 3595 6

Printed in Great Britain by
J.W. Arrowsmith Ltd, Bristol

Preface

This book is designed as a learning text for students of Computer Science, Electronic Engineering, Information Technology and other HND, B.Sc or M.Sc courses who are attending a module that introduces microcomputer architecture and the assembly language programming of the Motorola MC68000 microprocessor. In addition, the book may also be used for self-instruction by experienced programmers and home computer users who wish to program their 68000 based microcomputers in assembly language.

The Motorola MC68000 family of microprocessors is used in many modern microcomputers ranging from single board development systems up to professional CAD/CAE workstations. The MC68000 can address up to 16 Mbytes of memory and has a large range of powerful instructions that can process 8-, 16- or 32-bit data.

This learning text would normally be used to accompany a course of practical/tutorial sessions that may be backed by lectures. Each chapter is a self contained unit that can be read by the student and contains exercises that should be attempted during tutorial/practical sessions. Answers to the exercises are given in Appendix E. To complete each chapter is a problem that should be attempted and may be used by instructors as a means of assessment. At critical points in the text reviews of topics covered in the previous few chapters are presented. Depending upon the level of the course and the programming experience of the students, lectures can be given to reinforce the learning process (e.g. based upon the reviews).

Good programming practice is encouraged throughout the book by the use of modular and structured programming techniques. In particular, as an aid to modular programming, the use and writing of subroutines is described early in the text and a library of MC68000 assembly code subroutines is provided which enables student programs to read and write numeric and other data via the computer console. This enables the use of subroutines in the exercises and problems before the operation of the subroutine call and return instructions are described in detail.

Although this book is a self contained text, it is expected that a computer architecture/assembly language programming module that makes use of it would run in parallel with a module on high-level language programming (e.g. Pascal). This would serve to reinforce the modular and structured programming techniques outlined in this book, which could then be formalized in a Software Engineering module at a later stage.

This book has not been written with any particular MC68000 microcomputer system in mind and the only assumption made is that single character input/output routines are available which can be accessed from assembly language programs (Appendix C discusses some of the problems that may arise when using different 68000 based microcomputers).

Outline of the Book

An introduction to computer systems, information representation, and computer hardware and software fundamentals is presented in Chapters 1, 2 and 3 (these initial chapters may be omitted if these topics have already been covered in other modules). While these chapters are being covered, the students could be carrying out practical exercises on number systems (Appendix A describes the binary and hexadecimal number systems and the ASCII character code), and learning to use the monitor/operating system of the microcomputer to be used (a tutorial/practical learning system could be provided for this).

When the basic MC68000 architecture and programming is introduced in Chapters 4, 5 and 6, the student should be familiar with the use of the microcomputer and will be ready to attempt the programming exercises and problems. Software Engineering techniques are then presented in Chapter 7 (this could be omitted if the student has already covered this topic in another module) followed in Chapter 8 by an introduction to subroutine programming and the use of the assembly language subroutine library of Appendix D. Subroutines can then be written and used in the exercises and problems of Chapters 9 to 15 (well before the operation of the subroutine call and return instructions are described in detail in Chapter 14).

Basic input/output programming is introduced in Chapter 16 followed in Chapter 17 by an introduction to parallel communications and the MC6821 PIA (Peripheral Interface Adaptor). The exercises of this chapter (and Chapter 19) may be carried out using ready built plug-in cards to attach to the PIA, and/or may be amended to take account of the facilities available and the overall course objectives. For students who have covered basic digital sequential and combinational logical circuits, extra exercises and problems can be provided in which the students build circuits to be attached to the PIA and tested with assembly language programs. Chapter 18 gives details of MC68000 exceptions (interrupts are a type of exception) and Chapter 19 then describes the use of interrupts with the MC6821 PIA. Chapter 20 discusses serial communications and describes the MC6850 ACIA (Asynchronous Communications Interface Adaptor). The exercises and problems of this chapter will depend upon what external equipment may be connected.

Proficiency in programming can only be gained by practice. It is therefore recommended that programs given as exercises and problems are designed, coded, run, and tested before continuing to the next chapter. Depending upon the knowledge of the students and the overall aims of the course sections of the book may be skimmed over or omitted entirely, and extra exercises and problems provided. For example, if the microcomputer being used supports Pascal, it would be possible to provide exercises in the linking of Pascal main programs to assembly language subroutines.

Recommendations to Tutors

The tutor should initially scan the book to gain an overall impression of each chapter with its associated exercises and problems. Appendix C should then be examined to determine problems that may arise due to the target microcomputer configuration. Chapters 1 to 6 can then be read in detail and the exercises and problems attempted. Chapters 7 and 8 can then be read, the assembly language library of Appendix D implemented (test programs are provided to check the library) and the exercises and problems of Chapter 8 attempted. Once the library is operational the following chapters can be read. By attempting the exercises and problems the tutor should discover any difficulties associated with the particular microcomputer and assembler being used. The students could then be provided with a list of such difficulties and modifications (e.g. the use of a non-standard assembler), and other details such as I/O device register addresses,etc.

If the students are to attempt to program the MC6821 PIA (Chapters 17 and 19) and/or MC6850 ACIA (Chapter 20) input/output devices, plug-in cards and other external equipment will have to be provided (sample circuits that can be built for use with the MC6821 PIA are shown in Chapters 17 and 19).

Brian Bramer,1986

Contents

1 Introduction to Computer Systems

Until a few years ago computer systems were large, expensive and required many expert staff to maintain, operate and program them. The users of such systems were generally restricted to staff within large organizations that could afford to purchase and operate them. The advent of microcomputers has changed this, enabling the production of a computer system small enough to sit on a desk and cheap enough to be used in the home (1), the laboratory (2), the factory (3,4), in commerce (5) and in the office (6).

1.1 An end-user's viewpoint of a computer system

The vast majority of users of computer systems are **end-users**, in that they use computer systems as a tool to aid their everyday work. The computer system can be considered as a ´black box´ into which information is fed for processing and then results are produced. The information input can range from simple text (7) or numerical data (8) (entered via a keyboard) to diagrams (entered via a digitizing tablet). Similarly, information output may be text, numbers or diagrams and pictures (9) presented on a display screen or printer. To communicate with the system, users employ terms that are common to their everyday working environment, e.g. accountants use columns of numbers and artists use pictures.

In general end-users can purchase a complete microcomputer system that consists of the hardware (the physical components), and the software (the programs that tell the hardware what to do) that suits their application. Thus they do not need to have any knowledge of how a computer system works or how to write the programs. In fact, attempts by end-users to learn these skills will distract them from gaining a knowledge of the more important requirements, such as (10):

1 Applications of computers in their own field.
2 Limitations of computer systems, e.g. accuracy, size of problem that can be solved, etc.
3 How to draw up a specification of a computer system to suit their requirements.
4 How to draw up tender documents and arrange demonstrations.
5 How to assess the system´s performance in meeting the requirements specification.
6 How to install and then manage the computer system, e.g. taking disk backups, etc.

1.2 A computer programmer's viewpoint

1.2.1 Application programmers

In many application areas there is still a need for specialists with a knowledge of computer programming. These specialists may be professional computer scientists implementing computer systems in particular application areas, or scientists or engineers writing programs for their own applications. Such programs would generally be written in a high-level problem-solving language (11) that, apart from some appreciation of accuracy limitations, requires little knowledge of the internal workings of the computer system.

1.2.2 Systems programmers and hardware designers

Computer scientists and electronic engineers who have a need to write programs to control input and output devices do require a knowledge of the internal working of computer systems and how to write in assembly language. This book describes the assembly language for the Motorola MC68000 microprocessor and how it can be applied to program simple input/output devices. The MC68000 is a member of the Motorola range of microprocessors (12). It is an advanced 16-bit microprocessor (13) that, in its various versions, is used in many modern microcomputer systems (14,15).

1.3 A microcomputer system

At a superficial level a computer system can be considered as consisting of three components, namely hardware, software and data.

1.3.1 Hardware

The term hardware embraces the physical components of the system:

1 The box that contains the processing elements, memory, input/output interface circuits and power supply.
2 Display screen and keyboard for user interaction.
3 Peripheral devices such as disks and printers.

The internal electronic circuits of modern computers are made up from a number of integrated circuit chips and other components. An integrated circuit chip is a small packaged device a few centimetres square that contains complex electronic circuits. The heart of the modern microcomputer is the microprocessor which is an integrated circuit chip that contains the basic control and processing circuits of a small computer. The complete microcomputer system contains a microprocessor plus memory, input/output devices, power supplies, etc.

However, before the computer hardware can perform a task (for example add numbers or read a character from a keyboard), it requires a program to tell it what to do.

1.3.2 Software

Software comprises the programs that tell the hardware what to do. A program is a sequence of instructions stored in the memory of the computer system. The processor fetches an instruction, decodes it and then executes the required operation (e.g. to add two numbers). When an instruction execution is complete, the processor fetches the next one and so on. A program may be simple, for example, to calculate the average of ten numbers, or very complex, as would be required to draw a picture on a display screen.

1.3.3 Data

The data is the information to be processed by the computer system. Data may be simple numbers for mathematical calculations, text such as addresses or more complex structures such as pictures or drawings. The instructions that make up the program define what data is to be processed, in what form and at what time.

1.4 Instruction and data storage

Within the computer hardware there must be some **memory** that stores the instructions of the program to be executed and the data to be processed.

1.4.1 Representation of integer numbers

Within modern computer systems the basic element of storage is the **binary bit** which can represent a 0 or a 1. The reason for this is that it is very easy to build electronic switches to represent an off/on condition or 0/1 binary value. Although a single binary bit can only represent two states, 0 or 1, a sequence of bits can be used to store a larger range of values. Such a sequence is called a word of storage and is usually 8, 16, 32 or 64 bits in length. An 8-bit word, for example, can represent a positive number in the range 0 to 11111111 binary (0 to 255 decimal) as shown below:

bit	7	6	5	4	3	2	1	0
bit value	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

In the diagram above it can be seen that the least significant or rightmost bit, bit 0, represents 1 and the most significant or leftmost bit, bit 7, represents 128 decimal (the convention for identifying the bits within a word is that the least significant bit is numbered 0). The combinations of 1s and 0s of the 8-bit word thus represent the range 0 to 11111111 binary (0 to 255 decimal). The general term given to an 8-bit storage word is a **byte.** The majority of modern computer systems use a memory based on bytes of storage. To represent values that are too large to store in eight bits a number of bytes may be used. For example, a 16-bit number, that could be made up from two bytes, can represent a value in the range 0 to 65535 decimal (see Appendix A for more details of the binary number system).

Many scientific calculations require the use of signed numbers and in this case the majority of modern computer systems use a storage system known as **twos complement binary arithmetic** in which the most significant bit is used to store the sign (1 for a negative number and 0 for a positive number). Thus an 8-bit number can represent values in the range -128 to +127 and a 16-bit number values in the range -32768 to +32767 (see Appendix A for further discussion). The Motorola MC68000 allows calculations to be carried out on 8-, 16- and 32-bit signed and unsigned numbers.

In practice it would be both difficult and error prone to enter data directly in binary form, so hexadecimal (base 16) or decimal are more commonly used. It is a relatively easy task to convert between binary and hexadecimal (Appendix A describes some conversion techniques).

Exercise 1.1

Convert the following numbers to binary and hexadecimal; do the calculation in each case and then convert the result back into decimal (use signed 8-bit twos complement binary numbers):

```
 16      45      110      110
+32     +60     - 45     + 45
```

The last calculation gives a condition called overflow; explain what has happened.

1.4.2 Character data

Character data is used within the computer to represent text (such as names and addresses) and consists of all the usual printable characters (for example the alphabet A-Z and a-z, digits 0-9 and other characters such as +, -, *, /).

In practice, each character is stored in a byte of memory and represented by a particular binary pattern or character code. To enable different computers, terminals and printers to be connected together there are a number of standard character codes. The most commonly used character code is called ASCII (American Standard Code for Information Interchange), which is listed in Table A.2 of Appendix A. The character A, for example, is represented by the binary pattern 01000001 (41 hexadecimal), and B by 01000010 (42 hexadecimal), etc. The majority of computer users do not need to know or even be aware of these codes as the keyboard and display equipment converts between the characters and the internal codes automatically (i.e. if the user hits the key A on the keyboard the binary value 01000001 is sent to the computer).

Exercise 1.2

From Table A.2 of Appendix A determine the decimal, hexadecimal and binary values of the ASCII character code for the following characters:

A a B b I i X x 1 5 + - ? =

Can you see anything significant about the order of the ASCII character codes for letters and digits and why it may be useful ?

1.4.3 Instruction representation

A computer program is made up of a sequence of instructions which are represented by binary patterns. For example, the binary pattern 0100001001000011 (4243 hexadecimal), when executed by the Motorola MC68000 microprocessor, would set all the lower 16 bits of the data register D3 to 0. Each instruction that the computer hardware can execute has a particular binary pattern, with sequences of such binary patterns in the memory of the computer forming a program. Programs in this form are in a language called machine code, i.e. the language the hardware of the computer understands. It is clear that if humans had to write programs in this machine code form, programming would be a very error prone and time consuming task. In practice, machine code programming is limited to applications where there is no other way of getting a program into a machine (for example, testing a computer where the input/output system is faulty).

In practice, professional programmers use either an assembly language or a high-level language.

1.5 Assembly language

In an assembly language each machine instruction is represented by a meaningful mnemonic (e.g. ADD, SUB, DIV) and data can be represented in binary, octal, hexadecimal, decimal and character form. The MC68000 instruction that clears the lower 16 bits of data register D3, as above, would be written:

```
CLR.W     D3
```

where CLR.W is the instruction mnemonic and D3 is the position of the data being operated upon. The computer hardware can only understand machine code, so before it can be executed the assembly language program has to be converted into machine code. This is done by a program called an assembler that takes each assembly language statement and converts it on a one-to-one basis into the equivalent machine code. The resultant machine code is then executed.

Even this form of programming is difficult because it is only one level above machine code and orientated to a particular computer (each processor type has its own machine code language). For example, programmers who had a knowledge of the Motorola MC68000 microprocessor assembly language would have to learn a new assembly language if they then worked with an Intel 8086 microprocessor. In addition, any assembly or machine code language programs that had been written for the MC68000 would have to be totally rewritten for the 8086.

Even with the above disadvantages assembly language programming is still required in many cases, such as:

(a) for code to control input/output devices;
(b) for time critical code in real time applications;
(c) for some control applications.

Machine code and assembly languages are described as low-level languages in that they are orientated towards the computer hardware. High-level languages on the other hand are computer independent and problem orientated.

1.6 High-level problem solving languages

High-level languages are written in an English or mathematical notation that is orientated towards solving practical problems (11). Some examples of high-level languages are:

BASIC	Beginners All-purpose Symbolic Instruction Code: a simple language available on many home microcomputers;
FORTRAN	FORmula TRANslation: a language widely used for mathematical, scientific and engineering applications;
COBOL	COmmon Business Orientated Language: a language designed for commercial business applications;
PASCAL	a modern powerful general problem solving language.

After the program has been entered into the computer it has to be converted into machine code by a program called a compiler. Each statement in a high-level language can be converted into a number of machine code instructions. For example, the Pascal statement:

```
A:=B+C+20;
```

could become several machine code instructions.

The compilation process is not 100% efficient so a program written in a high-level language will take more memory and run slower than an equivalent assembly language program written by a **good** programmer. However, the advantages of working in a language that is orientated towards solving problems rather than towards the computer hardware means that the majority of application programs are written in high-level languages.

An additional advantage of using high-level languages is that such languages are less computer dependent than assembly languages (depending upon the quality of the international standard of the language and the particular implementation being used). A Pascal program written and tested on one make of computer system should run without problems on a different make.

1.7 Review of information representation

ALL information within the computer, either instructions or data, is represented in binary. To a programmer working in a high-level language this is not a problem as the compiler and run time system look after data storage and conversion between decimal and binary. Consider the following simple Pascal program which writes a number and a character:

```
PROGRAM TEST;
CONST I=20; X='D';
BEGIN
WRITELN(' number is ',I,' letter is ',X);
END.
```

When the above program is compiled (i.e. converted into machine code), the compiler assigns the storage for any variables and sets up the values defined by CONSTant expressions - the values of I and X are converted to the equivalent 16-bit and byte binary values 0000000000010100 and 01000100 respectively (0014 and 44 hexadecimal). When the program is executed the WRITELN statement converts the internal binary representation of the integer number I into a string of digit character codes to be transmitted to the display screen (the display hardware converts these into characters to be viewed).

When working in a low-level language such as machine code or assembly code, the binary pattern 01000010 01000011 (42 43 hexadecimal) could represent to the computer hardware:

1 two 8-bit integer numbers: 66 and 67 decimal;
2 a 16-bit number: 16963 decimal;
3 two characters in ASCII: B and C respectively;
4 the instruction to the Motorola MC68000 microprocessor to set the lower 16 bits of data register D3 to 0, i.e. CLR.W D3.

When working in a high-level language such as Pascal, the compiler and run time system look after the organization of data storage and conversion between external characters and the internal binary form. When working in machine code or assembly languages the programmer is responsible for seeing that instructions and data are kept separate and that the correct code is executed and data processed. It is very easy to get mixed up and try to add character data or even execute data. Careful program design and coding will avoid this problem.

Exercise 1.3

What particular problems could face an assembly language programmer when looking for a new job ?

1.8 Problems for Chapter 1

1 Why is the binary system used for information storage within modern computer systems ?

2 Convert the following numbers to binary and hexadecimal; do the calculation in each case and then convert the result back into decimal (use signed 16-bit twos complement binary numbers):

```
    67        67        456        456       1027
   +86       -86       +345       -345      +2056
```

3 Describe, in each case, the advantages/disadvantages and areas of application of:
   (a) machine code programming;
   (b) assembly language programming;
   (c) high-level language programming.

2
Computer Hardware: The Functional Components

The hardware of a microcomputer system can be broken down into a number of functional components:

High speed registers
Primary and secondary memory
The Control Unit
The Arithmetic/Logic Unit (ALU)
Input and Output

Fig. 2.1 is a diagrammatic representation of these components and shows the information and control flow within the system.

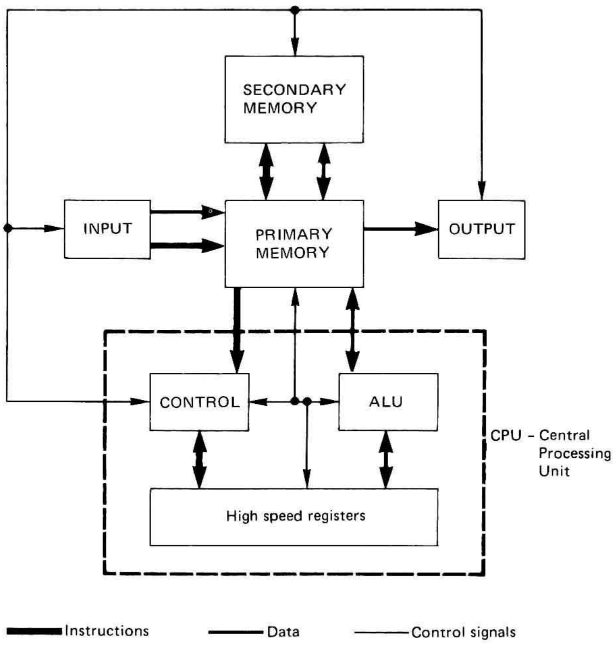

Fig. 2.1 Functional block diagram of a computer system

The following sections describe each functional component and its role within the overall system.