Design of Arithmetic Units for Digital Computers

John B. Gosling
Department of Computer Science,
University of Manchester

M

First published 1980 by
THE MACMILLAN PRESS LTD
London and Basingstoke
Associated companies in Delhi Dublin
Hong Kong Johannesburg Lagos Melbourne
New York Singapore and Tokyo

Typeset in 10/12 Press Roman by
STYLESET LIMITED
Salisbury · Wiltshire
and printed in Great Britain by
J. W. Arrowsmith Ltd, Bristol

ISBN 0 333 26397 9
ISBN 0 333 26398 7 pbk

Design of Arithmetic Units for Digital Computers

Macmillan Computer Science Series

Consulting Editor
Professor F. H. Sumner, University of Manchester

S. T. Allworth, *Introduction to Real-time Software*

G. M. Birtwistle, *Discrete Event Modelling on Simula*

Richard Bornat, *Understanding and Writing Compilers*

J. K. Buckle, *The ICL 2900 Series*

Derek Coleman, *A Structured Programming Approach to Data**

Andrew J. T. Colin, *Fundamentals of Computer Science*

Andrew J. T. Colin, *Programming and Problem-solving in Algol 68**

S. M. Deen, *Fundamentals of Data Base Systems**

J. B. Gosling, *Design of Arithmetic Units for Digital Computers*

David Hopkin and Barbara Moss, *Automata**

Roger Hutty, *Fortran for Students*

H. Kopetz, *Software Reliability*

A. Learner and A. J. Powell, *An Introduction to Algol 68 through Problems**

A. M. Lister, *Fundamentals of Operating Systems, second edition**

Brian Meek, *Fortran, PL/I and the Algols*

Derrick Morris and Roland N. Ibbett, *The MU5 Computer System*

John Race, *Case Studies in Systems Analysis*

I. R. Wilson and A. M. Addyman, *A Practical Introduction to Pascal*

* The titles marked with an asterisk were prepared during the Consulting Editorship of Professor J. S. Rohl, University of Western Australia.

Preface

The original motivation for the development of digital computers was to make it possible to perform calculations that were too large to be attempted by a human being without serious likelihood of error. Once the users found that they could achieve their initial aims, they then wanted to go into greater detail, and to solve still bigger problems, so that the demand for extra computing power has continued unabated, and shows no sign of slackening. This book is an attempt to describe some of the more important techniques used today, or likely to be used in the near future, to perform arithmetic within the computing machine.

There are, at present, few books in this field. Most books on computer design cover the more elementary methods, and some go into detail on one or two more ambitious units. Space does not allow more. In this text the aim has been to fill this gap in the literature.

In selecting the topics to be covered, there have been two main aims: first, to deal with the basic procedures of arithmetic, and then to carry on to the design of more powerful units; second, to maintain a strictly practical approach. The number of mathematical formulae has been kept to a minimum, and the more complex ones have been eliminated, since they merely serve to obscure the essential principles.

At the practical level, discussion has been restricted to the binary number system. Some may argue that there should be a discussion of other bases, and perhaps of a redundant number system (for example, a residue system). None of these has found great acceptance, and does not appear to be about to do so. For practical consideration also, iterative arrays have been omitted (unless the simultaneous multiplier is considered to be such). The most controversial omission is a discussion of error-detecting techniques and fault-finding considerations. These topics require more space for proper treatment than is available here.

The readership of the book is expected to range from undergraduate students to computer designers. First-year students might find it profitable to restrict consideration to chapter 2, sections 3.1, 3.2, chapter 4 and perhaps the first three sections of chapters 5 and 6. Final-year students should be able to tackle the whole of the book. It is hoped that the book will also prove a useful work for those involved in the design of real machines.

For the benefit of students, a number of tutorial and examination questions have been appended to most chapters. Those with acknowledgements are reprinted by permission of the University of Manchester from examination papers of the Department of Computer Science and of Electrical and Electronic Engineering. Answers to numerical parts are provided at the end of the book, together with brief notes for some other questions.

ACKNOWLEDGEMENTS

The author would like to express his thanks to the many people who have in some way contributed to the book. Professor D. B. G. Edwards of the Computer Science Department in the University of Manchester and Professor D. Aspinall of UMIST are responsible to a large degree for providing a baptism in the subject, as well as much subsequent help. The former has provided the opportunities for much of the practical work. Many helpful discussions have also been held with Professor D. J. Kinniment of the University of Newcastle, and with E. T. Warburton of ICL. Dr S. Hollock of Plessey (Research) Ltd has provided the opportunity for work on the uncommitted logic array mentioned in chapter 3. Dr L. E. M. Brackenbury kindly commented on the manuscript, and others are too numerous to mention individually; their colleagueship is none the less appreciated. Thanks are also due to my wife for her tolerance during the many evenings spent preparing the work.

J. B. GOSLING

Contents

1 *Preliminary Notes*

1.1 INTRODUCTION

One of the main pressures for the development of the modern digital computer was the need to perform calculations that were beyond the capability of a human operator, partly because of the sheer length of the calculation and partly because of the likelihood of errors arising through tiredness or other human factors. The machine should overcome both of these limitations to a considerable extent. Over the years machines have become increasingly more powerful, and users have continued to demand more and more capability. Computers have of course penetrated many other areas than mathematics, but this book is primarily concerned with the way in which the elementary mathematical processes can be, and are, implemented in digital computing machines.

The prime intention of the book is to give a practical description of the algorithms for performing the various operations, and to explain how they are implemented. Although covering the elementary algorithms described in most general textbooks on computer design, it will also deal with more advanced concepts and more powerful units which are generally omitted from these texts. The selection of algorithms described could be extended considerably, but the intention has been to restrict the list to those that either add to an understanding of the processes concerned, or have practical usefulness in the computers of today and the foreseeable future. In some cases an indication of other possibilities is given, and the bibliography provides further reading on these topics.

The arithmetic described in this book is limited strictly to binary arithmetic (base 2), since this is the predominant means of implementation. Decimal arithmetic must be coded in binary in some way for convenience of machine implementation. The main area for the use of decimal coding is in finance, and if the arithmetic is limited to the use of integers (for example, pounds and pence expressed in pence) then a binary coding is just as good as decimal, and is considerably faster. Other radices that have been proposed are ternary (base 3) and negabinary (base -2). Neither of these representations has gained any great acceptance and probably will not, though such predictions are hazardous to say the least. Other forms of representation have been suggested, some of which have error-detecting and/or error-correcting properties. None of these has yet found wide acceptance.

1.2 ASSUMPTIONS

This text will assume that the reader is familiar with the binary representation of numbers, and can recognise simple numbers. It will also assume that he is capable of understanding and following the manipulation of logical expressions in Boolean form, though knowledge of advanced logical techniques is not required. The symbols '.', '+' and an overbar are used to represent the AND, OR and NOT functions respectively. The symbols used in diagrams are those used in most manufacturers' data books. Details of specific commercial devices are not assumed, though anyone wishing to make use of the design techniques described would clearly require access to the relevant literature, and in some cases figures are quoted from these sources without comment.

1.3 TERMINOLOGY AND CONVENTIONS

The meaning of a number of terms used in the text will require a brief explanation. A flip–flop is a temporary storage platform of one *bit* (*b*inary dig*it*). Two important forms exist. The '*D* latch' of figure 1.1 transfers the data on one input, *D*, to the output *Q* whenever the second input, the 'clock', is in one state (high in figure 1.1). When the clock is in the other state, *Q* remains at the last value of *D* prior to the clock change. The second type of flip–flop is a master–slave type (figure 1.2). This is, in fact, two latches, one clock being the inverse of the other. The over-all effect is that the input, *D*, appears at the output, *Q*, following one *edge* of the clock waveform. Otherwise *D* and *Q* are isolated from each other.

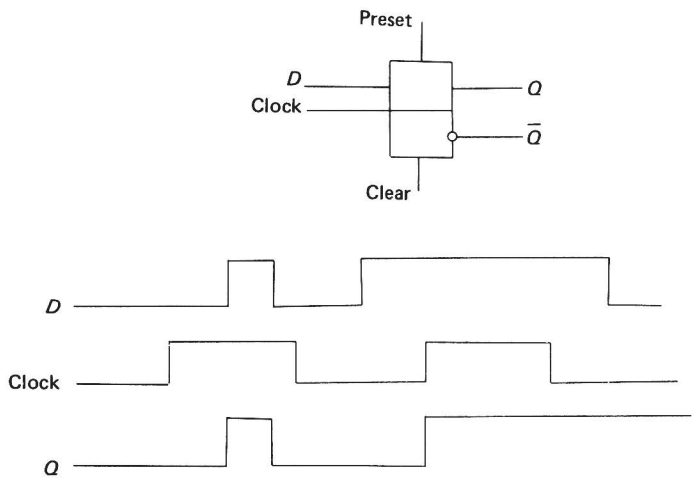

Figure 1.1 D-latch flip–flop

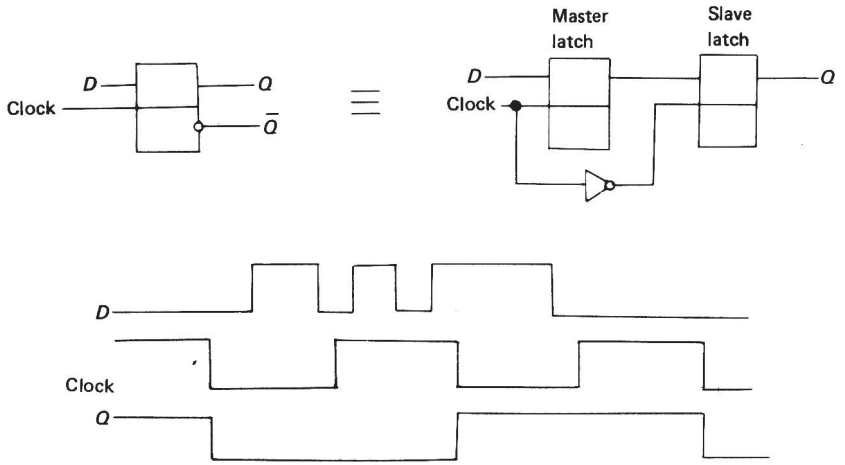

Figure 1.2 Master–slave flip–flop

Both types of flip–flop may have a preset and/or clear input (sometimes known as set and reset), which set Q to binary 1 or 0, respectively.

A *register* is a collection of flip–flops providing temporary storage for a number of bits. This number is usually a *word* of the machine. A word is a number of bits making up the basic numbers in the machine. In modern machines all the bits of a word are usually handled at the same time (in parallel). A shift register is a register in which the Q outputs are connected to the D inputs of adjacent device(s). Figure 1.3 shows a shift register capable of shifting both ways. With the control signal in the one state, data is shifted left to right. Application of an appropriate clock edge will cause a shift of one place. With the control in the zero state, shifting is right to left. Clearly the flip–flops must be master–slave types to ensure only one place shift per clock pulse.

It is a convention of engineering drawing that signals normally flow left to right and top to bottom as far as possible. However, in pencil-and-paper addition it is normal to place the least significant digit on the right, and work right to left. Thus a carry will flow right to left. In the diagrams in this book the

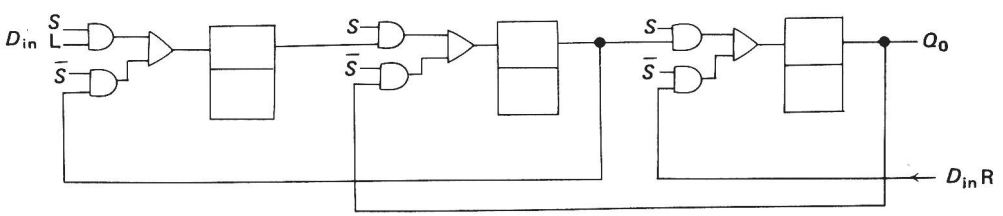

Figure 1.3 Bidirectional shift register

convention used is that familiar from the latter procedure: that is, the least significant (LS) bit of a number is on the right of a diagram and the most significant (MS) bit on the left.

1.4 NUMBER FORMATS

Numbers in digital computers are represented in one of two basic forms, fixed point and floating point. In fixed-point form the binary point (equivalent to the decimal point) is in a fixed place in the representation. In most computers this is to the right of the least significant bit, and hence the representation is of integers, and this will be assumed frequently. The other form of representation, floating point, is described fully in chapter 6, and is a means of providing an extension of the range of representable numbers without using more than a single word (two words for smaller machines). Variations of these forms exist, but are of insufficient interest from the present point of view.

Arithmetic is generally performed between two numbers. The description here is mostly in terms of a unit in which one of the numbers is initially held in the arithmetic unit in a register called the *accumulator*, and the other number is supplied from storage and is referred to as the 'operand'. There is clearly no incompatibility with other types of unit where both operands are supplied from storage, whether the storage is registers or another form.

It will also be clear that, with the limited number of bits available in a computer word, there is always a finite possibility of producing results that are too large to be held in the representation. For instance, in relation to probability calculations, 57! is too large a number to be held in many commercial machines. Yet it is not at all impossible for a program to call for even larger numbers as intermediate results. The solution to this difficulty is a programming problem, but it is necessary for the hardware to give warning of unexpected *overflows*. In the text this problem is largely ignored except for certain specific sections. Overflows are not in fact difficult to detect, usually involving the provision of a few (often only one) *guard bits*. In the simple case of the addition of two positive numbers the guard bit is the carry from the most significant bit.

1.5 COST AND TIME

Throughout the book an attempt has been made to give practical figures for costs and times. Cost is measured on the basis of the number of integrated circuits (ICs) used. This is a fairly accurate guide, since it is also related to the printed-circuit (PC) board area, power dissipation and cooling arrangements. It does not take into account differences in size and dissipation of ICs, however, and this can have some effect since, for example, an arithmetic logic unit is a

24-pin package which requires almost four times the PC board area of the more common 16-pin package.

Times of operations are calculated on the basis of the worst-case times quoted by the manufacturers. Additional allowance might be made for wiring delays where circuits of the highest speed are concerned. In all cases the times depend to some extent on the details of the implementation. An accuracy of ±10 per cent is probable, and comparative figures should be at least as good as this, since they are all made on the same basis. However, the reader is also referred to chapter 8 in this respect.

Cost figures will, of course, change rapidly as more and more circuitry is incorporated in each package. However, where systems are implemented on ICs, the figures given indicate the complexity (and hence the production difficulty) of such ICs. Changes in speed as technology changes are less important, since to some extent the figures are relative figures. What cannot be foreseen is what new algorithms may be discovered which will only be economically viable because of the higher degree of integration available.

2 *Addition*

The most important arithmetic operation in a computer is addition. Subtraction is commonly implemented by the addition of the negative of the subtrahend, and in this book will not be discussed separately. Both multiplication and division can be implemented by means of addition and subtraction. In order to keep the discussion unencumbered with the problems of representing negative numbers, this chapter will describe the most important techniques for performing addition, assuming unsigned binary numbers. The effect of introducing negative numbers, and the implementation of subtraction, will be delayed until chapter 4. For the purposes of this chapter all numbers will also be assumed to be 'fixed point'.

2.1 BASIC ADDITION

Figure 2.1 illustrates the principle of any addition. Two numbers $X_N \ldots X_2 X_1$ and $Y_N \ldots Y_2 Y_1$ are to be added together. At each digit position the addition results in a sum, S, and a carry, C. The carry occurs if the sum is greater than 9 in decimal, or 1 in binary, the 'sum' in this case being the sum of X_i, Y_i and C_{i-1}. The box marked '+' performs the addition. Table 2.1 describes the operation of

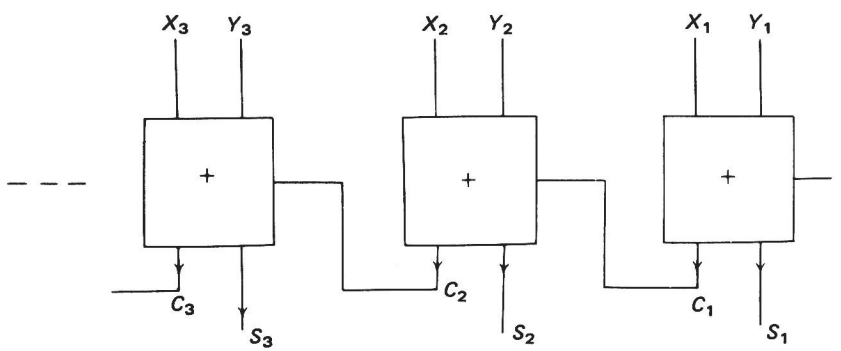

Figure 2.1 Principle of addition

Table 2.1 Truth table for binary addition

Inputs			Outputs	
X	Y	C	S	C
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

this box for binary numbers. The Boolean expressions for the sum and carry are

$$S_i = X_i\bar{Y}_i\bar{C}_{i-1} + \bar{X}_iY_i\bar{C}_{i-1} + \bar{X}_i\bar{Y}_iC_{i-1} + X_iY_iC_{i-1} \tag{2.1}$$

$$C_i = X_iY_i + X_iC_{i-1} + Y_iC_{i-1} \tag{2.2}$$

These two expressions are in minimal form, but there are several other forms and groupings of the terms which are useful in particular circumstances. Implementation of the expression as written can be achieved with a single AND–OR circuit having up to six inputs and an output. The AND–OR function (or AND–NOR) takes very little more time to perform than a simple AND (or NAND) function*. For present purposes the time to perform the AND–OR function will be designated t, and regarded as a basic time unit. The AND–OR circuit will be regarded as a basic cost unit in assessing the relative merits of different adder designs.

The adder of figure 2.1 is referred to as a ripple-carry adder, since the carry 'ripples' through each stage in turn. This corresponds with the pencil-and-paper procedure. To complete an addition a carry signal may start at the less significant (LS) end and propagate all the way to the more significant (MS) end. The following example illustrates this. The '*C*' bits are the carries produced by adding the 2 bits of the preceding column.

```
  0 1 0 1 1 0 1 0 1
  0 0 1 0 0 1 0 1 1
-------------------
  0 1 1 1 1 1 1 1 0   S
0 0 0 0 0 0 0 0 1     C
  1 0 0 0 0 0 0 0 0   Final sum
```

* See manufacturer's data books.