Digital Techniques for Technicians Levels 2 and 3

GRANADA
TEC
TECHNICIAN SERIES

McAllister

Digital Techniques for Technicians Levels 2 and 3

J. McAllister

Lecturer, Kingston College of Further Education

GRANADA
London Toronto Sydney New York

Granada Publishing Limited – Technical Books Division
Frogmore, St Albans, Herts AL2 2NF
and
36 Golden Square, London W1R 4AH
515 Madison Avenue, New York, NY 10022, USA
117 York Street, Sydney, NSW 2000, Australia
60 International Boulevard, Rexdale, Ontario, R9W 6J2, Canada
61 Beach Road, Auckland, New Zealand

British Library Cataloguing in Publication Data

McAllister, J.
Digital techniques for technicians levels 2 and 3.
1. Digital electronics
I. Title
621.3815   TK7868.D5

ISBN 0 246 11787 7

First published in Great Britain 1983 by Granada Publishing Ltd

Printed in Great Britain by William Clowes (Beccles) Limited,
Beccles and London

Digital Techniques for Technicians Levels 2 and 3

Preface

This book is intended for students taking any or all of the three half units in Digital Techniques, revised versions of which appeared in 1981. A practical approach is adopted throughout; the student encouraged to handle and experiment with the various logic ICs discussed in the text, and practical exercises are provided in appropriate cases. Exercises of the 'connect point A to point B' type have been avoided. Instead the student is encouraged to refer to databooks, draw up wiring and circuit diagrams, and to present results in the form of graphs, tables and written conclusions. It is estimated that a student taking all three half units would spend about a third of his time on practical work.

Over one hundred and fifty exercises are provided. Many of these are routine tests of knowledge and comprehension, but a few seek to extend the student by asking him to find new applications, or to anticipate future work. Practical applications are emphasised even in academic exercises. Up to date techniques are represented, especially in chapter ten, where less space is devoted to magnetic core stores than would seem justified by the TEC unit, and more space is given to semiconductor memories and programmable logic. The chapter sequence broadly follows the TEC scheme, but chapter ten is out of sequence, as it was felt that it would be better for the student to gain some experience of flip-flops and registers before looking at memory systems. Chapter nine on logic families could be read at any time after chapters one to three, or could be referred to in the course of work on other chapters.

I would like to thank my colleagues at Kingston who kindly read and commented on parts of the manuscript. Any errors or omissions are of course entirely my own responsibility. My thanks are also due to Eamonn Higgins for preparing some of th drawings, and for his advice on drawing. Finally I would lik thank Amy for the thousands of cups of coffee which were ne in the writing of the manuscript.

Contents

1 Binary arithmetic

TEC U81/750

General objectives

To understand the processes involved in the conversion to and from binary notation and methods of adding, subtracting, multiplying and dividing binary numbers.

Specific objectives

After studying this chapter the student should be able to:

1.1 *Convert compound denary numbers to binary and vice versa.*
1.2 *Use sign and magnitude notation.*
1.3 *Use the 1's complement and the 2's complement of a binary number.*
1.4 *Add two compound binary numbers.*
1.5 *Subtract one binary number from another.*
1.6 *Multiply one binary number by another.*
1.7 *Divide one binary number by another.*
1.8 *Demonstrate that the basic arithmetical operations may be carried out by repeated addition or subtraction.*

The binary number system

The study of digital techniques begins with the binary number system which plays an essential part in the operation of all modern electronic machines. Although the binary system is not used for everyday arithmetic, it is important to master the principles in this chapter to understand the problems that arise in the design of digital circuits, and to appreciate the techniques used to solve these problems.

The denary system which man has developed for normal use is a positional system based on the number ten. The position of a digit indicates a power of ten, and the digits 0 to 9 indicate how many times that power is to be counted. Starting from the least significant digit the powers of ten are, $10^0 = 1$, $10^1 = 10$, $10^2 = 100$...etc. The number 4321 is evaluated as $4 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 = 4000 + 300 + 20 + 1$.

While the denary (or decimal) system seems 'natural' to us from long familiarity, there is nothing inevitable about the choice of ten as a base. On the contrary, any number could be chosen, and mathematicians argue that the number twelve, for example, which has more factors than ten, would make a better base. Mathematics apart, the denary system has a serious handicap as a machine system; a machine working in denary would have to be able to take up any one of ten different states in order to represent each of the digits 0 to 9. Such a machine is possible in theory but would have so many practical limitations that it will not be considered further.

The binary system, based on the number two, needs only two distinct symbols, 1 and 0, therefore a two-state device is capable of distinguishing between them. This is a big advantage as two-state devices are readily available in electronics. To save writing the long-winded term 'binary digit', we use the word 'bit' to mean 1 or 0. Since a bit can take only the values 1 or 0, the maximum number that can be indicated in the units column is 1; therefore when a count of two is reached we have to move one place left and write 10. The sequence 00, 01, 10, 11, indicates a count of zero, one, two, three. The first two columns are now full so the only way we can indicate a count of four is to move left again and write 100.

The position of a bit in a binary number is proportional to a power of two. Starting from the least significant digit, the powers of two are, $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$...etc. The binary number 1010 is evaluated as $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 8 + 2 =$ denary 10. The denary value which corresponds to a given bit position is called the 'weight' of that bit. All the powers of two which could be accommodated in an eight bit number are set out in table 1.1, together with their denary weights.

Table 1.1

Power of two	7	6	5	4	3	2	1	0
Weight	128	64	32	16	8	4	2	1

Table 1.2 lists all the numbers that can be written in three bits. Notice that for numbers greater than one, more bits will be needed than denary digits. This is part of the price we pay for the relative simplicity of the binary system in terms of electronic circuitry.

Table 1.2

Binary	**Denary**
000	0
001	$2^0 = 1$
010	$2^1 = 2$
011	$2^1 + 2^0 = 3$
100	$2^2 = 4$
101	$2^2 + 2^0 = 5$
110	$2^2 + 2^1 = 6$
111	$2^2 + 2^1 + 2^0 = 7$

Using three bits we can write eight (i.e. two to the power of three) numbers. In general, using N bits we can write 2^N numbers, i.e. all the numbers from 0 to $2^N - 1$.

Exercise 1.1 Write down all the binary numbers that can be accommodated in four bits. Show the corresponding denary numbers as sums of powers of two as in table 1.2.

Binary to denary conversion

To convert a binary number to denary, add up all the powers of two which correspond to a 1 bit in the binary number. With practice this can be done mentally, but a systematic method is always advisable for large numbers. One such method is shown in table 1.3 for the conversion of binary 110101 to denary.

Table 1.3

Bit	**Power of two**	**Times**	**Result**
5	$32=2^5$	1	32
4	$16=2^4$	1	16
3	$8=2^3$	0	0
2	$4=2^2$	1	4
1	$2=2^1$	0	0
0	$1=2^0$	1	1

Add results: $32+16+0+4+0+1=53$.

Exercise 1.2 Convert the following binary numbers to denary, (a) 1101, (b) 1111, (c) 101110, (d) 111111.

Denary to binary conversion

Two methods will be given. A special circuit could be designed (a 'hardware solution'), or a computer program written (a 'software solution'), to implement either method. The student is free to pick the method which suits him best.

METHOD 1

First find the highest power of two which is **less** than the denary number (table 1.1 will be useful). Take the number 44 for example. From table 1.1, $2^5=32$, and $2^6=64$, so the highest power of two is the fifth power. Remembering that the bits are numbered from 0 upward, this tells us that the binary number has six bits, i.e. it is a number of the form 1*xxxxx*, where *x* indicates a bit still to be found. Now subtract 32 from 44 and test whether the next highest power of two can be taken from the remainder; if the answer is 'yes' then subtract it and put a 1 in the binary number; if 'no' then do not subtract anything and put a 0 in the binary number. Continue in this way down to 2^0. Table 1.4 illustrates the method for the number 44.

Table 1.4

Remainder	**Power of two**	**Subtract**	**Binary**
44	$2^5=32$	Yes	1
12	$2^4=16$	No	0
12	$2^3=8$	Yes	1
4	$2^2=4$	Yes	1
0	$2^1=2$	No	0
0	$2^0=1$	No	0

Read the binary number from the top: denary 44 = binary 101100.

METHOD 2

This method consists of repeated division of the denary number by two. The remainder (0 or 1) after each division is noted, and when read off in *reverse* order, the list of remainders gives the binary number.

Example 1.1 Convert denary 44 to binary.

Answer
$$\begin{aligned}44\div 2&=22+0\\22\div 2&=11+0\\11\div 2&=\ \ 5+1\\5\div 2&=\ \ 2+1\\2\div 2&=\ \ 1+0\\1\div 2&=\ \ 0+1\end{aligned}$$

The remainders in the right hand column are read from the bottom to give the binary number 101100 as before. With practice the division can be done mentally and only the remainders written down, but a systematic method such as that shown in example 1.1 is better for a beginner.

Exercise 1.3 Convert the following to binary, (a) 29, (b) 77, (c) 121, (d) 233.

Binary fractions

Fractions need present no problem if we remember that a fraction can be written as a **negative** power of the base. Thus in denary,

$$0.1=1/10=10^{-1}$$
$$0.01=1/100=10^{-2}\text{...etc.}$$

In binary we use negative powers of two, thus,

$$0.1=\tfrac{1}{2}=2^{-1}$$
$$0.01=\tfrac{1}{4}=2^{-2}$$
$$0.001=\tfrac{1}{8}=2^{-3}\text{...etc.}$$

It would be illogical to refer to the point in a binary fraction as a 'decimal point' because the word decimal implies the base ten. We will call it a binary point. Table 1.5 shows some binary fractions with corresponding powers of two and denary equivalents.

Table 1.5

Binary fraction	0.1	0.01	0.001	0.0001
Power of two	−1	−2	−3	−4
Denary	0.5	0.25	0.125	0.0625

Notice that a bit placed a given number of places after the binary point represents a much larger number than a denary digit the same number of places after the decimal point. This is because a given negative power of two represents a much larger number than the same negative power of ten. Two important consequences of this affect the accuracy to which numbers can be stored in a computer.

(1) Many binary places are needed to obtain high accuracy.
(2) Some denary fractions cannot be represented exactly in a finite number of bits.

Exercise 1.4 What is the largest denary (a) integer, and (b) fraction which can be represented in binary in eight bits?

Conversion of compound binary numbers to denary

To convert a compound binary number to denary, simply add up all the powers of two as for integers, but remember that bits after the binary point represent negative powers of two.

Example 1.2 Convert to denary, (a) 0.11, (b) 101.01, (c) 11.101.

Answers (a) $0.11 = 2^{-1}+2^{-2}$
$= ½ + ¼ = ¾$ or 0.75
(b) $101.01 = 2^{2}+2^{0}+2^{-2}$
$= 4 + 1 + ¼ = 5¼$ or 5.25
(c) $11.101 = 2^{1}+2^{0}+2^{-1}+2^{-3}$
$= 2 + 1 + ½ + ⅛ = 3⅝$ or 3.625

Exercise 1.5 Convert the following to denary, (a) 10.11, (b) 11.1111, (c) 110.0111, (d) 1001.1001.

Conversion of compound denary numbers to binary

To convert compound denary numbers to binary we can use either method 1 or a modified form of method 2 as given for integers. In both cases it is best to treat the integral and fractional parts separately.

Example 1.3 Convert 7.8125 to binary.

Answer The integer 7 converts to binary 111. We use method 1 to convert the fraction as follows:

Remainder	**Power of two**	**Subtract**	**Binary**
0.8125	$2^{-1}=0.5$	yes	1
0.3125	$2^{-2}=0.25$	yes	1
0.0625	$2^{-3}=0.125$	no	0
0.0625	$2^{-4}=0.0625$	yes	1
0.0	(when we get zero remainder the fraction is exact)		

Therefore 7.8125 = 111.1101 in binary.

If the denary fraction is not equivalent to an exact binary fraction then the process of subtracting ever decreasing powers of two will go on indefinitely. We have to decide how many binary places we need, work the answer out to one extra place, and round up or down according to whether the extra digit is 1 or 0.

Using method 2, conversion of a fraction is accomplished by successive *multiplication* by two.

Example 1.4 Convert 0.875 to binary.

Answer The 'carry' (1 or 0) after each multiplication is written apart from the rest of the work. Only the fraction is multiplied at each stage. The binary fraction is read off from the top of the carry column.

Fraction	**Carry**
$0.875 \times 2 = 1.750$	1
$0.750 \times 2 = 1.5$	1
$0.5 \quad \times 2 = 1.0$	1
0 (Again zero remainder indicates an exact fraction.)	

Therefore 0.875 = binary 0.111.

Example 1.5 Convert 9.5627 to binary.

Answer The integer 9 converts to 1001.

Fraction	**Carry**
$0.5627 \times 2 = 1.1254$	1
$0.1254 \times 2 = 0.2508$	0
$0.2508 \times 2 = 0.5016$	0
$0.5016 \times 2 = 1.0032$	1
$0.0032 \times 2 = \ldots$etc.	

By inspection we can see that 0.0032 will have to be multiplied by two many times before there is another carry, so we end the calculation at this point, but note that the fraction is not exact in binary.

Therefore $9.5627 \triangleq$ binary 1001.1001.

Representation of negative numbers

A computer must handle both positive and negative quantities, i.e. it must be able to represent the sign of the number as well as the number itself. There are three ways to do this,

(1) Sign and magnitude.
(2) 1's complement.
(3) 2's complement.

Sign and magnitude

In this notation the first bit is taken to indicate the sign. By convention 0 is taken to indicate a positive, and 1 to indicate a negative number. Thus 0101 is +5, and 1011 is −3. All the numbers which can be represented in three bits in sign and magnitude are set out in table 1.6.

Table 1.6

Negative numbers		**Positive numbers**	
Binary	Denary	Binary	Denary
100	−0	000	0
101	−1	001	1
110	−2	010	2
111	−3	011	3

One of the disadvantages of sign and magnitude notation is that there are two ways of writing zero; $100 = -0$ is, of course, identical to $000 = 0$. Secondly, instead of the expected $2^3 = 8$ numbers, we can only represent seven different numbers in three bits. Even more serious from a mathematical viewpoint is the fact that if a negative number is added to its positive counterpart the result is not zero. Sign and magnitude is also more complex to implement in hardware than other notations, but it has some advantages in circuits for multiplication and division.

Exercise 1.6

(a) Write down all the numbers which can be represented in sign and magnitude using four bits.
(b) Convert the following sign and magnitude numbers to denary, (i) 10011, (ii) 11111, (iii) 01111, (iv) 10000.
(c) How many different numbers can be represented in sign and magnitude using five bits?

1's complement

The complement of 0 is 1, and the complement of 1 is 0. To convert a positive binary number to its 1's complement, or negative form, simply complement each bit. Table 1.7 shows all the numbers which can be represented in 1's complement using three bits.

Table 1.7

Binary	**Denary**
100	−3
101	−2
110	−1
111	−0
000	0
001	1
010	2
011	3

Notice that again there are two ways of writing zero. However, 1's complement numbers when added to their inverses do at least give a result of zero (or $111 = -0$). Before we examine the 2's complement we need to do some binary addition.

Exercise 1.7 Write down all the numbers which can be represented in 1's complement notation using four bits.

Binary addition

We state four basic rules,

(1) $0 + 0 = 0$
(2) $0 + 1 = 1$
(3) $1 + 0 = 1$
(4) $1 + 1 = 0$, and carry 1.

Example 1.6

$$\begin{array}{r} 1001 \\ +\ \underline{0101} \\ =\ 1110 \end{array}$$

Example 1.7

$$\begin{array}{r} 1011 \\ +\ \underline{0111} \\ =10010 \end{array}$$

In example 1.7 there were three 1's to be added in the second column due to the carry from column one. When three or more binary numbers are to be added it can be a problem to decide which column should receive the carry. There is a fairly simple solution to this problem. First count the number of ones which occur in the column to be added; then express this number in binary. The least significant bit of this number is the sum bit, to be entered in the column being added; the other bits indicate which column(s) are to receive a carry. This procedure works for any number of bits, but in practice machines add numbers in pairs so there are never more than three bits to be added, two bits from the numbers to be added and one carry bit. Written as two-bit binary numbers the possible sums are 00, 01, 10 and 11, and in each of these the most significant bit is the carry and the least significant bit is the sum.

Example 1.8

$$\begin{array}{r} 111011 \\ +\ \underline{001111} \\ =1001010 \end{array}$$

In table 1.8 the 'partial sums', i.e. the results of adding each individual column are written as two-bit numbers to indicate the sum and carry.

Table 1.8

Column	Partial sum	Carry	Sum
0	10	1	0
1	11	1	1
2	10	1	0
3	11	1	1
4	10	1	0
5	10	1	0
6	01	0	1

Exercise 1.8 Convert to binary and add (a) 14+9, (b) 7+9, (c) 15+15, (d) 31+19.

Two's complement

A disadvantage of both the previous forms of negative numbers is that they do not provide a true additive inverse of a positive number. Let x be any positive number. We define the additive inverse of x to be a number y, such that $x+y=0$. In denary arithmetic the additive inverse of x is of course $-x$, since $x+(-x)=0$. We need a method of representing $-x$ in a machine in such a way that this property holds for all numbers, x.

We will first define the 2's complement of a binary number and then show that it is the additive inverse. The rule is simple; take the 1's complement and add 1.

Example 1.9 Find the 2's complement of 0111.

Answer 1's complement of 0111 = 1000. Add 1 to get 1001.

$$\text{Check answer,}\quad \begin{array}{r} 1001 \\ +0111 \\ \hline =0000 \end{array}$$

(the carry from bit four is ignored).

It is important to decide at the start how many bits are to be used and to keep this fixed. In practice the number of bits is fixed by the particular machine in use. All the numbers which can be written in four bits using 2's complement are shown in table 1.9.

Table 1.9

Negative		Positive	
Denary	**Binary**	**Denary**	**Binary**
−8	1000		
−7	1001	+7	0111
−6	1010	+6	0110
−5	1011	+5	0101
−4	1100	+4	0100
−3	1101	+3	0011
−2	1110	+2	0010
−1	1111	+1	0001
		0	0000

Note (i) there is no additive inverse for −8.
(ii) we can write $2^4=16$ different numbers in four bits.

Exercise 1.9 Write down the 2's complement form of all the numbers that can be written in five bits. Which of these does not have an additive inverse?

Binary arithmetic with signed numbers

Addition of positive numbers is the same in all three notations, and follows the rules already outlined. A point to note is that any carry or 'overflow' from the most significant bit indicates an error, which is one of the reasons why we earlier insisted on writing, for example, 00011 and not simply 11 when using five bits.