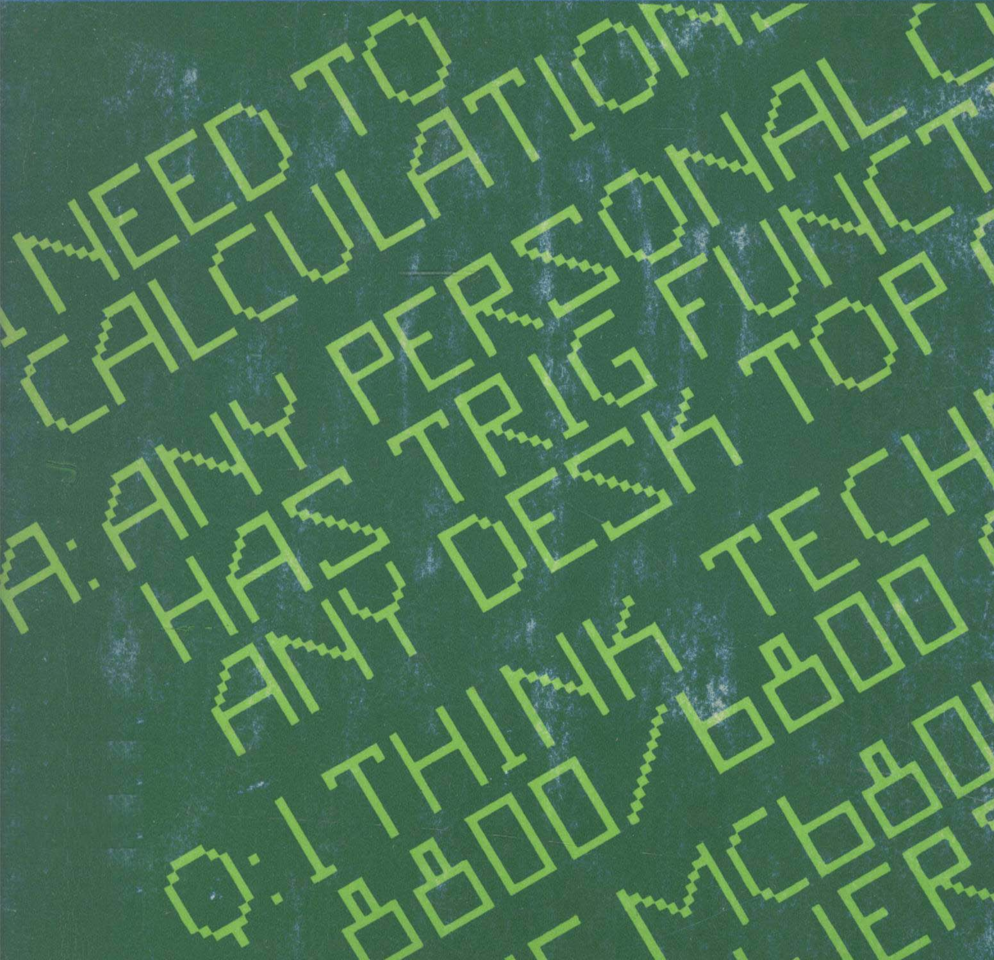
dilithium PRESS

HOME COMPUTERS: 2^{10} Questions & Answers Volume 2: Software

Rich Didday



HOME COMPUTERS: 2^{10} Questions & Answers Volume 2: Software

Rich Didday

dilithium PRESS P.O. Box 92, Forest Grove, Oregon 97116

The technical information, statements about specific products, and descriptions of specific computer languages in this book are based on fact and are believed to be accurate. The characters are fictional, and are not intended to portray any real persons, living or dead. Some fragments of conversations are based on real conversations.

HOME COMPUTERS: 2^{10} QUESTIONS AND ANSWERS
Volume 2: Software

Rich Didday

ISBN 0-918398-01-0

Library of Congress Cataloging in Publication Data

Didday, Richard L
Home computers.

Bibliography: p.
Includes index.
CONTENTS: v. 1. Hardware.—v. 2. Software.
1. Microcomputers—Miscellanea. I. Title.
TK7885.4.D53 001.6'4'04 77-9285
ISBN 0-918398-00-2 (v. 1)
ISBN 0-918398-01-0 (v. 2)

PRINTED IN THE UNITED STATES OF AMERICA

PREFACE

What's in this book

This is the second volume of the two-volume set 2^{10} *Questions and Answers About Home Computers.* This book has two main purposes. First, it's intended to give you a real feeling for what it's like to write programs for home computers. Second, it's intended to give people who come with an interest, but no specialized knowledge, a general background in programming (in general) and in programming microcomputers (in specific). Enough of a background so that you'll have no trouble understanding articles about advanced programming projects and software techniques in the computer hobbyist magazines, ads describing software products, and people who do have specialized software knowledge. There is *no* attempt to push specific products, nor is there an attempt to cover advanced, esoteric software techniques. The whole idea is to get you to the point where you can make your own, informed decisions about what software products you want, and what projects to attempt.

A glance at the Table of Contents and a few minutes of flipping through the book will show you how these purposes are accomplished. The material is expressed in the form of a dialog. One participant (A) has a substantial background in computing and home computing, the other (Q) is a bright, interested newcomer. In addition, an Editor adds occasional fine points and clarifications. Diagrams, Tables, and Appendices are used to provide additional information in a compact form.

Although there is a definite structure to the whole book, so that by reading from start to finish, you will find an orderly progression of material (the organization of specific microprocessors → programming in machine language → programming in assembly language → programming in Basic → generalizations about programming → things you can and can't expect to do with a home computer), the book is also designed to make it easy to skip around, covering topics of special interest to you (see Figure 0). Regardless of your specific background, I'm sure you'll find many topics of interest, simply because computing itself is such a rich, diverse, fascinating, lively activity.

And how I came to write it

I've been interested in computing for a long time (I wrote my first programs in 1964, soldered my first digital logic chips together in 1965), and when affordable computers became available, I jumped at the opportunity to have my own system, free from the constraints imposed by the companies and universities whose computers I had been using. As I went around talking to people, dropping into newly opened computer stores, going to conventions, I was struck by the high proportion of people I met who had substantial backgrounds in electronics and computer hardware — there seemed to be relatively few newcomers. ''Why is that?'', I wondered. Further investigation revealed what should have been obvious. Although dealing with computers is inherently no more difficult than working on your car, and although programming in a language like Basic is easily and commonly taught to children, the mystique and special jargon built up around computers serves as a barrier. People can't be expected to take a deep interest in computing if they can't really understand what's being said and if they have no way of knowing what they're in for.

My first plan was to produce a short book of answers to questions like ''What is a buffer?'', ''What is an array?'', ''What is the difference between machine and assembly language?'', ''What is flip-flop?'', etc. That failed because the terms, although each one is simple in itself, are densely interconnected and make little sense out of context. The solution was to write a more extensive book consisting of coherent conversations involving the terms to be explained. But now another problem arose. If enough material was included to make the book of real use to people with wide differences in background, that is, if it was to cover both the hardware *and* the programming aspects of home computing, the cost to the purchaser would be objectionably high. Fortunately, the material lent itself quite naturally to a division in two, so the person who is interested mainly in hardware aspects of home computing as well as the person who is interested mainly in programming can each find the material they want in a moderately priced book (Vol. 1 and Vol. 2, respectively).

Overall, it turned out to be much more work than I'd bargained for, but I feel it will be well worth the effort if it succeeds in helping people over the initial barriers, enabling them to discover the joys and excitements of computing.

Thanks to . . .
Many, many thanks to ''Nick'' Nichparenko, Dan Ross, Dennie Van Tassel, John Craig, the Byte Shop of Santa Cruz, Merl Miller, Rob Walker of Intel, Margaret Kinstler, Raymond Langsford, the late Walter Orvedahl, Rex Page, the lady in Albuquerque, and William Makepeace Thackeray.

December 19, 1976
Santa Cruz, California

Rich Didday

EDITOR'S INTRODUCTION

This book and the companion volume 2^{10} *Questions and Answers About Home Computers: Vol. 1* represent a heavily edited transcription of nine days of conversation focusing on home/hobby computing. Among the steps taken to create a useful book from the raw recordings are these:

—each Day's conversation has been grouped into sections, and each section given a title.

—presentable figures have been drawn from the rough sketches provided by the participants.

—the material has been cross-referenced to aid the reader in quickly finding related material.

—editorial insertions (denoted by [square brackets]) have been made to explain fine points or to correct potentially misleading statements.

—an extensive set of Appendices, a Bibliography, and an Index have been provided.

—much of the conversation has been condensed. Some has been converted into tabular form. Portions of questionable relevance have been deleted. In other cases, material representing qualifications of previous statements has been made into parenthetical phrases and inserted at the appropriate points.

—a numbering scheme has been placed on the material. Questions are numbered only where either the question or the answer (or both) add substantively to the discussion. Unnumbered material has been retained where necessary for continuity or where it gives an indication of the nature of the interaction between the participants.

The conversations divided into two self-sufficient volumes very naturally, with a few minor exceptions. The switch from a concern with hardware issues to a concentration on software occurs very gradually during Day 5, so gradually that there seems to be no single point at which the change can be said to occur. For this reason, the first three sections of Day 5 have been included in both volumes. In addition, some material, notably the discussion of binary, octal, and hexadecimal number systems, is needed in both volumes. Hence, a summary of the relevant information from the First Five Days has been included in the Last Five Days. In addition, there is a duplication of some of the Appendices, the Bibliography, and the Index. This last duplication is intended to aid those readers who possess both volumes.

The Editor

TABLE OF CONTENTS

SUMMARY OF RELEVANT INFORMATION FROM VOL. 1

The conceptual computer

There are two main aspects of computer systems: **hardware** and **software.** The term **hardware** refers to those parts of the computer system that you can touch — the integrated circuit components, the wires, switches, lights, keyboard, power supply, the chassis the subassemblies are mounted in, so forth. **Software** refers to entities that exist as patterns, i.e. programs, data, stored values.

All digital computers have a hardware organization that fits the conceptual framework shown in Figure E1. Programs, data, and temporary values are stored in **memory.** Memory is organized as a number of **locations,** each of which has a unique **address.** On most microcomputers, each location in memory stores one 8-bit binary pattern (a **byte**), and each address is a 16-bit binary value.

The **controller** (alternate names: **processor, microprocessor, central processing unit, cpu**) takes instructions (i.e. statements in a program) from memory and carries them out. Each command in a microprocessor's **instruction set** (see Appendices for the 8080 and 6800 instruction sets) causes the controller to take a specific action (e.g. store a value in a specific place in memory, get a value from a specific place in memory, perform a test on a value and if the test succeeds take the next instruction from some specific place in memory, halt, send a value to a specific input/output device, etc.).

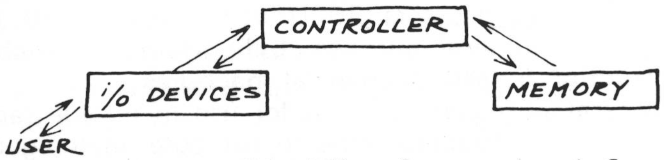


Figure E1 The Conceptual Computer

The **input/output (i/o) devices** are used to communicate values from and to the computer. Typical i/o devices are front panel switches and lights, terminals (such as Teletypes, keyboards and TV interfaces, video terminals, etc.), and external storage devices (such as cassette tape recorders, discs, etc.).
Figure E2 is an elaboration of Figure E1 and shows how the parts of the conceptual computer are tied together in most microcomputer systems.

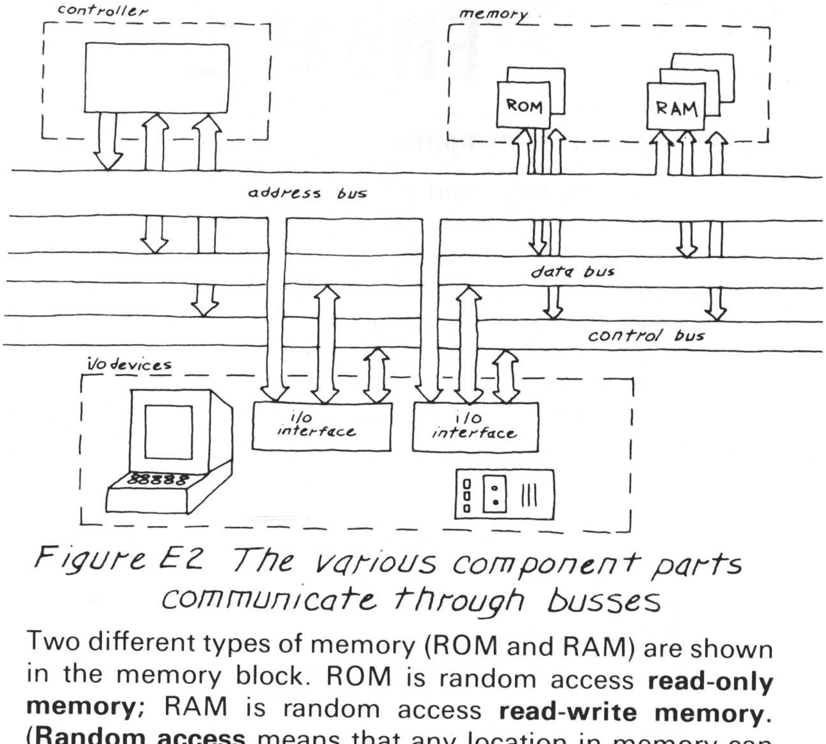


Figure E2 The various component parts communicate through busses

Two different types of memory (ROM and RAM) are shown in the memory block. ROM is random access **read-only memory;** RAM is random access **read-write memory.** (**Random access** means that any location in memory can be accessed in the same amount of time; an example of a memory device which is not random access is cassette tape in which values are accessed **sequentially.**)
The controller consists of a microprocessor chip plus any necessary support chips (e.g. circuitry to provide synchronization signals, devices which serve to connect the microprocessor to other parts of the conceptual computer, etc.). Most microprocessors (8080, 8085, 6800, Z-80, 6502, etc.) are **general purpose, byte-oriented, variable instruction length, sequential machines.**

general purpose: intended for use in a wide range of applications; able to compute anything that is **computable** (see Q982).

byte-oriented: the basic unit of storage is one **byte** (8 binary digits); basic machine commands are one byte long; memory is accessed one byte at a time.

variable instruction length: although each basic machine command is one byte long, a complete instruction may require additional information, thus instructions can be one, two, or three bytes in length. For instances, a Jump instruction is (typically) three bytes long — one byte to specify the command Jump and two additional bytes to specify a (16-bit) memory address (i.e. two additional bytes which specify where to Jump to).

sequential: a sequential computer carries out one instruction at a time. Computer systems which contain more than one processor and can therefore carry out more than one instruction at a time are said to be capable of **parallel processing.**

The three major components of the computer shown in Figure E2 are interconnected by groups of wires called **busses.** The **address bus** communicates 16-bit values from the controller to memory and i/o. The **data bus** communicates 8-bit values from any component to the others. The **control bus** is used to communicate control and synchronization signals among the components. For example, if the controller is carrying out an instruction which says to store a value in a specific spot in memory, the busses are used in the following way: The controller places the address of the desired memory location on the address bus. All devices on the address bus receive the signal, and it is up to circuitry on each device to determine if it should respond. Thus, only the block of memory which contains the specified address will allow the signals on the other busses to affect it. The controller places the value which is to be stored on the data bus and it too is communicated to all devices on the busses, including, of course, the block of memory previously mentioned. In addition, the controller places a signal on the control bus which specifies that a **memory write** (as opposed to **memory read**) operation is to take place, and this causes the specified memory location to store the value appearing on the data bus in the memory location whose address appears on the address bus. Since (most) microprocessors use **synchronous busses,** no further communication is necessary (i.e. in an **asynchronous bus,** the controller would wait until it received a signal informing it that the memory operation had taken place). (For a description of what happens if the memory used in the system responds so slowly that the controller can demand another memory operation before the first is completed, see Q97-99 in Vol. 1.)

Number Systems

There are four number systems in common use among people who program in machine and assembly language — **decimal, binary, octal,** and **hexadecimal.** All are **positional** number systems, which means that the position of a symbol in the number determines its value (the "3" in 13 means three, but the "3" in 327 means three hundred).

In the **decimal** number system (base 10), there are 10 different symbols,

0 1 2 3 4 5 6 7 8 9

and each column in a number is associated with a different power of 10.

In **binary** (base 2), there are 2 different symbols,

0 1

and each column in a number is associated with a different power of 2.

In **octal** (base 8), there are 8 different symbols,

0 1 2 3 4 5 6 7

and each column is associated with a different power of 8.

In **hexadecimal** (base 16), there are 16 different symbols

0 1 2 3 4 5 6 7 8 9 A B C D E F

and each column is associated with a power of 16.

Binary is used for the obvious reason — computers are built of two-state elements. **Octal** and **hexadecimal** are used because they provide a compact (hence easily re-memberable) representation for binary values, and the conversion to and from binary to octal or hexadecimal is extremely simple.

The notation y_x indicates that the string of digit symbols y represents a number expressed in base x. Thus 11_2 means "the binary value 11" ($11_2 = 3_{10}$).

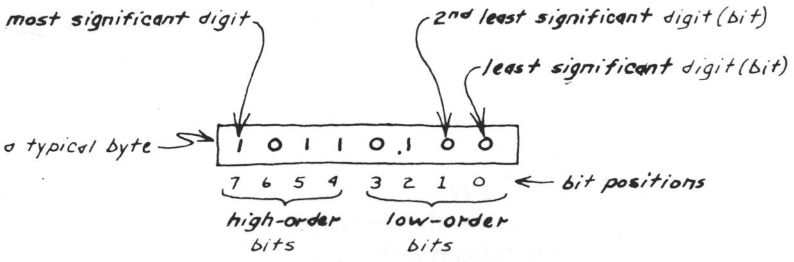
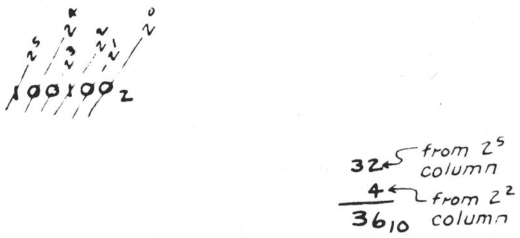


Figure E3 Notation

Conversions

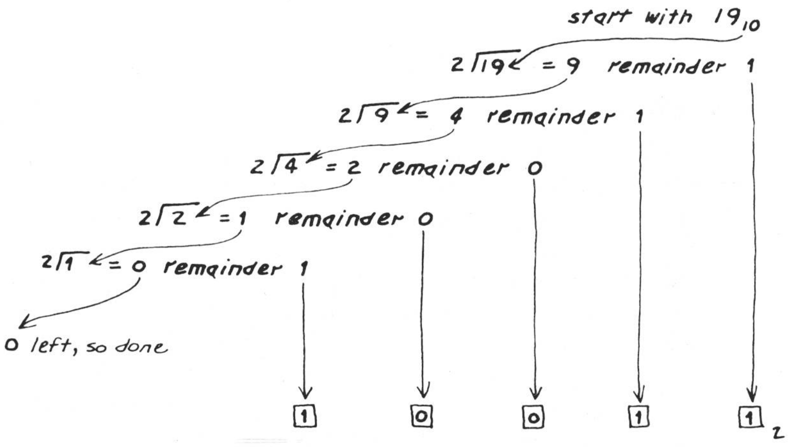
binary to decimal
The rightmost digit of a binary number (also called the **least significant digit**) tells how many (i.e. either 0 or 1) 2^0's there are in the number. The next digit to the left tells how many 2^1's there are. Thus, to convert a binary value to decimal, start with the rightmost digit. If it's a 1, write down 1_{10} ($=2^0$), otherwise go to the next digit to the left. If the second digit from the right is a 1, write down 2 ($=2^1$), otherwise go on. If the third digit from the right is a 1, write down 4 ($=2^2$), otherwise go on. After repeating this process until all digits in the binary number have been dealt with, add up the numbers you wrote down — their sum is the equivalent decimal value. (For powers of 2, 8, and 16 see Appendix — Powers.)



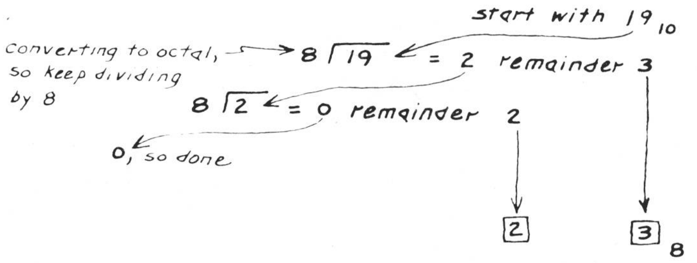
octal to decimal, hexadecimal to decimal
The procedures for converting from octal or hexadecimal to decimal are the same as for binary except that the value in each digit position is multiplied by the corresponding power of 8 (for octal) or 16 for (hexadecimal) instead of 2.

$8^2 8^1 8^0$	$16^2 16^1 16^0$
123_8	140_{16}
$3 \times 1 = 3$	0×1
$2 \times 8 = 16$	$4 \times 16 = 64$
$1 \times 64 = 64$	$1 \times 256 = 256$
83_{10}	320_{10}

decimal to binary
Repeatedly divide the decimal value by 2, writing down the remainders from *right to left* as you go, stopping when you are left with zero as the next dividend.



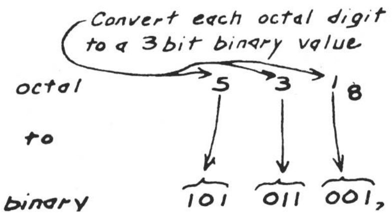
decimal to octal, decimal to hexadecimal
Follow the same procedure as for converting decimal to binary, but repeatedly divide by 8 or 16.

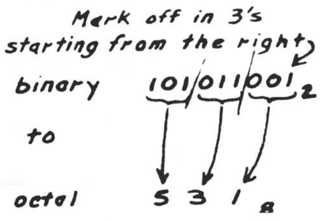


binary to octal
Each octal digit corresponds (exactly) to three binary digits. Starting from the right, mark off each set of three binary digits, convert each set (in place) to the equivalent octal digit.

3-bit binary value	equivalent octal digit
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

octal to binary
Replace each octal digit (in place) by the equivalent three digit binary value.





binary to hexadecimal
Each octal digit corresponds to (exactly) four binary digits. Starting from the right, mark off each set of four binary digits, convert each (in place) to the equivalent hexadecimal digit:

4-bit binary value	equivalent hexadecimal digit
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

hexadecimal to binary
Replace each hexadecimal digit (in place) by the equivalent four digit binary value.