Principles of Computer Design

Leonard R. Marino

Principles of Computer Design

Leonard R. Marino

San Diego State University

COMPUTER SCIENCE PRESS

Computer Science Press
1803 Research Blvd.
Rockville, Maryland 20850

1 2 3 4 5 6 Printing Year 91 90 89 88 87 86

Library of Congress Cataloging in Publication Data

Marino, Leonard R.
Principles of computer design.

Bibliography: p.
1. Electronic digital computers—Design and construction. 2. Computer architecture. I. Title.
TK7888.3.M3524 1985 621.3819'582 84-23812
ISBN 0-88175-064-6

Throughout the text, computers are discussed from the viewpoint of the designer, even though the primary objective is to provide an understanding of computer structure, and not necessarily to train designers. The design viewpoint is adopted because it offers the truest insights into the current structure of computers, and provides the best basis for understanding future developments. Also, the inclusion of practical design techniques facilitates the construction of meaningful exercises.

Microcircuit design (i.e., the design of a circuit to be implemented on the surface of an IC chip) and macrocircuit design (i.e., the design of systems using ICs as components) are given equal emphasis in the text. Microcircuit design has become increasingly important to computer system designers in recent years, as the size of the system that can be implemented on a single IC chip has increased. At the same time, macrocircuit design has remained important, both because high-performance computers are constructed from smaller-scale (and faster) chips, and because no matter how large the system on a single chip may be, there will always be the need for systems that are much larger.

The emphasis throughout this text is on the principles of the design process rather than on the details of currently available devices or current practice. This does not imply, however, that the text is oriented toward theory rather than practice. In fact, the reverse is true. Practical systematic design techniques are presented, and difficult issues such as loading, transmission line effects, and noise are considered. Specific devices are used for illustration, but the emphasis is on design principles that are independent of device details. After finishing the text, the student should be equipped to undertake design projects of considerable complexity, using any sort of digital components.

Chapter Summaries

Chapter 1 provides an overview of computer organization, design, and production. In Section 1.1 a particular computer architecture is described which is used in examples and exercises throughout the text. The 6800 microprocessor system architecture was selected for this purpose because it illustrates all of the important aspects of traditional (von Neumann) computer architecture, yet it is simple and uncluttered. It is also an architecture of continuing commercial importance, since all 68xx family processors are compatible with the 6800. Assembler language programming is introduced in this section, and is used throughout the text.

In Section 1.2 an important principle of computer science is introduced, that complexity in a digital computer arises not so much from the inherent complexity of the primitive components from which the machine is constructed (e.g., transistors are used simply as switches), but rather from the

overwhelming number of details that must be managed when interconnecting many of these simple components. One of the most important skills that a computer engineer must develop is the ability to manage this complexity of many details. The ideas of using multilevel models and modular design techniques to control complexity are introduced in this section, and applied throughout the text.

In Section 1.3 the major packaging levels, macrocircuit and microcircuit, are introduced, and the economics of computer production are discussed.

Chapter 2 provides a brief review of electric circuits, followed by an introduction to the fundamentals of digital electronics. Simple switching models of transistors are presented and used to analyze logic gate structures. The fundamental roles of capacitance and resistance in determining speed and power dissipation in digital circuits are discussed. The elementary physics and mechanics of integrated circuit operation and fabrication are presented.

A compact presentation of combinational network design is provided in Chapter 3. Several traditional topics are omitted, including formal Boolean algebra and tabular minimization of switching functions. These topics are no longer of fundamental importance in computer design, and are better left to an advanced course in logic design.

Digital arithmetic is presented in Chapter 4. Because of its fundamental importance, and because it is frequently a source of confusion for students, this topic is treated with particular care. A clear distinction is maintained between binary words and the numbers that they represent, and special attention is given to dealing with overflow and representing numerical relations.

Chapter 5 begins with a discussion of digital signalling, that is, the use of voltages to transmit information between components. The major classes of register transfer level components are then discussed: combinational networks, pulse generators, memory, and sequential networks. Microcircuit and macrocircuit implementations of each component are presented.

Chapter 6 describes systematic techniques for register transfer level design. Several design examples are presented in detail, including input-output controllers, a DMA controller, and the 6800 processor. Microprogramming is discussed in the last section.

Chapter 7 is devoted to topics associated specifically with macrocircuit design. Macrocircuit models for IC interconnections are described, and standard electrical and timing specifications are discussed. These models apply to all levels of macrocircuit design, including high-speed MSI/LSI-based systems, microprocessor-based systems, and high-performance systems constructed from custom LSI and VLSI components. Transmission line effects, which are crucial in high-speed systems, are considered in detail. Noise is also considered, but because of the limited background assumed, the discussion is necessarily qualitative. The objective is to provide an intuitive understanding of the principles involved.

Note to Instructors

One of the difficult aspects of teaching computer design is determining the proper amount of detail to include in lectures. A lecture with too much detail will be boring, while one with too little detail may be dismissed as superficial. The optimal amount and selection of detail depends upon the background of the students and the objectives of the course.

In order to allow sufficient latitude to accommodate different situations, a considerable amount of detail has been included in this text. It is not intended that all of this detail be laboriously presented in lectures, although at one time or another almost all of it has been included in my own lectures. It is intended, rather, that lectures focus on general principles, illustrated by selected examples, with much of the detail left as assigned or elective reading.

This text is suitable for a first course in computer design for students of computer science or computer engineering. The only prerequisite is completion of the standard lower division university physics and calculus sequences. With this minimal background, it should be possible to complete the text in two semesters.

The text is also suitable for a course that is preceded by one or more courses in electronics, logic design, or computer organization. Selected portions of the text may then be omitted or covered lightly, making a one semester course possible.

Acknowledgements

Many friends and colleagues have helped me in writing this book. I would especially like to thank Bill Brown, Alex Iosupovici, Chuck Seitz, and Jeffrey Ullman for reading parts of the manuscript and making many excellent suggestions. I am grateful to my brothers Al and Tony for frequent helpful discussions and valued advice. I am indebted to Tom Windeknecht and Hank D'Angelo, from whom I have learned a great deal, and who have significantly influenced the paths that I have taken. Thanks also to the hundreds of students who have sustained me with their interest and curiosity while reading draft versions of this text in the EE373, EE573, and EE503 courses at San Diego State University.

My wife suggested the following dedication for this book: "To my beautiful wife, Kay, and wonderful children, Jesse, Daniel, and Katie, in spite of whom this manuscript was completed." However, while it is conceivable that this work might have been completed sooner without the distractions of my family, it is far more likely that without them it would not have been completed at all. It certainly would not have been completed without Kay's generous support and understanding.

Quotations

I always enjoy reading quotations that provide a sense of history or capture the essense of an important idea with a brief statement. Here are several that are relevant to computer design.

Everything should be made as simple as possible, but not simpler.
Albert Einstein

Achilles: There must be some amazingly smart ants in that colony, I'll say that.
Anteater: I think you are still having some difficulty realizing the difference in levels here. Just as you would not confuse an individual tree with a forest, so here you must not take an ant for the colony. You see, all the ants in Aunt Hillary are as dumb as can be. They couldn't converse to save their little thoraxes.
Douglas Hofstadter

I regard programs as specific instances of mechanisms, and I wanted to express, at least once, my strong feeling that many of my considerations concerning software are, mutatis mutandis, just as relevant for hardware design.

The art of programming is the art of organizing complexity, of mastering multitude and avoiding its bastard chaos as effectively as possible.

(We should) restrict ourselves to simple structures whenever possible and avoid in all intellectual modesty "clever constructions" like the plague.

Program testing can be used to show the presence of bugs, but never their absence.
Edsger W. Dijkstra

As the result of the large capacity of computing instruments, we have to deal with computing processes of such complexity that they can hardly be constructed and understood in terms of basic general purpose concepts. The limit is set by the nature of our own intellect: precise thinking is possible only in terms of a small number of elements at a time. The only efficient way to deal with complicated systems is in a hierarchical fashion.
O. J. Dahl and C. A. R. Hoare

Inasmuch as the completed device will be a general-purpose computing machine it should contain certain main organs relating to arithmetic, memory-storage, control and connection with the human operator.

The utility of an automatic computer lies in the possibility of using a given sequence of instructions repeatedly.

A. W. Burks, H. H. Goldstine and J. von Neumann (1946)

The entire UNIVAC system is constructed of circuits which . . . have been designed as building blocks, and the entire computer is constructed around these blocks.

J. P. Eckert, et al (1951)

"*To be is to do*"—Socrates
"*To do is to be*"—Jean-Paul Sartre
"*Do be do be do*"—Frank Sinatra

Kurt Vonnegut

CONTENTS

Chapter 1

AN OVERVIEW

This chapter introduces the study of digital computers from three perspectives. Section 1.1 describes the general organization and operation of stored program computers. Section 1.2 discusses some of the issues involved in designing complex systems and presents general systematic methods for dealing with complexity. Computer design serves as a vehicle for the discussion, but the principles apply to the design of any complex system. Section 1.3 discusses the technologies involved in implementing and packaging digital computers. It is these matters more than any other that determine the cost of digital computers, and hence their impact on society.

1.1 COMPUTER ORGANIZATION [1-8]

In this section, the fundamentals of computer organization and programming are presented, using the 6800 microprocessor as an example. This particular processor was chosen because it is simple and yet it has all the fundamental characteristics of a modern processor. In addition, the 6800

architecture is likely to remain popular for many years, since it is used, with various modifications, by all of the 6800 family microprocessors. This family includes processors significantly more powerful than the 6800 (e.g., the 6809), as well as single-chip microcomputers (e.g., the 6805).

1.1.1 Fundamental Characteristics

In this section are described the most fundamental characteristics of computers. These characteristics are shared by essentially all digital computers.

Digital Representation of Information

Computers are machines that store and process information. Within a computer, information is represented by binary words. A *binary word* (or simply a word) is a sequence of bits. A *bit* is a "binary digit," either 0 or 1. For example, 01101011 is an 8-bit binary word.

Registers. Within a computer, words are stored in devices called *registers*. There are many different ways to implement registers (see Section 5.4). Regardless of the implementation, however, registers always serve the same purpose: to store binary words.

Placing a word into a register for storage is referred to as *loading* the register, or *writing* into the register. Examining a register to determine what word is stored is referred to as *reading* the register. The word in a register is not changed by reading. After loading a register, the word loaded remains in the register until another word is loaded.

A *memory* is a set of registers that share the same hardware for reading and writing. Memories are classified according to the type of read and write access that they provide.

Mathematically, a register is represented by a binary word variable. The value of the variable at any time is the word stored in the register at that time. We will normally use capital letters or strings of capital letters to denote binary word variables.

Individual bits of a binary word variable are binary variables. These are normally referred to by using the word variable name with a subscript to identify the specific bit, as illustrated in Figure 1.1. The subscript range associated with an n-bit word is usually $\langle n-1:0 \rangle$, as in the figure for the 8-bit register X.

In block diagrams, registers are represented by rectangular boxes. If it is necessary to represent individual bits (or "cells") of the register, this is done by subdividing the box, as in Figure 1.1. Each cell may contain either a 0 or a 1.

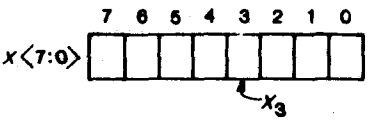

Figure 1.1 An 8-bit register

An 8-bit word is commonly referred to as a *byte*. The byte has become the standard unit for specifying storage capacities. For example, a computer may have 32 kilobytes (32K) of random access memory (RAM); a disk memory may have a capacity of 16 megabytes (16M). The prefixes "kilo" and "mega," when used to specify storage capacities, mean 2^{10} and 2^{20} (rather than 10^3 and 10^6 as in usual scientific notation). Hence, 32K is actually 32,768 bytes and 16M is actually 16,777,216 bytes..

For compactness, binary words are frequently represented by using *hexadecimal* notation. The bits of a word are separated into 4-bit groups, and each group is represented by one hexadecimal digit, as defined in Table 1.1. Hence, for example, the word 01101110 is represented as 6E and 1001010111000111 as 95C7.

Codes. In order to use binary words to represent information, it is necessary to define a *code* that specifies the particular information value that is represented by each word.

Table 1.1. Hexadecimal notation

4-bit word	Hex 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

The three general types of information that are most commonly represented in computers are: text, numbers, and programs.

Text. The most widely used code for representing text is the American Standard Code for Information Interchange (ASCII, pronounced ass-key), defined in Table 1.2.

Notice that the ASCII character set includes a number of "nonprinting" text and control characters. The SP (space) and CR (carriage return) characters, for example, are part of the text since they separate words and lines. Others such as SYN and ACK are not actually part of the text, but are used to control the transmission of text or the operation of input-output devices.

The ASCII code is a 7-bit code. The registers in most computers, however, are 8 bits or some multiple of 8 bits. For this reason, characters are normally stored as 8-bit words. The eighth bit is either ignored (e.g., made 0 always) or used as a "parity bit" for error detection. The value of the parity bit for a character is chosen so that the total number of 1's among the 8 bits is odd. Any single-bit error that occurs in the transmission of a character can then be detected. No matter which bit changes, the parity of the resulting word is even.

Numbers. The fundamental code for representing unsigned integers is the *base 2* code (abbreviated B2). If $[X]_2$ denotes the integer represented by the word $X\langle n-1:0\rangle$ under the B2 code, then

$$[X]_2 = X_0 \cdot 2^0 + X_1 \cdot 2^1 + \cdots + X_{n-1} \cdot 2^{n-1} \tag{1.1}$$

For example, $[1101]_2 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 = 13$.

The most common code for representing signed integers is *two's complement* code (2C code). The 2C code is a variation of the B2 code. Arithmetic under the B2 and 2C codes is discussed in Chapter 4.

Real numbers are represented by fixed-point and floating-point codes. These are also discussed in Chapter 4.

Programs. Programs are the third type of information commonly stored in computers. Programs are represented by a special code called *machine code*. Programs and machine code are discussed later in this section.

Register Transfer Operations

A computer processes information by doing operations on words. For example, the symbol $+_2$ denotes the operation that takes two words of the same

Table 1.2 ASCII code*

$X\langle 3:0\rangle$ \ $X\langle 6:4\rangle$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN.	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS←	CAN	(	8	H	X	h	x
1001	SKIP HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF→	FS	,	<	L	\	l	|
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

*Control character abbreviations:

NUL	null	DC1	device control 1
SOH	start of heading	DC2	device control 2
STX	start of text	DC3	device control 3
ETX	end of text	DC4	device control 4
EOT	end of transmission	NAK	negative acknowledge
ENQ	enquiry	SYN	synchronous idle
ACK	acknowledge	ETB	end of transmission block
BEL	bell	CAN	cancel
BS	backspace	EM	end of medium
HT	horizontal tabulation	SUB	substitute
LF	linefeed	ESC	escape
VT	vertical tabulation	FS	file separator
FF	form feed	GS	group separator
CR	carriage return	RS	record separator
SO	shift out	US	unit separator
SI	shift in	SP	space
DLE	data link escape	DEL	delete

length and produces a third word which represents their sum under the B2 code. The result word is one bit longer than the original words.

The word operation $\hat{+}_2$ is the same as $+_2$ except that the extra bit is dropped, so that the result is the same length as the operands. This is called a *fixed-range* operation. ("Overflow" is possible with fixed-range operations, but this need not be discussed here.)

In similar fashion $+_{2C}$ denotes addition under the 2C code, $-_{2C}$ is subtraction under the 2C code, $+_{FL}$ is addition under our floating-point code, and so on. The definition and implementation of arithmetic word operations are discussed in Chapter 4.

Logic operations on words are simpler (e.g., AND, OR, INVERT, SHIFT). No overflow is possible. These are discussed in Chapter 3.

A *register transfer* (RT) operation is a word operation with the operand words taken from specific registers and the result word loaded into a specific register. For example, $A \hat{+}_2 B \rightarrow C$ denotes the RT operation that takes the words in A and B, performs the fixed-range B2 addition operation, and puts the result into register C. A and B are called the *source* registers for the operation; C is called the *destination* register.

Only the destination register is changed by an RT operation. Source registers are unaffected. A particular occurrence of the operation $A \hat{+}_2 B \rightarrow C$ is illustrated in Figure 1.2. Notice that only register C is changed by the operation. There is no overflow in this example; the fixed-range operation is correct. The integers represented by the operand words are 5 and 7 and the integer represented by the result word is 12.

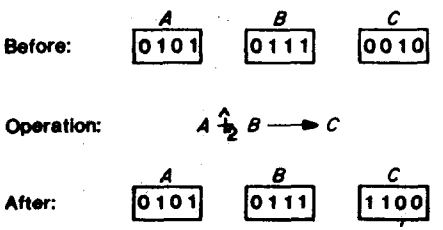

Figure 1.2 An RT operation

Program Controlled Operation

Computers perform complex information processing tasks by doing long sequences of simple RT operations. The RT operations that a computer is capable of performing are determined by the computer's instruction set. Each *instruction* specifies a particular operation, including the source and

destination registers for the operation. Instructions are represented by binary words, as defined by the machine code. The instruction set and machine code for a particular processor are presented in Section 1.1.3.

A *program* is a sequence of instructions. The information processing task that is performed by a computer is determined by a program that is stored in the computer's memory. The instructions that constitute the program are "fetched" (i.e., read from memory) and "executed" (i.e., the specified RT operation is performed) one at a time.

The processing task that a computer performs can be changed simply by changing the program that is stored in the computer's memory. This ability to "program" a computer to do different jobs is the most important economic characteristic of computers. The major benefits of this characteristic are two:

1. It allows a single expensive machine to be shared for many small tasks, none of which individually could justify the machine cost.

2. It allows the same machine to be used for many different applications, thus increasing the market size and allowing the machine to be mass-produced. This greatly reduces the cost per machine.

1.1.2 Structure and Operation

The structure of a typical small computer is shown in Figure 1.3. The P, M, IOC, and DMAC blocks communicate with each other via a single communication channel called the *system bus*. More complex communication structures are possible, involving multiple communication paths and switches (see Section 1.2.3). However, the same major components (P, M, IOC, IOD, and DMAC) are used in these more complex structures and the role played by each of these components in the operation of a computer remains essentially the same, regardless of the complexity of the communication structure.

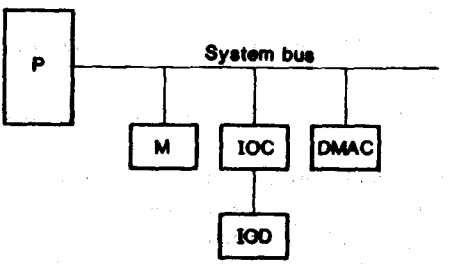

Figure 1.3 Single-bus architecture