Computer Systems

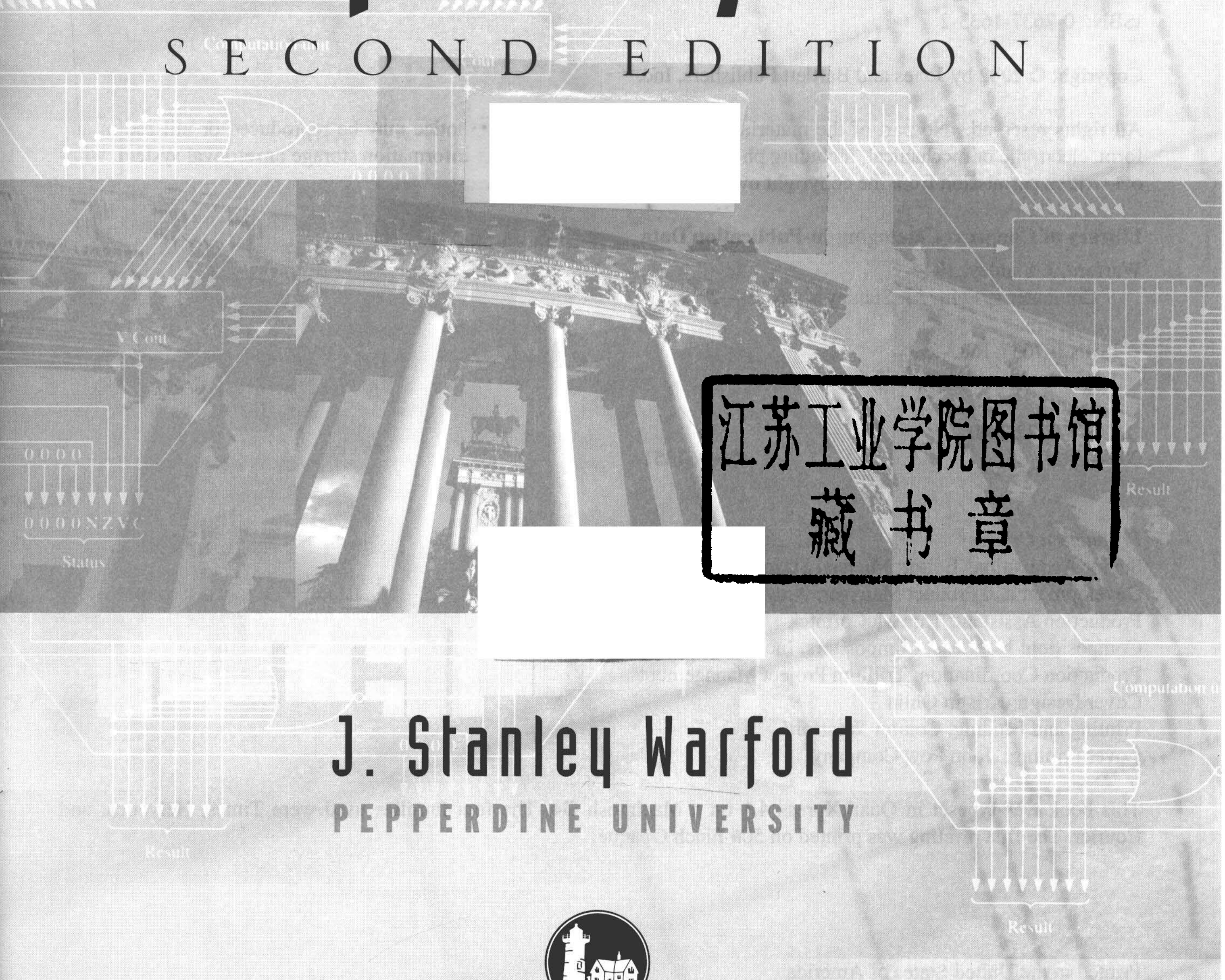
SECOND EDITION

J. Stanley Warford

TP333
W48 =2

Computer Systems

SECOND EDITION



J. Stanley Warford
PEPPERDINE UNIVERSITY

JONES AND BARTLETT PUBLISHERS
Sudbury, Massachusetts
BOSTON TORONTO LONDON SINGAPORE

World Headquarters
Jones and Bartlett Publishers
40 Tall Pine Drive
Sudbury, MA 01776
978-443-5000
info@jbpub.com
www.jbpub.com

Jones and Bartlett Publishers
Canada
2406 Nikanna Road
Mississauga, ON L5C 2W6
CANADA

Jones and Bartlett Publishers
International
Barb House, Barb Mews
London W6 7PA
UK

ISBN: 0-7637-1633-2

Library of Congress Cataloging-in-Publication Data

Warford, J. Stanley, 1944-
Computer systems / J. Stanley warford.—2nd ed.
p. cm.
ISBN 0-7637-1633-2
1. Computer systems. I. Title.

QA76.W2372 2001
004—dc21 00-067851

Production Credits
Senior Acquisitions Editor: Michael Stranz
Development and Product Manager: Amy Rose
Production Assistant: Tara McCormick
Composition: Northeast Compositors, Inc
Production Coordination: Trillium Project Management
Cover Design: Kristin Ohlin
Printing and Binding: Courier Westford
Cover printing: John Pow Company

This book was typeset in QuarkXpress 4.1 on a Macintosh G4. The font families used were Times, Helvetica, and Courier. The first printing was printed on 50# Finch Opaque.

Printed in the United States of America
06 05 04 03 02 10 9 8 7 6 5 4 3 2 1

Opcode	Mnemonic	Meaning	Addr. modes	Status bits
00000	STOP	stop execution	*U*	
00001	LOAD*R*	*R* := Oprnd	idsx	NZ
00010	STORE*R*	Oprnd := *R*	dsx	
00011	ADD*R*	*R* := *R* + Oprnd	idsx	NZVC
00100	SUB*R*	*R* := *R* - Oprnd	idsx	NZVC
00101	AND*R*	*R* := *R* AND Oprnd	idsx	NZ
00110	OR*R*	*R* := *R* OR Oprnd	idsx	NZ
00111	NOT*R*	*R* := NOT *R*	*U*	NZ
01000	ASL*R*	C := most significant bit; *R* := arithmetic shift left *R*	*U*	NZVC
01001	ASR*R*	C := least significant bit; *R* := arithmetic shift right *R*	*U*	NZC
01010	LDBYT*R*	*R* := byte Oprnd	idsx	NZ
01011	STBYT*R*	byte Oprnd := *R*	dsx	
01100	LOADB	B := Oprnd	idsx	NZ
01101	ADDSP	SP := SP + Oprnd	i	NZVC
01110	BR	PC := Oprnd	ix	
01111	BRLE	if NZVC ≡ ≤ then PC := Oprnd	ix	
10000	BRLT	if NZVC ≡ < then PC := Oprnd	ix	
10001	BREQ	if NZVC ≡ = then PC := Oprnd	ix	
10010	BRNE	if NZVC ≡ ≠ then PC := Oprnd	ix	
10011	BRGE	if NZVC ≡ ≥ then PC := Oprnd	ix	
10100	BRGT	if NZVC ≡ > then PC := Oprnd	ix	
10101	BRV	if V=1 then PC := Oprnd	ix	
10110	BRC	if C=1 then PC := Oprnd	ix	
10111	COMP*R*	*R* - Oprnd	idsx	NZVC
11000	JSR	SP := SP - 2; Mem[SP] := PC; PC := Oprnd	ix	
11001	RTS	PC := Mem[SP]; SP := SP + 2	*U*	
11010	RTI	return from interrupt	*U*	
11011	CHARI	byte Oprnd := input	dsx	
11100	CHARO	output := byte Oprnd	idsx	
11101	DECI	Oprnd := input	dsx	NZV
11110	DECO	output := Oprnd	idsx	
11111	HEXO	output := Oprnd	idsx	

Pep/7 Instruction Set.

This book is dedicated to my mother,
Susan Warford.

Preface

Computer Systems offers a clear, detailed, step-by-step exposition of the central ideas in computer organization, assembly language, and computer architecture. The book is based in large part on a virtual computer, Pep/7, which is designed to teach the basic concepts of the classic von Neumann machine. The strength of this approach is that the central concepts of computer science are taught without getting entangled in the many irrelevant details that often accompany such courses. This approach also provides a foundation that encourages students to think about the underlying themes of computer science. Breadth is achieved by emphasizing computer science topics that are related to, but not usually included in, the treatment of hardware and its associated software.

Summary of Contents

Computers operate at several levels of abstraction; programming at a high level of abstraction is only part of the story. This book presents a unified concept of computer systems based on the level structure of Figure P.1.

The book is divided into six parts corresponding to six of the seven levels of Figure P.1:

Level App7	Applications
Level HOL6	High-order languages
Level ISA3	Machine
Level Asmb5	Assembly
Level OS4	Operating system
Level LG1	Logic gate

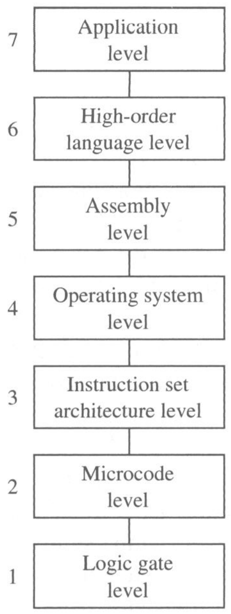


Figure **P.1**

The level structure of a typical computer system.

The text generally presents the levels top-down, from the highest to the lowest. Level ISA3, the instruction set architecture level, is discussed before Level Asmb5, the assembly level, for pedagogical reasons. In this one instance, it is more natural to revert temporarily to a bottom-up approach so that the building blocks of the lower level will be in hand for construction of the higher level.

Level App7 Level App7 is a single chapter on applications programs. It presents the idea of levels of abstraction and establishes the framework for the remainder of the book. A few concepts of relational databases are presented as an example of a typical computer application. It is assumed that students have experience with text editors or word processors.

Level Hol6 Level Hol6 consists of one chapter, which reviews the C++ programming language. The chapter assumes that the student has experience in some imperative language, not necessarily C++, such as Pascal or C. Advanced features of C++, including object-oriented concepts, are avoided. The instructor can readily translate the C++ examples to other common Level Hol6 languages if necessary.

The topic of recursion is treated in this chapter because it depends on the mechanism of memory allocation on the run-time stack. A fairly detailed explanation is given on the details of the memory allocation process for function calls, because this mechanism is revisited at a lower level of abstraction later in the book.

Level ISA3 Level ISA3 is the instruction set architecture level. Its two chapters describe Pep/7, a virtual computer designed to illustrate computer concepts. The Pep/7 computer is a classical von Neumann machine. The CPU contains an accumulator, an index register, a base register, a program counter, a stack pointer, and an instruction register. It has four addressing modes: immediate, direct, stack relative, and indexed. The Pep/7 operating system, in simulated read-only memory (ROM), can load and execute programs in hexadecimal format from students' text files. Students run short programs on the Pep/7 simulator and learn that executing a store instruction to ROM does not change the memory value.

Students learn the fundamentals of information representation and computer organization at the bit level. Because a central theme of this book is the relationship of the levels to one another, the Pep/7 chapters show the relationship between the ASCII representation (Level ISA3) and C++ variables of type `char` (Level Hol6). They also show the relationship between two's complement representation (Level ISA3) and C++ variables of type `int` (Level Hol6).

Level Asmb5 Level Asmb5 is the assembly level. The text presents the concept of the assembler as a translator between two levels—assembly and machine. It introduces Level Asmb5 symbols and the symbol table.

The unified approach really comes to play here. Chapters 5 and 6 present the compiler as a translator from a high-order language to assembly language. Previously, students learned a specific Level Hol6 language, C++, and a specific von Neumann machine, Pep/7. These chapters continue the theme of relationships between the levels by showing the correspondence between (a) assignment statements at Level Hol6 and load/store instructions at Level Asmb5, (b) loops and `if` statements at Level Hol6 and branching instructions at Level Asmb5, (c) arrays at

Level Hol6 and indexed addressing at Level Asmb5, (d) procedure calls at Level Hol6 and the run-time stack at Level Asmb5, (e) function and procedure parameters at Level Hol6 and stack-relative addressing at Level Asmb5, and (f) `switch` statements at Level Hol6 and jump tables at Level Asmb5.

The beauty of the unified approach is that the text can implement the examples from the C++ chapter at this lower level. For example, the run-time stack illustrated in the recursive examples of Chapter 2 corresponds directly to the hardware stack in Pep/7 main memory. Students gain an understanding of the compilation process by translating manually between the two levels.

This approach provides a natural setting for the discussion of central issues in computer science. For example, the book presents structured programming at Level Hol6 versus the possibility of unstructured programming at Level Asmb5. It discusses the goto controversy and the structured programming/efficiency tradeoff, giving concrete examples from languages at the two levels.

Chapter 7, Language Translation Principles, introduces students to computer science theory. Now that students know intuitively how to translate from a high-level language to assembly language, we pose the fundamental question underlying all of computing: What can be automated? The theory naturally fits in here because students now know what a compiler (an automated translator) must do. They learn about parsing and finite state machines—deterministic and nondeterministic—in the context of recognizing C++ and Pep/7 assembly language tokens. This chapter includes an automatic translator between two small languages, which illustrates lexical analysis, parsing, and code generation. The lexical analyzer is an implementation of a finite state machine. What could be a more natural setting for the theory?

Level OS4 Level OS4 consists of two chapters on operating systems. Chapter 8 is a description of process management. Two sections, one on loaders and another on interrupt handlers, illustrate the concepts with the Pep/7 operating system. Three instructions have unimplemented opcodes that generate software interrupts. The operating system stores the process control block of the user's running process on the system stack, and the interrupt service routine interprets the instruction. The classic state transition diagram for running and waiting processes in an operating system is thus reinforced with a specific implementation of a suspended process. The chapter concludes with a description of concurrent processes and deadlocks. Chapter 9 describes storage management, both main memory and disk memory.

Level LG1 Level LG1 uses two chapters to present combinational networks and sequential networks. Chapter 10 emphasizes the importance of the mathematical foundation of computer science by starting with the axioms of boolean algebra. It shows the relationship between boolean algebra and logic gates, and then describes some common SSI and MSI logic devices. Chapter 11 illustrates the fundamental concept of a finite state machine through the state transition diagrams of sequential circuits. It concludes with the construction of the data section of the Pep/7 computer. The same machine model is thus used from the C++ level to the logic gate level, providing a complete, unifying picture of the entire system.

Use in a Course

This book offers such broad coverage that instructors may wish to omit some of the material when designing the course. Chapters 1–5 should be considered core. Selections can be made from Chapters 6 through 11.

In the book, Chapters 1–5 must be covered sequentially. Chapters 6 (Compiling to the Assembly Level) and 7 (Language Translation Principles) can be covered in either order. I often skip ahead to Chapter 7 to initiate a large software project, writing an assembler for a subset of Pep/7 assembly language, so students will have sufficient time to complete it during the semester. Chapter 11 (Sequential Networks) is obviously dependent on Chapter 10 (Combinational Networks), but neither depends on Chapter 9 (Storage Management), which may be omitted. Figure P.2, a chapter dependency graph, summarizes the possible chapter omissions.

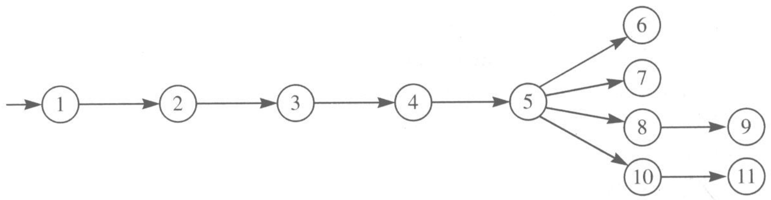


Figure **P.2**

A chapter dependency graph.

Support Materials

The support material listed below is available from the publisher's web site

```
http://computersystems.jbpub.com
```

Pep/7 Assembler and Simulator The Pep/7 machine is available for MSWindows, MacOS, and Unix systems. The assembler features

- an integrated text editor for the MSWindows and MacOS versions,
- error messages in red type that are inserted within the source code at the place where the error is detected,
- student-friendly machine language object code in hexadecimal format,
- the ability to code directly in machine language, bypassing the assembler,
- the ability to redefine the mnemonics for the unimplemented opcodes that trigger synchronous interrupts.

The simulator features

- simulated ROM that is not altered by load instructions,
- a small operating system burned into simulated ROM that includes a loader and an interrupt handler system,
- an integrated debugger that allows for break points, single step execution, CPU tracing, and memory tracing,
- the option to trace an application, the loader, or the operating system in any combination,
- a user-defined upper limit on the statement execution count to recover from endless loops,
- the ability to modify the operating system by designing new interrupt handlers for the unimplemented opcodes.

Computer Systems Figures Every figure in the book is enlarged and reproduced to be used as transparency masters for overhead projection.

Solutions Manual Solutions to selected exercises are provided in an appendix. Solutions to the remaining exercises are available to instructors who adopt the book. For security reasons, the solutions are available only in hardcopy form directly from the publisher.

Changes to the Second Edition

In addition to many small changes, updates, and corrections throughout the text, this second edition differs in three respects from the first.

A new section in Chapter 3 describes the IEEE floating point standard. The description is unusually thorough, and includes the special values NaN, infinity, and denormalized numbers. Problems are added in Chapter 8 for students to implement floating point instructions using the interrupt mechanism of the virtual hardware.

The installed memory of Pep/7 has been increased to 32 Mbytes from 4 Mbytes for Pep/6.

The C++ coding style is now more conventional than in the first edition. The readability of the code is enhanced at the expense of lengthier program listings.

[1] Harmes, Douglas and Berque, Dave, Using a PDP-11/10 to Teach Content and History in Computer Organization Courses, *SIGCSE Bulletin–Conference proceedings of the thirty-second SIGCSE Technical Symposium on Computer Science Education* (2001), 209–213.

Computing Curricula 2001

As this edition was going to press the new Curriculum 2001 guidelines for Computer Science had not yet been finalized. Preliminary versions of the curriculum report present a taxonomy of bodies of knowledge with a specified core. *Computer Systems* applies to the category Architecture and Organization (AR) and covers practically all of the core topics from the AR body of knowledge. The AR core areas from the preliminary report, together with the chapters from this text that cover each area, are

AR1. Digital logic and digital systems, Chapters 10, 11

AR2. Machine level representation of data, Chapter 3

AR3. Assembly level machine organization, Chapters 4, 5, 6

AR4. Memory system organization and architecture, Chapter 9

AR5. Interfacing and communication, Chapter 8

AR6. Functional organization, Chapter 11

AR7. Multiprocessing and alternative architectures, Chapter 8

Acknowledgments

Pep/1 had 16 instructions, one accumulator, and one addressing mode. Pep/2 added indexed addressing. John Vannoy wrote both simulators in ALGOL W. Pep/3 had 32 instructions and was written in Pascal as a student software project by Steve Dimse, Russ Hughes, Kazuo Ishikawa, Nancy Brunet, and Yvonne Smith. In an early review, Harold Stone suggested many improvements to the Pep/3 architecture that were incorporated into Pep/4 and carried into Pep/5 and Pep/6. Pep/4 had special stack instructions, simulated ROM, and software interrupts. Pep/5 was a more orthogonal design, allowing any instruction to use any addressing mode. John Rooker wrote the Pep/4 system and an early version of Pep/5. Gerry St. Romain implemented a MacOS version and an MS-DOS version. Pep/6 simplified indexed addressing and includes the complete set of conditional branch instructions. John Webb and Greg Kaestle wrote the trace facility using the BlackBox development system. Pep/7 increased the installed memory from 4 Mbytes to 32 Mbytes.

More than any other book, Tanenbaum's *Structured Computer Organization*1 has influenced this text. This text extends the level structure of Tanenbaum's book by adding the high-order programming level and the applications level at the top.

The following reviewers of the manuscript and users of the previous edition shaped the final product significantly: Wayne P. Bailey, Fadi Deek, William Decker, Peter Drexel, Gerald S. Eisman, Victoria Evans, Myers Foreman, David Garnick, Ephraim P. Glinert, Dave Hanscom, Michael Hennessy, Michael Johnson, Andrew Malton, Robert Martin, Richard H. Mercer, Randy Molmen, John Motil,

Peter Ng, Bernard Nudel, Carolyn Oberlink, Wolfgang Pelz, James F. Peters III, James C. Pleasant, Eleanor Quinlan, Glenn A. Richard, David Rosser, Scott Smith, Harold S. Stone, J. Peter Weston, and Norman E. Wright.

Thanks especially to Gerry St. Romain and John Motil, who provided, based on their experience, literally hundreds of ideas that were incorporated into the text. Scott Smith from the State University of New York at Plattsburgh wrote the sidebars. Joe Piasentin provided artistic consultation.

At Jones and Bartlett Publishers, Editor J. Michael Stranz and Production Editor Amy Rose provided valuable support and were a true joy to work with (especially because of our mutual love of chocolate). Kristin Ohlin captured the flavor of the book with her striking cover design.

I am fortunate to be at an institution that is committed to excellence in undergraduate education. Pepperdine University, in the person of Ken Perrin, provided the creative environment and the professional support in which the idea behind this project was able to evolve. My wife, Ann, provided endless personal support. To her I owe an apology for the time this project has taken, and my greatest thanks.

Stan Warford
Malibu, California

APPENDIX

Pep/7 Architecture

This appendix summarizes the architecture of the Pep/7 computer.

Figure A.1

The hexadecimal conversion chart.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0_	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1_	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2_	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3_	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4_	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5_	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6_	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7_	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8_	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9_	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A_	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B_	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C_	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D_	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E_	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F_	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

Figure A.2

The relationship between hexadecimal and binary.

Char	Bin	Hex	Char	Bin	Hex	Char	Bin	Hex	Char	Bin	Hex
NUL	000 0000	00	SP	010 0000	20	@	100 0000	40	`	110 0000	60
SOH	000 0001	01	!	010 0001	21	A	100 0001	41	a	110 0001	61
STX	000 0010	02	"	010 0010	22	B	100 0010	42	b	110 0010	62
ETX	000 0011	03	#	010 0011	23	C	100 0011	43	c	110 0011	63
EOT	000 0100	04	$	010 0100	24	D	100 0100	44	d	110 0100	64
ENQ	000 0101	05	%	010 0101	25	E	100 0101	45	e	110 0101	65
ACK	000 0110	06	&	010 0110	26	F	100 0110	46	f	110 0110	66
BEL	000 0111	07	'	010 0111	27	G	100 0111	47	g	110 0111	67
BS	000 1000	08	(	010 1000	28	H	100 1000	48	h	110 1000	68
HT	000 1001	09	)	010 1001	29	I	100 1001	49	i	110 1001	69
LF	000 1010	0A	*	010 1010	2A	J	100 1010	4A	j	110 1010	6A
VT	000 1011	0B	+	010 1011	2B	K	100 1011	4B	k	110 1011	6B
FF	000 1100	0C	,	010 1100	2C	L	100 1100	4C	l	110 1100	6C
CR	000 1101	0D	-	010 1101	2D	M	100 1101	4D	m	110 1101	6D
SO	000 1110	0E	.	010 1110	2E	N	100 1110	4E	n	110 1110	6E
SI	000 1111	0F	/	010 1111	2F	O	100 1111	4F	o	110 1111	6F
DLE	001 0000	10	0	011 0000	30	P	101 0000	50	p	111 0000	70
DC1	001 0001	11	1	011 0001	31	Q	101 0001	51	q	111 0001	71
DC2	001 0010	12	2	011 0010	32	R	101 0010	52	r	111 0010	72
DC3	001 0011	13	3	011 0011	33	S	101 0011	53	s	111 0011	73
DC4	001 0100	14	4	011 0100	34	T	101 0100	54	t	111 0100	74
NAK	001 0101	15	5	011 0101	35	U	101 0101	55	u	111 0101	75
SYN	001 0110	16	6	011 0110	36	V	101 0110	56	v	111 0110	76
ETB	001 0111	17	7	011 0111	37	W	101 0111	57	w	111 0111	77
CAN	001 1000	18	8	011 1000	38	X	101 1000	58	x	111 1000	78
EM	001 1001	19	9	011 1001	39	Y	101 1001	59	y	111 1001	79
SUB	001 1010	1A	:	011 1010	3A	Z	101 1010	5A	z	111 1010	7A
ESC	001 1011	1B	;	011 1011	3B	[	101 1011	5B	{	111 1011	7B
FS	001 1100	1C	<	011 1100	3C	\	101 1100	5C	|	111 1100	7C
GS	001 1101	1D	=	011 1101	3D	]	101 1101	5D	}	111 1101	7D
RS	001 1110	1E	>	011 1110	3E	^	101 1110	5E	~	111 1110	7E
US	001 1111	1F	?	011 1111	3F	_	101 1111	5F	DEL	111 1111	7F

Abbreviations for Control Characters

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

Figure **A.3**

The American Standard Code for Information Interchange (ASCII).

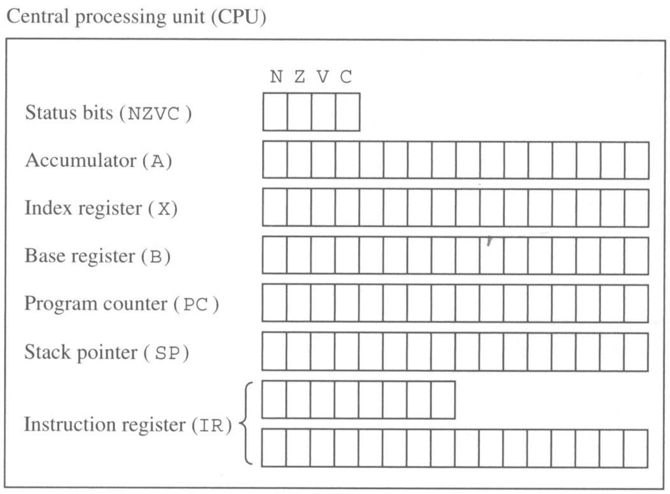


Figure **A.4**

The central processing unit of the Pep/7 computer.

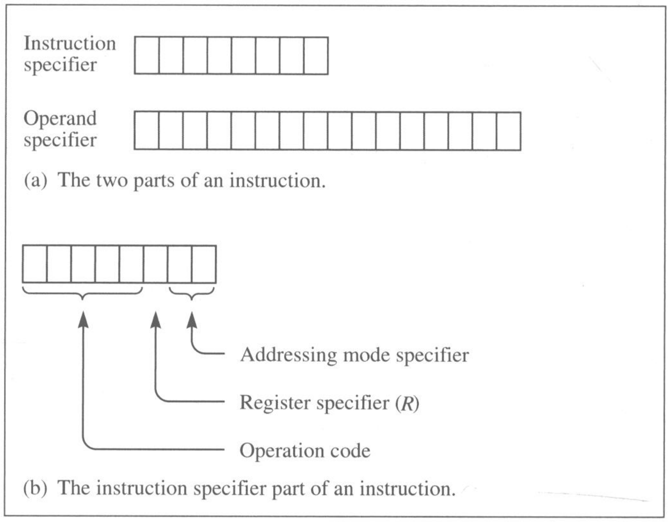

Instruction specifier
Operand specifier
(a) The two parts of an instruction.
Addressing mode specifier
Register specifier (*R*)
Operation code
(b) The instruction specifier part of an instruction.

Figure **A.5**

The Pep/7 instruction format.

Register specifier	Register specified (*R*)
0	Accumulator (A)
1	Index register (X)

Figure **A.6**

The register specified by the register specifier part of the opcode.

Addr. mode specifier	Addr. mode specified
00	Immediate (i)
01	Direct (d)
10	Stack relative (s)
11	Indexed (x)

Figure **A.7**

The addressing mode specified by the addressing mode specifier part of the opcode.

- Immediate addressing:

```
Oprnd = OprndSpec
```

- Direct addressing:

```
Oprnd = Mem[OprndSpec]
```

- Stack-relative addressing:

```
Oprnd = Mem[SP + OprndSpec]
```

- Indexed addressing:

```
Oprnd = Mem[B + X]
```

Figure **A.8**

The relationship between the operand and the operand specifier.

Opcode	Mnemonic	Meaning	Addr. modes	Status bits
00000	STOP	stop execution	*U*	
00001	LOAD*R*	*R* := Oprnd	idsx	NZ
00010	STORE*R*	Oprnd := *R*	dsx	
00011	ADD*R*	*R* := *R* + Oprnd	idsx	NZVC
00100	SUB*R*	*R* := *R* - Oprnd	idsx	NZVC
00101	AND*R*	*R* := *R* AND Oprnd	idsx	NZ
00110	OR*R*	*R* := *R* OR Oprnd	idsx	NZ
00111	NOT*R*	*R* := NOT *R*	*U*	NZ
01000	ASL*R*	C := most significant bit; *R* := arithmetic shift left *R*	*U*	NZVC
01001	ASR*R*	C := least significant bit; *R* := arithmetic shift right *R*	*U*	NZC
01010	LDBYT*R*	*R* := byte Oprnd	idsx	NZ
01011	STBYT*R*	byte Oprnd := *R*	dsx	
01100	LOADB	B := Oprnd	idsx	NZ
01101	ADDSP	SP := SP + Oprnd	i	NZVC
01110	BR	PC := Oprnd	ix	
01111	BRLE	if NZVC ≡ ≤ then PC := Oprnd	ix	
10000	BRLT	if NZVC ≡ < then PC := Oprnd	ix	
10001	BREQ	if NZVC ≡ = then PC := Oprnd	ix	
10010	BRNE	if NZVC ≡ ≠ then PC := Oprnd	ix	
10011	BRGE	if NZVC ≡ ≥ then PC := Oprnd	ix	
10100	BRGT	if NZVC ≡ > then PC := Oprnd	ix	
10101	BRV	if V=1 then PC := Oprnd	ix	
10110	BRC	if C=1 then PC := Oprnd	ix	
10111	COMP*R*	*R* - Oprnd	idsx	NZVC
11000	JSR	SP := SP - 2; Mem[SP] := PC; PC := Oprnd	ix	
11001	RTS	PC := Mem[SP]; SP := SP + 2	*U*	
11010	RTI	return from interrupt	*U*	
11011	CHARI	byte Oprnd := input	dsx	
11100	CHARO	output := byte Oprnd	idsx	
11101	DECI	Oprnd := input	dsx	NZV
11110	DECO	output := Oprnd	idsx	
11111	HEXO	output := Oprnd	idsx	

Figure **A.9**

The Pep/7 instruction set.