APPENDIX

Pep/9 Architecture

This appendix summarizes the architecture of the Pep/9 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

FIGURE A.2
The relationship between hexadecimal and binary.

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.3
The American Standard Code for Information Interchange (ASCII).

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	**SYN**	synchronous idle		
VT	vertical tabulation	**ETB**	end of transmission block		

FIGURE A.4
The central processing unit of the Pep/9 computer.

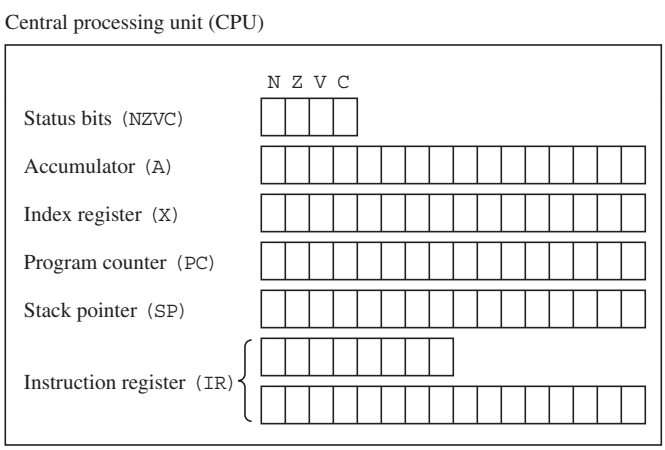
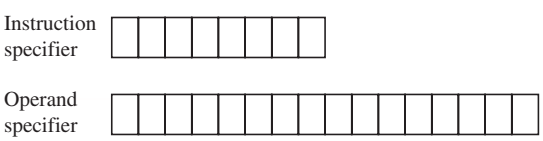


FIGURE A.5
The Pep/9 instruction format.



(a) The two parts of a nonunary instruction.

(b) A unary instruction.

FIGURE A.6
The Pep/9 instruction specifier fields.

aaa	Addressing Mode
000	Immediate
001	Direct
010	Indirect
011	Stack-relative
100	Stack-relative deferred
101	Indexed
110	Stack-indexed
111	Stack-deferred indexed

(a) The addressing-aaa field.

a	Addressing Mode
0	Immediate
1	Indexed

(b) The addressing-a field.

r	Register
0	Accumulator, A
1	Index register, X

(c) The register-r field.

FIGURE A.7
The Pep/9 addressing modes.

Addressing Mode	aaa	Letters	Operand
Immediate	000	i	OprndSpec
Direct	001	d	Mem[OprndSpec]
Indirect	010	n	Mem[Mem[OprndSpec]]
Stack-relative	011	s	Mem[SP + OprndSpec]
Stack-relative deferred	100	sf	Mem[Mem[SP + OprndSpec]]
Indexed	101	x	Mem[OprndSpec + X]
Stack-indexed	110	sx	Mem[SP + OprndSpec + X]
Stack-deferred indexed	111	sfx	Mem[Mem[SP + OprndSpec] + X]

FIGURE A.8
The Pep/9 instruction set at Level Asmb5.

Instruction Specifier	Mnemonic	Instruction	Addressing Mode	Status Bits
0000 0000	`STOP`	Stop execution	U	
0000 0001	`RET`	Return from `CALL`	U	
0000 0010	`RETTR`	Return from trap	U	
0000 0011	`MOVSPA`	Move SP to A	U	
0000 0100	`MOVFLGA`	Move NZVC flags to A⟨12..15⟩	U	
0000 0101	`MOVAFLG`	Move A⟨12..15⟩ to NZVC flags	U	
0000 011r	`NOTr`	Bitwise invert r	U	NZ
0000 100r	`NEGr`	Negate r	U	NZV
0000 101r	`ASLr`	Arithmetic shift left r	U	NZVC
0000 110r	`ASRr`	Arithmetic shift right r	U	NZC
0000 111r	`ROLr`	Rotate left r	U	C
0001 000r	`RORr`	Rotate right r	U	C
0001 001a	`BR`	Branch unconditional	i, x	
0001 010a	`BRLE`	Branch if less than or equal to	i, x	
0001 011a	`BRLT`	Branch if less than	i, x	
0001 100a	`BREQ`	Branch if equal to	i, x	
0001 101a	`BRNE`	Branch if not equal to	i, x	
0001 110a	`BRGE`	Branch if greater than or equal to	i, x	
0001 111a	`BRGT`	Branch if greater than	i, x	
0010 000a	`BRV`	Branch if V	i, x	
0010 001a	`BRC`	Branch if C	i, x	
0010 010a	`CALL`	Call subroutine	i, x	
0010 011n	`NOPn`	Unary no operation trap	U	
0010 1aaa	`NOP`	Nonunary no operation trap	i	

(*continued*)

FIGURE A.8
The Pep/9 instruction set at Level Asmb5. (*continued*)

0011 0aaa	DECI	Decimal input trap	d, n, s, sf, x, sx, sfx	NZV
0011 1aaa	DECO	Decimal output trap	i, d, n, s, sf, x, sx, sfx	
0100 0aaa	HEXO	Hexadecimal output trap	i, d, n, s, sf, x, sx, sfx	
0100 1aaa	STRO	String output trap	d, n, s, sf, x	
0101 0aaa	ADDSP	Add to stack pointer (SP)	i, d, n, s, sf, x, sx, sfx	NZVC
0101 1aaa	SUBSP	Subtract from stack pointer (SP)	i, d, n, s, sf, x, sx, sfx	NZVC
0110 raaa	ADDr	Add to r	i, d, n, s, sf, x, sx, sfx	NZVC
0111 raaa	SUBr	Subtract from r	i, d, n, s, sf, x, sx, sfx	NZVC
1000 raaa	ANDr	Bitwise AND to r	i, d, n, s, sf, x, sx, sfx	NZ
1001 raaa	ORr	Bitwise OR to r	i, d, n, s, sf, x, sx, sfx	NZ
1010 raaa	CPWr	Compare word to r	i, d, n, s, sf, x, sx, sfx	NZVC
1011 raaa	CPBr	Compare byte to r⟨8..15⟩	i, d, n, s, sf, x, sx, sfx	NZVC
1100 raaa	LDWr	Load word r from memory	i, d, n, s, sf, x, sx, sfx	NZ
1101 raaa	LDBr	Load byte r⟨8..15⟩ from memory	i, d, n, s, sf, x, sx, sfx	NZ
1110 raaa	STWr	Store word r to memory	d, n, s, sf, x, sx, sfx	
1111 raaa	STBr	Store byte r⟨8..15⟩ to memory	d, n, s, sf, x, sx, sfx	

FIGURE A.9
The pseudo-ops of Pep/9 assembly language.

Pseudo-op	Assembler Directive
.ADDRSS	The address of a symbol
.ALIGN	Padding to align at a memory boundary
.ASCII	A string of ASCII bytes
.BLOCK	A block of zero bytes
.BURN	Initiate ROM burn
.BYTE	A byte value
.END	The sentinel for the assembler
.EQUATE	Equate a symbol to a constant value
.WORD	A word value

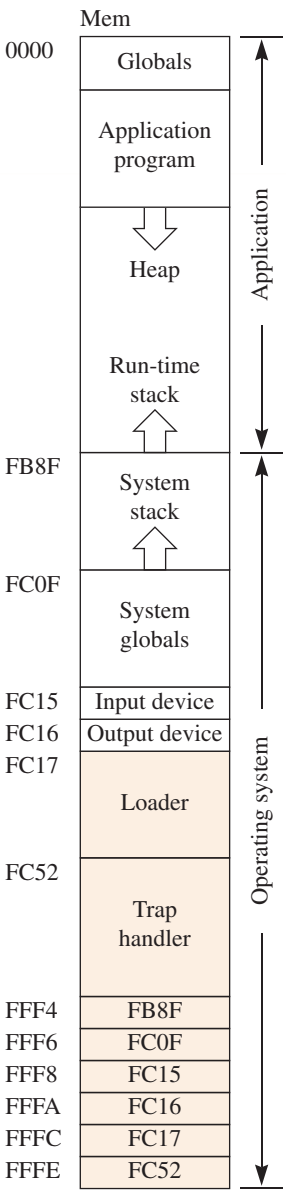


FIGURE A.10
A memory map of the Pep/9 memory. The shaded part is read-only memory.

FIGURE A.11
The RTL specification of Pep/9 instructions.

Instruction	Register Transfer Language Specification
`STOP`	Stop execution
`RET`	PC ← Mem[SP] ; SP ← SP + 2
`RETTR`	NZVC ← Mem[SP]⟨4..7⟩ ; A ← Mem[SP + 1] ; X ← Mem[SP + 3] ; PC ← Mem[SP + 5] ; SP ← Mem[SP + 7]
`MOVSPA`	A ← SP
`MOVFLGA`	A⟨8..11⟩ ← 0 , A⟨12..15⟩ ← NZVC
`MOVAFLG`	NZVC ← A⟨12..15⟩
`NOTr`	r ← ¬ r ; N ← r < 0 , Z ← r = 0
`NEGr`	r ← −r ; N ← r < 0 , Z ← r = 0 , V ← {*overflow*}
`ASLr`	C ← r⟨0⟩ , r⟨0..14⟩ ← r⟨1..15⟩ , r⟨15⟩ ← 0 ; N ← r < 0 , Z ← r = 0 , V ← {*overflow*}
`ASRr`	C ← r⟨15⟩ , r⟨1..15⟩ ← r⟨0..14⟩ ; N ← r < 0 , Z ← r = 0
`ROLr`	C ← r⟨0⟩ , r⟨0..14⟩ ← r⟨1..15⟩ , r⟨15⟩ ← C
`RORr`	C ← r⟨15⟩ , r⟨1..15⟩ ← r⟨0..14⟩ , r⟨0⟩ ← C
`BR`	PC ← Oprnd
`BRLE`	N = 1 ∨ Z = 1 ⇒ PC ← Oprnd
`BRLT`	N = 1 ⇒ PC ← Oprnd
`BREQ`	Z = 1 ⇒ PC ← Oprnd
`BRNE`	Z = 0 ⇒ PC ← Oprnd
`BRGE`	N = 0 ⇒ PC ← Oprnd
`BRGT`	N = 0 ∧ Z = 0 ⇒ PC ← Oprnd
`BRV`	V = 1 ⇒ PC ← Oprnd
`BRC`	C = 1 ⇒ PC ← Oprnd
`CALL`	SP ← SP − 2 ; Mem[SP] ← PC ; PC ← Oprnd
`NOPn`	Trap: Unary no operation
`NOP`	Trap: Nonunary no operation
`DECI`	Trap: Oprnd ← {*decimal input*}
`DECO`	Trap: {*decimal output*} ← Oprnd
`HEXO`	Trap: {*hexadecimal output*} ← Oprnd
`STRO`	Trap: {*string output*} ← Oprnd
`ADDSP`	SP ← SP + Oprnd
`SUBSP`	SP ← SP − Oprnd
`ADDr`	r ← r + Oprnd ; N ← r < 0 , Z ← r = 0 , V ← {*overflow*} , C ← {*carry*}
`SUBr`	r ← r − Oprnd ; N ← r < 0 , Z ← r = 0 , V ← {*overflow*} , C ← {*carry*}

`ANDr`	r ← r ∧ Oprnd ; N ← r < 0 , Z ← r = 0
`ORr`	r ← r ∨ Oprnd ; N ← r < 0 , Z ← r = 0
`CPWr`	T ← r − Oprnd ; N ← T < 0 , Z ← T = 0 , V ← {*overflow*} , C ← {*carry*} ; N ← N ⊕ V
`CPBr`	T ← r⟨8..15⟩ − byte Oprnd ; N ← T < 0 , Z ← T = 0 , V ← 0 , C ← 0
`LDWr`	r ← Oprnd ; N ← r < 0 , Z ← r = 0
`LDBr`	r⟨8..15⟩ ← byte Oprnd ; N ← 0 , Z ← r⟨8..15⟩ = 0
`STWr`	Oprnd ← r
`STBr`	byte Oprnd ← r⟨8..15⟩
Trap	T ← Mem[FFF6] ; Mem[T − 1] ← IR⟨0..7⟩ ; Mem[T − 3] ← SP ; Mem[T − 5] ← PC ; Mem[T − 7] ← X ; Mem[T − 9] ← A ; Mem[T − 10]⟨4..7⟩ ← NZVC ; SP ← T − 10 ; PC ← Mem[FFFE]

FIGURE A.12
The 16 functions of the Pep/9 ALU.

ALU Control			Status Bits			
(bin)	(dec)	Result	N	Zout	V	Cout
0000	0	A	N	Z	0	0
0001	1	A plus B	N	Z	V	C
0010	2	A plus B plus Cin	N	Z	V	C
0011	3	A plus $\overline{B}$ plus 1	N	Z	V	C
0100	4	A plus $\overline{B}$ plus Cin	N	Z	V	C
0101	5	A · B	N	Z	0	0
0110	6	$\overline{A \cdot B}$	N	Z	0	0
0111	7	A + B	N	Z	0	0
1000	8	$\overline{A + B}$	N	Z	0	0
1001	9	A ⊕ B	N	Z	0	0
1010	10	$\overline{A}$	N	Z	0	0
1011	11	ASL A	N	Z	V	C
1100	12	ROL A	N	Z	V	C
1101	13	ASR A	N	Z	0	C
1110	14	ROR A	N	Z	0	C
1111	15	0	A<4>	A<5>	A<6>	A<7>

FIGURE A.13
The data section of the Pep/9 CPU.

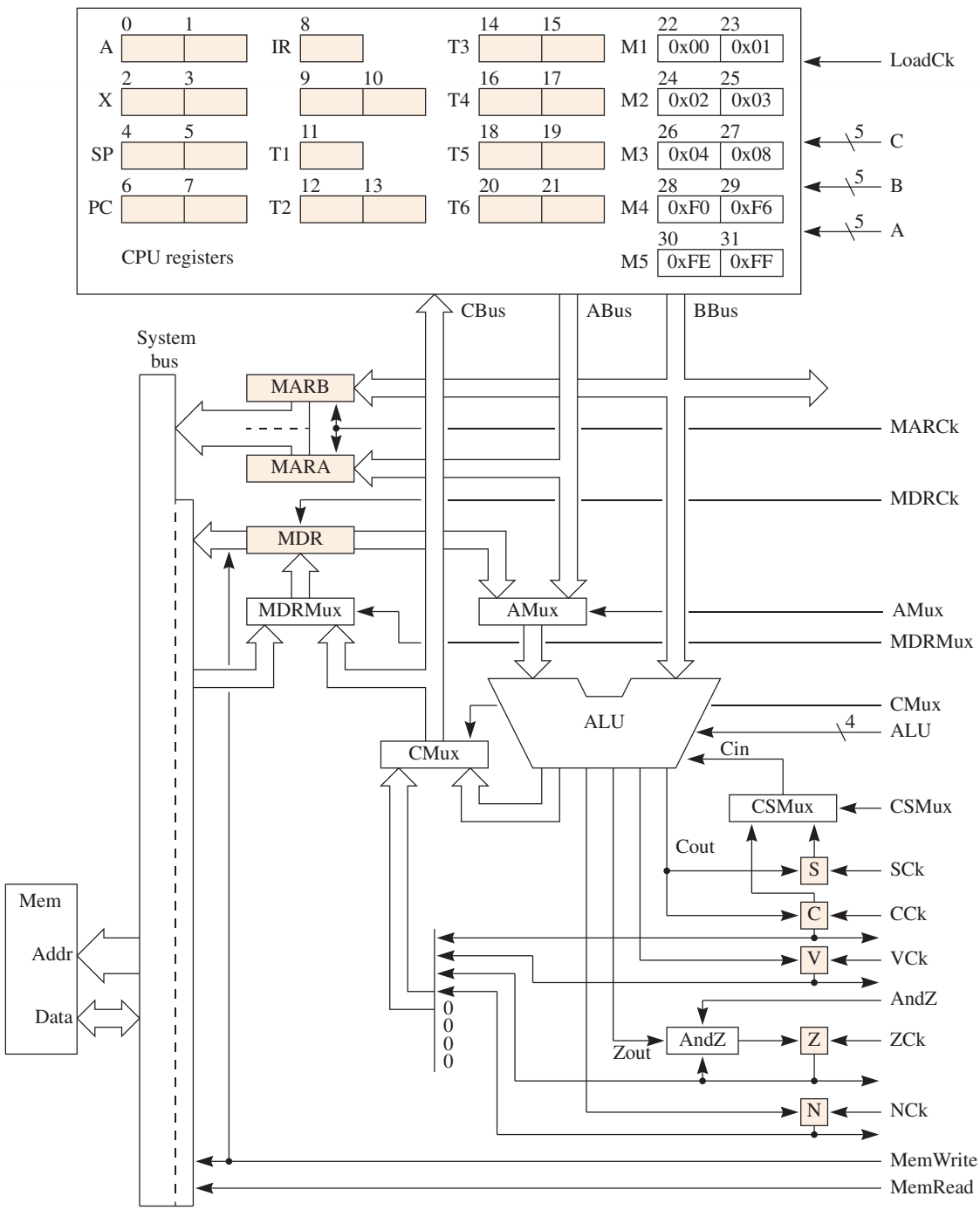
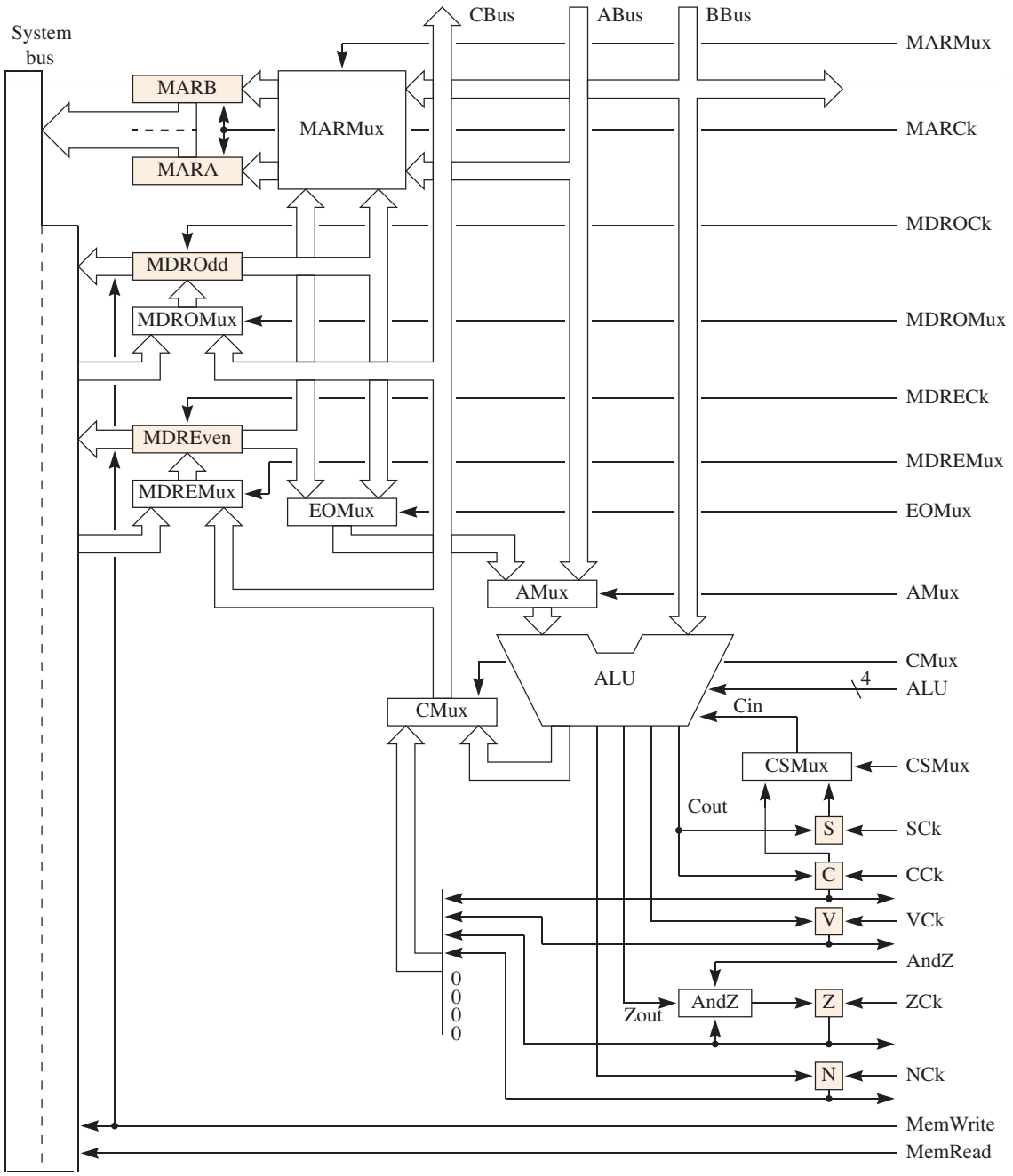


FIGURE A.14
The Pep/9 CPU with a two-byte data bus.

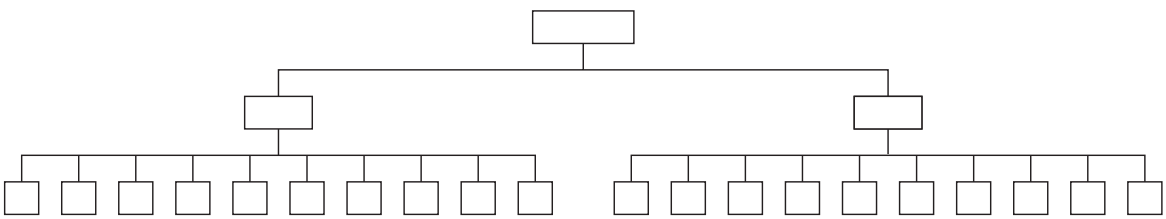


Solutions to Selected Exercises

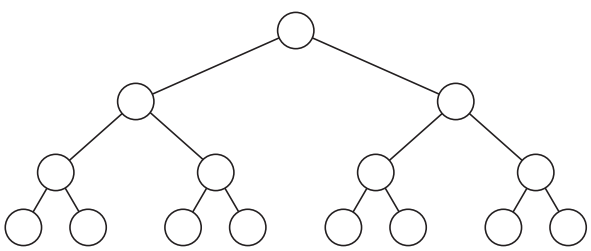
Chapter 1

2. (a) 11,110 not counting Khan

3. (a)



4. (a)



(b) 31

9. 43 ms

12. 32 b/s

15. (a) 1936 b (b) 152 characters

21. `Temp5`

F.Name	F.Major	F.State
Ron	Math	OR

`Temp6`

S.Name	S.Class	S.Major	S.State
Beth	Soph	Hist	TX
Allison	Soph	Math	AZ

22. (a)
```
select Sor where S.Name = Beth giving Temp
project Temp over S.State giving Result
```

Chapter 2

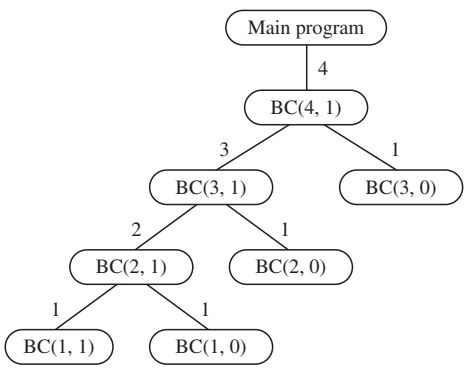
1. (a) Called four times.

2.

(a, part 1)

```
Main program
    Call BC(4, 1)
        Call BC(3, 1)
            Call BC(2, 1)
                Call BC(1, 1)
            Return to BC(2, 1)
                Call BC(1, 0)
            Return to BC(2, 1)
        Return to BC(3, 1)
            Call BC(2, 0)
        Return to BC(3, 1)
    Return to BC(4, 1)
        Call BC(3, 0)
    Return to BC(4, 1)
Return to main program
```

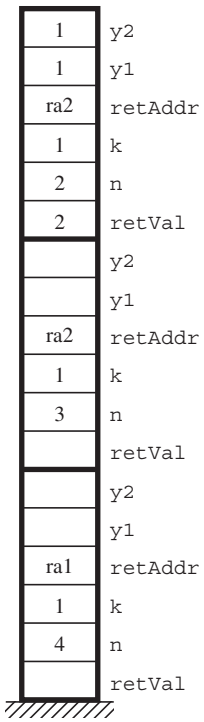
(a, part 2)



(a, part 3) Called seven times.

(a, part 4) Maximum of five stack frames.

(a, part 5)



Chapter 3

1. (a) Octal: 267, 270, 271, 272, 273, 274, 275, 276, 277, 300, 301

(b) Base 3: 2102, 2110, 2111, 2112, 2120, 2121, 2122, 2200, 2201, 2202, 2210

(c) Binary: 10101, 10110, 10111, 11000, 11001, 11010, 11011, 11100, 11101, 11110, 11111

(d) Base 5: 2433, 2434, 2440, 2441, 2442, 2443, 2444, 3000, 3001, 3002

3. (a) 18 (b) 6 (c) 11 (d) 8 (e) 31 (f) 85

5. (a) 11001 (b) 10000 (c) 1 (d) 1110 (e) 101 (f) 101001

7. (a) 00 to 11 (bin) and 0 to 3 (dec) (b) 000 to 111 (bin) and 0 to 7 (dec)

8. (a) 111 0100, C = 0 (b) 001 0000, C = 1

(c) 111 1110, C = 1 (d) 000 0000, C = 0

11. (a) $7 \times 8^4 + 0 \times 8^3 + 1 \times 8^2 + 4 \times 8^1 + 6 \times 8^0$

13. (a) 011 0001 (b) 110 0101 (c) 000 0000 (d) 100 0000 (e) 111 1111

(f) 111 1110 (g) 100 0000 to 011 1111 (bin) and –64 to 63 (dec)

15. (a) 29 (b) –43 (c) –4 (d) 1 (e) –64 (f) –63

17. (a) 011 1001, NZVC = 0000 (b) 000 0110, NZVC = 0001
(c) 001 1011, NZVC = 0011 (d) 101 0110, NZVC = 1001
(e) 100 0001, NZVC = 1011 (f) 111 0100, NZVC = 1000

19. (a) 10 to 01 (bin) and −2 to 1 (dec)
(b) 100 to 011 (bin) and −4 to 3 (dec)

20. (a) 010 1000, NZ = 00 (b) 000 0101, NZ = 00 (c) 110 1110, NZ = 10
(d) 101 1111, NZ = 10 (e) 100 0110, NZ = 10 (f) 101 1010, NZ = 10
(g) 101 0100 (h) 001 0101

22. (a) 24 (dec) = 001 1000 (bin)
ASL 001 1000 = 011 0000 (bin) = 48 (dec), NZVC = 0000
ASR 001 1000 = 000 1100 (bin) = 12 (dec), NZC = 000
(b) 37 (dec) = 010 0101 (bin)
ASL 010 0101 = 100 1010 (bin) = −54 (dec), NZVC = 1010
ASR 010 0101 = 001 0010 (bin) = 18 (dec), NZC = 001
(c) −26 (dec) = 110 0110 (bin)
ASL 110 0110 = 100 1100 (bin) = −52 (dec), NZVC = 1001
ASR 110 0110 = 111 0011 (bin) = −13 (dec), NZC = 100
(d) 1 (dec) = 000 0001 (bin)
ASL 000 0001 = 000 0010 (bin) = 2 (dec), NZVC = 0000
ASR 000 0001 = 000 0000 (bin) = 0 (dec), NZC = 011
(e) 0 (dec) = 000 0000 (bin)
ASL 000 0000 = 000 0000 (bin) = 0 (dec), NZVC = 0100
ASR 000 0000 = 000 0000 (bin) = 0 (dec), NZC = 010
(f) −1 (dec) = 111 1111 (bin)
ASL 111 1111 = 111 1110 (bin) = −2 (dec), NZVC = 1001
ASR 111 1111 = 111 1111 (bin) = −1 (dec), NZC = 101

25. (a) C = 1, ROL 010 1101 = 101 1011, C = 0
(b) C = 0, ROL 010 1101 = 101 1010, C = 0
(c) C = 1, ROR 010 1101 = 101 0110, C = 1
(d) C = 0, ROR 010 1101 = 001 0110, C = 1

28. (a) 3AB7, 3AB8, 3AB9, 3ABA, 3ABB, 3ABC

29. (a) 11,614

30. (a) 68CF

32. (a) 5D (hex) = 101 1101 (bin) = −35 (dec)
(b) 2F (hex) = 010 1111 (bin) = 47 (dec)
(c) 40 (hex) = 100 0000 (bin) = −256 (dec)

34. (a) −27 (dec) = 110 0101 (bin) = 65 (hex)
(b) 63 (dec) = 011 1111 (bin) = 3F (hex)
(c) −1 (dec) = 111 1111 (bin) = 7F (hex)

36. Have a nice day!

38. 101 0000 110 0001 111 1001 010 0000 010 0100
011 0000 010 1110 011 1001 011 0010

40. (a) D5 82

43. (a) An octal digit represents three bits.

44. (a) 6.640625 (b) 0.046875 (c) 1.0

46. (a) 1101.00101 (b) 0.000101 (c) 0.1001100110011 . . .

50. (a) −12.5 (dec) = −1100.1 (bin), which is stored as 1 110 1001

51. (a) 0.90625

53. (a) 1.0×2^6

Chapter 4

1. (a) 65,536 bytes (b) 32,768 words (c) 524,288 bits
(d) 92 bits (e) 5699 times bigger

3.

For instruction 6AF82C	For instruction D623D0
(a) opcode = 0110	(a) opcode = 1101
(b) It adds to register r	(b) It loads a byte from memory to register r
(c) r = 1	(c) r = 0
(d) The index register, X	(d) The accumulator, A
(e) aaa = 010	(e) aaa = 110
(f) Indirect	(f) Stack-indexed
(g) OprndSpec = F82C	(g) OprndSpec = 23D0

5.

	A	X	Mem[0A3F]	Mem[0A41]
Original content	19AC	FE20	FF00	103D
(a) C1 = Load word accumulator	FF00	FE20	FF00	103D
(b) D1 = Load byte accumulator	19FF	FE20	FF00	103D
(c) D9 = Load byte index register	19AC	FE10	FF00	103D
(d) F1 = Store byte accumulator	19AC	FE20	FF00	AC3D
(e) E9 = Store word index register	19AC	FE20	FE20	103D
(f) 79 = Subtract index register	19AC	EDE3	FF00	103D
(g) 71 = Subtract accumulator	1AAC	FE20	FF00	103D
(h) 91 = OR accumulator	FFAC	FE20	FF00	103D
(i) 07 = Invert index register	19AC	01DF	FF00	103D

9. (a) `M`

Chapter 5

1. (a) `ORX 0xEF2A,n` (b) `MOVSPA` (c) `LDBA 0x003D,sfx`

3. (a) `0A` (b) `33 00 0F` (c) `1A 01 E6`

5. (a) `42 65 61 72 00` (b) `F8` (c) `03 16`

7. `mug`

10.
```
-57
72
0048
Hi
```

12. (a) Object code is:

```
38 00 6D D0 00 0A F1 FC 16 38 6D 6D D0 00 0A F1
FC 16 D0 00 26 F1 FC 16 00 zz
```

Output is:

```
109
28013
&
```

13. Object code is:

```
12 00 05 00 09 39 00 03 00 zz
```

Symbol `here` has value 0003 (hex). Symbol `there` has value 0005 (hex).

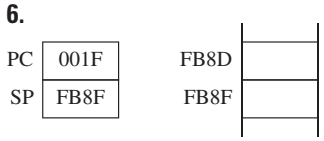
15. Symbol `this` has value 0000 (hex). The output is `4100`. The output comes from the hexadecimal output instruction, which outputs its own instruction specifier followed by the first byte of its operand specifier.

18. The compiler uses its symbol table to store the type of each variable. It consults the symbol table whenever it encounters an expression or assignment statement to verify that the types are compatible.

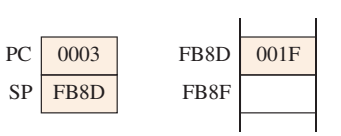
Chapter 6

3. Because the current value of `j` will be in the accumulator regardless of whether control comes from the `STWA` at 0009 or from the `BR` at the bottom of the loop. Before the `BR` at the bottom of the loop, the accumulator was used to increment `j`; hence, its current value will still be in the accumulator when `CPWA` executes.

6.



(a) Before execution of the second `CALL`.



(b) After execution of the second `CALL`.

8. The branch address is calculated as

$$\begin{aligned} \text{Oprnd} &= \text{Mem}[\text{OprndSpec} + \text{X}] \\ &= \text{Mem}[0013 + 8] \\ &= \text{Mem}[001\text{B}] \\ &= 4900 \end{aligned}$$

You cannot tell from the program listing what bits are at 4900, but assuming that they are all 0's, the von Neumann cycle blindly interprets the 00 at address 4900 as the `STOP` instruction.

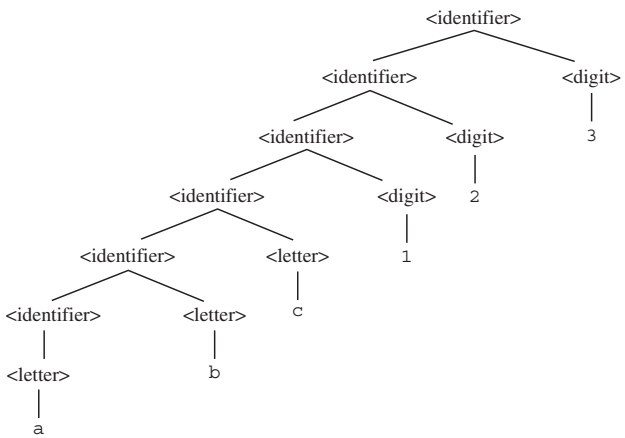
Chapter 7

1. The fundamental question of computer science is: What can be automated?

3. (a)

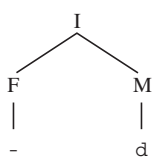
<identifier>

⇒ <identifier> <digit>	Rule 3
⇒ <identifier> `3`	Rule 9
⇒ <identifier> <digit> `3`	Rule 3
⇒ <identifier> `23`	Rule 8
⇒ <identifier> <digit> `23`	Rule 3
⇒ <identifier> `123`	Rule 7
⇒ <identifier> <letter> `123`	Rule 2
⇒ <identifier> `c123`	Rule 6
⇒ <identifier> <letter> `c123`	Rule 2
⇒ <identifier> `bc123`	Rule 5
⇒ <letter> `bc123`	Rule 1
⇒ `abc123`	Rule 4



4. (a)

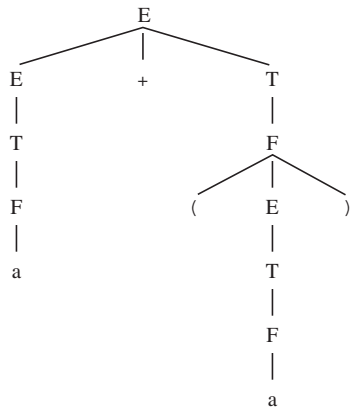
I	⇒ FM	Rule 1
	⇒ –M	Rule 3
	⇒ `-d`	Rule 6



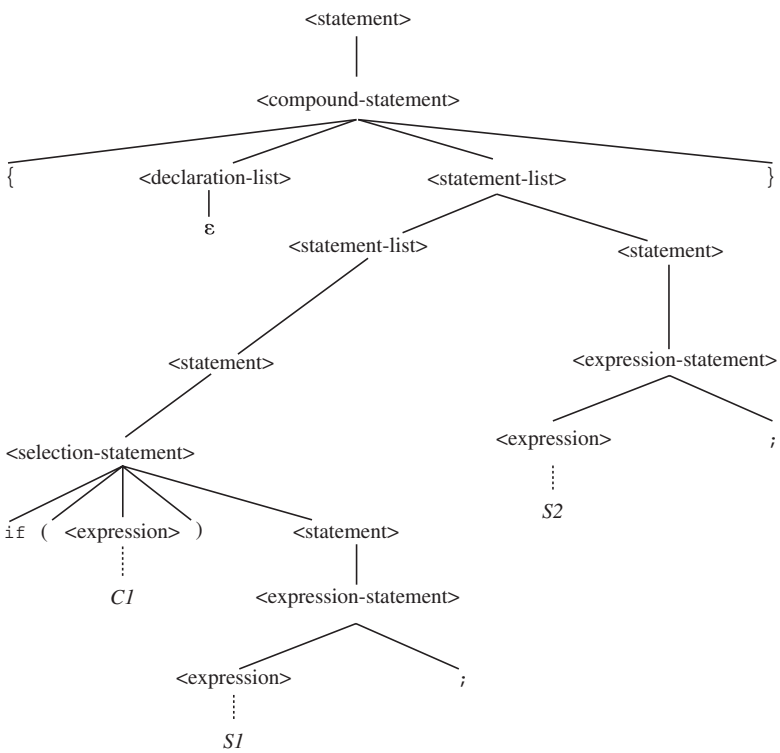
5. (a)

A	⇒ `abC`	Rule 2
	⇒ `abc`	Rule 5

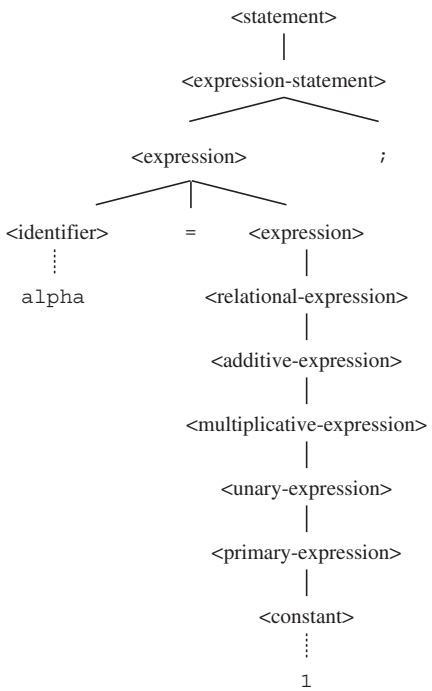
6. (a)


E
E
+
T
T
F
F
(
E
)
a
T
F
a

7. (a)

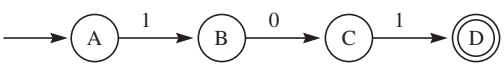

<statement>
<compound-statement>
{
<declaration-list>
<statement-list>
}
ε
<statement-list>
<statement>
<statement>
<expression-statement>
<selection-statement>
<expression>
;
S2
if (<expression>)
<statement>
C1
<expression-statement>
<expression>
;
S1

8. (a)



11. (a) The machine is deterministic. There are no inaccessible states.

13. (a)



Chapter 8

2. (a) 36 (hex), the ASCII code for 6 in the 30th byte, 6C
(b) 36 (hex)
(c) 60 (hex)

4. (a) 31 (hex), the instruction specifier of the interrupted instruction
(b) 0006 (hex)

5. (a) 39 (hex) the instruction specifier of the interrupted instruction
(b) 0007

6. (a) 49 (hex) the instruction specifier of the interrupted instruction
(b) 0009

7. (a) 0003 (hex), the numeric value of the ASCII character 3
(b) 0007 (hex), the numeric value of the ASCII character 7
(c) 0000 (hex), the value of the `init` state

8. Hint: The first character input is the ASCII hyphen character

9. (a) 0025 (hex), which is 37 (dec) (b) 0025 (hex), not negated because it was already nonnegative (c) The `CALL` from FF78 is for writing the 100's place. The accumulator still contains 0025 (hex), which is 37 (dec), because 37 mod 100 is 37.

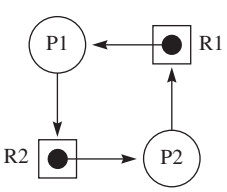
18. (a) The algorithm no longer guarantees mutual exclusion. Suppose P1 and P2 are both in their remainder sections with `enter1` and `enter2` both false. P1 could execute its `while` loop test and be interrupted, after which P2 could execute its `while` loop test. They could then assign their respective `enter` variables to true and enter their critical sections simultaneously.

20. (a) `s` = 0 and there are no blocked processes.

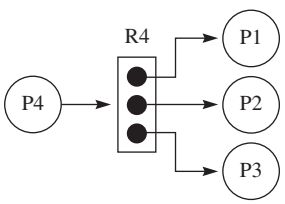
22. (a) The algorithm does guarantee mutual exclusion. If you remove the `t` semaphore altogether, you get the algorithm of Figure 8.20, which guarantees mutual exclusion regardless of any other code present in the algorithm.

24. (a) Mutual exclusion is no longer guaranteed. Can you find a trace that allows P1 and P2 to enter their critical sections simultaneously? Deadlock, however, cannot occur.

25.



(a) Contains a deadlock cycle.



(b) Does not contain a cycle and therefore does not have a deadlock.

Chapter 9

2. (a) A single bound register would suffice because only one process at a time can execute. If a user process tries to access a memory location outside its logical address space, the hardware must interrupt the access before the page table because there is no such page in main memory. The operating system must keep track of one bound value for each process.

4. (a) 2^{12} or 4096 bytes

6. Those that have a dirty bit value of N—namely, the ones in frames 2, 5, and 6.

8. The start of one page reference sequence for a job that has three frames allocated is 1, 2, 3, 1, 4, 2, . . . , which has four faults for FIFO and five for LRU. Can you complete the sequence in such a way that FIFO is better in this special case?

10. It produces five faults compared to seven for FIFO and six for LRU. You should trace the algorithm to verify this figure.

11. (a) Hint: The worst case occurs when the start of the block has just rotated past the read/write head when the head has reached the track. Hence, it is time for the disk to make one complete revolution. You can calculate it from the RPM number.

12. (a) Four data bits. (b) One parity bit.

16. (a) An error did occur in position 2. The corrected code word is 1101 1010 1001.

Chapter 10

1. (a)

$$
\begin{aligned}
& x + 1 \\
= \quad & \langle \text{complement of OR} \rangle \\
& x + (x + x') \\
= \quad & \langle \text{associative of OR} \rangle \\
& (x + x) + x' \\
= \quad & \langle \text{idempotent of OR} \rangle \\
& x + x' \\
= \quad & \langle \text{complement of OR} \rangle \\
& 1
\end{aligned}
$$

4. To show that the complement of $a + b$ is $a' \cdot b'$, you must show that

$$(a + b) \cdot (a' \cdot b') = 0 \quad \text{and} \quad (a + b) + (a' \cdot b') = 1$$

First part:

$$
\begin{aligned}
& (a + b) \cdot (a' \cdot b') \\
= \quad & \langle \text{commutative of AND} \rangle \\
& (a' \cdot b') \cdot (a + b) \\
= \quad & \langle \text{distributive of AND over OR} \rangle \\
& (a' \cdot b') \cdot a + (a' + b') \cdot b \\
= \quad & \langle \text{commutative and associative of AND} \rangle \\
& b' \cdot (a \cdot a') + a' \cdot (b \cdot b') \\
= \quad & \langle \text{complement of AND} \rangle \\
& b' \cdot 0 + a' \cdot 0
\end{aligned}
$$

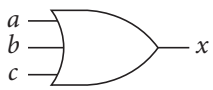
$=$ ⟨the zero theorem of AND, $x \cdot 0 = 0$⟩

$0 + 0$

$=$ ⟨idempotent of OR⟩

0

8. (a)

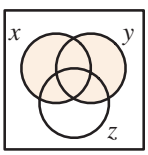


$x = a + b + c$

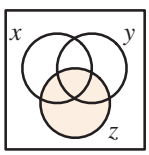
a	b	c	x
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

9. (a) The union of any set with the empty set is the set itself.

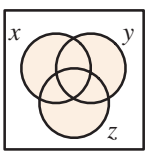
10. (a)



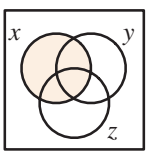
(1) $(x + y)$



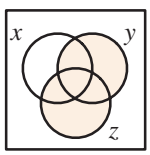
(2) z



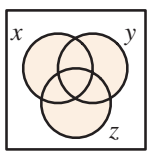
(3) $(x + y) + z$



(4) x



(5) $y + z$

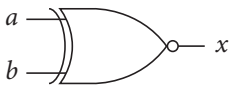


(6) $x + (y + z)$

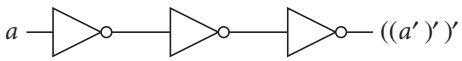
13. (a)

x	y	x AND y
0	0	0
0	A	0
0	B	0
0	0	0
A	1	0
A	0	A
A	A	0
A	B	A
B	1	0
B	0	0
B	A	B
B	B	B
1	1	0
1	A	A
1	B	B
1	1	1

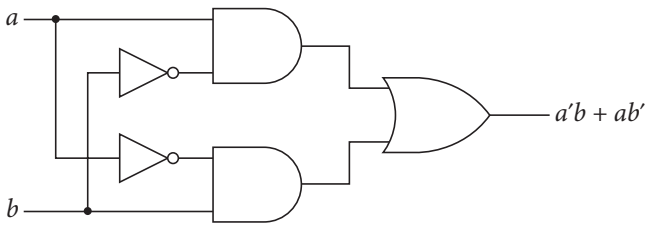
14. (a)



15. (a)



(c)



19. (a) $y(a, b, c) = a'bc + ab'c'$

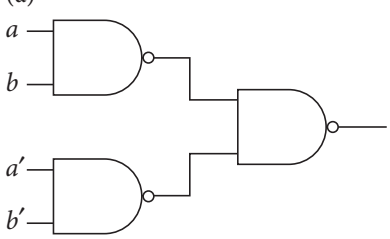
20. (a) $y(a, b, c) = (a+b+c)(a+b+c')(a+b'+c)(a'+b+c')(a'+b'+c)$
$(a' + b' + c')$

21. (a) $ab + a'b$ (d) $a'b + ab$

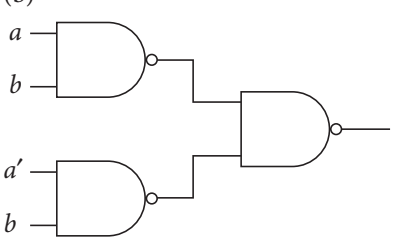
a	b	21(a)	21(d)
0	0	1	0
0	1	0	1
1	0	0	0
1	1	1	1

22.

(a)



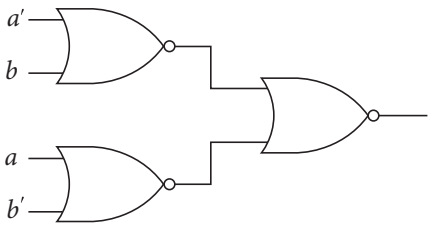
(b)



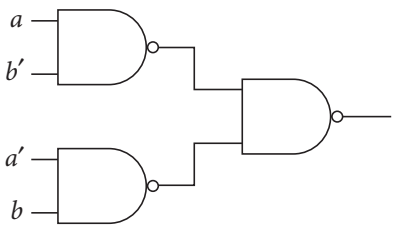
23. (b) $(a' + b)(a + b')$

a	b	
0	0	1
0	1	0
1	0	0
1	1	1

24. (b)



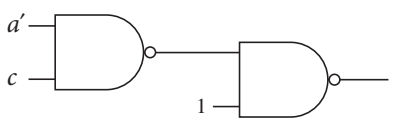
25. (a)



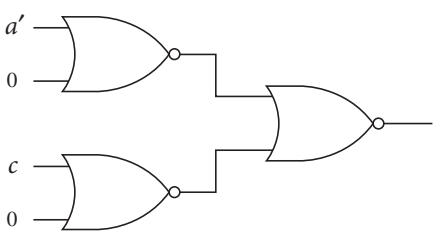
27. (a) $\Sigma(0,3)$ (d) $\Sigma(1,3)$

28. (b) $\prod(1,2)$

29. (a) $x = a'c$



30. (a) $x = (a')(c)$



31. (a) $x(a, b, c) = ac + b'c'$

32. (a) $x(a, b, c) = \prod(1, 2, 3, 6) = (a + c')(b' + c)$

33. (a) $x(a, b, c, d) = bc' + a'b'c + b'cd'$

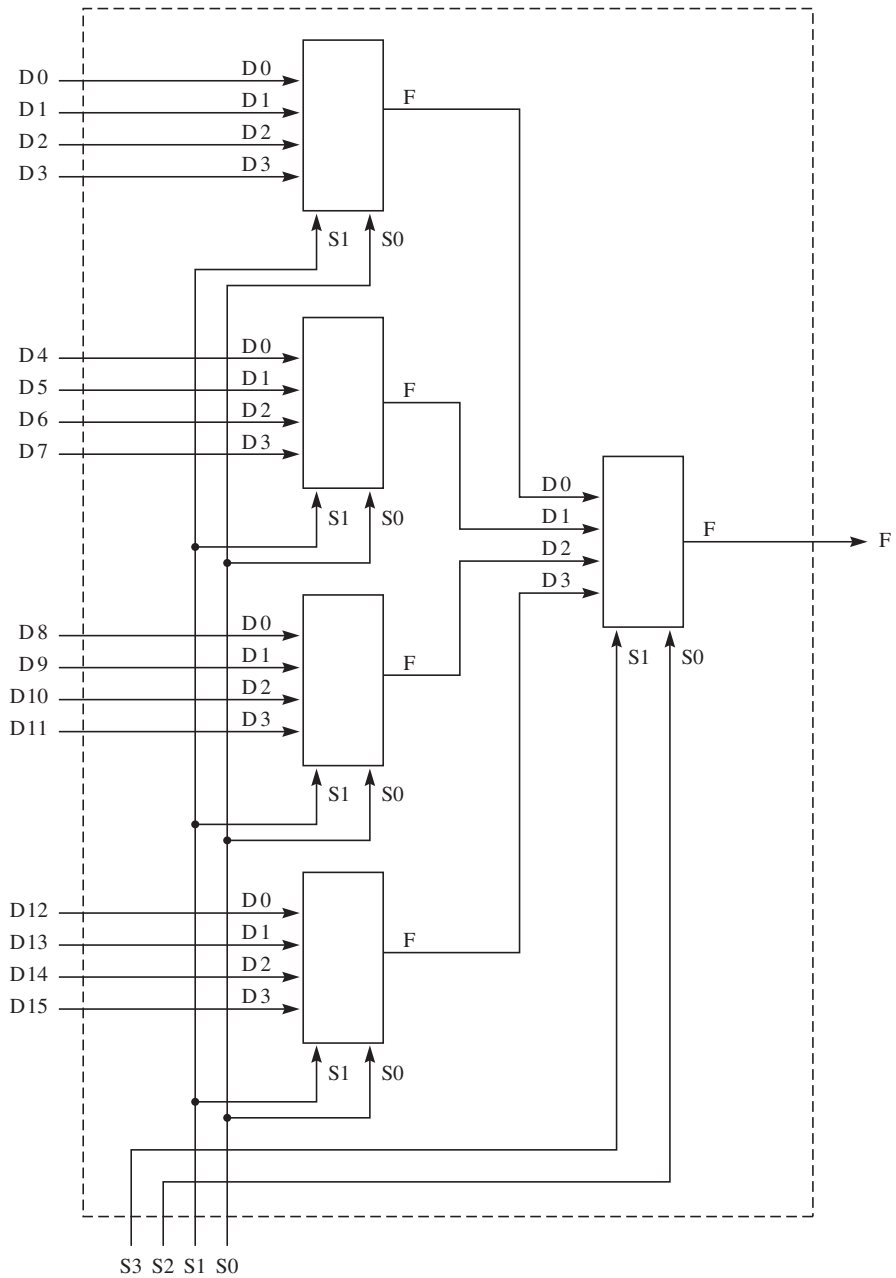
34. (a) $\prod(0, 1, 6, 7, 8, 9, 11, 14, 15)$, $x(a, b, c, d) = (b + c)(b' + c')(a' + c' + d')$ or $x(a, b, c, d) = (b + c)(b' + c')(a' + b + d')$

35. (a) $x(a, b, c) = a'b' + ab$

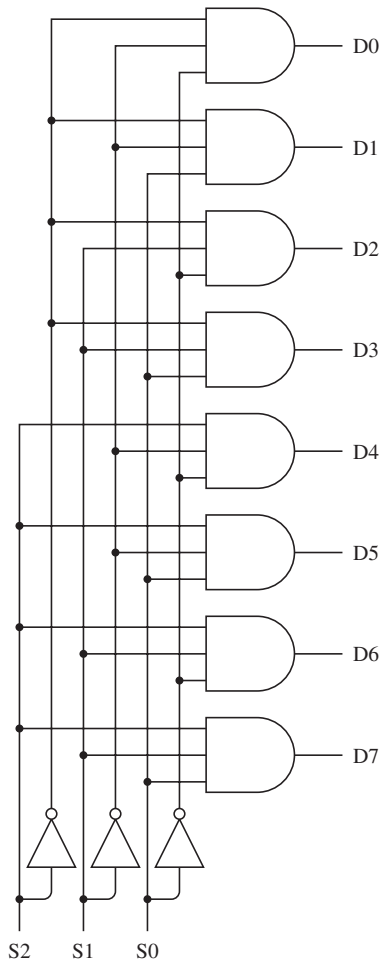
36. (a) $x(a, b, c, d) = bc + bd$

38. (a) The control line acts as an enable that passes the data through unchanged when it is 0. When the control line is 1, it disables the output, which is set to 1, regardless of the data input.

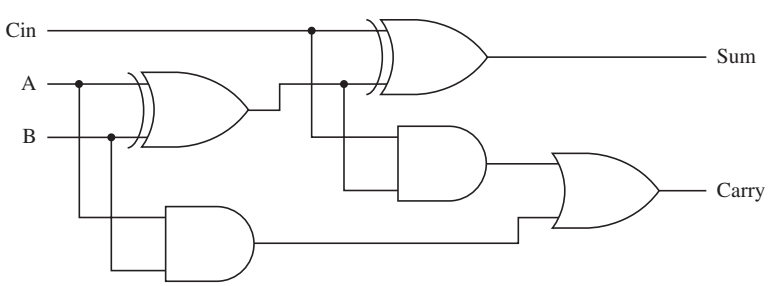
40.



42. (a)



46. (a)



(b) Maximum of three gate delays.

47. (b) Hint: If you look at Figure 10.52 and use the fact that a full adder has three gate delays and a half adder has one gate delay, you might conclude that the total gate delay is 10. However, it is less than that.

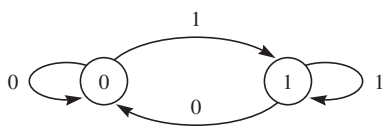
Chapter 11

1. The network will be stable if there is an even number of inverters.

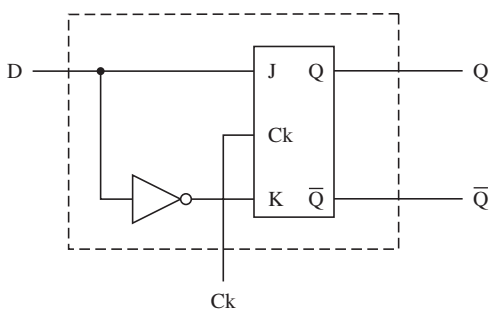
3.

	A	B	C	D	E	R2	S2	Q	$\bar{Q}$
(a)	0	0	1	1	0	1	0	0	1
(b)	0	0	0	0	0	1	0	0	1

5. (b)



8. (a)



13 (a)

$DA = A\,\overline{X1} + \overline{A}\,X1 + A\,B\,\overline{X2} + B\,X1\,X2$

$DB = \overline{B}\,\overline{X1} + B\,X1\,X2 + \overline{A}\,B\,X1$

$Y = \overline{A}\,X2 + A\,X1\,X2$

15 (a)

$JA = \overline{B}\,\overline{C}\,\overline{X} + B\,C\,X$ $\qquad$ $KA = \overline{B}\,\overline{C}\,\overline{X} + B\,C\,X$

$JB = \overline{C}\,\overline{X} + C\,X$ $\qquad$ $KB = \overline{C}\,\overline{X} + C\,X$

$JC = 1$ $\qquad$ $KC = 1$

Chapter 12

5. Hint: (a) Using 16 billion GiB equal to 16×1024^6 bytes, the chip would be 4.3×4.3 meters square, but you must show your calculation. (b) Yes, but you must explain.

12. (a)

```
lw $t0,20($s6)  # Register $t0 gets b[5]
add $t0,$s5,$t0 # Register $t0 gets g + b[5]
sw $t0,16($s4)  # a[2] gets g + b[5]
```

(b)

```
100011 10110 01000 0000000000010100
000000 10101 01000 01000 00000 100000
101011 10100 01000 0000000000010000
```

Index

C

D

H

I

J

K

L

M

N

O

S

T

U

V

W

X

Z