

Worked solutions to selected problems

CHAPTER 2

$$\begin{aligned}
 2.1 \text{ (a)} [1000110101]_2 &= 1 \times 2^9 + 0 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \\
 &\quad \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
 &= 512 + 32 + 16 + 4 + 1 \\
 &= [565]_{10}
 \end{aligned}$$

$$\begin{aligned}
 (b) [0.1000110101]_2 &= 1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} + 1 \\
 &\quad \times 2^{-5} + 1 \times 2^{-6} + 0 \times 2^{-7} + 1 \times 2^{-8} + 0 \times 2^{-9} + 1 \times 2^{-10} \\
 &= 0.5 + 0.03125 + 0.015625 + 3.90625 \times 10^{-3} + 9.765625 \\
 &\quad \times 10^{-4} \\
 &= [0.551\,757\,812\,5]_{10}
 \end{aligned}$$

- 2.2 To convert a decimal integer into binary the algorithm is to repeatedly divide it by two. The sequence of remainders written last (the MSB) to first (the LSB) is the resulting binary number. For a fractional number the algorithm is to multiply the fraction repeatedly when the sequence of carries over the decimal point form the binary number.

(a) Integer

(b) Fraction

2) <u>21345</u> (1	0.673
2) <u>10672</u> (0	<u>2</u>
2) <u>5336</u> (0	<u>1.346</u>
2) <u>2668</u> (0	<u>2</u>
2) <u>1334</u> (0	<u>0.692</u>
2) <u>667</u> (1	<u>2</u>
2) <u>333</u> (1	<u>1.384</u>
2) <u>166</u> (0	<u>2</u>
2) <u>83</u> (1	<u>0.768</u>
2) 41 (1	<u>2</u>
	<u>1.536</u>

$$\begin{array}{r}
 2) \underline{20} (0) & 2 \\
 2) \underline{10} (0) & 1.072 \\
 2) 5 (1) & 2 \\
 2) 2 (1) & 0.144 \\
 1 (0) & 2 \\
 \hline
 & 0.288
 \end{array}$$

$$= [0101001101100001]_2$$

Breaking this into groups of four, starting from the right, gives:

$$\begin{array}{ccccccccc}
 0101 & 0011 & 0110 & 0001 \\
 \text{or } [5 & 3 & 6 & 1]_{16}
 \end{array}$$

$$\begin{array}{r}
 2 \\
 0.576
 \end{array}$$

$$= [0.1010110]_2$$

Breaking into groups of four, working right from the binary point gives:

$$\begin{array}{cc}
 0.1010 & 1100
 \end{array}$$

$$\text{or } [0.AC]_{16}$$

2.3 $[3FDA104E]_{16}$

Convert each digit from hexadecimal to binary:

$$\begin{array}{ccccccccccccc}
 0011 & 1111 & 1101 & 1010 & 0001 & 0000 & 0100 & 0111
 \end{array}$$

Now divide into groups of three starting at the right-hand side:

$$\begin{array}{cccccccccccc}
 00 & 111 & 111 & 110 & 110 & 100 & 001 & 000 & 001 & 000 & 111
 \end{array}$$

Interpret each group as an octal number giving:

$$[0 \ 7 \ 7 \ 6 \ 6 \ 4 \ 1 \ 0 \ 1 \ 0 \ 7]_8$$

2.4 (a) $[1220]_4$; (b) $[0.11233]_4$.

2.5	(a) Signed magnitude	(b) 2's complement	(c) Offset binary
+63	00111111	00111111	10111111
-125	11111101	10000011	00000011
-0.125	10010000	11100000	01000000
	↑ Binary point	↑ Binary point	↑ Binary point

2.6 The received number comprises the data to be checked together with remainder, both of which could have become erroneous during transmission. The check is to divide the complete number by the constant ($x^5 + x^3 + x + 1$):

$$\begin{array}{r}
 101010) 101011111100 \mid 1000011 \\
 \underline{101010} \\
 \begin{array}{r}
 111111 \\
 \underline{101010} \\
 101010 \\
 \underline{101010} \\
 000000
 \end{array}
 \end{array}$$

The remainder is zero, indicating that the data 10101111 is correct.

$$2.7 \text{ (a)} \quad 11011.1101 \times 2^2 = 0.110111101 \times 2^7$$

The IEEE P754 single-precision format is a 32-bit grouping interpreted as:

$$(-1)^s 2^{e-127} (1.f)$$

where s is the sign digit, e is an 8-bit biased exponent and f is a 23-bit unsigned integer.

For the above number, $e - 127 = 7$ and $e = 134 = [10000111]_2$

The floating-point representation is then:

0 10000111 1101111010000000000000000

Table S1 Problem 3.2(a).

(i)

Location (Hex)	Instruction (Hex)	Mnemonic	Comment
00	28 06	LDM 06	Fetch loop constant to modifier
01	0009	CLR	Clear accumulator
→02	58 FF/M	STA FF/M	Store accumulator in modified address
03	80 03	JMPM 03	Jump if modifier zero to own location, i.e. stop
04	0008	INCM	Add +1 to modifier
05	80 02	JMP 02	Jump to start of loop
06	FF61		Loop constant (2's complement of 9F)

(ii)

Location (Hex)	Instruction (Hex)	Mnemonic	Comment
→00	0009	CLR	Clear accumulator
01	58 10/I	STA 10/I	Store accumulator in indirect address
02	50 08	LDA 08	Fetch operand address
03	48 09	SUB 09	Subtract 00F, i.e. last location cleared
04	98	JMP0 04	Jump to self if accumulator zero, i.e. stop
05	40 0A	ADD 0A	Add 0100, i.e. 00FF + 1 (incrementing by 1)
06	58 08	STA 08	Replace modified address
07	80 00	JMP 00	Jump back to start of loop
08	60		Operand address
09	00FF		Loop constant
0A	0100		Increment constant

Table S2 Problem 3.2(b).

(i)

<i>Location (Hex)</i>	<i>Instruction (Hex)</i>	<i>Mnemonic</i>	<i>Comment</i>
00	28 08	LDM 08	Fetch loop constant to modifier
01	0009	CLR	Clear accumulator
02	40 07	ADD 07	Add +1 to accumulator
03	58 FF/M	STA FF/M	Store accumulator in modified address
04	A0 04	JMPM 04	Jump to self if zero, i.e. stop
06	0008	INCM	Add +1 to modifier
06	80 02	JMP 02	Jump back to start of loop
07	1		Constant +1
08	FF61		Loop constant (2's complement of 9F)

(ii)

<i>Location (Hex)</i>	<i>Instruction (Hex)</i>	<i>Mnemonic</i>	<i>Comment</i>
00	50 0A	LDA 0A	Fetch count to accumulator
01	40 0B	ADD 0B	Add +1
02	58 0A	STA 0A	Replace in memory
03	58 0C/I	STA 0C/I	Store count in indirect address
04	50 0C	LDA 0C	Fetch operand address
05	48 0D	SUB 0D	Test for end of loop
06	98 06	JMP0 06	Jump to self if zero
07	40 0E	ADD 0E	Increment operand address by +1
08	58 0C	STA 0C	Replace in memory
09	80 00	JMP 00	Jump back to start of loop
0A	()		Accumulating number
0B	1		Constant +1
0C	60		Operand address
0D	00FF		Loop constant
0E	0100		Increment constant

$$(b) \ 1.01110101 \times 2^{-12} = 0.101110101 \times 2^{-11}$$

and $e - 127 = 11$ giving $e = 138$ and the floating-point representation as:

0 10001011 1011101010000000000000000

$$(c) -0.110101011 \times 2^6 \text{ gives}$$

1 10000110 1101010110000000000000000

$$(d) -0.000001101 \times 2^{-1} \text{ gives}$$

1 01111001 1101000000000000000000000

Table S3 Problem 3.3.

<i>Location (Hex)</i>	<i>Instruction (Hex)</i>	<i>Mnemonic</i>	<i>Comment</i>
00	28 10	LDM 10	
→01	50 A0	LDA A0/M	
02	98 06	JMP0 06	
→03	A0 03	JMPM 03	Test for zero loop
04	0008	INCM	
→05	80 01	JMP 01	
→06	0005	EXAM	
07	50 11	STA 11	
08	0005	EXAM	
09	50 11	LDA 11	
0A	40 12	ADD 12	
0B	58 13/I	STA 13/I	Unpacking of address of zero location and storing in table
0C	50 13	LDA 13	
0D	40 14	ADD 14	
0E	58 13	STA 13	
0F	80 03	JMP 03	
10	FFC0		Test loop constant, -0040
11	()		Temporary store
12	A0		
13	A1		
14	1		Address table constants

CHAPTER 3

3.2 Machine-code programs for this problem are given in Tables S1 and S2.

3.3 A typical program for this problem is shown in Table S3. Note that in this program we are handicapped by having only one modifier register and not being able to increment a memory location directly. The 'test for zero' loop employs the modifier register to increase the speed of operation of the program and the 'unpacking' of the address is performed using indirect addressing. An alternative approach is to use the indirect address function for the test loop, since the actual address could then be held in the indirectly addressed location.

3.4 In this program, given in Table S4, we can take advantage of the fact that the same modifier constant can be used; therefore we can employ a modified instruction for both transfers.

3.7 This program is shown in Table S5. The basic logic is to extract the 4-bit BCD digits using the AND function and then to multiply by

Table S4 Problem 3.4.

<i>Location (Hex)</i>	<i>Instruction (Hex)</i>	<i>Mnemonic</i>	<i>Comment</i>
00	28 0A	LDM 0A	
01	50 80/M	LDA 80/M	
02	58 0A	STA 0A	
03	50 F0/M	LDA F0/M	
04	58 80/M	STA 80/M	
05	50 0A	LDA 0A	
06	58 F0/M	STA F0/M	
07	A0 07	JMPM 07	
08	0008	INCM	
09	80 01	JMP 01	
0A	FFE0		
0B	()		

Table S5 Problem 3.7.

<i>Location (Hex)</i>	<i>Instruction (Hex)</i>	<i>Mnemonic</i>	<i>Comment</i>
00	28 0E	LDM 0E	Fetch constant to modifier
01	50 0F	LDA 0F	
02	98 06	JMP0 06	
03	50 0F	LDA 0F	Fetch BCD word to accumulator
04	13 04	SRL 4	Shift right four places
05	58 0F	STA 0F	Store shifted word
06	60 11	AND 11	Extract BCD digit
07	40 10	ADD 10	Add contents of temporary store
08	A0 08	JMPM 08	Stop with binary word in accumulator
09	0008	INCM	
0A	A8 12	MUL 12	Multiply by binary 10, result in X-reg
0B	0006	EXAX	Exchange accumulator with X-register
0C	58 10	STA 10	
0D	80 03	JMP 03	
0E	FFFF		Count constant -3
0F	()		BCD word
10	0000		Temporary storage
11	000F		AND constant binary 15
12	000A		Binary 10

binary 10; the result is stored away and added to the next BCD digit, which is multiplied by binary 10, and so on.

Note that multiplying two integers $\times 2^{-15}$ results in the product being placed in the least significant end of the X-register (the sign digit of the X-register is not normally used).

Table S6 Problem 3.8.

<i>Location (Hex)</i>	<i>Instruction (Hex)</i>	<i>Mnemonic</i>	<i>Comment</i>
00	50 04	LDA 04	Fetch a to accumulator
01	B8 05	XOR 05	Form $a \oplus b$
02	B8 06	XOR 06	Form $a \oplus b \oplus c$
03	0000	STOP	
04	()	a	
05	()	b	
06	()	c	

Table S7 Problem 3.10.

<i>Location (Hex)</i>	<i>Instruction (Hex)</i>	<i>Mnemonic</i>
00	28 0A	LDM 0A
01	50 0C/IM	LDA 0C/IM
02	58 0B	STA 0B
03	50 0D/IM	LDA 0D/IM
04	58 0C/IM	STA 0C/IM
05	50 0B	LDA 0B
06	58 0D/IM	STA 0D/IM
07	A0 07	JMPM 07
08	0008	INCM
09	80 01	JMP 01
0A	FFED	
0B	()	
0C	0280	
0D	0180	

3.8 This is a trivial problem if the original expression is manipulated correctly; for example,

$$\begin{aligned} S &= c(\bar{a}\bar{b} + ab) + \bar{c}(\bar{a}\bar{b} + a\bar{b}) \\ S &= c\bar{z} + \bar{c}z, \text{ where } z = \bar{a}\bar{b} + a\bar{b} = a \oplus b \\ S &= c \oplus z = c \oplus a \oplus b \end{aligned}$$

A suitable program is shown in Table S6.

3.10 The main difference with this program, shown in Table S7, compared to that of problem 3.4 is that we need to use indirect addressing because the available address bits have been exceeded: that is, 0160 → 0180 (sector 1) and 02D0 → 02F0 (sector 2). Since we

Table S8 Problem 3.11.

<i>Label</i>	<i>Instruction</i>	<i>Comment</i>
	ORG 0000	Locates program at 0000
C1	EQU n	
C2	EQU -k	
C3	EQU 0OFF	
C4	EQU 0001	
C5	EQU 0000	
		Constants
	LDM C2	Load modifier with -k
	CLA	Clear accumulator
	ADD C1	Add n to accumulator
START	STA MEMREF	Store n in MEMREF
	LDA MEMREF/I	Load contents of address n into accumulator
	AND C3	AND with 00FF
	OUTA	Output to I/O register
	LDA MEMREF/I	Load contents of address n into accumulator
	SRL 8	Shift right 8 places
	OUTA	Output to I/O register
	LDA MEMREF	Load address n into accumulator
	ADD C4	Increment accumulator by +1
	STA MEMREF	Store in MEMREF
	INCM	Increment modifier register by +1
LOOP	JMPM LOOP	If modifier register equals zero loop, i.e. stop
	JMP START	Jump to START
	END	Pseudo-operation to signify end of code
MEMREF	RMB 2	Reserve two memory bytes, i.e. 1-word

are working in sector 0 we may specify directly within the sector, but locations outside the sector must be specified indirectly. This means including two extra 'pointer' locations; the addresses in these locations may be modified in the usual way by setting the modifier bit in the instruction to 1. Note that when the modifier and the indirect address bits are both set to 1, it is the content of the indirect address that is modified.

- 3.11 The assembler code program is shown in Table S8. Note that the EQU pseudo-operation is liberally used to set up constants. This is good practice since it permits easy changes to these to be made at a later date without having to modify any of the code below. For example, in this case the values of n and k are not defined absolutely here. MEMREF is used as a temporary store for the current address of the memory location to be output to the I/O register and all such space is allocated after the END pseudo operation. The AND function is used as a masking function to enable only the least significant byte to be meaningful.

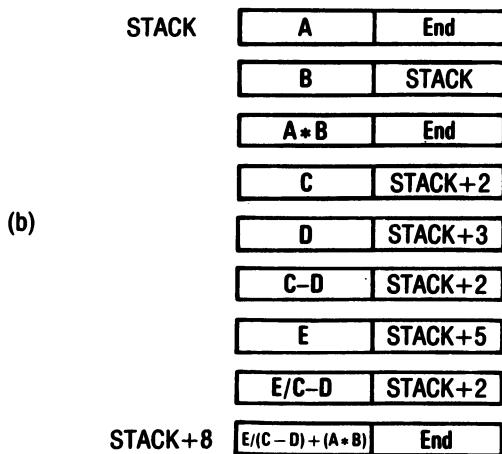
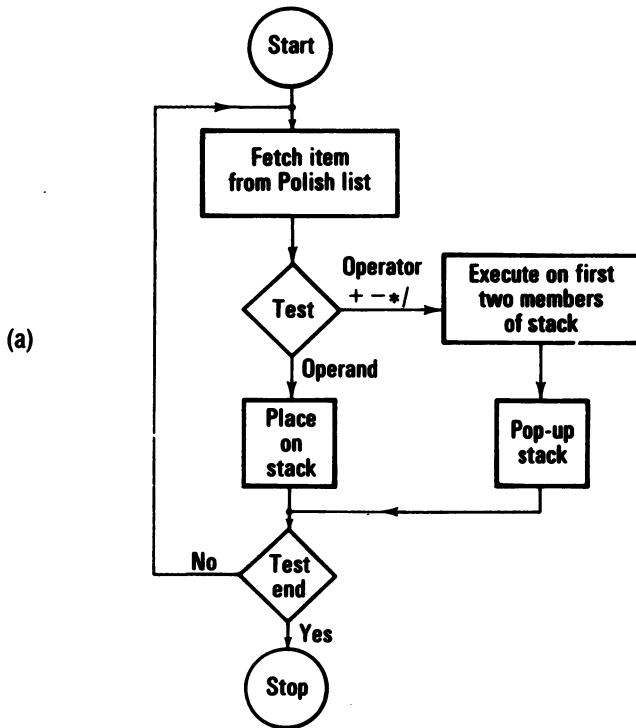


Figure S1 Problem 3.12.

Table S9 Problem 3.12.

<i>Label</i>	<i>Instruction</i>	<i>Comment</i>
C1	ORG 0000	Locates program at 0000
C2	EQU AE	Address of first word of Polish list
C3	EQU 0D	Terminating character
C4	EQU 0015	
C5	EQU 0002	
	EQU 0001	
START	CLR EXAM	Operator constants
J1	LDA C1/M SUB C2	Clear modifier
SELF	JMPN TABLE_1	Fetch first word from Polish list
	JMP0 SELF	Test for operator/operand/END operator
	ADD C2	
	STA TEMP	
	LDA ST+2	
	STA ST+3	
	LDA ST+1	
	STA ST+2	
	LDA ST	
	STA ST+1	
	LDA TEMP	
B1	STA ST INCM	
	JMP J1	
TABLE_1	ADD C3	Push down stack
	JMP0 J2	
	ADD C4	
	JMP0 J3	
	ADD C4	
	JMP0 J4	
	ADD C5	
	JMP0 J5	
J2	LDA ST+1	Place operand on stack
	DIV ST	
B2	STA ST	Increment modifier register
	JMP J6	
J3	LDA ST+1	
	SUB ST	
	JMP B2	
J4	LDA ST+1	Divide
	ADD ST	
	JMP B2	
J5	LDA ST+1	
	MUL ST	
	EXAX	
	JMP B2	
J6	LDA ST+2	Subtract
	STA ST+1	
	LDA ST+3	
	STA ST+2	
	JMP B1	
	END	Add
STACK TEMP	RMB 8 RMB 2	Multiply
		Pop-up stack
		Operand stack of depth four
		Temporary store

Table S10 Problem 3.12.

<i>Label</i>	<i>Instruction</i>	<i>Comment</i>
SPACE	ORG 0000	Locates program at 0000
C1	EQU F0	Address of start of stack
C2	EQU AE	Address of first word of Polish list
	EQU 0001	Constant +1
	CLR	
	EXAM	}
	LDA C1/M	
	STA SPACE/M	}
	INCM	
	LDA C1/M	
	STA SPACE/M	
	INCM	
	EXAM	
	SUB C2	
	EXAM	}
	LDA SPACE/M	
	EXAM	
	SUB C2	
	EXAM	
	LDA SPACE/M	
	EXAM	
	SUB C2	
	EXAM	
	ADD SPACE/M	
	STA SPACE/M	
	INCM	

3.12 The Polish notation (see Chapter 3, reference [5]) is well suited to stack operation, and places the operators after, rather than between, the operands. We assume initially that the Polish list consists of fixed maximum-length expressions, located from AE Hex upwards and terminated with the marker @. In this case it would be very simple to construct a push-down/pop-up store, or stack, of finite depth to perform the evaluation. Figure S1(a) shows a typical flowchart for such a routine, and a simple assembler code program is given in Table S9. In this program the stack has been programmed by physically moving the contents of the memory locations; this is not the best way since, in general, it is always better to manipulate addresses rather than words.

An alternative approach is shown in Table S10. In this case the modifier register acts as a **pointer** (the same results could be achieved by indirect addressing) pointing to the address of the top member of the stack. Note that only the 'top' of the stack is moved

during the stacking and unstacking processes. In this case it is also necessary to make special provisions to ensure that the arithmetic operations are performed in the correct order ($A * B$) since the normal application of this technique gives $B * A$ if B is at the top of the stack. It is still necessary with this type of stack to know its precise size beforehand; a perfectly general stack for handling Polish lists of any size may be constructed. In this case the operations of push-down and pop-up become simply a matter of altering list pointers, as shown in Figure S1(b).

If at the start of the process a block of continuous words is allocated in memory for the list cells, sooner or later these will become exhausted. Moreover, the memory will contain deleted cells as a result of amending the lists. In order to ensure maximum utilization an **available space list** is created (using a stack principle) which is employed to note the addresses of all free cells. When a new cell is required it is obtained from the available space list and, conversely, when no longer required it is returned to the list. Note that the top of the list is continually changing, and is given by the pointer obtained from the available space list, in this particular case taken in sequence from **STACK** to **STACK+8**.

CHAPTER 4

4.1 A possible microprogram for the LINK instruction is given in Table S11. Alternative techniques could include placing the link address in special modifier registers, or using memory locations with prewired addresses. Neither of these techniques, however, allows 'nesting' of the subroutines, and a better approach would be to use a push-down/push-up stack register with modification facilities provided on the top register.

Table S11 Problem 4.1.

Next address	Micro-operation	Comment
01 : 06 }		;fetch instruction
21	E_0, M_i	;content of control register to memory register
22	$C_{ad}, R/W$;instruction address bits to c-bus
23	WAIT	;wait for end of memory write cycle
24	$C_{os}, X_i, +1lsb$;increment instruction address by +1
25	AU_o, E_i, END	;transfer incremented address to control register

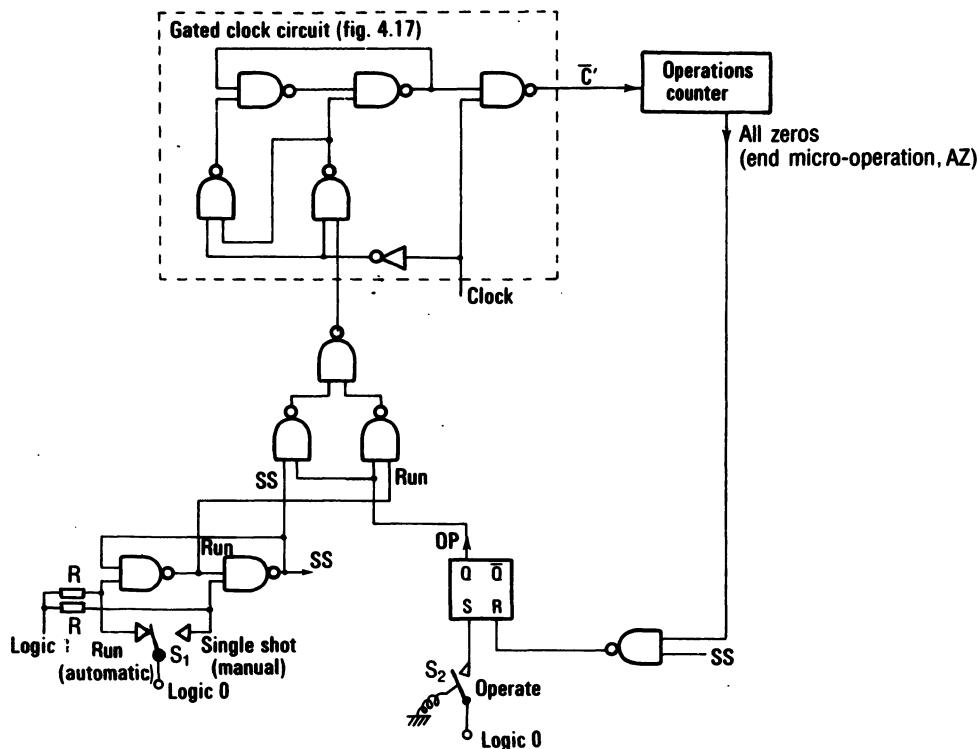


Figure S2 Problem 4.4.

4.4 A typical circuit to perform this function is shown in Figure S2. To operate the computer in the automatic mode switch S_1 is put to 'run' and the operate key depressed; to stop the machine S_1 is changed over to the single-shot position. The gated clock circuit ensures that, though there may be a momentary pause in the clock pulses, only complete pulses reach the counter. Note that all the switches must be buffered (by a flip-flop) to eliminate contact bounce.

When working in the manual mode it is necessary to depress the operate key (which is spring-loaded) in order to initiate the execution of an instruction. The operate key sets the OP flip-flop, which is reset in turn by the micro-operation END. Note that when the operate key is depressed in the single-shot mode the flip-flop has both the S and the R inputs equal to 1: that is, the 'indeterminate' condition. However, the effect of this is to put both the Q and \bar{Q} outputs to

Table S12 Problem 4.5.

<i>Next address</i>	<i>Micro-operation</i>	<i>Comment</i>
27	C _{ad} ,R/W	;Fetch multiplier
28	WAIT	
29	M _o ,MD _i	;Multiplier to modifier register
30	A _c ,M _i	;Use M-register to hold Multiplicand
31	A _c ,K ₄ \bar{K}_3 \bar{K}_2 \bar{K}_1 \bar{K}_0	;Clear accumulator, set K-counter to 16
32	BRC(MD ⁰ = 0) 35	;Test multiplier value
33	A _o ,X _i ,M _{au}	;Add multiplicand to partial product
34	AU _o ,A _i	
35	A _o ,MD _i ,R	;Shift accumulator and modifier one place right
36	K _d ,BRC(K = 0) 32	;Decrement K-counter, branch if non-zero
37	MD _o ,X _i ,END	;Transfer least significant part of product to the X-register

logic 1 (using NAND logic) which will allow the machine to operate, thereby changing the counter and causing AZ to go to 0; this assumes of course that the machine responds before the key is released.

4.5 The register set of our primitive computer does not really contain enough registers to be able to undertake multiplication and division efficiently. The modifier register can be used to store the multiplier and has to have associated with it a bit-test facility for MD⁰ (bit position 0). The multiplication algorithm used is simply to test the least significant bit of the multiplier and add the multiplicand to the partial product, held in the accumulator, or not, depending whether it is a 1 or 0 respectively, and then to shift the partial product right one place. In our case the LSB of the accumulator is assumed to move into the MSB of the modifier: i.e. the modifier is an extension of the accumulator and both shift together. The memory register is used to hold the multiplicand. A microprogram is shown in Table S12.

CHAPTER 5

5.1 (a) Arithmetic using sign and magnitude numbers is very simple for multiplication and division (it is only necessary to compare signs: if different, the result is negative, if the same, the result is positive). However, addition and subtraction becomes a complicated process. The magnitude of their operands themselves must be compared as well as the signs:

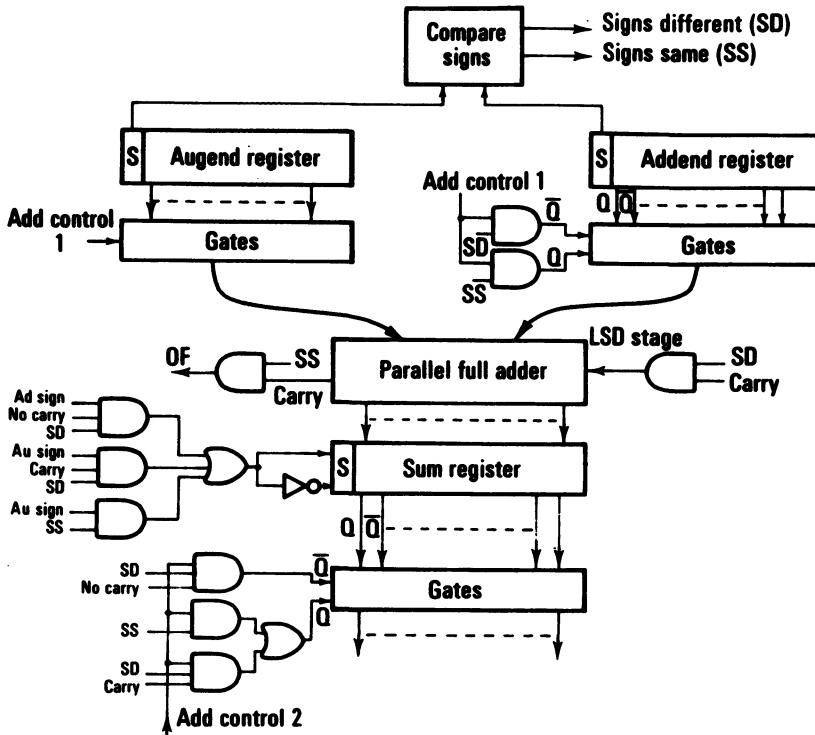
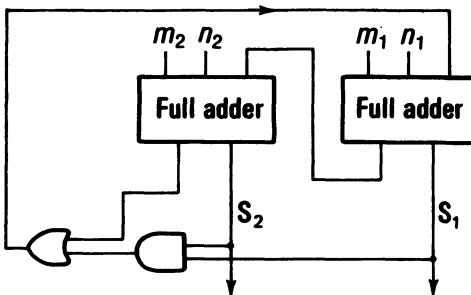


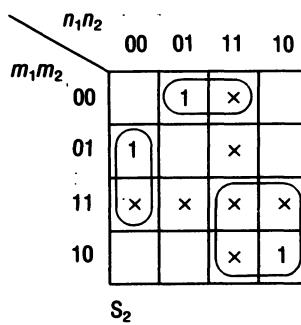
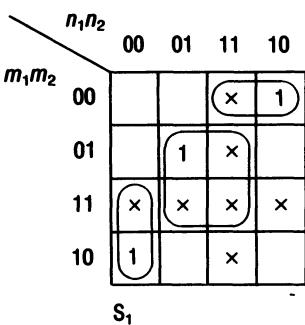
Figure S3 Problem 5.1.

1. Signs the same: add magnitudes, duplicate sign in result.
2. Signs different: (i) *Augend larger*, add 1's complement of addend to augend (with end around carry) result given sign of augend.
(ii) *Addend larger* (or equal), add 1's complement of addend to augend, complement result, result given sign of addend.

To implement the algorithm in hardware it is necessary to be able to compare the augend and addend for the $>$, $<$, or $=$ relationships using, for example, the circuit shown in Figure 5.36. A conventional parallel full adder can be used, with the overflow (that is, carry from MS stage) being fed back to the carry input of the LS stage (to give an end-around carry). The complemented result can be obtained by gating out the \bar{Q} outputs of the sum register in place of the usual Q outputs. Similarly the \bar{Q} register outputs can be gated to the full-adder circuits to obtain the complement of the addend. However, it may be shown that an end-around carry only occurs if the augend is larger than the addend; consequently, the final carry output may be used as a control signal thus allowing the comparator

**Figure S4** Problem 5.3.**Table S13** Problem 5.3.**(a) Mod 3 addition**

	0	1	2	
+	00	01	10	n_1n_2
m_1m_2	00	01	10	
0	00	00	01	10
1	01	01	10	00
2	10	10	00	01

(b) K-Maps

to be dispensed with. If this technique is used special precautions must be taken with the overflow resulting from the addition of two like sign numbers. Subtraction may be performed by changing the sign of the subtrahend. A suggested block diagram for a sign and magnitude arithmetic unit is shown in Figure S3.

(b) The 1's complemented arithmetic unit is quite straightforward, the arithmetic operations being performed in much the same way as for the normal 2's complemented system, with the exception that the end-around carry scheme must be incorporated.

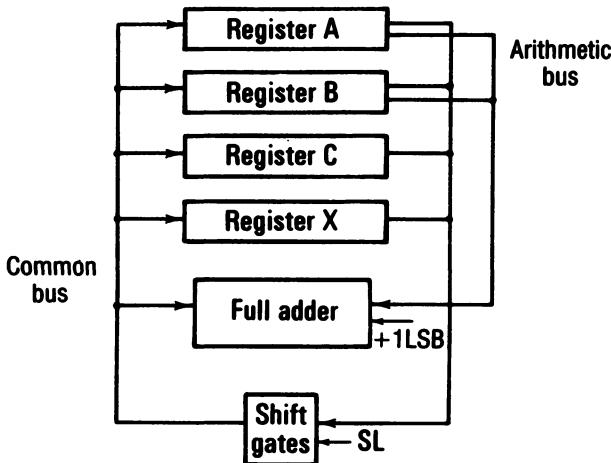


Figure S5 Problem 5.4.

Table S14 Problem 5.4.

<i>Next address</i>	<i>Micro-operation</i>	<i>Comment</i>
27	RA_o, X_i, R, RB_{au}	; RA to X-reg. shifted one place right, output of RB taken to input of arithmetic unit
28	AU_o, RA_i	; Output of AU to RA
29	$RA_o, X_i, +1lsb$; Complemented output RA to X-register
30	AU_o, RC_i, END	; Output of AU to RC

5.3 The design tables for this problem, treated as a switching network, are shown in Table S13; note that only mod 3 numbers (0, 1, 2) are allowed. The equations, shown below, may be implemented in the usual way:

$$\begin{aligned} S_1 &= \bar{m}_1 \bar{m}_2 n_1 + m_2 n_2 + m_1 \bar{n}_1 \bar{n}_2 \\ S_2 &= \bar{m}_1 m_2 n_2 + m_1 n_1 + m_2 \bar{n}_1 \bar{n}_2 \end{aligned}$$

An alternative design approach is to use binary full adders and to correct the resulting binary sum by adding 1 into the least significant stage. This may be done by detecting the overflow (carry-out) and $S_1 S_2 = 1$ conditions; a logic diagram is shown in Figure S4.

5.4 We assume that the registers (RA, RB and RC) are connected to a common parallel highway system, complete with shifting gates, similar to that discussed in the main text; a typical logic diagram is shown in Figure S5, and a suitable microprogram in Table S14. Note

that the same symbolism as used in the text is employed for the microinstructions: that is, RA_0 puts the output of register RA to the common highway; \bar{RA}_0 puts the complemented outputs to the highway, etc.

5.6 (a) The first part of this problem is fairly straightforward, and a suggested flowchart is shown in Figure S6. The main difficulties are caused through a lack of double-length mode operation (this becomes even more apparent when the microprogram is considered later). Consequently, it is necessary to program the double-length shift operation required for multiplication. This is done by extracting the least significant digit of the most significant half of the partial product, before shifting, and inserting it in the most significant digit position of the least significant half of the partial product, *after* shifting. Note also the formation of a code word (TAG) from the inspection of the sign digits of the multiplicand and multiplier; the codeword is used in the program to determine the appropriate correction operations.

Table S15 Problem 5.6.

Next address	Micro-operation	Comment
21	$C_{ad}, R/W$	
22	A_o, X_i	;transfer M'plicand to X-reg
23	$A_c, K_4 \bar{K}_3 \bar{K}_2 \bar{K}_1 K_0$;clear Acc. set K-counter to $K_4 \bar{K}_3 \bar{K}_2 \bar{K}_1 K_0$
24	WAIT	
25	$M_o, MD_i, MD_o^{16}, G_i$;transfer M'plier to Modifier reg., set flag G
26	$BR(MD^0 = 0) 29$	
27	A_o, M_i, M_{au}	};add M'plicand to most significant half of partial product, Set flag F
28	AU_o, A_i, A_o^0, F_i	
29	A_o, M_i, R	};shift P.P right
30	M_o, A_i	
31	$BR(X^{16} = 0) 33$	
32	A_o^{15}, A_i^{14}	;arithmetic shift
33	MD_o, M_i, R	};shift multiplier right
34	M_o, MD_i	
35	F_o, MD_i^{15}	;set MSB of M'plier to flag F
36	$K, K_d, BR(K \neq 0) 26$;decrement K-counter and Jump back
37	$G, BR(G = 0) 42$	
38	$X_o, M_i, M_{au} + 1lsb$	};flag G set add 2's complement of M'plicand to MSPP
39	AU_o, X_i	
40	A_o, M_i, M_{au}	
41	AU_o, A_i	
42	MD_o, X_i, END	;transfer LSPP to X-register

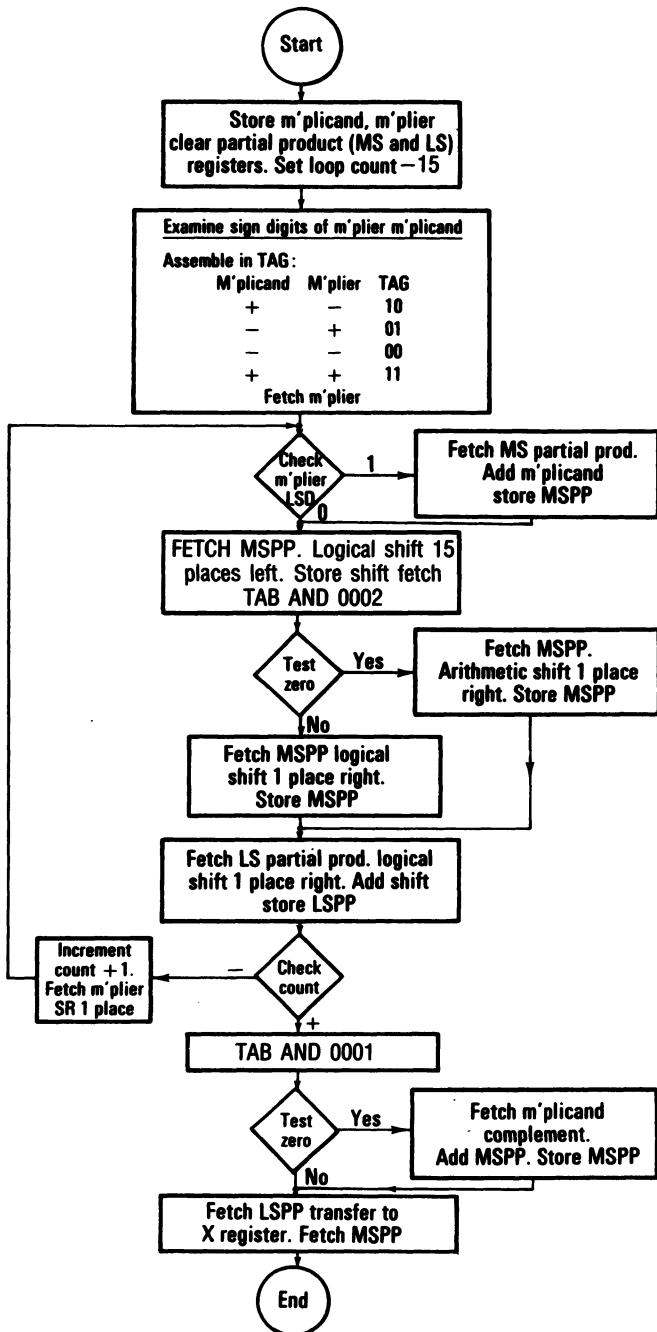


Figure S6 Problem 5.6(b).

(b) To implement the algorithm in hardware presents a number of problems. Strictly speaking the machine structure is too primitive (due mainly to an insufficient number of registers) to incorporate multiplication/division algorithms effectively. In this example it is necessary to use the modifier register as part of the multiplication logic; this limits the operation of the machine since the programmer must store away the contents of the modifier register (which must be retrieved later) before executing a multiplication instruction. This function could also be included as part of the microprogram, by initially writing away the contents of the modifier (at micro-operation level) into a predetermined location using a wired address technique. A suitable microprogram for the multiplication algorithm is shown in Table S15, and follows a similar logic to the flowchart given in Figure S6.

The initial micro-operations are concerned with loading the registers; the multiplier is held in the modifier register, and the X-register contains the multiplicand. Partial products are accumulated in the accumulator (most significant part only) which is initially cleared to zero; register M is left free and is used as a temporary buffer store. The least significant part of the partial product is inserted bit by bit into the most significant end of the modifier register during the double-length shift operations. Note that it is necessary to invent some new micro-orders concerned primarily with bit manipulations. These are represented by $A_i^n A_o^j$ which refers to input or output from a particular bit stage (n, j) of a register: for example, $A_o^{14} A_i^{15}$ sets the bit contained in bit 14 into b15 (MSB) of register A; this operation is required for the arithmetic right shift function. In addition it is necessary to be able to set flags or flip-flops to store certain transitory conditions: for example the sign digit of the multiplier (in the example flag G is used for this purpose). Flag F is used to store the least significant digit of the partial product; this is required during the right shifting of the multiplier. The most significant digit of the multiplicand (held in the X-register) is treated as a condition (X_{MSD}) in the usual way.

- 5.8 The control logic for the multiplier follows conventional lines and is based on the microprogram principle. For example, the decoded output of the multiplier compare logic would be used in conjunction with the microinstruction sequence to generate the necessary switching waveforms. The control unit must perform the following operations.
1. Load multiplicand and multiplier registers. Clear input register. Set cycle counter to n (number of bits).

2. Examine decoder outputs from multiplier compare logic:

$\bar{M}_2 M_3$: generate normal and gate signals
 M_2 : generate left shift and gate signals

(At this stage the contents of the input register would be added to the contents of the multiplicand register.)

3. Transfer contents of partial product register to input register.
 4. Examine decoder outputs:

if $M_2 M_3 \neq 1$ jump to 6
 $M_2 M_3 = 1$ generate normal and gate signals

5. Transfer partial product register to input register.
 6. Generate: multiplicand shift two places left, multiplier shift two places right.
 7. Check end of multiplication instruction

$n = 0$ stop
 $n \neq 0$ decrement cycle counter, jump to 2

Note that two counters are required:

- (a) An N -bit cycle counter (where $n = 2^N$) capable of being preset to n (generally by setting to the all 1s condition) and counting down for each input pulse. The count-down requirement is easily achieved by inverting the literals of the normal count-up flip-flop terms. For example, in the 3-bit binary counter of Figure 4.7(d) the input terms would become:

$$J_A = K_A = 1; \quad J_B = K_B = \bar{A}; \quad J_C = K_C = \bar{A}\bar{B}$$

- (b) A 3-bit clocked control counter capable of being reset to allow jump operations.

The logical circuits required for register shifting, gating, etc., are shown in Figure S7.

5.10 Assume the normal computer structure described in the text.

- (a) The mask or AND operation may be performed using special gating logic ($n \times 3$ input gates) connecting A and B to the common bus. It is necessary to form $A \cdot B$ transferring the result first to the X-register, and then back to the A-register.
 (b) The inclusive OR operation may be performed in an analogous manner to that described in (a) above for the AND function. The exclusive OR operation may be performed very easily by generating a micro-operation which suppresses the carries in the full-adder unit (effectively performing modulo 2 addition only).
 (c) To generate the 2's complement of register A it is first necessary

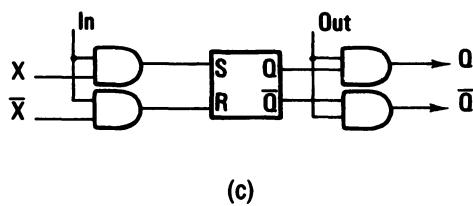
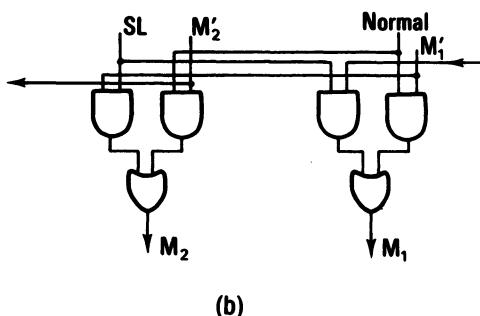
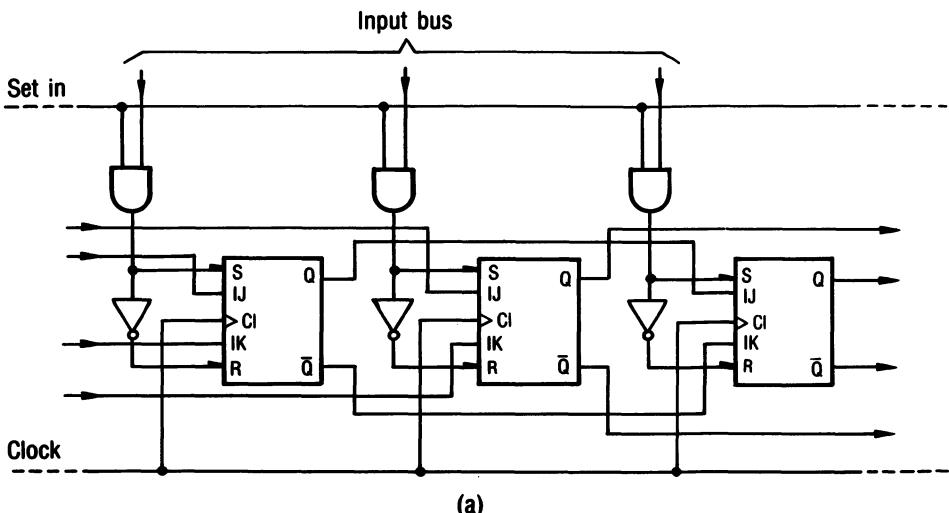


Figure S7 Problem 5.8: (a) register shifting two places right; (b) gates shifting one place left; (c) buffer register and gates.

to find the inverse. This may be done by implementing the micro-operation ‘Output complementary outputs of A to common bus’ (A_o) which effectively interchanges the Q and \bar{Q} outputs on the common bus. It is now a simple matter to form the 2’s complement by using the micro-operations:

$$\begin{array}{l} \bar{A}_o X_i + 1\text{LSB} \\ AU_o, A_i \end{array}$$

which has the effect of forming the complement of register A in the X-register, adding +1, and then putting the result back into the A-register.

(d) The NAND function can be performed either by using direct NAND gating to the common bus or by using the AND function followed by the micro-operations:

$$\begin{array}{l} \bar{A}_o X_i \\ X_o A_i \end{array}$$

CHAPTER 6

- 6.2 Eight $4K \times 8$ RAM chips and two $16K \times 8$ ROM chips are required to satisfy the memory requirement of this microcomputer. The chip enables of the memory chips can be used to select them and since ten are required in total, four address lines (most significant) will need to be allocated for this purpose. In practice the most significant address line, A15, can be used to select either ROM or RAM. Address lines A(14:12) can be decoded by a 3-to-8-line decoder enabled by A15 (low for assertion) to select one of the eight RAM chips, and A(15:14) decoded directly to select one of the two ROM chips. A block diagram of this arrangement is shown in Figure S8(a). An alternative arrangement using standard cell macros would be to use one 3-to-8-line decoder to select the RAM chips and a 2-to-4-line decoder for the ROMs. The associated memory map is given in Figure S8(b).
- 6.3 The efficiency of recording is directly related to the space occupied on the disk by the bit-magnetization state. It is, then, inversely proportional to the number of flux changes per bit and so the solution requires the number of flux transitions per bit to be evaluated. These are best seen by sketching the waveforms for each of the recording modes, covering all of the possible bit-transitions for each case, i.e. $0 \rightarrow 1$, $1 \rightarrow 1$, $1 \rightarrow 0$, $0 \rightarrow 0$ and averaging the result. Figure S9 shows these and tabulates the efficiencies normalized to the NRZ mode.
- 6.4 In the worst case of randomly stored data the table-look-up program is required to search for any bit combination within the word, and each word must be examined in turn; a typical program is shown in Table S16. If we assume a single address machine with an average instruction time of $1\mu s$, the maximum time to search through the

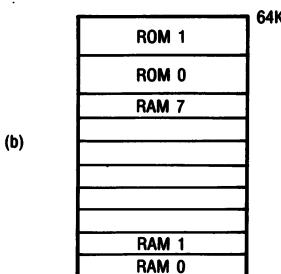
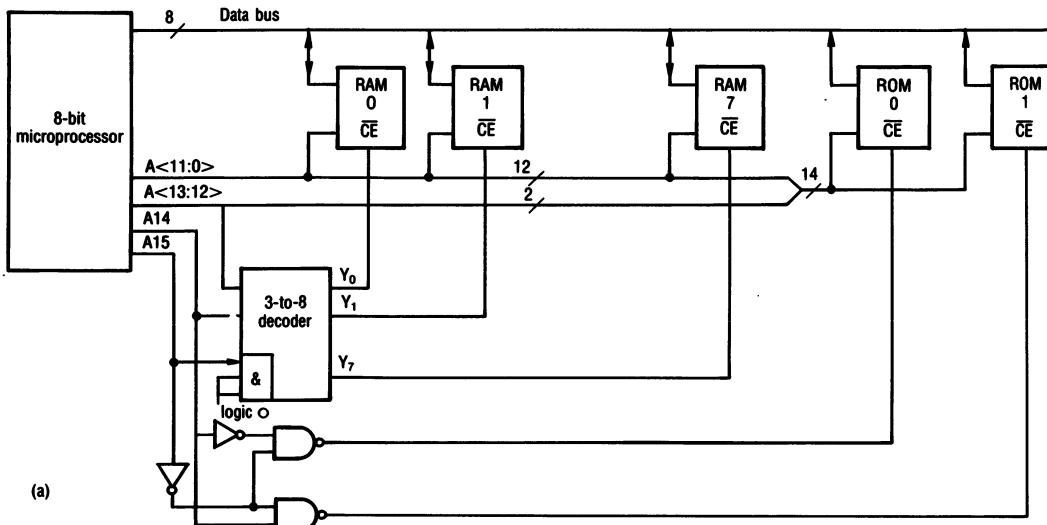


Figure S8 Problem 6.2: (a) microcomputer memory block diagram; (b) memory map.

table is $1 + (6 \times 65536 \times 1) \approx 400\text{ ms}$ which on average gives a 200 ms search time. A special table-look-up instruction can be provided by combining the instructions

AND	CONS
SUB	VARS
JMP0	
INCM	
JMP	BACK

The instruction would need to be a specified address-type instruction, the address specifying the mask constant, with the tag word being stored immediately after it. Allocating a mnemonic TLU for this

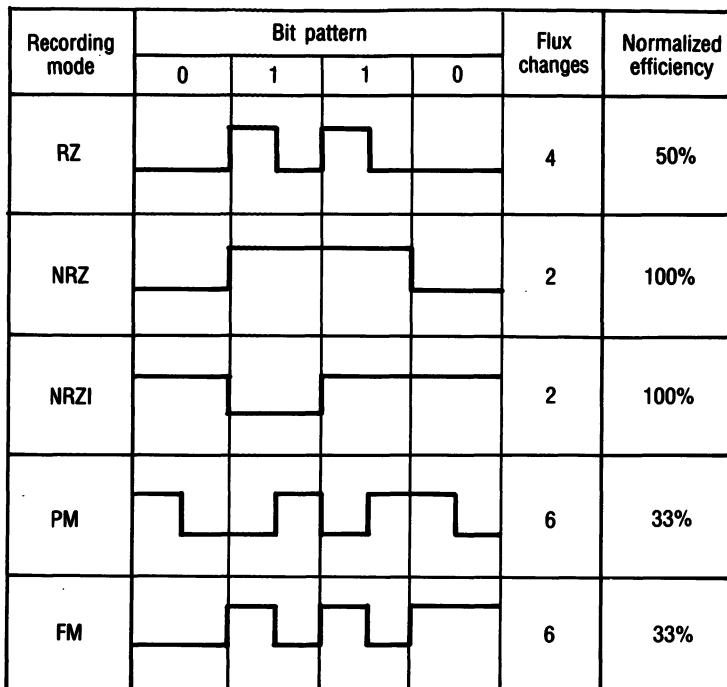


Figure S9 Problem 6.3.

Table S16 Problem 6.4.

<i>Label</i>	<i>Op-code</i>	<i>Operand</i>	<i>Comment</i>
BACK	LDM	(-65536)	;Fetch count constant to modifier
	LDA	WORD/M	;Fetch word to accumulator
	AND	CONS	;AND with mask constant
	SUB	VARS	;Subtract required tag-bits
	JMP0	FND	;Test accumulator, if zero jump to FND
	INCM		;Increment modifier register
FND	JMP	BACK	
	LDA	WORD/M	
	STOP		
CONS	()	;Mask constant
VARS	()	;Tag-bits

instruction and assuming that it has the same average execution time, the table-look-up program now becomes

LDM	(-65536)
LDA	WORD/M
TLU	CONS

LDA WORD/M
STOP

The number of memory locations required for the program has been reduced from 11 to 7 and the maximum table-look-up time to $1 + (1 \times 65536 \times 1) + 2 \approx 66$ ms. Thus the use of a special instruction has effectively reduced the search time by a factor of 6. However, there is a practical limit to the improvement of speed that can be achieved since it will depend on how well the microprogram can be optimized. Moreover in practice the search times could be improved by suitable structuring of the stored data and accessing the table by means of a generated address pointer. The most efficient technique for table look-up, however, is to use an associative memory (see Figure S10). The memory must be capable of matching to any part of the word: hence it must be possible to examine all bits. To operate the memory the desired pattern (tag bits) must first be set into the descriptor register and the match indicators of each word set to 1.

All words in the memory are searched simultaneously on a bit-by-bit basis. The first access (read cycle) is to bit 15 in all words, and the outputs are compared with bit 15 of the desired tag. The match indicators of those words in which bit 15 equals the tag are left set to 1, all mismatches causing the indicators to be set to 0. The final result is that only those words which contain bits identical with the specified descriptor have their match indicators set to 1. The address of the

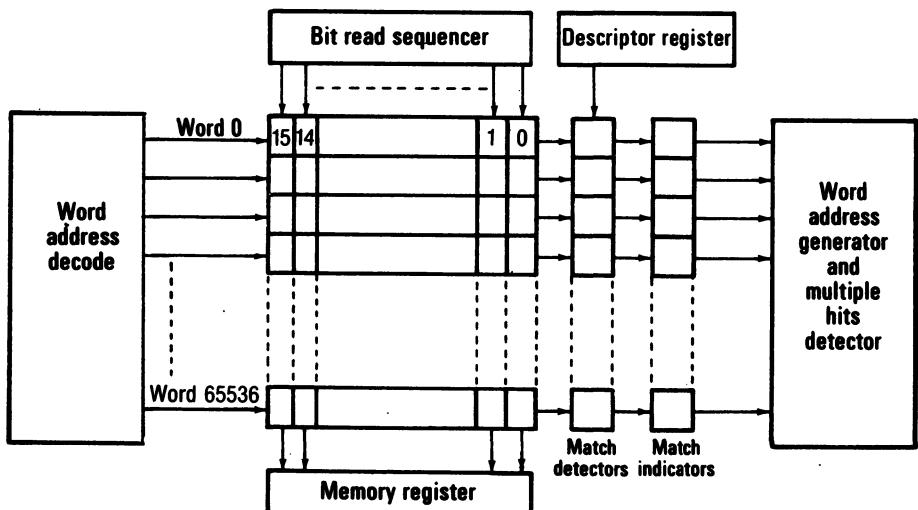


Figure S10 Problem 6.4.

word must now be obtained, and the required word read into the M-register in the usual way. Note that n read operations are required, where n is the number of bits in the word. If we assume a 500 ns access time the complete search takes 8 μ s. A special instruction is required of the form 'Search for word specified in address, contents to be placed in the accumulator'. After the basic search operation is completed, the microprogram must access the associative memory and transfer the required word to the accumulator. The complete table-look-up routine then consists of one instruction with an execution time of the order of 10 μ s.

- 6.5 (a) For direct mapping the main memory address is made up of the tag field and the index field, a total of 19 bits in this case ($2^{19} = 512K$). The index field is used to address the cache memory and requires 11 bits to do so, giving a tag field length of eight bits. The cache memory is partitioned into blocks so that the index field comprises a **block number** field together with a **word address** field to identify the required word in that block. The block size is 16 so that the word field is four bits. The total number of blocks available is, then, 128 ($=2^7$).

(b) The hit ratio is defined as:

$$\text{Hit ratio} = \text{number of hits}/\text{total number of accesses}$$

or in other words, for a hit ratio $|_r$, of 0.9, 90% of the total memory accesses are successfully read from the cache and 10% from the main memory, giving:

$$\begin{aligned}\text{Average access time} &= (0.1 \times 150 + 0.9 \times 30) \text{ ns} \\ &= 42 \text{ ns}\end{aligned}$$

(c) Because a write-through method is used, every write will take a main memory cycle time. Thus the overall hit ratio for read and write cycles is reduced and is given by:

$$\begin{aligned}\text{Hit ratio}|_{r+w} &= (0.9 \times 0.75) \\ &= 0.675\end{aligned}$$

CHAPTER 7

- 7.1 The DMA mode of operation may be implemented using the form of microprogram shown in Table S17. It is assumed that multiple DMAs are allowed (on a hardwired priority basis) with each peripheral device setting up a particular bit of a DMA request register (DMAR). At the conclusion of each computer instruction the DMAR register

Table S17 Problem 7.1.

<i>Next address</i>	<i>Micro-operation</i>	<i>Comment</i>
21	BR _C (DMAR = 0) END	;test for DMA request
22	DMAC _{ad} , R/W	;output of encoder to Address Bus
23	WAIT	
24	M _o C _i	
25	IO _o ,M _i	;write away data word
26	C _{ad} ,R/W	
27	C _o ,X _i , + llsb, + lwc	;add +1 to word count and address
28	C _i ,AU _o	
29	BR _C (A < 0) 25	;check word count, if zero set DMAF
30	DMAF,END	

must be examined for zero contents (using the condition DMAR = 0) to determine if a DMA request has been made. A dedicated memory location (DML) holds the word count and address of the location into which the first word is to be transferred of the device requiring DMA service. The address of this control word is obtained by encoding the contents of the DMAR, the outputs of this encoder being gated directly to the address bus using a new micro-operation DMAC_{ad}. The peripheral equipment must be program-initiated, and the contents of the DML control word primed (with the number of words, $-n$, and the start address of the data block) before the DMA mode of operation takes place. Both parts of the control word must be incremented by 1 in the microprogram for each word transfer. When the word count goes to zero a flag is set (DMAF) indicating to the external device that the data transfers have been completed.

- 7.3 To allow for the fact that two keys may be pressed inadvertently, a two-character buffer store should suffice. It can also be assumed that a character will remain on the parallel output lines of the VDU interface long enough for data transfer to take place. It is also assumed that the characters must be parity-checked and then packed, two to a word, in the main computer memory. The input message must be preceded and terminated by suitable control signals, say @ and *. The erase symbol is used to erase the character immediately preceding it.

Characters are input on an interrupt basis, that is each time a key is pressed an interrupt is sent to the CPU. The interrupt procedure may be assumed to be a standard one as described in the text and includes the usual supervisory program (see Figure 3.9). The input program must, of course, be initiated from the main program.

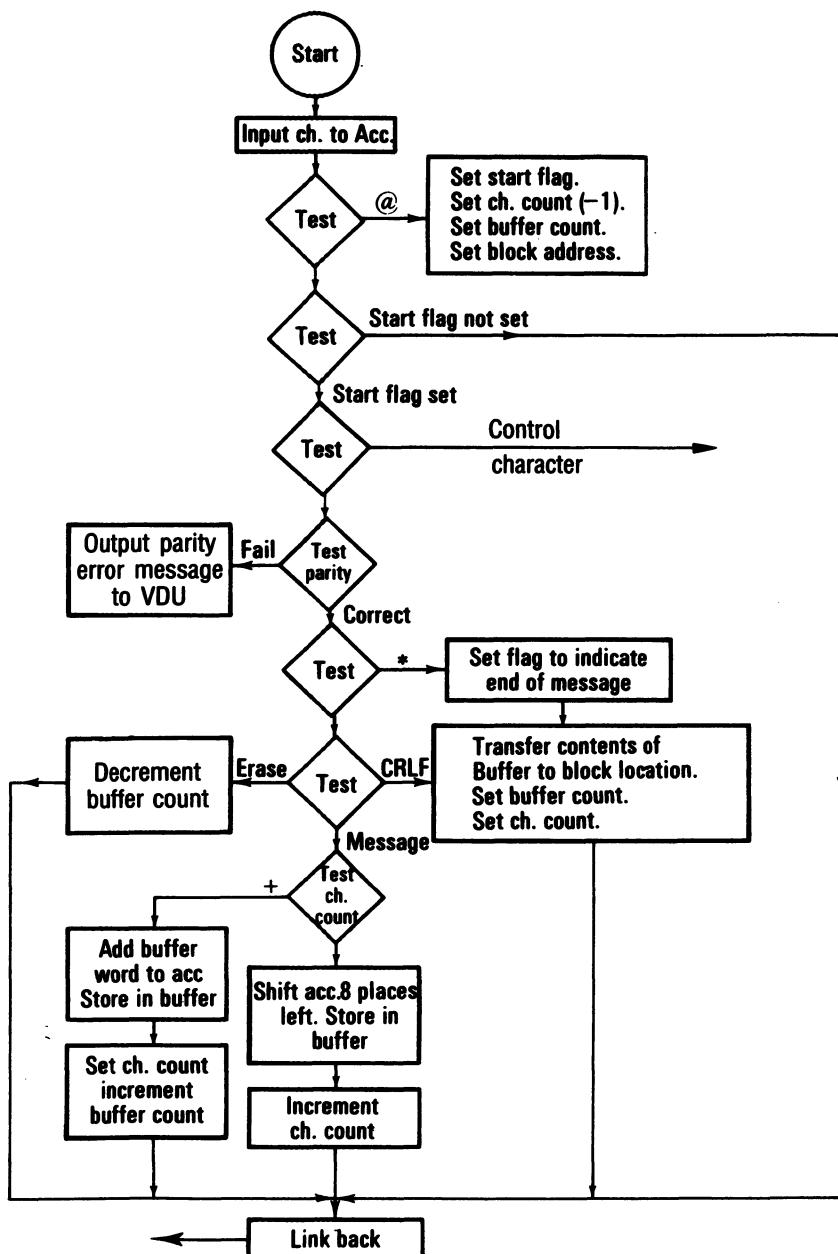


Figure S11 Problem 7.3.

A flowchart for the character input process is shown in Figure S11. Note that the character input is held in a buffer store until CRLF is received, when the entire line is stored away into the reserved memory locations (starting at the specified block address). If an erase symbol is received the buffer count is decremented by one allowing the character to be erased to be overwritten in the buffer store. The start of the message is indicated by the @ character which is used to set a start flag; this is to allow, for example, the message to always start on a new line. Control characters are tested for and if found cause a branch to a service routine to analyse and act accordingly.

The program presented in this example is considerably oversimplified, since multiple key depressions and the generation of specific control characters are ignored.

- 7.4 In order to transmit digital data over normal voice telephone links the binary output of the computer is modulated before transmission. At the receiving end of the line the signal must be correspondingly demodulated back into binary form; this is normally done using a modulator-demodulator unit known as a **modem**. Various methods of modulation are possible and data transmission can be either synchronous or asynchronous. Chapter 9 should be referred to for further details.

Asynchronous working is best for this problem and the 3.3 ms bit-rate allows each message bit to be registered into a single flip-flop stage and input a bit at a time on an interrupt basis. Using this technique it is possible to have up to 16 separate channels working into a single word buffer. Though this technique requires the minimum of external equipment, the software load (assembly and checking of the message) is considerable. Alternatively, the message bits may be clocked serially into a shift register and then transferred as a parallel word directly into the CPU. Thus if each character message is 12 bits long (including start/stop and parity checks) one complete word can be assembled every 39.6 ms. An interrupt signal to the CPU can be initiated by the stop signal with the data transfer occurring at the beginning of the next message start signal; alternatively, a timing gap or dummy message could be allowed between each actual message.

One of the major decisions that has to be made by the computer systems engineer is the allocation of tasks between hardware and software. In this case we are primarily concerned with the assembly and checking functions. The final decision depends on cost, speed, and availability of equipment. For example, one solution would be to use a computer dedicated to the assembly and decoding (including channel multiplexing) of all incoming messages. Alternatively, for

Table S18 Problem 7.4.

<i>Next address</i>	<i>Micro-operation</i>	<i>Comment</i>
21	M_c, A_o, X_i	;clear M-register, transfer accumulator to X-register
22	M_6, M_4, M_2, M_0	;set vector into M-register
23	I_m, A_i	;AND M-register with accumulator
24	$BRC(PC=0) \ 26$;even parity check on 8 LSBs of accumulator
25	K_1	
26	M_c, X_o, A_i	
27	M_6, M_4, M_2, M_1	
28	I_m, A_i	
29	$BRC(PC=0) \ 31$	
30	K_2	
31	M_c, X_o, A_i	
32	M_6, M_8, M_4, M_3	
33	I_m, A_i	
34	$BRC(PC=0) \ 36$	
35	K_3	
36	M_c, X_o, A_i	
37	A_o, X_i, R	;right shift to lose check bit
38	X_o, A_i, R	;lose second check bit
39	A_o, MD_i, R	;move LSB to Modifier
40	A_o, X_i, R	;lose third check bit
41	X_o, A_i	
42	MD_o, A_i, L	;restore LSB to form 4-bit message word
43	K_{ou}, MD_i, END	;error bits in modifier

common protocols off-the-shelf interface devices could be used and non-standard cases could be implemented in an ASIC.

Table S18 shows a microprogrammed routine to decode a 7-bit Hamming code held in the least significant end of the accumulator. Note the need to incorporate special micro-operations and logic to perform a parity check on the contents of the accumulator and to set individual bits of the M-register to 1. Moreover, provision must also be made to output the contents of the K-register to the common bus and to set up a parity condition.

CHAPTER 9

- 9.2 (a) **EtherNet:** The format has 26 bytes of control information, operates at 10 Mbps and waits for an acknowledgement of the receipt of data before the next transmission. If we assume the propagation delay between station A and station B to be t_s , then the interval between transmissions of new data is $2t_s$, provided the data is always

received correctly. The maximum message length is 1500 bytes, so that only two packets are required in this example. For packet 1 (60 bytes of data) the transmission time is

$$(86 \times 8)/10 \times 10^6 = 69 \mu\text{s}$$

For packet 2 (1500 bytes) the transmission time is

$$(1526 \times 8)/10 \times 10^6 = 1221 \mu\text{s}$$

Total transmission time is then $(69 + 1221 + 4t) = (1290 + 4t) \mu\text{s}$.

(b) **Cambridge ring:** Here the frame size is 38 bits of which 22 are control and 16 data. The ring normally operates at 10 Mbps and the packet has to circulate back to the source to indicate correct data transmission. Let this time be $2t$ so as to allow a fair comparison with the other network types. For packet 1, 30 frames will have to be transmitted and for packet 2, 750, giving a total of 780 frames. The transmission time per frame is $38/10 \times 10^6 = 3.8 \mu\text{s}$, and the total transmission time is $(780 \times 3.8 + 1560t) = (2964 + 1560t) \mu\text{s}$.

(c) **IBM Token ring:** The data field of this network is variable and so only two packets are required. Each packet has 21 control bytes associated with it and the ring typically operates at 4 Mbps. Ring-propagation delay is again assumed to be $2t$. For packet 1 the transmission time is

$$(81 \times 8)/4 \times 10^6 = 162 \mu\text{s}$$

For packet 2 the transmission time is

$$(1521 \times 8)/4 \times 10^6 = 3042 \mu\text{s}$$

The total time for transmission is, then, $(3204 + 4t) \mu\text{s}$.

- 9.3 The Cambridge ring operates on the insertion mode principle so that the buffer register of the transmitting station becomes part of the ring and lengthens it effectively by one station. At 10 Mbps a $50 \mu\text{s}$ propagation delay is equivalent to 500 bit-times, or $500/38 = 13$ packets. The maximum number of stations that can be connected at any one time is, then, 14.
- 9.4 (a) The HDLC bit-string has to be checked from left to right and a zero inserted after every five consecutive 1s. The start and stop fields are not part of the given bit-stream and are excluded from this algorithm.

Writing the given bit-stream in binary we have

F	B	F	F	4	E	F	2	8	0	0
1111	1011	1111	1111	0100	1110	1111	0010	1000	0000	0000

Inserting the bit-stuffing zeros,

111 1100 1111 1011 1110 0100 1110 1111 0010 1000 0000 0000

adding the start and stop fields,

01111110 111 1100 1111 1011 1110 0100
1110 1111 0010 1000 0000 0000 01111110

writing this as a Hex number, right justified gives the transmitted bit-stream, 3F7CFBE4EF28007E.

(b) In the given bit-stream, FBFF4EF2800:

FB is the address byte

FF is the control field

4EF is data and 2800 are the CRC bits

The CRC will have been correctly computed if there is zero remainder when the above bit-stream is divided modulo-2 (refer to worked solution of problem 2.6) by the constant 101011. The arithmetic is not shown here, but zero remainder does result.

(c) The control field bits are FF, or 11111111 and the two leading 1s identify the frame as an unnumbered frame.

CHAPTER 10

10.1 In the case of Figure 10.24 there are three input terminals and therefore eight different tests, designated t_0 – t_7 . The 16 faults f_1 – f_{16} are all the possible s-a-0 and s-a-1 faults occurring on the eight connections C1–C8 and are referred to in the usual way as C1/0, C1/1, etc. Thus the fault matrix consists of 8 rows and 17 columns including the correct output; this is shown in Table S19. Examining the derived G_D matrix, shown in Table S20, we see that the essential tests are t_3 (testing f_5 , that is, C3/0), t_1 (for f_8) and t_2 (for f_6 and f_{10}). These essential tests will cover all the faults except f_1 and f_{11} , which may be covered by including test t_7 . A possible test set (in fact the minimal) is thus t_1 , t_2 , t_3 , and t_7 , which in terms of input/output values becomes:

001/1, 010/1, 011/0, 111/1

10.3 The reliability of the RAM blocks is given by:

$$[1 - (1 - r_2)(1 - r_2)] = 1 - (1 - r_2)^2$$

The reliability of the 2-out-of-3 disk system may be calculated in the simplified way using a truth-table format:

Table S19 Problem 10.1.

	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
000	t_0	1	1	0	1	1	1	1	1	1	1	1	1	0	1	0	1
001	t_1	1	1	0	1	0	1	1	0	1	1	1	0	1	0	1	0
010	t_2	1	1	1	1	1	0	1	1	1	0	1	1	0	1	0	1
011	t_3	0	0	1	1	0	1	0	1	0	1	0	0	1	0	1	0
100	t_4	0	1	0	0	1	0	0	1	0	1	0	0	1	0	1	0
101	t_5	0	1	0	0	1	0	0	1	0	1	0	0	1	0	1	0
110	t_6	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	0
111	t_7	1	0	1	0	1	1	1	1	1	1	1	0	1	1	1	0

Table S20 Problem 10.1.

$$\begin{array}{ccc}
 D_1 & D_2 & D_3 \\
 0 & 1 & 1 & (1 - r_4)(r_4)(r_4) = r_4^2 - r_4^3 \\
 1 & 0 & 1 & (r_4)(1 - r_4)(r_4) = r_4^2 - r_4^3 \\
 1 & 1 & 0 & (r_4)(r_4)(1 - r_4) = r_4^2 - r_4^3 \\
 1 & 1 & 1 & (r_4)(r_4)(r_4) = r_4^3
 \end{array}$$

The overall reliability is the sum of the individual reliabilities:

$$R_D = 3r_4^2 - 2r_4^3$$

The system reliability is given by the expression:

$$\begin{aligned}
 R_s &= [1 - (1 - r_2)^2]r_1r_2[r_4^2(3 - 2r_4)]r_5 \\
 &= [1 - (1 - 0.92)^2]0.95 \times 0.85[(0.8)^2(3 - 2 \times 0.80)]0.90
 \end{aligned}$$

Thus

$$R_s = 0.61$$

The mean time between failures may be calculated using the expression:

$$R_s = e^{-\lambda t} = e^{-\lambda M} = e^{-t/M}$$

where λ = failure rate, M = mean time between failures, and t is the mission time.

Expressing $e^{-\lambda 1000}$ as a series expansion we have:

$$R_s = 1 - \lambda 1000 + \frac{(\lambda 1000)^2}{2} + \dots$$

The failure rate λ is very small so the expression approximates to:

$$R_s = 1 - \lambda 1000$$

$$\text{Now } M = \frac{1000}{1 - R_s} = 1/\lambda$$

$$\text{Thus } M = \frac{1000}{0.39} = 2564 \text{ hours}$$

Therefore the mean time between failures is 2564 hours.

Logic symbols

A1 INTRODUCTION

MIL-STD-806B (US Department of Defense) symbols (see section A4) are used extensively for describing logic circuits both in textbooks and papers and in the graphics of schematic capture in CAD packages. However, with the introduction of the ASIC and the general move towards customized LSI and with it the use of cell libraries, the traditional discrete component logic symbols are at too low a level to represent complex logic elements.

Internationally, Working Group 2 of the IEC Technical Committee TC-3 has consolidated all the work that has taken place on developments of logic symbols, and in the USA the IEEE has produced a standard containing all of the IEC work which is known as IEEE Std 91-1984.

This standard represents a new symbolic language which extends, rather than making obsolete, the use of the well-used MIL-STD-806B symbols, and the two can exist comfortably together as can be seen from their use in this and other texts.

It is not the intention to cover the whole of the IEEE standard in this appendix, but rather to introduce the principles associated with it, so as to encourage its wider acceptance by design engineers.

A2 SYMBOL STRUCTURE

A symbol for an element is made up from a combination of outlines, together with one or more qualifying symbols (see Figure A1 and Table A1). The function of these qualifying symbols is to define what logical operations are taking place at the inputs and outputs. The outlines of elements may be abutted or embedded. When a circuit has one or more inputs that are common, or where control is to be exercised over all

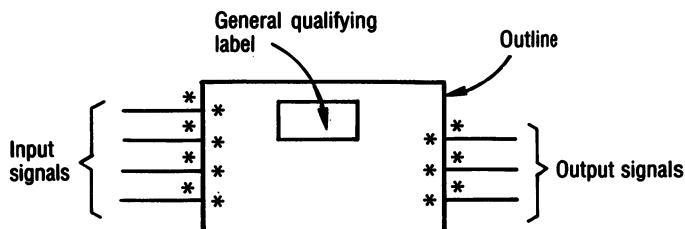


Figure A1 Symbol representation: * denotes possible position for qualifying symbol.

Table A1 General qualifying symbols (subset).

Symbol	Description
&	AND gate or function
>1	OR gate function, i.e. at least one active input is needed to activate the output
=1	Exclusive OR, i.e. one and only one input must be active to activate the output
=	Logic identity. All inputs must be at the same state
2k	An even number of inputs must be active
2k + 1	An odd number of inputs must be active
1	The one input must be active
▷	A buffer or element with more than the usual output capacity
■■	Schmitt trigger or element with hysteresis
X/Y	Coder, code converter, e.g. DEC/BIN, BIN/HEX
MUX	Multiplexer/data selector
DMUX or DX	Demultiplexer
Σ	Adder
P-Q	Subtractor
CPG	Look-ahead carry generator
π	Multiplier
COMP	Magnitude comparator
ALU	Arithmetic logic unit
I = 0	Element powers up cleared to the 0 state
I = 1	Element powers up set to the 1 state
Φ	Highly complex function; 'black box' symbol with limited detail shown under special rules

of the inputs, a common control block is used of the shape shown in Figure A2.

Symbols appear both outside and inside the outline. Table A2 shows the input and output qualifying symbols which are *outside* the outline, while Table A3 shows a small subset of those which can be used *inside* the outline.

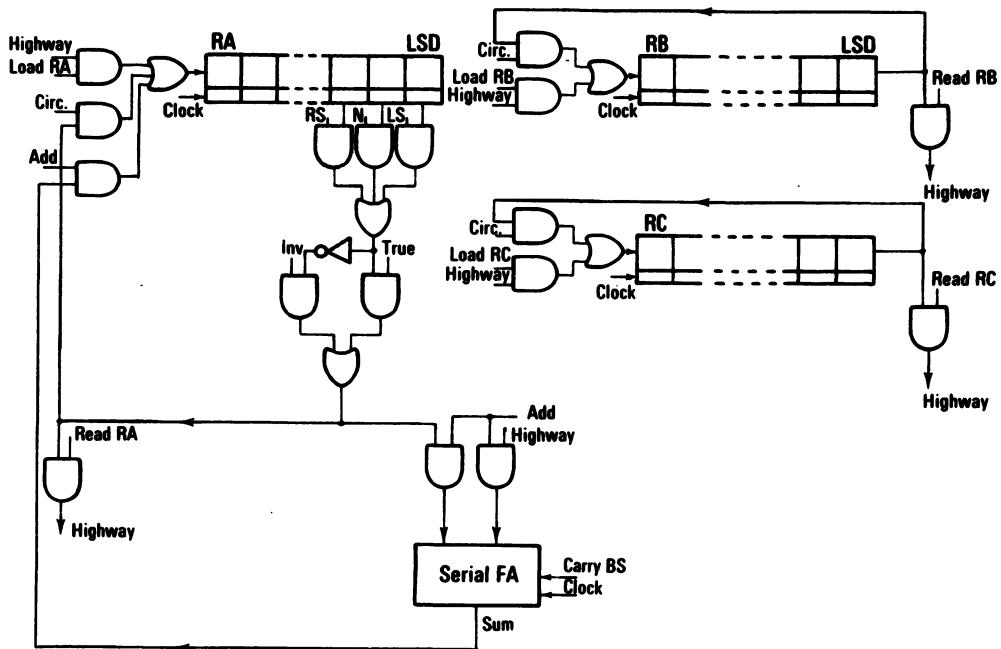
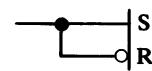


Figure A2 Common control block.

Table A2 Qualifying symbols for inputs and outputs (subset).

<i>Symbol</i>	<i>Description</i>
—○—	Logic negation at input. External 0 produces internal 1
—○—	Logic negation at output. Internal 1 produces external 0
—■—	Active low input. Equivalent to in positive logic
—■—	Active low output. Equivalent to in positive logic
—○— } —○— }	Dynamic inputs {
—○— } —○— }	active on {
—○— } —○— }	indicated transition
—○— }	Input for analogue signals (on an analogue symbol)
—○— }	Input for digital signals (on an analogue symbol)

Table A3 Symbols inside the outline (subset).

<i>Symbol</i>	<i>Description</i>
$\nabla \overline{ }$	Three-state output
$\triangleright \overline{ }$	Output with more than usual output capability (symbol in direction of signal flow)
EN	Enable input
J,K,R,S	Usual meanings associated with flip-flops: e.g. R = reset to 0, S = set to 1
$\neg \overline{ }$	Postponed output (of a pulse-triggered flip-flop). The output changes when the input initiating the change returns to its initial state
$\rightarrow T$	Toggle input causes internal state of output to change to its complement
$\rightarrow D$	Data input to a storage element equivalent to:
	 A logic symbol for a flip-flop. It consists of a rectangle with two inputs on the left labeled 'S' and 'R'. The 'S' input has a small circle at its top end, indicating it is an active-low input.
$CT=9 \overline{ }$	The output is active if the content of the register is as indicated
$\rightarrow CT=15$	The contents-setting input, when active, causes the register to be set to that value

A3 DEPENDENCY

This is a feature of the IEEE standard which raises the level of complexity that can be described by a logic element symbol. It permits the relationship between inputs or outputs, or inputs and outputs, to be logically defined without showing all of the elements and interconnections involved. Eleven types of dependency have been defined (see Table A4), but for the purposes of use in the diagrams of this book only the principles involved within a subset of these will be described, the reader being referred to the full IEEE Standard for a complete description.

The general labelling rules for applying dependency are as follows.

1. Label the input (or output) affecting other inputs or outputs with the standard letter symbol (as in Table A4) indicating the relationship involved, followed by an identifying number.
2. Label each input (or output) affected by the standard letter symbol input (or output) with the same identifying number.
3. If it is the complement of the affecting input that is to be used with another input then a bar is placed over *that* input and *not* the affecting input.
4. If two affecting inputs (outputs) have the same letter and the same identifying number, they form an OR relationship with each other.

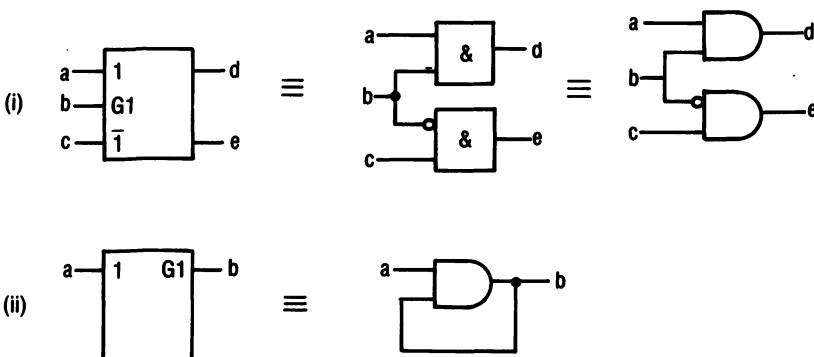
Table A4 Dependency types.

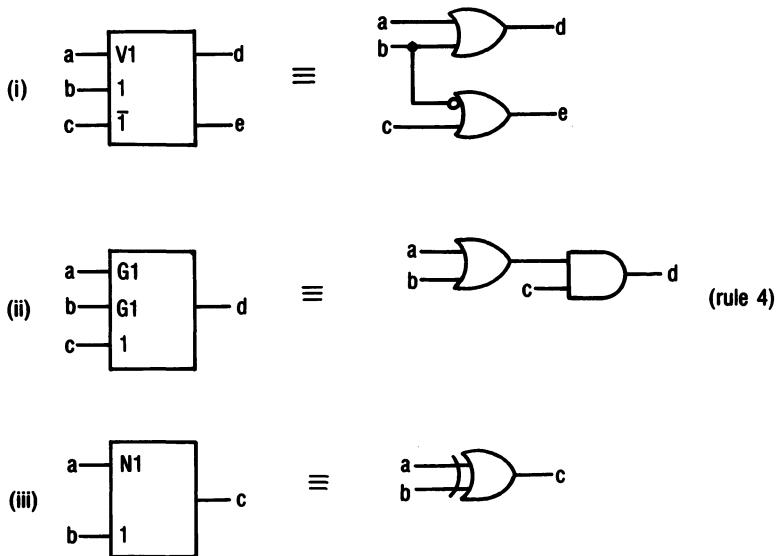
<i>Dependency type</i>	<i>Qualifying symbol</i>
Address	A
AND	G
Control	C
Enable	EN
Negate (Exclusive-OR)	N
Interconnection	Z
Mode	M
OR	V
Reset	R
Set	S
Transmission	X

5. If the affected input (output) requires a label to denote its function, i.e. G, this label is *prefixed* by the identifying number of the affecting input. If there is more than one affecting input the identifying numbers of these are prefixed in the same way, and separated by commas.
6. For multiple affecting inputs in (5) above the order of logical implementation is taken as being from left to right.
7. In cases where ambiguity may arise through the use of numbers, another character may be used in their place, i.e. where the labels denoting the functions associated with an affected input are themselves numbers, such as the outputs of a coder.

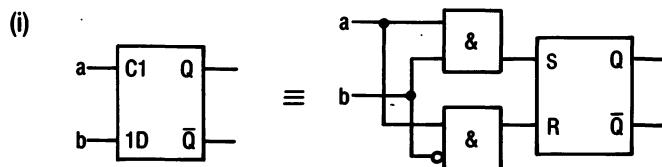
Some examples are given in sections A3.1 to A3.6.

A3.1 AND (G)

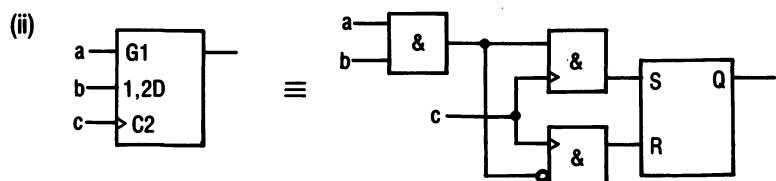


A3.2 OR (V) and Exclusive-OR (N)**A3.3 Control (C)**

Control inputs are used to enable (disable) the data inputs of storage elements. Dynamic inputs are shown with the appropriate symbol.



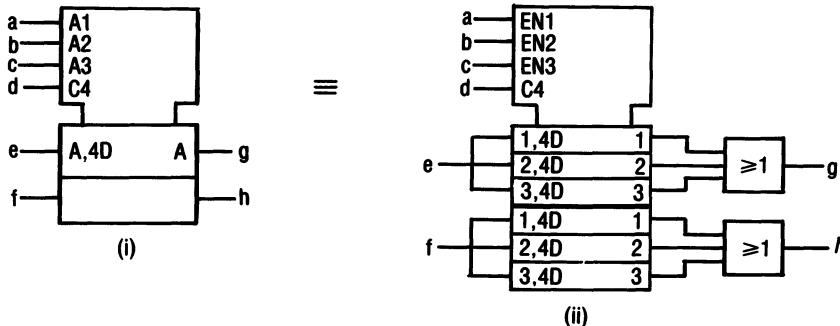
The D-type flip-flop is shown as an SR type driven from a single input and its complement.



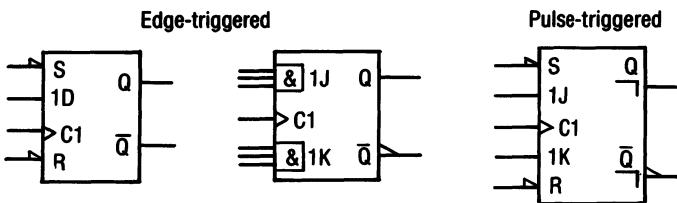
The dynamic input c affects the input b after it has been ANDed with input a, control being exercised as an AND function.

A3.4 Address (A)

This provides a symbolic representation of those elements, such as memory, which use address control inputs to select specified sections of an array. The output of the array is the OR function of corresponding elements of selected sections. As an example; a 3-word by 2-bit memory is shown below in (i) with an expansion of the functionality in (ii). From the latter it can be seen that A1 selects both bits of word 1, and if selected the values of the inputs on e and f can then be clocked in under control of input d, which is a control input. Similarly A2 and A3 select words 2 and 3 respectively. (i) is really a shorthand notation of (ii), and although it does not display the array structure of the memory, is concise and correct in its description of the input and control functions.



A3.5 Bistable elements



(\neg see Table A3)

A3.6 Coders

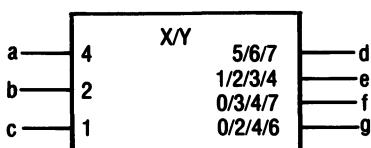
The general symbol for a code or coder is shown as the general qualifying symbol, for example Y/Z, X/Hex, etc., and gives some indication of the code and the relationship between inputs and outputs. The internal logic relationship is further indicated by labelling the inputs and outputs as follows. Either

1. label the inputs with numbers such that the internal value equals the sum of the weights associated with those inputs in their 1-state; or
 2. replace Y by an appropriate indication of the input code and label the inputs with characters which refer to this code.

The output relationships with the internal logic value are indicated by either

1. labelling each output with a list of numbers (using '/' as a separator) representing those internal values that lead to the 1-state of the output; or
2. replacing Z by an appropriate indication of the output code and label the outputs with characters which refer to this code.

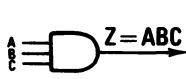
Alternatively the general symbol for a coder may be used together with a truth table showing the input/output relationship. This is the recommended way of showing a programmed PROM.



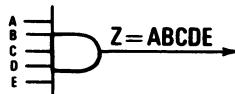
Inputs			Outputs			
a	b	c	d	e	f	g
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

A4 MIL-STD-806B LOGIC SYMBOLS

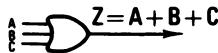
A4. 1 AND gates



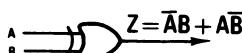
A4. 2 NAND gates



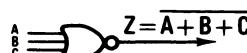
A4. 3 OR gates



A4. 4 Exclusive OR gates



A4. 5 NOR gates



A4. 6 Inverter amplifiers



Glossary of abbreviations

AD	address strobe
ADB	ASP data buffer
ADCCP	Advanced Data Communications Control Procedure
ALU	arithmetic and logic unit
APE	associative processing element
ASCII	American Standard Code for Information Interchange
ASIC	application-specific integrated circuit
ASM	algorithmic state machine
ASP	associative string processor
ATE	automatic test equipment; automated test environment
ATM	asynchronous transfer mode
BCD	binary-coded decimal
BER	bit error rate
BiCMOS	Bipolar CMOS
BIST	built-in self-test
BSC	Binary Synchronous Communications
CA	collision avoidance
CAD	computer-aided design
CAFS	content-addressable file system
CASE	computer-aided software evaluation; computer-aided system evaluation
CCD	charge-coupled device
CCITT	Consultative Committee for International Telegraph and Telephone
CD	collision detection
CISC	complex instruction set computer
CML	current-mode logic
CMOS	complementary metal oxide semiconductor
CPU	central processor unit
CRC	cyclic redundancy check

CSMA/CD	carrier sense multiple access-collision detection
DAP	distributed array processor
DART	dual asynchronous receiver transmitter
DD	data dictionary
DDCMP	Digital Data Communication Message Protocol
DFD	data flow diagram
DIL	dual-in-line
DMA	direct memory access
DML	dedicated memory location
DMR	double mode redundancy
DS	data strobe
DSP	digital signal processor
DTL	diode-transistor logic
DXF	Drawing Interchange Format
E ² ROM	electrically erasable programmable read-only memory
ECL	emitter-coupled logic
EEPROM	electrically erasable programmable read-only memory
EMPL	Extensible MicroProgramming Language
EPLD	electrically programmable logic device
EPROM	erasable programmable read-only memory
ERD	entity relationship diagram
FET	field-effect transistor
FIR	finite impulse response
FLOPS	floating-point operations per second
FM	frequency modulation
GCR	group code recording
HDL	hardware description language
HDLC	high-level data link control
I ² L	integrated injection logic
IPSS	International Packet Switched Service
ISDN	Integrated Services Digital Network
ISP	Instruction Set Processor
ISPS	Instruction Set Processor Specification
JLCC	J-leaded chip carrier
LAN	local area network
LAPB	link access procedure-balanced
LCC	leadless chip carrier
LED	light-emitting diode
LFSR	linear feedback shift register
LLC	logical link control
LRU	least recently used
LSB	least significant bit
LSI	large-scale integration
MAC	medium access control

MAL	minimal average latency
MAN	metropolitan area network
MFM	modified frequency modulation
MIMD	multiple instruction multiple data
MIPS	millions of instructions per second
MISD	multiple instruction single data
MOPS	millions of operations per second
MOS	metal oxide semiconductor
MSB	most significant bit
MSI	medium-scale integration
MTBF	mean time between failures
MTRF	mean time to repair a fault
MTTF	mean time to failure
NRZ	non-return to zero
NRZ1	non-return to zero one
OSI	Open Systems Interconnection
PAL	programmable array logic
PC	personal computer
PCB	printed circuit board
PCM	pulse code modulation
PDX	primary data exchanger
PE	processing element
PGA	pin grid array
PLA	programmable logic array
PM	phase modulation
PROM	programmable read-only memory
PSS	Packet Switchstream Service
RAM	random-access memory
RISC	reduced instruction set computer
RISP	reduced intrusion scan path
RLL	run length limited
ROM	read-only memory
RTL	resistor-transistor logic
RZ	return to zero
SDLC	synchronous data link control
SIMD	single instruction multiple data
SIMPL	Simple Identity MicroProgramming Language
SISD	single instruction single data
SP	stack pointer
STD	state transition diagram
TMR	triple mode redundancy
TTL	transistor-transistor logic
UART	universal asynchronous receiver transmitter
VDM	Vienna Development Model

VDU	visual display unit
VGA	video graphics array
VHDL	VHSIC hardware description language
VHSIC	very high-level silicon integrated circuit
VLSI	very large-scale integration
WF	weighting factor
WORM	write once read many
ZM	zero modulation

Index

- Acceptance tests 399
- Access time 196, 255
- Accumulator 9, 11, 132
- Ada language 67, 343
- ADC 331
- ADCCP 357
- Addend 132
- Adder
 - 4-bit full adder 384
 - autoasynchronous 139
 - auxiliary carry look-ahead 145
 - binary-coded decimal 151–2
 - carry equations 144
 - carry generate 142
 - carry look-ahead 142–8
 - carry propagate 142
 - carry-bypass 145–6
 - carry-completion 139–42
 - carry-save 148–9
 - cascaded serial 134–5
 - conditional 145
 - full 127–8
 - half 127
 - LSI ALU chip 145–7
 - multiple input binary 134–5
 - parity checked 189
 - self-timing 139
 - serial 131–7
 - synchronous state table 137
 - synchronous three-input 135–6
- Addition
 - out-of-range 150–1
 - overflow and underflow 149–50
- Address 7, 251
- Address bus 55, 72, 265
- Address field 239
- Address highway (bus) 13
- Addressing modes 38–43
 - auto-increment/decrement 42
 - direct 40
 - immediate (zero-order) 39
 - indexed 41
 - indirect 40
 - relative 43
- ALGOL language 67
- Alphanumeric codes 19
- ALU 78
- ALU chip (eight function) 116–18
- AMD 2900 co-processor 185
- AMD2900 bit-slice family
 - 4-bit processor 114–19
 - AM2910 sequence controller 114–19
 - double-length algorithms 118
 - multiplication, division 118
- Analogue-to-digital converter 253
- Arbitration schemes 321–5
 - asynchronous requests 324–5
 - daisy chaining 321–2
 - polling 322–4
- Archival data storage 213
- Arithmetic error procedure 186
- Arithmetic logic error detection
 - complete error-detection 191
 - division 193
 - error LEDs 185–6
 - error-detecting codes 186
 - interrupt 186
 - multiplication 193
 - parity codes 186
 - parity-checked adder 189
 - residue codes 186
 - residue-checked adder 191
 - trap instruction 186
- Arithmetic logic unit (ALU) 125

- Arithmetic operations**
 addition 126–48
 division 170–3
 fixed-point 125
 floating-point 125
 multiplication 153–69
 normalization 184
 residue 187
 scaling 25–7
 software implementation 185
 subtraction 126–34
- Arithmetic unit** 3, 11, 116–18, 194
- Array processors** 302–10
 DAP 303
 geometric control 304
 illiac IV 303
 mode control 304
 network control unit 303
 processing elements 302
 solomon 303
 vectram 303
- Arrays** 69
ASCII 257, 357, 383
- Assembler** 64
- Assembly language** 63, 77
- Associative mapping** 238–9
 address field 239
 data field 239
- Associative memory** 200, 236–7
- Associative processors** 289, 338–42
 bit-serial 340
 block-orientated 340
 fully parallel 340
 word-serial 340
- Associative string processor (ASP)** 342
- Asynchronous requests** 324–5
- Augend** 132
- Automatic test equipment (ATE)** 403
- Average carry length** 141
- Babbage, Charles** 15
- Backing memory** 200
- Backplane standards** 282–3
- Bandwidth** 221
- Base pointer** 43
- Base register** 245
- Baseband signalling** 350, 353
- Baud rate** 277
- Below range number** 184
- BER** 349
- Bias field** 232
- BiCMOS** 382
- Binary addition, rules** 126
- Binary arithmetic** 25–30
- Binary codes** 17–20
- Binary division** 170–3
- Binary multiplication** 153–69
- Binary number, fractional** 26
- Binary subtraction, rules** 126
- Binary system** 2
- Binary-coded decimal (BCD)** 18
- Binary-coded decimal adder** 151–2
- Binary-to-decimal conversion** 31
- Bipolar CMOS (BiCMOS)** 382
- Bipolar method** 219
- Bipolar ROM** 212
- Bisync (BSC)** 357
- Bit error rate (BER)** 349
- Bit stuffing** 358, 374
- Bit-organized** 201
- Bit-serial** 340
- Bit-slice processors** 114–19
- Block** 224
- Block-orientated** 340
- Body** 420
- Boolean difference** 400
- Boolean semaphores** 344
- Booth's algorithm** 175–7
- Branch and test** 332
- Bridge, network** 351, 371
- Broadband signalling** 350, 353
- Buffer register** 87
- Bus protocols** 55
- Byte** 6
- Byte-count protocol** 357
- C language** 67
- Cache memory** 74, 200, 237–42
 associative mapping 238–9
 data valid bit 242
 direct mapping 239–41
 efficiency or hit ratio 238, 251
 first-in-first-out 242
 least recently used 242
 locality of reference 237
 replacement algorithms 241–2
 set-associative mapping 241
 write back 238
 write through 238
- CAD techniques** 399
- CAD tools** 388, 419
- CAD VDU** 252
- Cambridge ring** 365–6
 phase modulation 366

- register insertion 364–6, 374
repeater station 366
Capacitive noise 389
Capstan 226
Carrier, network 364
Carry equations 144
Carry errors 189
Carry generate 130, 142, 190
Carry look-ahead 142–8
 fan-in factor 144
 group levels 145
Carry propagate 130, 142, 190
Carry propagation 138–9
Carry-bypass 88, 145
Carry-save adder 148–9
Cartridge unit 229
CASE tools 377
Cathode-ray tube display 262, 283
CCD channel 216
 CCD channel 216
 charge detection 217
 charge injection circuit 217
 charge regeneration 217
 data buffers 217
 frequency of operation 217
 linear array 216
 MOS capacitor 216
 n-channel MOS 213
 phased voltage (clocks) 216
 potential well 216
 read/write circuits 217
 serial shift register 213
 storage time 217
 transfer efficiency 217
CCITT 357
Central processor unit (CPU) 71
CHAMIL language 121
Channels 53, 316
Characteristic impedance 390
Charge detection/regeneration 217
Charge injection circuit 217
Charge-coupled device (CCD) 213–18
Chevron pattern 232
Chip-enable line CE 204
Chrominance signals 262
Circuit switching 352
Clock 11, 81
CMOS 82, 382
Codes
 9's complement 19
 ASCII 19
 BCD 18–19
cyclic 19
error correcting 23–5
error detection 21–3
gray 19
hamming 23
minimum distance three 23
minimum distance two 23
weighted 18–20
XS3 19
Collision 296
Collision avoidance (CA) 361
Collision detection (CD) 361
Collision matrix 300
Combinatorial counter 160
Combined adder-subtractor 130
Comment statements 64
Common highway (bus) 11
Communication channel 252
Communication process 350
Communication protocol 267
Communications interfaces 277–80
 baud rate 277
 DART 277
 double buffering 279
 multiplexed 280
 UART 277
Comparator 96–8
Comparator, iterative 182
Complementary MOS (CMOS) 382
Component redundancy 413
Computer
 accumulator based 34
 architecture 3
 commissioning 377–8
 complex instruction set (CISC) 33
 digital 2
 hybrid 2
 memory 3
 production 377
 register based 34
 analogue 1
 integrations 2
 performance 33
 reduced instruction set (RISC) 33
 simple program 10
Computer applications 13–15
Computer architecture,
 hypothetical 7–13
Computer networks
 local area (LANS) 350
 metropolitan area (MANS) 350
 wide area (WANS) 350

- Computer operation 11–13
Computer word 5, 11
Computer-aided design (CAD) 252
Computer-aided system evaluation 419
Concurrent Pascal language 292, 343
Condition code register 115
Conditional inputs, branch 103
Conditional logic 103
Contact bounce 260
Content-addressable memory 200,
 236–7
Control bit 7, 50
Control bus 72
Control register 11
Control requirements 79
Control strategy 78, 124
Control unit 3, 11
 microprogrammed 12
 timed 11
Control, global 291
Control, local 291
Cost per bit 196
Counter, ASIC 384
Counter circuits
 3-bit synchronous up/down 94
 asynchronous ripple through 87
 chain-code 89–91
 gray-code 89
 master-slave 89
 ring 93–4
 synchronous 87
Coupling 314
Co-processor 17, 184, 289, 378
Crossbar switch 318
Cross-assembler 66
CSMA/CD 361–2
Curie point 235
Current mode logic (CML) 382
Custom design 385
Cycle stealing 277
Cyclic code checker 405
Cyclic memory 197
Cyclic Redundancy checks (CRC) 21,
 31
Cylinder 225

Daisy-chain method 274, 321–2
Damping network 393
DAP 303
DART 277
Data bus 55, 72, 265
Data dictionary 424

Data field 239
Data flow diagram 424
Data link layer protocols
 ADCCP 357
 bisync (BSC) 357
 DDCMP 357
 HDLC 357–8
 LAPB 357
 LAPD 357
 SDLC 357
Data logging 14
Data memory 327
Data representation 4–6
Data transfer checks 397
Data transmission 255
Data transmission switching methods
 circuit switching 352
 message switching 352
 packet switching 352
Data types
 2's complement 17
 alphanumeric 17
 binary coded decimals 17
 external 17
 floating-point 17
 integers 17
 internal 16
 multiple-byte 17
 numeric 17
 single precision 17
 variables 16
Data valid bit 242
Datagram 352
Data-flow graph 310
Data-flow processors 310–2
 data-flow graph 310
 dynamic task allocation 311
 functional blocks 310
 granularity, coarse, fine 310
 Manchester computer 312
 petri net 310
 SISAL language 312
 SPNLL language 312
 static task allocation 311
 TASS language 312
 tokens 310
DC-to-DC converter 391
DDCMP 357
Decoders
 1-out-of-M 94–5
 3-bit serial 96
 3-to-8 bit MSI 95–6, 384

- PAL, ROM 96
Dedicated bus 318
Delay modulation 222
Delayed branching 336
Descriptor base register 248
Design entity 420
Desktop publishing 252
Diagnostic test programs 396
Diagnostic testing 403
Differential Manchester encoding 367
Digital plotter 254
Digital signal processors (DSPs) 33, 289, 326–34
 applications 326
 Harvard architecture 327
 TMS 320C25 327–34
Digital-to-analogue converter 254
Digitizer 253
Diode matrix 212
Diode-transistor logic (DTL) 380
Dipulse 219
Direct digital I/O 253–4
Direct mapping 239–41
 address field 239
 index field 239
 read operation 239
 Tag field 239
Direct memory access 56, 256, 275–7, 28
Directives 63
Disk characteristics 254
Disk controller 225, 378
Disk memory 224–6
 block 224
 cylinder 225
 disk controller 225
 DMA transfer 225
 floppy 225
 flying head 225
 hard sectored 225
 latency 225
 magnetic 224
 magneto-optical 224
 optical 224
 recording, RLL, MFM 225
 sector 224
 single/multiple disk 225
 soft sectored 225
 track 224
 Winchester 225
Display file 284
Distance 23
Distributed computer systems 313
Dividers, Repeated subtraction 170–1
 restoring 171
Division algorithms, unsigned
 iterative 172–3
 non-restoring 172
 repeated subtraction 170–1
 restoring 171–2
Division, signed 178
Divisor 171
DMA 256, 275–7, 286
DMA transfer 225
Double buffering 279
DRAM 200
 access times 210
 charge storage 208
 multiplexed address lines 209
 read operation 209
 refresh 208
 write cycle 210–11
Drawing interchange format (DXF) 283
Drum plotter 283
DSP 121, 326–34
DSP architecture 74, 326–34
DTL 380
Dual-in-line (DIL) package 380
Duplicate carry 190
DXF 283
Dynamic number range 18
Dynamic processor allocation 313
Dynamic RAM (DRAM) 200, 208–10
Dynamic relocation 60, 244
Dynamic storage 197
Dynamic task allocation 311
D-algorithm 403
Earth loops 392
Earth plane 393
ECL 382
Efficiency or hit ratio 238
Efficiency, magnetic recording 221
Electrically Erasable PROM (EEPROM) 212–13
Electromagnetic coupling 390
Electromagnetic radiation 394
Emitter-coupled logic (ECL) 382
EMPL microprogramming language 121
Emulation 80, 419, 421–2
Encoder, binary 96
Entity relationship diagram 424

- EPLD 383
EPROM erasure 213
Equality 97
Erasable programmable ROM (EPROM) 212–13
Error-detecting codes 186
Essential tests 401
EtherNet 361–3
 file transport protocol 363, 374
 Manchester encoding 363
 non-rooted tree 361–2
 VLSI IEEE 802.3 standard 363
Even parity 189
Execute cycle 13
Exponential failure law 409
Exponents 28, 180
Extended FORTRAN language 343

Failure rate 408
Failure, network 364
Fan-out 403
Fault dictionary method 405
Fault matrix 400–2, 429
Fault testing 395–9
Fault-tolerant design 418
Field declarations 70
Field effect transistor (FET) 380
Field-programmable array logic 383
File gap 228
File transport protocol 363
Files 228
Firmware 80
First generation languages 343
First-in-first-out 242
Fixed-point numbers 26–7
Flash memory 201, 212–13
Flat pack 380
Flip-flop memory cell 205–8
Flip-flop, ASIC 384
Floating control mode 313
Floating gate 213
Floating-point arithmetic 180–5
 addition/subtraction 180–5
 co-processor 184
 DSPs 185
 hardware 181–3
 multiplication/division 184
 non-normalized form 184
 normalized form 184
 post-normalization 184
 small-magnitude operands 184
 software 185
Floating-point hardware 285, 338
Floating-point numbers 17, 27–30
 double-precision 29
 exponent 28
 IEEE standard P754 29, 31
 mantissa 28
 normalized form 28
 single-precision 29
Floating-point software routines 185
Floppy disk 225
FLOPS 289
Flying head 225
Formats, instruction 36–7
 register source/destination 36
 single address 36
 single operand 36
 three-address 36
 two-address 36
 zero-address 36
FORTRAN language 67, 292
Four-bit processor slice 384
Fowler-Nordheim tunnelling 213
Fractional number representation 18, 178
Frame 352
Frequency modulation (FM) 221
Full custom design 383
Full adder, parallel 11
Full adder/subtractor
 circuit 127
 switching equations 127–8
 truth table 127
Functional blocks 310
Functional languages 343–5
 HOPE 343
 ID 343
 Lambda calculus 344–5
 LISP 343
 SISAL 343
 VAL 343
Futurebus 72

Gallium arsenide 383
Gap length, recording head 218
Gas discharge display 283
Gate array 383
Gateway, network 351, 371
GCR 5-bit code 222
Generalized interface unit 281–3
Geometric control 304
Global memory 328
Glue logic 119, 383

- Graceful degradation 313
Granularity 310, 406
Graphics systems 283–5
Greedy control 300
Greedy cycles 300
Ground returns 392
Group code recording (GCR) 222
Group encoding, micro-operations 108

Half-adder 126–7
Half-subtractor 127
Hamming codes 23–4
Hammock network 413
Handshaking 268
Hard sectored disk 225
Hardware associative store 247
Hardware description language (HDL) 419
Hardware design 4
Hardware faults 186
Hardware interface 255
Hardware redundancy 414
Hardwired control 337
Harvard architecture 74, 200, 327
HDLC 357–8, 374
Helical scan 230
Helically scanned magnetic tape 201
Hexadecimal 5, 6
Hexadecimal keyboard 257–8
Hexadecimal-to-binary conversion 31
Hierarchical control 80
High frequency radiation 394
High level languages 16, 67–8
Hit, cache 238
HOPE language 343
Horizontal format 107
Hybrid circuits 379
Hypothetical machine, processor state 70

IBM token ring 365, 374
differential Manchester encoding 367
IEEE 802.2 standard 365
packet-switching protocol 367
phase-locked loop 368
star configuration 367
ID language 343
Ideal network 349
Identifiers 69
IEEE 488 bus 72
IEEE 796 283

IEEE 796 bus 72
IEEE 802 LAN model 358
ILLIAC IV 305–10
arithmetic facilities 307
array control unit 306
processing element (PE) 308
quadrant 305
Image scanner 253
Inclusive OR 195
Incremental plotter 283
Index field 239
Indirect address bit 6–7, 107
Indirect addressing 62
Information redundancy 413
INMOS T425 314
Inner computer 80
Input devices 253
Input/output
instructions 53–5
memory-mapped 53–5
Input/output unit 4
Instruction cycle 69
Instruction execution
cache memory 74
DSP architecture 74
fetch 13, 104–5
fetch and execute cycles 74
Harvard architecture 74
pipelining 74
Instruction formats 36–7
register source/destination 36
single address 36
single operand 36
three-address 36
two-address 36
zero-address 36
Instruction representation 4–7
Instruction set 32–3
hypothetical machine 8
Instruction & data caches 336
Instructions 4
ADD with carry 185
arithmetic 52
data-handling 51–3
end-around 51
fetch, store, add, subtract 43–4
format 36–7
input and output 53–6
input/output 7, 53–5
IOS 7, 9
jump 44
logical, AND, OR, XOR 51–2

- machine-code 7
 masking 271
 mnemonic form 10
 modification 46–51
 modified 13
 multiply, divide 44
 operate 315
 overlap 294
 prefix 315
 PUSH, POP 58
 skip 44–5, 50, 269
 SUB with borrow 185
 trap 186
 ZA 7, 9
 Instruction-based I/O 264
 Integrated circuit 14
 Integrated injection logic (IIL) 382
 Integrated services digital network (ISDN) 351
 Intel 8086
 co-processor 185
 memory addressing 38
 Intensity signals 262
 Interactive processes 316
 Interface unit, generalized 281–3
 Interfaces 14, 286, 420
 Interfaces, communications 277–80
 Interlocks 296
 Internal hardware 329
 Interrupt mechanisms 271–5
 Interrupts 56, 107, 186, 256
 daisy-chain method 274
 interrupt acknowledge 274
 interrupt disable 271
 interrupt enable 271
 interrupt line 267, 271
 interrupt request 272
 masking 271
 multiple interrupts 273–4
 nesting interrupts 274
 polling method 272
 real-time systems 275
 RISC 337
 skip chain method 272–3
 use of stacks 59
 vectored interrupt 274
 ISDN 351
 Iterative circuits 134
 Iterative division 172–3
 Iterative operations 2
 I/O control 252
 I/O processors 280–3
 J-leaded chip carrier (JLCC) 380
 Kerr effect 235
 Keyboard
 ASCII 253, 256–61
 hexadecimal 257–8
 Lambda calculus 344–5
 LAN operating system 370
 LAN servers 369–70
 Language compilers 378
 LAPB protocol 357
 LAPD protocol 357
 Latches/flip-flops
 clocked J-K 83
 master-slave clocked J-K 83
 tri-state D and S-R type 82–3
 Latency 225, 298
 Leadless chip carrier (LCC) 380
 Least recently used 242
 Left rotation 52
 LFSR 406
 Light pen 285
 Line termination 390
 Linear array 216
 Linear feedback shift register (LFSR) 406
 Link interfaces 317
 Liquid crystal display 283
 LISP language 343
 Loading rules 82
 Local area networks (LANs) 350, 360–9
 Cambridge ring 364
 collision avoidance (CA) 361
 collision detection (CD) 361
 CSMA/CD 361–2
 EtherNet 361–2
 IBM token ring 365
 ring-based 363–8
 token ring 361
 Locality of reference 237
 Logic analyser 396, 398
 Logic circuits
 comparators 96–8
 counters 87–92
 decoders 94–6
 dividers 93–4
 PLA, MSI, ASIC, custom-design 81
 shift registers 83–7
 storage registers 82–3
 timed control 99

- Logic design 376–7, 419
Logic level changers 281
Logic simulation 406
Logic testing 399
Logical address 243
Look-ahead circuits 296
Look-ahead parity 223
Look-back parity 223
Loop constant 47
Loop test 49
Loosely coupled processor system 317–20
 crossbar switch 318
 dedicated bus 318
 multiported systems 320
 time-shared buses 319
Low power laser 235
Low power Schottky TTL 380
LSI 14
LTD instruction 332

MACD instruction 332
Machine-code instructions 7
Machine independent language 68
Macrocell 383
Magnetic bubble memory 197, 230–3
 addressing 233
 bias field 232
 chevron pattern 232
 drive frequency 232
 magnetic domains 230
 magnetic material 230
 major-minor loop 233
 non-volatile 232
 organisation 233–4
 read operation 232
 shift register 233
 T-bar pattern 232
 write and erase 232
Magnetic cartridge 201
Magnetic disk memory 197, 201, 224
Magnetic domains 230
Magnetic drum 225
Magnetic material 230
Magnetic recording 281–24
 bandwidth 221
 bipolar 219
 delay modulation 222
 dipulse 219
 frequency modulation (FM) 221
 group code recording (GCR) 222
 Manchester encoding 221
modified FM (MFM) 222
non-return to zero (NRZ) 220
NRZ one (NRZ1) 221
phase encoding (PE) 221
phase modulation (PM) 221
pulse crowding 219
return to zero (RZ) 219
run length limited (RLL) 222
self-clocking 221
surface magnetic flux 218
zero modulation (ZM) 222
 horizontal 218
 reading head 218
 techniques 218–24
 vertical 218
Magnetic recording head 218
Magnetic tape memory 197, 226–30
 capstan type 226
 cartridge unit 229
 file gap 228
 files 228
 helical scan type 230
 pinch roller type 226
 record 228
 record gap 228
 recording method 227–9
 tape mark 228
 tape streamer type 229
 vacuum type 227
Magneto-optical disk 224
Magneto-optical recording 235
Magnitude comparator 384
Main memory 200
Main program 56
Mainframe computers 14
Maintainability 412
Majority voting 417
Major-minor.loop 233
Manchester dataflow computer 312
Manchester encoding 221, 363
Mantissae 28, 180
Manual keyboard 256–61
Masking 53
Master ‘talker’ 267
Mathematical checks 396–7
Mean time between failures (MTBF) 409
Mean time to repair fault (MTRF) 412
Medium access control (MAC) 360
Memory
 byte-addressable 37–8
CCD 213–17

- CMOS 206
 flip-flop 205–8
 magnetic bubble 230–5
 magnetic disk 224–6
 magnetic tape 226–30
 optical 235–6
 read/write cycle 11
 semiconductor 201–13
 synchronous SRAM 206–8
- Memory address unit 11, 13
 Memory capacity 62–3
 Memory categories 198
 Memory cell 196
 Memory characteristics 199
 access time 196
 cost per bit 196
 dynamic 197
 non-volatile 197
 random access 197
 read only 197
 serial access 197
 static 197
 volatile 197
 Memory expansion 212
 Memory management 63, 244–9
 base register 245
 descriptor base register 248
 dynamic relocation 244
 hardware associative store 247
 memory map 246
 overlays 244
 page address 245
 page map 246
 pages 245
 paging techniques 245
 segment 247
 segment description 248
 segment descriptor table 248
 segmentation method 247
 static relocation 244
 table look-up 246
 Memory map 246
 Memory mapped I/O 264
 Memory state 69
 Memory structure 201–5
 Message switching 352
 Metal conductors 213
 Metal oxide semiconductor (MOS) 380
 Metropolitan area networks (MANs)
 350
 Microcomputer, single-chip 15
 Microinstruction 80, 101
 Microinstruction formats 107–11
 bit steering 111
 format shifting 111
 horizontal 107
 single-level encoding 110
 two-level encoding 110–11
 vertical 107
 Microinstruction register 101, 115
 Microinstruction word 107
 Microinstruction, partitioning 107–11
 Microprocessor revolution 14
 Microprogram 80, 101
 Microprogram unit cycle time 104
 Microprogram (control) memory 101,
 111–12
 blocked memory 111
 split memory 111
 two-level memory 111–12
 Microprogrammed control 78–80
 asynchronous operation 81–2
 conditional data 105
 interrupt, error overflow 106–7
 K-counter, repeat 105–6
 synchronous operation 81–2
 Wilkes model 99–101
 Microprogramming
 assembly language 120–1
 dynamic control unit 119
 emulation 119
 high level languages 121
 microcode 119–20
 static control unit 119
 Microsequencer 384
 Micro-operation 101
 MIMD computer 291–2
 Minicomputer 14
 Minimal average latency 300
 Minimum distance codes 23
 MIPS 33, 289
 MISD computer 291–2
 Miss, cache 238
 Mnemonics 64
 Mode bits 36
 Mode control 304
 Modems 253, 349, 378
 Modification registers 47, 49
 Modified FM (MFM) 222
 Modifier bit 6–7, 33, 47, 107
 Modular concept 377
 Modula-2 language 343
 Modulo 2 22, 89
 Modulo 2 addition 186

- Monitor** 343
Monitor station 363
MOPS 289
MOS capacitor 216
MOS charge storage 208
MOSFET 382
Motorola M68000
 MC68881/2 co-processors 185
 memory addressing 38
 memory capacity 62
Motorola M68020 read/write 268
Mouse 253, 263–4
MPGL language 121
MPL language 121
MTBF 409
Multibus 72, 283
Multipchip circuits 378
Multiple interrupts 273–4
Multiple processor systems 312–25
 coupling 314
 distributed systems 313
 dynamic allocation 313
 floating control mode 313
 graceful degradation 313
 loosely coupled systems 317
 static allocation 313
 tightly coupled 314
 transputer 314
Multiple shifts 165
Multiplexer 212, 384
Multiplication, unsigned
 array 157–9
 dedicated hardware 155–6
 quasi-serial 159–60
 quaternary 167
 repeated addition 153–5
 shifting across zeros 164
 simultaneous multiplier 163
 table look-up 167–8
 uniform multiple shifts 165–6
Multiplication, signed
 Booth's algorithm 175–7
 correction terms 174–5
Multipliers, fast unsigned
 partitioned 168
 reduced addition cycles 164–7
 residue 188
 simultaneous 163–4
 table look-up 167–9
Multiported systems 320
Multi-access working 60
Multi-byte operations 185
Nanoinstructions 112
Negative numbers 24
Nested loops 57
Nesting interrupts 274
Netlist 420
Network
 bridge 351
 data structure 351
 gateway 351
 protocols 351
Network control unit 303
Noise 375
Noise spikes 389
Noise, in circuits 388–95
Non-restoring division 172, 178
Non-return to zero one (NRZ1) 221
Non-return to zero (NRZ) 220
Non-rooted tree 361–2
Non-volatile 197, 232
Normalization 184
Number representation
 1's complement 25
 2's complement 24
 fixed-point 26–7
 floating-point 27–30
 fractional, binary 26
 negative 24
 offset binary 24
 signed magnitude 24
n-channel MOS 213
Object program 64
Occam language 292, 314–15, 345–6
Octal 5
Odd parity 189
Offset binary 24
Off-line supervisory control 14
One-carry 139
One's complement 25
On-chip cache 200
Open systems interconnection (OSI)
 351
Operating systems 378
Operation code (op-code) 7
Operational amplifier functions 1
Optical coupler 392
Optical disk 201, 224
Optical disk storage 235–6
 Curie point 235
 Kerr effect 235
 low power laser 235
 magneto-optical recording 235

- optical recording 235
polarized beam splitter 235
WORM 235
Optical recording 235
Opto-isolator 392
Op-code 32
Op-code decoder 12, 13
Op-code structure 7
Ordered ring access 363
Orthogonal instructions 336
Oscillator 384
OSI 351
OSI seven-layer model 355
Output devices 254
Out-of-range 397
Over range 184
Overflow 25, 397
Overlaying 244

Packet 352
Packet flow 363
Packet switching 352
Packets finished with 363
Packet-switching protocol 367
Page 62
Page address 245
Page map 246
Pages 245
Paging techniques 245
PAL 386
Parallel associative processor 340
Parallel channel 351
Parallel computer 11
Parallel interface 282
Parallel port 256, 378
Parallel processing languages 343–6
Ada 343
concurrent pascal 343
extended FORTRAN 343
first generation 343
modula-2 343
occam 345–6
Parallel processor systems 289
Parallel redundancy 415
Parallelism, applied 290
Parallelism, natural 290
Parity codes 186
Parity checking 21–3
Parity, row/column 21
Parity-checked adder 189
Partial dividend 171
Partial products 153

Partitioning 400
Pascal 67
Path sensitizing 400–3
PCM 349
PDP-11
 address modes 39–43
 memory addressing 38
 memory capacity 62
 register set 34–5
Performance, computer 33
Peripherals 252
Personal computer (PC) 375
Petri net 310
Phase encoding (PE) 221
Phase modulation 221, 366
Phased voltage (clocks) 216
Phase-locked loop 368
Pin grid array (PGA) 380
Pinch roller 226
Pipeline computers 292–302
 collision 296
 collision matrix 300
 collision vector 298
 forbidden interval set 298
 forbidden latency set 298
 greedy control 300
 greedy cycles 300
 instruction overlap 294
 interlocks 296
 latency 298
 look-ahead circuits 296
 minimal average latency 300
 pipeline delay 296
 precedence rules 295
 shift register controller 299
 unified state diagram 300
Pipeline delay 296
Pipeline register 115
Pipeline test register 384
Pipelining 74, 114, 335
Pixel 262, 284
PLA 386
PLE 386
Pneumatic logic 378
Poisson distribution 417
Polarized beam splitter 235
Polling 55, 322–4
Polling method 272
Port 53
Post-increment 42
Potential well 216
Power filter network 391

- Precedence rules 295
Pre-decrement 42
Printer 254
Priority 56
Probability of failure 416
Procedure 70
Process specification 425
Process synchronization 343
Processing elements 302
Processor state 61, 69
Processor structure 71–3
 data transmission bandwidth 72
 disk drives 72
 futurebus 72
 IEEE 488 bus 72
 IEEE 796 (multibus) 72
 real-time applications 72
 VME bus 72
Program controlled I/O 55
Program counter 11, 34, 43
Program loop 47
Program parameters 57
Programmable array logic (PAL) 386
Programmable logic array (PLA) 386
Programmable logic element (PLE)
 386
Programmable ROM (PROM) 212,
 386
Programmer's model 330–1
Program-controlled data transfer
 255–6
PROM 212, 386
Propagation delay 81, 407
Protocols 349
 data link layer 355
 OSI seven-layer model 355
 RS232-C/422 355
 V24 355
 X.25 355
Protracted carries 136
Pseudo-operations 63, 65–7
Pseudo-operORG, EQU, FCB, RMB,
 END 66
Pulse code modulation (PCM) 349
Pulse crowding 219
Push-down stack registers 57
- Quadrant 305
Quaternary algorithm 167
Queue 57–8
Quotient 171
- Radix 18, 28
RAM 197
Random access memory 197
Random replacement 242
Read cycle 206
Read only memory 197
Read only memory (ROM) 210–3
Read operation 239
Read/write signal WE 205
Real estate 198, 387
Real-time control 72
Real-time systems 275
Record 228
Record gap 228
Recording, RLL, MFM 225
Reduced instruction set computers
 (RISC) 334–8
 delayed branching 336
 floating-point hardware 338
 hardwired control 337
 instruction & data caches 336
 interrupts 337
 orthogonal instructions 336
 pipelining 335
 principles 335
 register scoreboard 336
 register sets 335
 register windows 335
 three-operand architecture 336
Reduced intrusion scan path 406
Redundancy 21, 399, 413–18
Refresh amplifier 209
Refresh controller 209
Register insertion 364–6
Register scoreboard 336
Register sets 34, 335
Register structure 315
Register transfer 11
Register transfer languages 68–71
 APL, ISP, ISPS 69–71
 HDL, HILO 71
Register windows 335
Relays 378
Reliability 375, 377, 407–13
Reliability prediction 410
Remainder 171
Removable magnetic disk 201
Repeater station 366
Replacement algorithms 241–2
Residue codes 186–8
Residue number multiplication 187
Residue of a number 186–7

- Residue-checked adder 191
 Resistor-transistor logic (RTL) 380
 Restoring division 171–2
 Return to zero (RZ) 219
 Reverse Polish notation 57
 Re-entrancy 60
 Re-entrant routines 59–60
 Right rotation 51
 Ringing 391
 Ring-based networks 363–8
 carrier 364
 failure 364
 monitor station 363
 ordered ring access 363
 packet flow 363
 packets finished with 363
 Ripple carry 88
 RISC 121
 RISC principles 335
 RLL 222
 RLL m/n code 222
 Rollback 414
 Rolling-ball principle 263
 ROM 197, 384
 Round-off 178–80
 Round-robin allocation 322
 RPTK instruction 332
 RS232-C/422 355
 RTL 380
 Run length limited (RLL) 222
 Safety critical 186
 Scalar operations 288
 Scaling 18
 Scaling factor 27
 Scan path 403
 Schematic capture 407
 Schottky diode 380
 SDLC 357
 Sector 62, 224
 Sector bit 6–7, 107
 Segment 247
 Segment description 248
 Segment descriptor table 248
 Segmentation method 247
 Self-clocking 221
 Semaphore flags 320
 Semiconductor Integrated circuits
 application specific (ASIC) 378
 large scale integration (LSI) 378
 medium scale integration (MSI) 378
 monolithic 378, 380–3
 very large scale (VLSI) 378
 Semiconductor memory 197
 Semi-custom design 385
 Serial access memory 197
 Serial adder
 D-type flip-flop equations 131
 full 131–7
 J-K flip-flop equations 131
 multiple-input binary 134–5
 shift register storage 132–3
 Serial port 256, 260, 378
 Set-associative mapping 241
 Shaft encoder 263
 Shaft position converter 253
 Shift register 213, 233, 384
 Shift register controller 299
 Shift registers
 bidirectional shift 85
 end-around shift 84
 two-rank parallel 86
 Shifted operands 192
 Shifting across zeros 164
 Sign digits 175
 Signature analysis 405
 Signed magnitude 24
 Signed operands 178
 SIMD computer 291–2
 SIMPL language 121
 Single address machine 9
 Single address space 243
 Single stuck fault model 400
 Single-level encoding 110
 Single/multiple disk 225
 SISAL high level language 312, 343
 SISD computer 291–2
 Sketchpad 253
 Skip chain method 272–3
 Slave ‘listener’ 268
 SNOBOL4 language 121
 Soft sectored disk 225
 Software design 4
 Software engineering 398, 422–6
 Software faults 398
 Software packages 378
 Software redundancy 414
 Software tools 375, 419–26
 Software validation 406, 419
 Software verification 419
 Solomon computer 303
 Source instruction 65
 Source program 64
 SPNLN assembler language 312
 SRAM 201
 SSADM method 424

- Stack overflow 58
Stack pointer 34, 43, 58
Stack underflow 58
Stacks 57–61, 116
Standard bus protocols 282–3
Standard cell 383
Star configuration 367
State transition diagram 424
Static processor allocation 313
Static RAM (SRAM) 201, 384
Static relocation 244
Static storage 197
Static task allocation 311
Status flags 320
Status register 34
Std-bus 283
Stored program 4
Straight line coding 332
Structured analysis 424
Structured design 419
Structured programming 57, 426
STRUM language 121
Subroutine 56–7
Subroutine link 56
Subroutine link procedures 59–60
Subroutine nesting 60
Subracter
 full 127–8
 half 127
Super VGA 284
Surface magnetic flux 218
Symbolic address 64
Synchronization 262
Synchronous character protocol 357
Synchronous SRAM 200, 206–8
System design 376
System redundancy 413
System specification 419
System stack 60
System table 60
System testing 375

Table look-up 246
Table look-up multiplier 167–9
Tablet stylus 285
Tag 237
Tag comparison 237
Tag field 239
Tape equipment 253–4
Tape mark 228
Tape streamer 201, 229
TASS assembler language 312
Temporal languages 71

Temporary register 49
TEMPURA language 71
Terminator 64
Test cycle 399
Test pattern generation 400
Test vector 400, 403
Thick-film circuits 379
Thin-film circuits 378
Third-party software 375
Three-element FIR filter 330–3
Three-operand architecture 336
Tightly coupled processors 314
Time-division-multiplex 351
Time-shared buses 319
Timing circuit
 frequency oscillator 99
 two-phase clock 99
TMS 320C25 327–34
 3-element FIR filter 330–3
 branch and test 332
 data memory 327
 global memory 328
 internal hardware 329
 LTD instruction 332
 MACD instruction 332
 programmer's model 330–1
 RPTK instruction 332
 straight line coding 332
Token ring 361
Tokens 310
Trace program 398
Track 224
Transceiver 384
Transfer efficiency 217
Transfer function 1
Transistor–transistor logic (TTL) 380
Transparency 349
Transputer 314–17
 channels 316
 INMOS T425 314
 interactive processes 316
 link interfaces 317
 Occam language 314–15
 operate instruction 315
 prefix instruction 315
 register structure 315
Triple mode redundancy (TMR) 417
Tri-state logic 82
TTL 380
Twisted-pair lines 353, 390
Two-dimensional array 209
Two-dimensional selection 203–4
Two-level encoding 110–11

- Two-out-of-three majority 186
Two's complement 24, 130, 174
 addition 25–6
 division 26
 multiplication 26
T-bar pattern 232
- UART 277
Underflow, arithmetic 25
Unified state diagram 300
Unreliability 416
Up-time ratio 412
User-defined control bits 116
User-programmable ROM 212–3
- V process model 424
V24 355
Vacuum magnetic tape unit 227
VAL language 343
Validation 396, 426
Validation, hardware 121
Validation, software 121
Variable-bit-rate 351
VDM method 424
VDU 19, 254, 262–3, 28
Vector operations 288
Vector processors *see Array* 302–10
Vectored interrupt 274
Vectram computer 303
Verification 55, 426
Vertical format 107
VHDL 407, 420–1
VHSIC hardware description language
 (VHDL) 420
Video memory 262
Virtual address 243
Virtual circuit 352
Virtual memory 242–4
Virtual packet format 352
VLSI 14
 CPU chip 72
 architectures 289
 IEEE 802.3 standard 363
VME bus 72, 283
- Volatile 197
Voltage spikes 87, 389
Von Neumann model 288
Voting circuits 186
- Wait micro-operation 104
Wallace tree 158–9
WAN commercial networks
 PSS, IPSS, TYMNET 372
WAN Company networks
 XEROX, DEC, IBM, AT&T 372
WAN co-operative networks
 FIDONET, JUNET 372
WAN metanetworks
 AUSEAnet 372
WAN research networks
 ARPANET, JANET 372
Watch-dog 398
Waterfall model 423
Weighting factor 409
Wide area networks (WANS) 350,
 370–2
Winchester disk 225
Words 69
Word-organized 205
Word-serial 340
WORM 235
Write back 238
Write cycle 206
Write through 238
- X.25 355
- YALLL language 121
Yourdon method 424
- Z formal language 426
Zero modulation (ZM)
 encoding algorithm 222–4
 look-ahead parity 223
 look-back parity 223
Zero-carry 139
Zilog Z800 38