APPENDIX A
COMMANDS & STATEMENTS

Commands & Statements

The following is a list of all CC-40 BASIC commands and statements in alphabetical order. Commands are listed first. Statements are listed next. Most statements can be executed immediately as well as used in a program line. Those statements that can be used only in a program line have an asterisk (*) after them. Commands and statements that can be abbreviated have the acceptable abbreviation in *italics*.

Commands

CALL ADDMEM
CALL CLEANUP
*CON*TINUE
LIST
NEW
*NUM*BER
OLD
*REN*UMBER
SAVE
VERIFY

Statements

ACCEPT *
ATTACH
BREAK
CALL
CALL CHAR
CLOSE
DATA
CALL DEBUG
DEG
*DEL*ETE
DIM
DISPLAY
END
CALL ERR
CALL EXEC
FOR TO STEP
FORMAT
CALL GETLANG
CALL GETMEM
GOSUB *
GOTO *
GRAD
IF THEN ELSE
IMAGE
CALL INDIC
INPUT *
CALL IO
CALL KEY
LET
LINPUT *
CALL LOAD
NEXT
ON BREAK
ON ERROR
ON GOSUB *
ON GOTO *
ON WARNING
OPEN
PAUSE
CALL PEEK
CALL POKE
PRINT
RAD
RANDOMIZE
READ *
RELEASE
CALL RELMEM
REM
RESTORE
RETURN *
RUN
CALL SETLANG
STOP
SUB *
SUBEND *
SUBEXIT *
UNBREAK
CALL VERSION

Built-In Functions

The following list gives a brief description of each CC-40 BASIC function in alphabetical order.

Function	Value Returned and Comments
ABS	Absolute value of a numeric expression.
ACS	Trigonometric arccosine of a numeric expression given in the angular measure indicated in the display.
ASC	The numeric ASCII code of the first character of a string expression.
ASN	Trigonometric arcsine of a numeric expression given in the angular measure indicated in the display.
ATN	Trigonometric arctangent of a numeric expression given in the angular measure indicated in the display.
CHR$	A one-character string that corresponds to an ASCII code.
COS	Trigonometric cosine of a numeric expression calculated using the angular measure indicated in the display.
EOF	End-of-file condition of a file.
EXP	Exponential value (e^x) of a numeric expression.
FRE	Information about available memory.
INT	Integer value of a numeric expression.
INTRND	Integer random number with a specified maximum value.
KEY$	A one-character string that corresponds to a key pressed.
LEN	Number of characters in a string expression.
LN	Natural logarithm of a numeric expression.
LOG	Common logarithm of a numeric expression.
NUMERIC	Number that denotes whether a string expression is a valid representation of a numeric constant.
PI	The value of π (3.14159265359).
POS	Position of the first occurrence of one string expression within another.
RND	Random number from 0 to 1.
RPT$	String that is a specific number of repetitions of a string expression.

(continued)

(continued)

Function	Value Returned and Comments
SEG$	Substring of a string expression, starting at a specified point in that string and ending after a certain number of characters.
SGN	Sign of a numeric expression.
SIN	Trigonometric sine of a numeric expression calculated using the angular measure indicated in the display.
SQR	Square root of a numeric expression.
STR$	String equivalent of a numeric expression.
TAB	Column position for the next item in the *print-list* of PRINT or DISPLAY.
TAN	Trigonometric tangent of a numeric expression calculated using the angular measure indicated in the display.
VAL	Numeric value of a string expression which represents a number.

RESERVED WORDS

Reserved Words

The following is a list of all CC-40 BASIC reserved words. A reserved word may not be used as a variable name, but may be a portion of a variable name.

ABS	GOSUB	RELATIVE
ACCEPT	GOTO	RELEASE
ACS	GRAD	REM
ALL	IF	REN
ALPHA	IMAGE	RENUMBER
ALPHANUM	INPUT	RESTORE
AND	INT	RETURN
APPEND	INTERNAL	RND
ASC	INTRND	RPT$
ASN	KEY$	RUN
AT	LEN	SAVE
ATN	LET	SEG$
ATTACH	LINPUT	SGN
BEEP	LIST	SIN
BREAK	LN	SIZE
CALL	LOG	SQR
CHR$	NEW	STEP
CLOSE	NEXT	STOP
CON	NOT	STR$
CONTINUE	NULL	SUB
COS	NUM	SUBEND
DATA	NUMBER	SUBEXIT
DEG	NUMERIC	TAB
DEL	OLD	TAN
DELETE	ON	THEN
DIGIT	OPEN	TO
DIM	OR	UALPHA
DISPLAY	OUTPUT	UALPHANUM
ELSE	PAUSE	UNBREAK
END	PI	UPDATE
EOF	POS	USING
ERASE	PRINT	VAL
ERROR	PROTECTED	VALIDATE
EXP	RAD	VARIABLE
FOR	RANDOMIZE	VERIFY
FORMAT	READ	WARNING
FRE	REC	XOR

ASCII CODES & KEYCODES LIST

ASCII Codes and Keycodes

The following table lists the ASCII character codes in decimal and hexadecimal notation. The ASCII codes produced and/or character(s) displayed when the key or key sequence is pressed are shown in the column titled CHARACTER. The characters that can be displayed using the CHR$ function are shown in the column titled DISPLAYED USING CHR$. The keys that are pressed to generate the ASCII code are shown in the column titled KEY SEQUENCE.

User-defined character codes (0-6) and the user-assigned keys (codes 128-137) are shown as two asterisks (**).

ASCII Code			Displayed	Key
DEC	HEX	Character	Using CHR$	Sequence
00	00	NULL	**	**[CTL] 0**
01	01	SOH	**	**[CTL] A**
02	02	STX	**	**[CTL] B**
03	03	ETX	**	**[CTL] C**
04	04	EOT	**	**[CTL] D**
05	05	ENQ	**	**[CTL] E**
06	06	ACK	**	**[CTL] F**
07	07	BEL		**[CTL] G**
08	08	BS		**[CTL] H**
09	09	HT		**[CTL] I**
10	0A	LF		**[CTL] J**
11	0B	VT		**[CTL] K**
12	0C	FF		**[CTL] L**
13	0D	CR		**[CTL] M** or **[ENTER]**
14	0E	SO		**[CTL] N**
15	0F	SI		**[CTL] O**
16	10	DLE		**[CTL] P**
17	11	DC1		**[CTL] Q**
18	12	DC2		**[CTL] R**
19	13	DC3		**[CTL] S**
20	14	DC4		**[CTL] T**
21	15	NAK		**[CTL] U**
22	16	SYN		**[CTL] V**
23	17	ETB		**[CTL] W**
24	18	CAN		**[CTL] X**
25	19	EM		**[CTL] Y**
26	1A	SUB		**[CTL] Z**

(continued)

(continued)

ASCII Code DEC	HEX	Character	Displayed Using CHR$	Key Sequence
27	1B	ESC		[CTL] [CLR]
28	1C	FS		[CTL] =
29	1D	GS		[CTL] ;
30	1E	RS		[CTL] .
31	1F	US		[CTL] ,
32	20	Space	Space	Space
33	21	!	!	[SHIFT] !
34	22	"	"	[SHIFT] "
35	23	#	#	[SHIFT] #
36	24	$	$	[SHIFT] $
37	25	%	%	[SHIFT] /
38	26	&	&	[SHIFT] &
39	27	'	'	[SHIFT] '
40	28	(	(	[SHIFT] (
41	29	)	)	[SHIFT])
42	2A	*	*	•
43	2B	+	+	+
44	2C	,	,	,
45	2D	–	–	–
46	2E	.	.	.
47	2F	/	/	/
48	30	0	0	0
49	31	1	1	1
50	32	2	2	2
51	33	3	3	3
52	34	4	4	4
53	35	5	5	5
54	36	6	6	6
55	37	7	7	7
56	38	8	8	8
57	39	9	9	9
58	3A	:	:	[SHIFT] :
59	3B	;	;	;
60	3C	<	<	[SHIFT] ,
61	3D	=	=	=
62	3E	>	>	[SHIFT] .
63	3F	?	?	[SHIFT] ?

(continued)

(continued)

ASCII Code DEC	HEX	Character	Displayed Using CHR$	Key Sequence
64	40	@	@	[CTL] 2
65	41	A	A	[SHIFT] A
66	42	B	B	[SHIFT] B
67	43	C	C	[SHIFT] C
68	44	D	D	[SHIFT] D
69	45	E	E	[SHIFT] E
70	46	F	F	[SHIFT] F
71	47	G	G	[SHIFT] G
72	48	H	H	[SHIFT] H
73	49	I	I	[SHIFT] I
74	4A	J	J	[SHIFT] J
75	4B	K	K	[SHIFT] K
76	4C	L	L	[SHIFT] L
77	4D	M	M	[SHIFT] M
78	4E	N	N	[SHIFT] N
79	4F	O	O	[SHIFT] O
80	50	P	P	[SHIFT] P
81	51	Q	Q	[SHIFT] Q
82	52	R	R	[SHIFT] R
83	53	S	S	[SHIFT] S
84	54	T	T	[SHIFT] T
85	55	U	U	[SHIFT] U
86	56	V	V	[SHIFT] V
87	57	W	W	[SHIFT] W
88	58	X	X	[SHIFT] X
89	59	Y	Y	[SHIFT] Y
90	5A	Z	Z	[SHIFT] Z
91	5B	[	[	[CTL] 8
92	5C	¥	¥	[CTL] /
93	5D	]	]	[CTL] 9
94	5E	^	^	[SHIFT] ^
95	5F	_	_	[CTL] 5
96	60	`	`	[CTL] 3
97	61	a	a	A
98	62	b	b	B
99	63	c	c	C
100	64	d	d	D

(continued)

(continued)

ASCII Code DEC	HEX	Character	Displayed Using CHR$	Key Sequence
101	65	e	e	E
102	66	f	f	F
103	67	g	g	G
104	68	h	h	H
105	69	i	i	I
106	6A	j	j	J
107	6B	k	k	K
108	6C	l	l	L
109	6D	m	m	M
110	6E	n	n	N
111	6F	o	o	O
112	70	p	p	P
113	71	q	q	Q
114	72	r	r	R
115	73	s	s	S
116	74	t	t	T
117	75	u	u	U
118	76	v	v	V
119	77	w	w	W
120	78	x	x	X
121	79	y	y	Y
122	7A	z	z	Z
123	7B	{	{	[CTL] 6
124	7C	|	|	[CTL] 1
125	7D	}	}	[CTL] 7
126	7E	→	→	[CTL] 4
127	7F	DEL	←	[SHIFT] ↓
128	80	**		[FN] 0
129	81	**		[FN] 1
130	82	**		[FN] 2
131	83	**		[FN] 3
132	84	**		[FN] 4
133	85	**		[FN] 5
134	86	**		[FN] 6
135	87	**		[FN] 7
136	88	**		[FN] 8
137	89	**		[FN] 9
138	8A			
139	8B			

(continued)

(continued)

ASCII Code DEC	HEX	Character	Displayed Using CHR$	Key Sequence
140	8C	.		
141	8D			[SHIFT] /
142	8E			[SHIFT] *
143	8F			[SHIFT] −
144	90			[SHIFT] +
145	91			[CTL] *
146	92			[CTL] −
147	93			[CTL] +
148	94	DELETE		[FN] ←
149	95			[FN] →
150	96			[FN] ↑
151	97	NUMBER		[FN] ↓
152	98	VERIFY		[FN] /
153	99	SAVE		[FN] *
154	9A	OLD		[FN] −
155	9B	LIST		[FN] +
156	9C	CALL		[FN] .
157	9D	ELSE		[FN] ,
158	9E	CHR$(		[FN] ;
159	9F	GOTO		[FN] =
160	A0			[FN] [CLR]
161	A1	ASN(	｡	[FN] A
162	A2	PAUSE	｢	[FN] B
163	A3	GRAD	｣	[FN] C
164	A4	ATN(	､	[FN] D
165	A5	TAN(	･	[FN] E
166	A6	LN(	ｦ	[FN] F
167	A7	LOG(	ｧ	[FN] G
168	A8	LINPUT	ｨ	[FN] H
169	A9	NEXT	ｩ	[FN] I
170	AA	INPUT	ｪ	[FN] J
171	AB	PRINT	ｫ	[FN] K
172	AC	USING	ｬ	[FN] L
173	AD	THEN	ｭ	[FN] M
174	AE	IF	ｮ	[FN] N
175	AF	GOSUB	ｯ	[FN] O
176	B0	RETURN	ｰ	[FN] P
177	B1	SIN(	ｱ	[FN] Q
178	B2	PI	ｲ	[FN] R

(continued)

(continued)

ASCII Code DEC	HEX	Character	Displayed Using CHR$	Key Sequence
179	B3	ACS(	ウ	[FN] S
180	B4	SQR(	エ	[FN] T
181	B5	TO	オ	[FN] U
182	B6	EXP(	カ	[FN] V
183	B7	COS(	キ	[FN] W
184	B8	RAD	ク	[FN] X
185	B9	FOR	ケ	[FN] Y
186	BA	DEG	コ	[FN] Z
187	BB	BREAK	サ	[FN] [BREAK]
188	BC		シ	[SHIFT] [RUN]
189	BD		ス	[CTL] [RUN]
190	BE	CONTINUE	セ	[FN] [RUN]
191	BF	RUN	ソ	[RUN]
192	C0		タ	[SHIFT] [FN] 0
193	C1		チ	[SHIFT] [FN] 1
194	C2		ツ	[SHIFT] [FN] 2
195	C3		テ	[SHIFT] [FN] 3
196	C4		ト	[SHIFT] [FN] 4
197	C5		ナ	[SHIFT] [FN] 5
198	C6		ニ	[SHIFT] [FN] 6
199	C7		ヌ	[SHIFT] [FN] 7
200	C8		ネ	[SHIFT] [FN] 8
201	C9		ノ	[SHIFT] [FN] 9
202	CA		ハ	
203	CB		ヒ	
204	CC		フ	
205	CD		ヘ	
206	CE		ホ	
207	CF		マ	
208	D0		ミ	
209	D1		ム	
210	D2		メ	
211	D3		モ	
212	D4		ヤ	
213	D5		ユ	
214	D6		ヨ	
215	D7		ラ	
216	D8		リ	
217	D9		ル	

(continued)

(continued)

ASCII Code DEC	HEX	Character	Displayed Using CHR$	Key Sequence
218	DA		レ	
219	DB		ロ	
220	DC		ワ	
221	DD		ン	
222	DE		゛	
223	DF		゜	
224	E0		α	
225	E1		ä	
226	E2		β	
227	E3		ε	
228	E4		μ	
229	E5	PB	σ	[SHIFT] ↑
230	E6	OFF	ρ	[OFF]
231	E7	BREAK	g	[BREAK]
232	E8	UP	√	↑
233	E9	DOWN	⁻¹	↓
234	EA	SHIFT	j	[SHIFT] [ENTER]
235	EB		ˣ	
236	EC		¢	
237	ED		£	
238	EE		ñ	
239	EF		ö	
240	F0		p	
241	F1		q	
242	F2		θ	
243	F3		∞	
244	F4		Ω	
245	F5		ü	
246	F6	DEL	Σ	[SHIFT] ←
247	F7	INS	π	[SHIFT] →
248	F8	HOME	x̄	[CTL] ↑
249	F9	SKIP	y	[CTL] ↓
250	FA	CLR	千	[CLR]
251	FB	BTAB	万	[CTL] ←
252	FC	←	円	←
253	FD	FTAB	÷	[CTL] →
254	FE	→		→
255	FF			

Trigonometric Calculations and Restrictions

The following information provides restrictions for trigonometric functions, a list of trigonometric identities, and a table of trigonometric conversions.

Restrictions for SIN, COS, TAN

The approximate valid range for the arguments of SIN, COS, and TAN is given below for radians, degrees, and grads.

$|X| < PI/2*10^{10}$ radians
$|X| < 90*10^{10}$ degrees
$|X| < 10^{12}$ grads

Restrictions For Inverse Trigonometric Functions

The largest angle resulting from an arc function is 180°, π radians, or 200 grads. Because each resultant value has many angle equivalents (for example, ASN(.5) = 30°, 150°, 390°, ...), angles calculated by inverse trigonometric functions are restricted as follows.

	Range of calculated angles		
ARC Function	**Degrees**	**Radians**	**Grads**
Arcsine (x)	−90 to 90	$-\pi/2$ to $\pi/2$	−100 to 100
Arccos (x)	0 to 180	0 to π	0 to 200
Arctan (x)	−90 to 90	$-\pi/2$ to $\pi/2$	−100 to 100

Trigonometric Identities

The following trigonometric functions are not part of CC-40 BASIC, but may be calculated using the BASIC expressions described below. The expressions for functions that are frequently used can be assigned to any of the ten user-assigned keys as described in chapters 1 and 2.

Function	**Symbol**	**BASIC Expression Equivalent**
Secant	SEC(X)	1/COS(X)
Cosecant	CSC(X)	1/SIN(X)
Cotangent	COT(X)	1/TAN(X)
Inverse Secant	ARCSEC(X)	SGN(X)*ACS(1/X)
Inverse Cosecant	ARCCSC(X)	SGN(X)*ASN(1/X)+(SGN(X)−1)*PI/2
Inverse Cotangent	ARCCOT(X)	PI/2−ATN(X) or PI/2+ATN(−X)
Hyperbolic Sine	SINH(X)	(EXP(X)−EXP(−X))/2
Hyperbolic Cosine	COSH(X)	(EXP(X)+EXP(−X))/2
Hyperbolic Tangent	COTH(X)	−2*EXP(−X)/(EXP(X)+EXP(−X))+1
Hyperbolic Secant	SECH(X)	2/(EXP(X)+EXP(−X))
Hyperbolic Cosecant	CSCH(X)	2/(EXP(X)−EXP(−X))
Hyperbolic Cotangent	COTH(X)	2*EXP(−X)/(EXP(X)−EXP(−X))+1
Inverse Hyperbolic Sine	ARCSINH(X)	LN(X+SQR(X*X+1))
Inverse Hyperbolic Cosine	ARCCOSH(X)	LN(X+SQR(X*X−1))
Inverse Hyperbolic Tangent	ARCTANH(X)	LN((1+X)/(1−X))/2
Inverse Hyperbolic Secant	ARCSECH(X)	LN((1+SQR(1−X*X))/X)
Inverse Hyperbolic Cosecant	ARCCSCH(X)	LN((SGN(X)*SQR(X*X+1)+1)/X)
Inverse Hyperbolic Cotangent	ARCCOTH(X)	LN((X+1)/(X−1))/2

Radian, Degree, and Grad Conversions

It may be necessary to convert angular values from one unit of angle measurement to another. The following table provides the factors needed to make these conversions.

From/To	Degrees	Radians	Grads
Degrees		× PI/180	÷ 0.9
Radians	× 180/PI		× 200/PI
Grads	× 0.9	× PI/200	

Because these conversions are independent of the computer's angle setting, use care when using the results for further calculations. Before you use the result in subsequent trigonometric calculations, make certain that the appropriate angle setting has been selected.

Accuracy Information

Calculation Accuracy

The CC-40, like all computers, operates under a fixed set of rules within preset limits. The mathematical tolerance of the computer is controlled by the number of digits it uses for calculations.

The CC-40 uses a minimum of 13 digits to perform calculations. The results are rounded to 10 digits when displayed in the default display format. The computer's 5/4 rounding technique adds 1 to the least significant digit of the display if the next nondisplayed digit is five or more. If this digit is less than five, no rounding occurs. Without these extra digits, inaccurate results such as the following would frequently be displayed.

```
1/3*3=.9999999999
```

This result occurs because 1/3 is maintained as .3333333333 in the finite internal representation of a number. However, when 1/3 × 3 is rounded to 10 digits, the answer `1.` is displayed.

The more complex mathematical functions are calculated using iterative and polynomial methods. The cumulative rounding error is usually kept beyond the tenth digit so that displayed values are accurate. Normally there is no need to consider the undisplayed digits. However, certain calculations may cause the unexpected appearance of these extra digits as shown below.

2/3 = .66666666666667 and 1/3 = .33333333333333
2/3 − 1/3 − 1/3 = .00000000000001 (displayed 1.E − 14)

Such possible discrepancies in the least significant digits of a calculated result are important when testing if a calculated result is equal to another value. In testing for equality, precautions should be taken to prevent improper evaluation.

A useful technique is to test whether two values are sufficiently close together rather than absolutely equal as shown below.

Instead of

```
IF X=Y THEN ...
```

use

```
IF ABS(X−Y) < 1E−11 THEN ...
```

Internal Numeric Representation

The CC-40 uses radix-100 format for internal calculations. A single radix-100 digit ranges in value from 0 to 99 in base 10. The computer uses a 7-digit mantissa which results in 13 to 14 digits of decimal precision. A radix-100 exponent ranges in value from −64 to +63 which yield decimal exponents from 10^{-128} to 10^{+126}. The exponent and the 7-digit mantissa combine to provide a decimal range from −9.9999999999999E+127 through −1.E−128; zero; and then +1.E−128 through +9.9999999999999E+127.

The internal representation of the radix-100 format requires eight bytes. The first byte contains the exponent, and the algebraic sign of the entire floating-point number. The exponent is a 7-bit hexadecimal value offset or biased by 40_{16} (the 16 subscript indicates hexadecimal values in this appendix). The correspondence between exponent values is shown below.

Biased hexadecimal value	00_{16}	to	40_{16}	to	$7F_{16}$
Radix-100 value	−64	to	0	to	+63
Decimal value	−128	to	0	to	+126

If the floating-point number is negative, the first byte (the exponent value) is inverted (1's complement). Each byte of the mantissa contains a radix-100 digit from 0 to 99 represented in binary coded decimal (BCD) form. In other words, the most significant four bits of each byte represent a decimal digit from 0 to 9 and the least significant four bits represent a decimal digit from 0 to 9. The first byte of the mantissa contains the most significant digit of the radix-100 number. The number is normalized so that the decimal point immediately follows the most significant radix-100 digit.

The following examples show some decimal values and their internal representations.

Decimal Number	**Internal Value**							
1	40_{16}	01_{16}	00_{16}	00_{16}	00_{16}	00_{16}	00_{16}	00_{16}
10	40_{16}	10_{16}	00_{16}	00_{16}	00_{16}	00_{16}	00_{16}	00_{16}
100	41_{16}	01_{16}	00_{16}	00_{16}	00_{16}	00_{16}	00_{16}	00_{16}
1234	41_{16}	12_{16}	34_{16}	00_{16}	00_{16}	00_{16}	00_{16}	00_{16}
PI	40_{16}	03_{16}	14_{16}	15_{16}	92_{16}	65_{16}	35_{16}	90_{16}
− PI	BF_{16}	03_{16}	14_{16}	15_{16}	92_{16}	65_{16}	35_{16}	90_{16}

System Power Up & Down Procedure

This appendix describes the actions taken when the system is powered up and down.

System Power Up

When the CC-40 is turned off, the power continues to be supplied to the CMOS RAM chips as part of the *Constant Memory*™ feature. This power supply gives the computer the capability to retain information in memory even when the computer is turned off. Pressing the **[ON]** key turns on the full power supply and causes the TMS70C20 microprocessor to execute the power up code. The power up code resets all hardware with power up defaults and performs several operations to initialize the system.

Next the power up code checks to see if the expected values are stored in RAM locations 0802_{16} and 0803_{16} (the 16 subscript indicates hexadecimal values). One of these locations must be an $A5_{16}$ and the other a $5A_{16}$ or the system is coldstarted as described below. If the expected values are correct, an exclusive-OR checksum is calculated for all of the RAM in the system. This checksum is compared to the checksum value that was stored when the computer was turned off. If the checksums are identical, the system is warmstarted as described below. If the checksums are different and a CALL ADDMEM is in effect, the system is coldstarted. If the checksums are different and a CALL ADDMEM is not in effect, the message `Memory contents may be lost` is displayed and only essential parts of the system are initialized. This latter operation leaves the contents of program memory intact and is described below under "Partial Initialization".

Warmstart

When the CC-40 is warmstarted, a bus reset command is sent over the I/O bus. If a CALL ADDMEM is not in effect, the cartridge port is checked for a cartridge. If one is installed, pointers to the program/subprogram header list and BASIC extension information are copied into the system reserved area and the speed of the system is matched with the cartridge speed. The cartridge is then checked for a program that is to be executed at power up. If one exists, it is executed; otherwise, the system enters the system command level.

Coldstart

A coldstart of the CC-40 initializes the system by:

- Setting the language flag to English
- Initializing the expected values used at power up
- Initializing the BASIC program space
- Initializing the user-defined strings
- Initializing all important registers, RAM based trap vectors, etc.

The I/O bus is then reset and the cartridge is checked as in warmstarting the system. **Note:** Entering the command **NEW ALL** is the same as coldstarting the system without checking the cartridge port.

Partial Initialization

When the expected values are correct, but the checksum of the RAM is incorrect and CALL ADDMEM is not in effect, the message `Memory contents may be lost` is displayed. The system is powered up essentially 'as is', except that registers necessary for it to run are initialized and the cartridge port is checked.

System Power Down

When the **[OFF]** key is pressed while in system command level, the power down code is entered. This code closes all open files, resets the I/O bus, and calculates the exclusive-OR checksum of memory. This value is stored in memory for the next power up.

Logical Operations on Numbers

The logical operators AND, OR, NOT, and XOR can be used on integer numbers in the range -32768 to 32767. This appendix briefly describes the binary number system, conversion of decimal numbers to their binary equivalents, and the operation of the logical operators.

Binary Notation

Binary (base 2) notation is another way to express the value of a number. Our usual system, decimal (base 10) notation, uses combinations of the ten digits zero through nine. Numbers written in binary notation use only the two digits zero and one. Each position occupied by a binary digit (a 0 or 1) is called a bit.

In decimal notation, each digit in a number represents a power of 10. For example, the number 2408 in decimal notation can be written in expanded form as follows.

$$(2\times10^3)+(4\times10^2)+(0\times10^1)+(8\times10^0)$$

This is equal to 2408 as shown below.

$$\begin{array}{l}
2\times10^3=2\times1000=2000\\
4\times10^2=4\times100=400\\
0\times10^1=0\times10=0\\
8\times10^0=8\times1=\underline{8}\\
\phantom{8\times10^0=8\times1=}2408
\end{array}$$

In binary notation, each digit represents a power of two. For example, the binary number 101101 can be written as

$$(1\times2^5)+(0\times2^4)+(1\times2^3)+(1\times2^2)+(0\times2^1)+(1\times2^0)$$

For reference purposes, the powers of two and their decimal values are as follows.

...	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
...	128	64	32	16	8	4	2	1

The decimal equivalent of 101101 can be calculated as shown below.

$$\begin{array}{l}
1\times2^5=1\times32=32\\
0\times2^4=0\times16=0\\
1\times2^3=1\times8=8\\
1\times2^2=1\times4=4\\
0\times2^1=0\times2=0\\
1\times2^0=1\times1=\underline{1}\\
\phantom{1\times2^0=1\times1=}45
\end{array}$$

To convert a number from decimal notation to binary notation, repeatedly reduce the decimal number by the greatest power of 2 not larger than the number until there is no remainder.

For example, the decimal number 77 can be converted to binary notation using the following technique.

The largest power of 2 contained in the number 77 is 64 (2^6). A 1 is placed in that position of the binary number as shown below.

128	64	32	16	8	4	2	1
0	1	0	0	0	0	0	0

Reducing 77 by 64 leaves a remainder of 13. The largest power of 2 contained in 13 is 8 (2^3) and a 1 is placed there. Reducing 13 by 8 leaves a remainder of 5. The largest power of 2 contained in 5 is 4 (2^2) and a 1 is placed there. Reducing 5 by 4 leaves a remainder of 1. Place a 1 in the 2^0 position.

The decimal number 77 in binary notation is shown below.

128	64	32	16	8	4	2	1
0	1	0	0	1	1	0	1

You can check the accuracy of the conversion as follows.

$$\begin{array}{l}
1\times2^6=1\times64=64\\
0\times2^5=0\times32=0\\
0\times2^4=0\times16=0\\
1\times2^3=1\times8=8\\
1\times2^2=1\times4=4\\
0\times2^1=0\times2=0\\
1\times2^0=1\times1=\underline{1}\\
\phantom{1\times2^0=1\times1=}77
\end{array}$$

Logical Operations

When logical operations are performed on numbers within the valid range, the CC-40 first converts the values to their 16-bit binary equivalents. The logical operations are performed on a bit-by-bit basis, and the resulting binary number is converted back to decimal notation.

The left-most bit is reserved to indicate the sign (0 = positive; 1 = negative). Therefore, the largest number that can be represented by the remaining 15 bits is 32,767.

If a decimal number with a fractional part is used with a logical operator, the number is rounded before any logical operation is performed.

LOGICAL OPERATIONS ON NUMBERS

The following are the rules for the four logical operators.

Operator	**Rule**
AND	If both bits are 1s, the result is 1. If either bit is 0, the result is 0.
OR	If either bit is a 1, the result is 1. If both bits are zero, the result is 0.
XOR	If either bit, but not both, is 1, the result is 1. If both bits are the same, the result is 0.
NOT	If the bit is 0, the result is 1. If the bit is 1, the result is 0.

The following table shows the results of the four logical operations on all the possible combinations of bits.

AND	First bit	0	0	1	1
	Second bit	0	1	0	1
	Results	0	0	0	1
OR	First bit	0	0	1	1
	Second bit	0	1	0	1
	Results	0	1	1	1
XOR	First bit	0	0	1	1
	Second bit	0	1	0	1
	Results	0	1	1	0
NOT	Bit	0	1		
	Results	1	0		

For example, when the logical operations are performed on the numbers 77 and 67, the numbers are first converted to binary notation. The number 77 is represented in 16 bits as 0000000001001101 and the number 67 is represented in 16 bits as 0000000001000011. The results of performing an AND, an OR, and an XOR on the two values are shown below.

```
     AND                       OR
(77) 0000000001001101     (77) 0000000001001101
(67) 0000000001000011     (67) 0000000001000011
(65) 0000000001000001     (79) 0000000001001111

     XOR
(77) 0000000001001101
(67) 0000000001000011
(14) 0000000000001110
```

The results of performing an AND, OR, and an XOR on 77 and 67 can be obtained on your CC-40 by entering the following.

```
PRINT 77 AND 67; 77 OR 67; 77 XOR 67
```

Using the logical operator NOT on 77 and 67 is shown below.

```
      NOT 77                      NOT 67
(77)  0000000001001101     (67)  0000000001000011
(-78) 1111111110110010     (-68) 1111111110111100
```

To display the results of NOT 77 and NOT 67, enter the following.

```
PRINT NOT 77; NOT 67
```

Note that the results of NOT 77 and NOT 67 have a 1 in the left-most bit which denotes that they represent negative numbers. In the CC-40 a negative binary number is represented as the two's complement of the absolute value of the number.

To obtain the two's complement of a binary number, change each 0 bit to 1 and each 1 bit to 0. Then add 1 to this changed number. For example, the two's complement of 77 is obtained as shown below.

```
77 in binary                    0000000001001101
Change each bit                 1111111110110010
add 1                                          1
-77 in two's-complement form    1111111110110011
```

A more detailed description of binary arithmetic is beyond the scope of this appendix. Refer to a standard reference book on this subject for more information.

DEBUG Monitor

The DEBUG subprogram is used to access the debug monitor. The debug monitor is designed to be used with the Editor/Assembler to read and modify memory and to run and debug assembly language programs and subprograms. More detailed information on the debug monitor is available in the Editor/Assembler manual. Indiscriminate use of the debug monitor may result in loss of data in memory.

Running the DEBUG Monitor

To execute the debug monitor, type **CALL DEBUG** and press **[ENTER]**. The prompt `MONITOR:` is displayed, followed by the flashing cursor, to indicate that the debug subprogram is active. The prompt changes to : after the first command is entered. CALL DEBUG can be used as a statement in a BASIC program to allow debugging of assembly language subprograms called from a BASIC program. The following notational conventions are used in this appendix.

- The characters that are **bold** in examples must be typed by the user and if the characters are to be entered, the **[ENTER]** key must be pressed.
- The space bar and the **[ENTER]** key are used to execute most commands.
- The **[CLR]** and **[BREAK]** keys are used to cancel commands.
- The ← key can be used to erase the previous character typed when entering an address or data.
- Memory addresses can be entered in either hexadecimal or decimal notation. A number is assumed to be in hexadecimal notation unless it is preceded with a decimal point, in which case it is assumed to be in decimal notation.

Displaying Memory—The D Command

The display command is D. It displays the contents of memory eight bytes at a time in hexadecimal notation. Execute the display command by entering

D nnnn

where **nnnn** is the address of the start of the first eight-byte block of memory to be displayed.

The monitor responds by displaying

```
nnnn b0 b1 b2 b3 b4 b5 b6 b7
```

where b0 represents the first byte and b7 represents the eighth byte.

The ↑ or – key can be used to display the values in the next lower addresses in multiples of eight. The ↓ or + key or the space bar can be used to display the values in the next higher addresses in multiples of eight.

To leave the D command, press the **[CLR]**, **[ENTER]**, or **[BREAK]** key.

Examining and Modifying Memory—The M Command

The examine and modify command is M. It can be used to read and modify individual bytes of memory. Execute the command by entering

M nnnn

where **nnnn** is the address of the first byte to be examined or modified.

The monitor responds by displaying

```
nnnn=xx
```

where nnnn is the address of the byte and xx is the hexadecimal value stored in that byte. A new value can be stored in that byte by entering the value.

The ↑ or – key can be used to display the value in the next lower address. The ↓ or + key or the space bar can be used to display the value in the next higher address.

To leave the M command, press the **[CLR]**, **[ENTER]**, or **[BREAK]** key.

Copying Memory—The C Command

The copy command is C. It can be used to copy a block of memory to a specified location. Execute the copy command by entering

C ssss dddd llll

where **ssss** specifies the lowest address of the block to be copied, **dddd** specifies the first memory location to be copied to, and **llll** specifies the number of bytes to copy. **llll** bytes are copied one at a time from **ssss** to **dddd**.

Modifying Processor Information—The P Command

The modify program information command is P. It can be used to modify the microprocessor's program counter (PC), status register (SR), and stack pointer (SP). Execute the command by typing

```
P
```

The monitor responds by displaying

```
PC=nnnn
```

where nnnn is the current hexadecimal value of the program counter. A new value can be entered for the program counter. If the program counter is not to be modified, press the space bar.

The monitor responds by displaying

```
ST=xx
```

where xx is the current hexadecimal value of the status register. A new value can be entered for the status register. If the status register is not to be modified, press the space bar.

The monitor responds by displaying

```
SP=yy
```

where yy is the current value of the stack pointer. A new value can be entered for the stack pointer. If the stack pointer is not to be modified, press the space bar to exit from the command.

Note: Indiscriminate modification of the program counter or stack pointer followed by the E commmand may cause undesirable results.

Setting Break Points—The B Command

The breakpoint command is B. It can be used to set up to two breakpoints. A breakpoint is set by entering an address for either of the breakpoints. Entering an address for a breakpoint causes a break to occur when that location is executed. To set a breakpoint, type the following.

```
B
```

The monitor responds by displaying

```
B nnnn
```

where nnnn is the current value of the first breakpoint. (A 0000 value means no breakpoint has been set.) To set only one breakpoint, type the address, press **[ENTER]**, and the monitor prompts for another command.

To set a second breakpoint, press the space bar instead of **[ENTER]** after the first address has been typed. The monitor responds by displaying

```
nnnn -
```

where nnnn is the address of the first breakpoint set. The address for the second breakpoint can then be entered.

When a break occurs, the monitor displays the prompt

```
· nnnn xx yy:
```

where nnnn is the hexadecimal address where the breakpoint occurred, xx is the hexadecimal value of the status register at the time the breakpoint occurred, and yy is the hexadecimal value of the stack pointer.

Executing a breakpoint automatically clears any breakpoint(s) set.

Single Stepping—The S Command

The single step command is S. It can be used to execute the instruction at the address in the program counter. Execute the command by typing the following.

```
S
```

This instruction has the same effect as executing a breakpoint at the instruction following the current one. (See the B command.)

Executing—The E Command

The execute command is E. It can be used to start execution at the address given in the program counter. Execute the command by entering the following.

```
E
```

Paging—The R Command

The page command is R. It can be used to change the page on which code is executing. The page can be either the system ROM page or a cartridge page. Execute the command by typing the following.

```
R
```

The monitor responds by displaying

```
CARTRIDGE PAGE=x
```

where x is the current page that is selected for the cartridge. If the cartridge page is not to be modified, press the space bar to display the system ROM page. Otherwise, type the new cartridge page number and then press the **[ENTER]** key to exit from the command or the space bar to modify the system page. When the space bar is pressed, the monitor responds by displaying

```
SYSTEM PAGE=n
```

where n is the current system ROM page that is selected while the machine language program is running. The new system ROM page can then be entered.

Help—The ? Command

The help command is ?. It can be used to display a list of the commands used in the debug monitor. Execute the command by typing the following.

?

The monitor responds by displaying

```
COMMANDS=Q,B,E,M,C,S,D,P,R
```

Press the space bar or **[ENTER]** to leave the command.

Exiting—The Q Command

The exit command is Q. It is used to leave the debug monitor by typing the following.

Q

The monitor responds by displaying

```
:Q
```

B can be typed to continue program execution at the next BASIC statement or **I** can be typed to return to system command level.

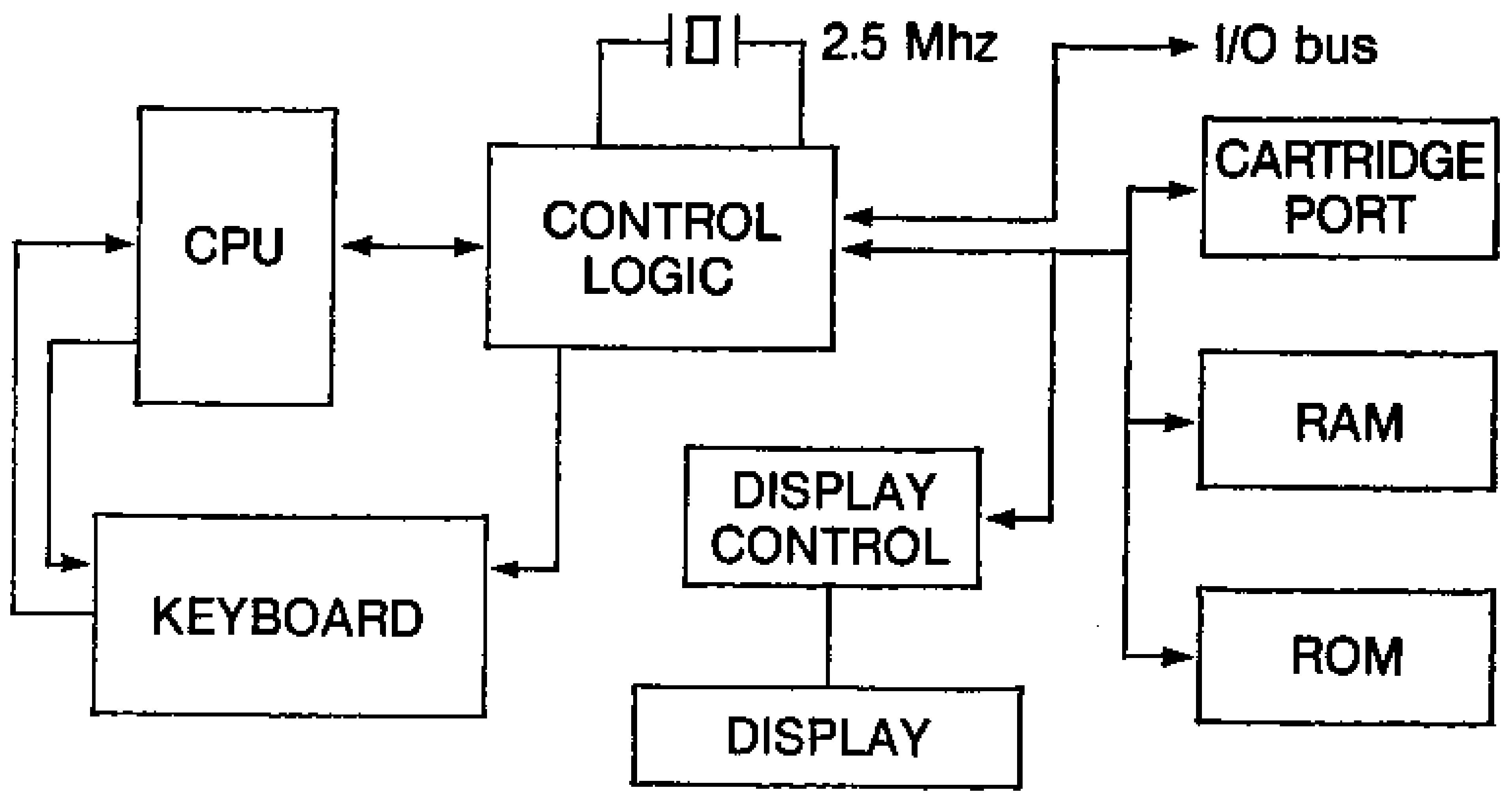
Technical Information

This appendix provides technical information on the Texas Instruments Compact Computer Model CC-40 and presumes some knowledge of digital circuits and assembly language programming. The CC-40 hardware, memory organization, memory expansion, system command level, and the *HEX-BUS*™ interface are described in this appendix. More detailed system information is given in the Editor/Assembler manual.

CC-40 Hardware

The CC-40 is built around the TMS70C20 CMOS microprocessor. The 70C20 is an 8-bit microprocessor with 2K bytes of internal ROM and 128 bytes of RAM (called the register file). A 256-byte block, starting at 0100_{16} (the subscript 16 indicates a hexadecimal number) is used for memory-mapped peripheral ports.

Composing the rest of the system is a 32K-byte ROM, up to 18K bytes of RAM, the display controller subsystem, Liquid Crystal Display (LCD), keyboard, power supply, and control logic. A block diagram is shown below.



CC-40 Memory Organization

The TMS70C20 microprocessor can access a total of 64K bytes of memory. This memory is mapped into several distinct sections.

- A 128-byte register file
- A peripheral file
- System RAM
- The cartridge port
- System ROM
- Processor ROM

Each of these sections is addressed at a specific area in the memory map as shown in the following table.

Address: Decimal	Hex	Description	
0 127	0000 007F	Register File	(128 bytes)
128 255	0080 00FF	unused	(128 bytes)
256 511	0100 01FF	Peripheral File	(256 bytes)
512 2047	0200 07FF	unused	(1.5K bytes)
2048 20479	0800 4FFF	System RAM	(up to 18K bytes)
20480 53247	5000 CFFF	Cartridge port	(32K bytes)
53248 61439	D000 EFFF	System ROM	(8K bytes)
61440 63487	F000 F7FF	unused	(2K bytes)
63488 65535	F800 FFFF	Processor ROM	(2K bytes)

System Memory Map

Note: When a RAM cartridge is added to less than 18K of built-in RAM, the cartridge overlays the memory starting at 1000_{16}.

The Register File

The register file contains the following groups of registers used in BASIC.

1. The A register
2. The B register
3. The assembly language subroutine stack
4. BASIC reserved (program pointer, current program character, etc.)
5. General purpose temporary registers (floating-point operations, I/O temporaries, etc.)

The general layout of the register file is shown below.

Address: Decimal	Hex	Description
0	0000	A Register
1	0001	B Register
2 57	0002 0039	Assembly language subroutine stack
58 74	003A 004A	BASIC statement temporaries
75 87	004B 0057	BASIC reserved area
88 126	0058 007E	General purpose temporaries
127	007F	Floating-point status

Register File

The Peripheral File

The TMS70C20 contains special instructions for performing I/O. These instructions access a particular section of the memory map called the peripheral file. This area contains several built-in peripheral registers such as the I/O control register, timer control registers, A and B ports, and the peripheral file expansion area. The general layout of the peripheral file is shown below.

Address: Decimal	Hex	Description
256	0100	I/O control register
257	0101	Reserved
258	0102	Timer data register
259	0103	Timer control register
260	0104	A port input data
261	0105	Reserved
262	0106	B port output data
263 271	0107 010F	unused
272	0110	Address control register
273	0111	Power on hold latch
274	0112	I/O bus-data
275	0113	I/O bus-bus available
276	0114	I/O bus-handshake ctl
277	0115	Piezo control
278	0116	Low battery sense line
279 280	0117 0118	unused
281	0119	Page control register
282	011A	Clock control register
283 511	011B 01FF	unused

Peripheral File

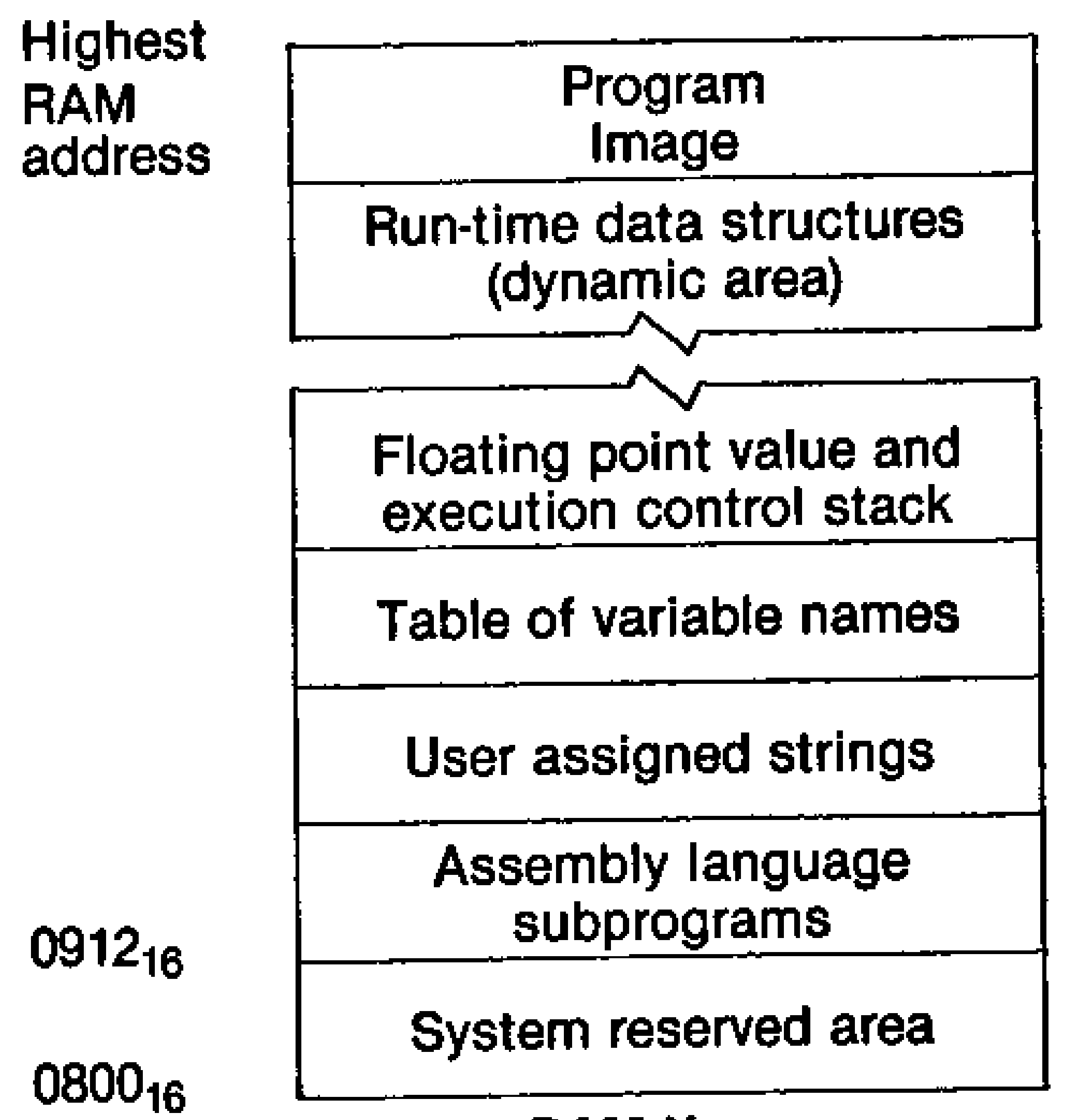
System RAM

RAM starts at 0800_{16} in the CC-40. The RAM can be increased by using a *Memory Expansion* cartridge. A minimum of 402 bytes of the memory is reserved by BASIC for the following.

- RAM-based trap vectors
- List pointers
- Random number seeds
- Permanent buffers (such as the keyboard input buffer)
- Other necessary information

The rest of memory is used to store the floating point stack, the dynamic area, any program in memory, and any user loaded assembly language subprograms. Each user-assigned string requires 1 byte plus the length of the string.

The use of RAM is outlined in the table below.



RAM Usage

BASIC Program Image

A BASIC program requires the following quantities of memory.

- Eleven bytes for overhead information.
- Four bytes of overhead for each program line.
- Two bytes of overhead plus the length of the variable name for each variable name. Each additional use of the same variable name requires only one byte.
- One byte of memory for each use of the following BASIC program elements.

  ABS, ACCEPT, ACS, ALL, AND, APPEND, ASC, ASN, AT, ATN, ATTACH, BEEP, CALL, CHR$, CLOSE, COS, DATA, DIM, DISPLAY, END, EOF, ERASE, EXP, FOR, FRE, GOSUB, GOTO, IF, IMAGE, INPUT, INT, INTERNAL, INTRND, KEY$, LEN, LET, LINPUT, LN, LOG, NEXT, NOT, NULL, NUMERIC, ON, OPEN, OR, OUTPUT, PAUSE, PI, POS, PRINT, RANDOMIZE, READ, REC, RELATIVE, RELEASE, REM, RESTORE, RETURN, RND, RPT$, SEG$, SGN, SIN, SIZE, SQR, STEP, STOP, STR$, SUB, SUBEND, SUBEXIT, TAB, TAN, THEN, TO, UPDATE, USING, VAL, VALIDATE, VARIABLE, XOR, statement separator :, comma ,, semicolon ;, left parenthesis (, right parenthesis), not-equal < >, less-than-or-equal < =, greater-than-or-equal > =, equal =, less-than <, greater-than >, concatenation &, addition or unary plus +, subtraction or unary minus −, multiplication *, division /, exponentiation ^, file number # .
- Two bytes of memory for each use of the following BASIC program elements.

  ALPHA, ALPHANUM, BREAK, CONTINUE (CON), DEG, DELETE (DEL), DIGIT, ELSE, ERROR, FORMAT, GRAD, LIST, NEW, NUMBER (NUM), OLD, PROTECTED, RAD, RENUMBER (REN), RUN, SAVE, UALPHA, UALPHANUM, UNBREAK, VERIFY, WARNING, tail remark ! .
- The number of bytes required for the following BASIC program elements are shown below.
  a. Three bytes for each line reference which appears in control transfer statements such as GOTO, GOSUB, ON GOTO, ON GOSUB, and IF THEN ELSE. Line references can also be in statements and commands such as ON ERROR, RESTORE, RUN, and DELETE.
  b. One byte of overhead plus two to eight bytes for each numeric constant. The number of bytes depends upon the number of significant digits in the floating-point representation of the constant. Trailing zeros are truncated from the normal representation to generate the program representation.

c. Two bytes of overhead plus the length of the string characters contained between the quotation marks for each quoted string constant. The length does not include the quotation marks. Within the quoted string two consecutive quotation marks count as a single quotation mark.

d. Two bytes of overhead plus the length of the string for each unquoted string constant. Leading and trailing spaces are ignored. Subprogram names in SUB, CALL, ATTACH, and RELEASE statements are unquoted strings. Unquoted strings also appear in REM, IMAGE, and DATA statements.

Run-time Data Structures

The following memory requirements are necessary to run a BASIC program. These structures are allocated dynamically during program execution.

- Four bytes of overhead for each variable in the main program. In addition the following memory is required for those variables.
  a. Eight bytes for each simple numeric variable.
  b. Two bytes of overhead for each dimension of each numeric array plus 8 bytes for each element value.
  c. Four bytes of overhead for the value of each simple string variable plus the length of the value. (**Exception:** If the variable is assigned a simple constant value in the program, the overhead for the value is reduced to 2 bytes. For example, A$="HELLO" requires 4 bytes of overhead for the variable A$ and 2 bytes of overhead for the value. A$="HELLO"&B$ requires 4 bytes of overhead for A$, plus 4 bytes of overhead for the value, plus the length of the value.)
  d. Two bytes of overhead for each dimension of a string array plus 4 bytes of overhead for each element value plus the length of each element value (see exception above).
- Eleven bytes of overhead for each BASIC subprogram plus 2 bytes for each variable (including arrays), plus 2 bytes for each dimension of each array. In addition each active and each attached subprogram has two bytes of overhead plus memory space for variables as described previously. If the subprogram is attached, the additional memory space remains allocated until the subprogram is released. Otherwise, the memory space is released when the subprogram terminates. See the CALL, ATTACH, and RELEASE statements.

- Twenty-one bytes of overhead for each open file or device plus the maximum record length specified in the OPEN statement. If record length is not specified in the OPEN statement, it is specified by the device when the OPEN statement is executed. This memory is released when the file or device is closed.
- Twenty-four bytes of the execution control stack during execution of each FOR NEXT loop.
- Eight bytes of the execution control stack during execution of the subroutine for each GOSUB or ON GOSUB.
- Sixteen bytes of the execution control stack during execution of the error handling subroutine for ON ERROR line-number.
- Twenty-four bytes of the execution control stack during execution of the subprogram for each BASIC subprogram CALL.
- Sixteen bytes of the execution control stack for an occurrence of a breakpoint until the program is continued or the capability to continue is destroyed.

Memory Expansion

The amount of memory added by a *Memory Expansion* cartridge depends upon the amount of resident memory and the size of the *Memory Expansion* cartridge. The table below shows the memory capacities resulting from adding a particular *Memory Expansion* cartridge to a specific amount of resident memory.

Cartridge Memory Size (K bytes)

Resident Memory (K bytes)		2	8	16
	2	4	10	18
	4	4	10	18
	6	4	10	18
	10	4	10	18
	18	20	26	34

System Command Level

The system command level of the BASIC interpreter is a loop which repetitively performs three phases of operation.

1. An input line is accepted from the keyboard and echoed in the display.
2. The input line is translated into an internal representation which can be processed by the execution level of the BASIC interpreter.
3. Based on the content of the input line and the key used to terminate the input, the command level determines how to use the input and processes it accordingly. After processing the current input line, the command level loops back to the input phase to accept another line from the user.

The key to the proper functioning of the command level is the decision concerning how to use the input line. This process begins in the line compression routine which translates the input line into its corresponding internal representation. This routine decides whether or not the input is a BASIC program line. If the input line begins with a valid line number (an integer from 1 to 32766), followed by one or more spaces, followed by an alphabetic character, the at sign, the underline, or tail remark symbol (!), it is translated as a program line. Otherwise, the input is translated as a statement (or command) for immediate execution or an equation for immediate calculation.

BASIC program lines are edited into the current BASIC program in memory. If the current program is not a BASIC or a protected program, an error occurs. If the input is not a program line, the command level must decide whether it is a statement/command or an equation. A statement/command is executed immediately as if it were a one line program. An equation is evaluated and the result is displayed left justified in the display. Other calculations may be appended to the result using the +, −, *, /, and ^ operators. However, if a new equation or statement/command is input, the result is automatically cleared before the input is accepted.

An input line must be one of the following to be a statement/command.

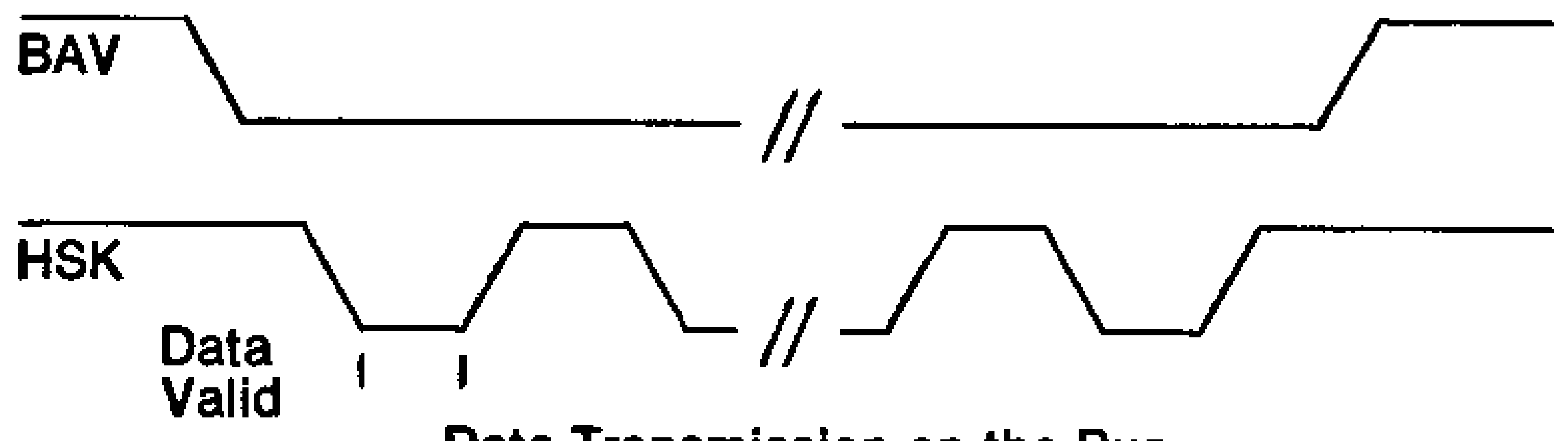
1. The line begins with a statement/command keyword.
2. The line begins with a variable name and contains one equal sign (=) which is *not* the last character of the line.
3. The line begins with a variable name and contains more than one equal sign.

All other input lines are considered to be equations.

The HEX-BUS™ Intelligent Peripheral Interface

The *HEX-BUS*™ interface is a 4-bit, medium-speed I/O bus. The CC-40 can transfer data at speeds up to 6000 bytes/second over the *HEX-BUS* interface. The bus communicates over an eight-line cable which consists of four data lines, two bus control lines, a ground line, and one line that is reserved for future expansion of the bus.

Transfers on the bus are controlled by two lines. The first, Bus AVailable (BAV), controls the start and finish of a transmission. BAV is pulled low to indicate that a command is about to be transmitted over the bus. The second line, HandShaKe (HSK), indicates that a nibble is available on the bus when it is low.



Data Transmission on the Bus

Each transmission consists of the bus master (CC-40) transmitting a command to a particular device along with any required data, and the device transmitting a response with any returned data and the status of the operation.

Error Messages

The following lists describe the cause of each error message generated by the CC-40. The first list, arranged alphabetically by message, provides detailed information about the probable cause of each error. The second list, arranged in ascending order by error code, serves as a cross reference to locate the message associated with a particular error code.

When an error message is displayed, the →, ←, ↑, ↓, and **[SHIFT] [PB]** keys can be used to display additional system error information and to edit an erroneous line.

[SHIFT] [PB] is used when an error occurs after a line is entered. **[SHIFT] [PB]** displays the erroneous entry which can then be edited and entered again

→ is used when an error occurs during program execution. → displays the error code and the line number of the line being executed (when the error occurred) in an `Enn Lmmmmm` format where `nn` is the error code and `mmmmm` is the line number. (This line is not necessarily the one that is the source of the problem since an error may occur because of values generated or actions taken elsewhere in the program.)

When an I/O error occurs, → displays either the error code, file number, and line number in an `EO, xxx #yyy mmmmm` format or the error code, device number, and line number in an `EO, xxx "yyy" mmmmm` format. `xxx` is the I/O error code, `#yyy` is the file number or `"yyy"` is the device number, and `mmmmm` is the line number of the line that was executing when the error occurred.

← can be used to redisplay the error message immediately after the → key has been pressed.

↑ or ↓ is used when an error occurs during program execution to display the program line that was executing when the error occurred.

Errors can be handled in a program using ON ERROR and CALL ERR. Refer to chapters 4 and 5 for more information.

Messages Listed Alphabetically

Code Message/Cause

29 `Bad argument`
- Invalid argument provided for one of the built-in numeric, string, or file functions such as LOG, CHR$, and EOF.
- Invalid argument provided for one of the option clauses in an input/output statement such as AT, SIZE, VALIDATE, and TAB.
- Arguments in a CALL statement did not match the requirements for the subprogram called.

07 `Bad INPUT data`
- Entered more than one value at a time in an INPUT or ACCEPT statement.
- Invalid data from a file in an INPUT or LINPUT statement.

17 `Bad line number`
- Line number specified in a statement or command was less than 1 or greater than 32766.
- RENUMBER command generated a line number greater than 32766.

18 `Bad program type`
- Entered a BASIC program line with an assembly language or other non-BASIC program in memory.
- Entered a SAVE, VERIFY, BREAK *line-list*, UNBREAK *line-list*, NUMBER, RENUMBER, LIST, CONTINUE *line-number*, RUN *line-number*, or DELETE *line-group* command with an assembly language or other non-BASIC program in memory.
- Attempted to CALL a main program or RUN a subprogram.
- Attempted to ATTACH a main program or an assembly language subprogram.
- File specified for LOAD subprogram did not contain a relocatable, assembly language subprogram.

32 `Bad subscript`
- Subscript value too large.
- Missing comma between subscripts or missing parentheses around subscripts.
- Incorrect number of subscripts.

(continued)

(continued)

Code Message/Cause

04 `Bad value`

- Index value in ON GOTO or ON GOSUB statement was zero or greater than the number of line number entries.
- Raised a negative value to a non-integer power.
- Invalid value provided for one of the option clauses in an input/output statement such as AT, SIZE, REC, and VARIABLE.
- Attempted a logical operation (AND, OR, XOR, or NOT) with a value less than −32768 or greater than 32767.

31 `BASIC extension missing`

- Attempted to execute an extended BASIC statement or function without the extension in the system.
- May also occur when the contents of memory have been improperly modified (see System error).

37 `Break`

- A breakpoint occurred or the break key was pressed.

10 `Can't do that`

- Attempted to perform a string operation as an immediate calculation.
- Entered CONTINUE command when not stopped at a breakpoint.
- A SUBEXIT or SUBEND statement was encountered when no subprogram was called. For example, CONTINUE *line-number* specified a line in a subprogram after the main program stopped at a breakpoint.

43 `DATA error`

- Out of data in the current program or subprogram.
- Improper data list in a DATA statement. For example, items not separated by commas.
- During an attempt to read a numeric item, the data read was not a valid representation of a numeric constant.

34 `Division by zero`

- Evaluation of a numeric expression includes division by zero; result is replaced by 9.9999999999999E+127 with the appropriate algebraic sign.

(continued)

(continued)

Code Message/Cause

23 `Error in image`

- Null string provided as image string.
- Numeric format field specified more than 14 significant digits.
- *Print-list* included a print-item but image string had only literal characters.

02 `Expression too complex`

- Too many functions, operators, or levels of parentheses pending evaluation; expression must be simplified or must be performed in two or more steps in separate statements.

24 `File error`

- *File-number* specified in an OPEN statement refers to a file already opened.
- *File-number* in an input/output statement, other than OPEN, did not refer to an open file.
- *File-number* or *device* number in an input/output statement was greater than 255.
- Attempted to INPUT or LINPUT from a file opened in OUTPUT or APPEND mode.
- Attempted to LINPUT from an internal-type file.
- Attempted to PRINT to a file opened in INPUT mode.
- Used REC clause in an input/output statement which accessed a sequential file.
- Missing period or comma after device number in *device* or *filename* specification.

30 `FOR without NEXT`

- More FOR statements than NEXT statements in a program or subprogram. **Note:** the line number reported is the last line of the current program or subprogram, not the line containing the unmatched FOR statement.

11 `Illegal after SUBEND`

- Statement other than REM, !, END, or SUB used after a SUBEND statement.

(continued)

(continued)

Code Message/Cause

13 `Illegal FOR-NEXT nesting`

- Too many levels of nested FOR NEXT loops.
- Same control variable used in nested FOR NEXT loops.

19 `Illegal in program`

- Used CALL ADDMEM, CALL CLEANUP, CONTINUE, DELETE *line-group*, LIST, NEW, NUMBER, OLD, RENUMBER, SAVE, or VERIFY in a program.

01 `Illegal syntax`

- Missing parentheses or quotation mark(s).
- Missing statement separator (:) or tail remark symbol (!).
- Missing or extra comma(s). For example:
  - –between arguments in *argument-list*
  - –between line numbers in *line-number-list*
  - –between variables in *variable-list*
  - –after *file-number* in input/output statements
- Missing hyphen in line sequence.
- Missing argument or clause. For example:
  - –no limit value after TO or increment value after STEP
  - –no line number or statement after THEN or ELSE
  - –no *string-constant* following IMAGE
  - –no *line-number* or *string-expression* after USING
  - –no value before or no value after a binary operator such as *, /, ^, or &
  - –no input variable following INPUT, LINPUT, ACCEPT, or READ
- Invalid argument or clause. For example:
  - –a string variable is used as *control-variable* in FOR
  - –a numeric variable is used as input variable in LINPUT
  - –VALIDATE or NULL is used in a DISPLAY statement
  - –USING or TAB is used with an internal-type file
  - –the size of print item exceeds record size for an internal-type file

(continued)

(continued)

Code Message/Cause

- Missing keyword. For example:
  - –no TO after FOR
  - –no THEN after IF
  - –no GOTO or GOSUB after ON *numeric-expression*
  - –no STOP, NEXT, or ERROR after ON BREAK
  - –no PRINT, NEXT, or ERROR after ON WARNING
- Improperly placed keyword. For example:
  - –DIM or SUBEND is used after a DIM statement in a multiple statement line
  - –a statement begins with a non-statement keyword such as TO, ERROR, VARIABLE, SIZE
  - –a misspelled variable results in a keyword or a misspelled keyword in a variable
  - –a keyword is used as a variable, such as ON VAL GOTO or IF STOP=1 THEN
- Duplicated option in input/output statement. For example:
  - –more than one AT, SIZE, ERASE ALL is in ACCEPT or DISPLAY
  - –more than one string expression is in VALIDATE
  - –more than one *open-mode*, *file-type*, *file-organization* is in OPEN
- Missing or invalid *filename* in OLD, SAVE, VERIFY, or DELETE file command.
- Invalid character in statement. For example "%", "?", ",", "O", "C", etc., are valid only within quoted strings or in an IMAGE or REM statement.
- Invalid character within a numeric constant.

08 `Invalid dimension`

- Specified array dimension was negative or was not a numeric constant.
- Too many elements specified for an array.
- More than three dimensions specified for an array
- Missing comma between dimensions or missing parentheses around dimensions of an array.

(continued)

(continued)

Code	Message/Cause
00	`I/O error` • An error was returned by a peripheral device during an input/output (I/O) statement or command, or while using the EOF function. A special I/O code is returned by the device and is displayed after the message. Common I/O error codes are described in the I/O ERROR CODES section of this appendix. The error code is followed by the *file-number* or the *device* number, whichever is appropriate to the statement or command being executed. A number sign indicates a *file-number* and quotation marks indicate a *device* number. Both the common codes and other device-dependent I/O error codes are described in the peripheral manuals.
16	`Line not found` • Could not find a line number specified in BREAK, CONTINUE, DELETE, GOSUB, GOTO, ON ERROR, USING, RESTORE, RUN, or BREAK. • RENUMBER could not find a referenced line. The command replaced the reference by 32767, which is not a valid line number.
12	`Line reference out of range` • BASIC statement referred to a line number which was lower than the first (or higher than the last) line number of the current program or subprogram.
27	`Line too long` • The internal representation of a program line or immediate statement(s) was too long. • The LIST representation of a program line exceeded 80 characters.
35	`Memory contents may be lost` • When the power was turned on, the computer determined that the contents of the constant memory were not the same as when the power was turned off. However, some system data was correct, so the loss may or may not be serious. This message often appears when the reset key is pressed while the power is on.

(continued)

(continued)

Code	Message/Cause
127	`Memory full` • Insufficient space to add, insert, or edit a program line. • Insufficient space to allocate variables for a program or subprogram. • Insufficient memory to allocate space for a string value. • Insufficient space to load a program or subprogram into memory. • Insufficient space to OPEN a file or device. • Insufficient space to assign a user-assigned string. • Attempted to allocate more than the largest available block of memory using the GETMEM subprogram.
14	`Missing RETURN from error` • An error processing subroutine terminated with a SUBEXIT or SUBEND statement instead of a RETURN statement.
42	`Missing SUBEND` • SUBEND missing in a subprogram. • Encountered a SUB statement within a subprogram; a subprogram cannot contain another subprogram.
44	`Must be in program` • ACCEPT, CALL with BASIC language subprograms, GOSUB, GOTO, INPUT, LINPUT, ON ERROR *line-number*, ON GOSUB, ON GOTO, READ, RESTORE *line-number*, SUB, SUBEXIT, and SUBEND statements can be executed only in a program.
39	`Must be in subprogram` • SUBEXIT or SUBEND statement encountered in a main program.
25	`Name table full` • Defined more than 95 variable names. The CLEANUP subprogram can be used to delete all variable names not used in the current program in memory.
28	`Name too long` • More than 15 characters in a variable or subprogram name.

(continued)

(continued)

Code	Message/Cause
06	`NEXT without FOR` • More NEXT statements than FOR statements in a program or subprogram. • *Control-variable* in NEXT statement did not match *control-variable* in corresponding FOR statement. • Executed a NEXT statement without previously executing the corresponding FOR statement.
40	`No RAM in cartridge` • Called ADDMEM subprogram with no cartridge installed or with a cartridge which did not contain RAM memory.
33	`Overflow` • A numeric value was entered or a numeric expression was evaluated which resulted in a number whose absolute value was greater than 9.9999999999999E+127; the value is replaced by 9.9999999999999E+127 with the appropriate algebraic sign.
15	`Program not found` • RUN statement did not find the specified program. • CALL statement did not find the specified subprogram.
20	`Protection violation` • Attempted to insert, delete, or edit a line with a protected program in memory. • Attempted to LIST, SAVE, NUMBER, or RENUMBER a protected program.
45	`RETURN without GOSUB` • Executed a RETURN statement without previously executing the corresponding GOSUB statement.
05	`Stack underflow` • Attempted to remove a value from the execution control stack when it was empty. This error only occurs when the contents of memory have been improperly modified (see System error).
41	`Statement must be first on line` • SUB statement used after the first statement in a multiple statement line.

(continued)

(continued)

Code	Message/Cause
03	`String-number mismatch` • Used a string argument where a numeric argument was expected or a numeric argument where a string argument was expected. • Assigned a string value to a numeric variable or a numeric value to string variable. • A numeric variable or expression was provided as a prompt in an INPUT or LINPUT statement.
36	`String truncation` • String operation (concatenation or RPT$) resulted in a string with more than 255 characters; the extra characters are discarded.
21	`Subprogram in use` • Called an active subprogram; subprograms may not call themselves, directly or indirectly.
126	`System error` • This error generally occurs when the contents of memory have been lost or improperly modified. For example, memory may be modified by a loss of power or by improper use of the POKE, RELMEM, EXEC, or DEBUG subprogram(s).
38	`System initialized` • Displayed when circumstances force the complete initialization of the system. The system is initialized when the power is turned on and one of the following occurs. –The computer determines that the contents of memory have been destroyed (may occur after changing the batteries). –The computer determines that previously appended expansion RAM (through ADDMEM subprogram) is no longer in the system. • The message may also appear when the reset button is pressed because much of the same memory checking is performed,. (The system initialization procedure is described in appendix G.)

(continued)

(continued)

Code	Message/Cause
26	Unmatched parenthesis • A statement or expression did not contain the same number of left and right parentheses. • Left and right parentheses in a statement or expression did not match up. For example, SIN(1+)PI/2)(where SIN(1+(PI/2)) was intended.
22	Variable not defined • Attempted to perform a calculation with a variable which has not been defined. • Encountered an undefined variable in a program or subprogram. This error can occur when CONTINUE *line-number* specifies a line which is not in the same program or subprogram where the breakpoint occurred.
09	Variable previously defined • Variable in a DIM statement appeared previously in the current program or subprogram. • Variable referenced using the wrong number of dimensions. For example, a variable was first used as a simple variable and later used as an array in the same program or subprogram.

Error Codes List in Ascending Order

Code	Message
00	I/O error
01	Illegal syntax
02	Expression too complex
03	String-number mismatch
04	Bad value
05	Stack underflow
06	NEXT without FOR
07	Bad INPUT data
08	Invalid dimension
09	Variable previously defined
10	Can't do that
11	Illegal after SUBEND
12	Line reference out of range
13	Illegal FOR-NEXT nesting
14	Missing RETURN from error
15	Program not found

(continued)

(continued)

Code	Message
16	Line not found
17	Bad line number
18	Bad program type
19	Illegal in program
20	Protection violation
21	Subprogram in use
22	Variable not defined
23	Error in image
24	File error
25	Name table full
26	Unmatched parenthesis
27	Line too long
28	Name too long
29	Bad argument
30	FOR without NEXT
31	BASIC extension missing
32	Bad subscript
33	Overflow
34	Division by zero
35	Memory contents may be lost
36	String truncation
37	Break
38	System initialized
39	Must be in subprogram
40	No RAM in cartridge
41	Statement must be first on line
42	Missing SUBEND
43	DATA error
44	Must be in program
45	RETURN without GOSUB
126	System error
127	Memory full

I/O ERROR CODES

The following list details the standard input/output (I/O) error codes. Some peripherals may have additional error codes; if so, they are explained in the peripheral manual.

I/O errors are displayed in one of the following forms.

- `I/O error ccc #fff`
- `I/O error ccc "ddd"`

where `ccc` is the I/O error code listed below or in the peripheral manual, `fff` is the file number assigned in an OPEN statement, and `ddd` is the device code associated with the peripheral device.

Code **Definition**

1 DEVICE/FILE OPTIONS ERROR
- Incorrect or invalid option specified in "*device.filename*".
- *Filename* too long or missing in "*device.filename*".

2 ERROR IN ATTRIBUTES
- In an OPEN statement, incorrect attributes (*file-type*, *file-organization*, *open-mode*, *record-length*) were specified for an existing file.

3 FILE NOT FOUND
- The file specified in one of the following operations does not exist.
  - –OPEN statement using the INPUT attribute
  - –OLD "*device.filename*"
  - –RUN "*device.filename*"
  - –DELETE "*device.filename*"
  - –CALL LOAD("*device.filename*")

4 DEVICE/FILE NOT OPEN
- Attempted to access a closed file with a INPUT, LINPUT, PRINT, or CLOSE operation.
- File specified in EOF function is closed.

5 DEVICE/FILE ALREADY OPEN
- Attempted to OPEN or DELETE an open file.
- Attempted to FORMAT storage medium on a device which has a file open.

(continued)

(continued)

Code **Definition**

6 DEVICE ERROR
- A failure has occurred in the peripheral. This error can occur when directory information on a tape was lost, the peripheral detected a transmission error or a medium failure, etc.

7 END OF FILE
- Attempted to read past the end of the file.

8 DATA/FILE TOO LONG
- Attempted to output a record which was longer than the capacity of the device.
- A file exceeded the maximum file length for a device.

9 WRITE PROTECT ERROR
- Attempted to FORMAT a write-protected storage medium.
- Attempted to OPEN a write-protected file in OUTPUT or UPDATE mode.
- Attempted to DELETE a file from a write-protected medium.

10 NOT REQUESTING SERVICE
- Response to a service request poll when the specified device did not request service. (This code is used in special applications and should not be encountered during normal execution of BASIC programs.)

11 DIRECTORY FULL
- Attempted to OPEN a new file on a device whose directory is full.

12 BUFFER SIZE ERROR
- When an existing file was opened for input or update, the specified record length (VARIABLE XXX) was less than the length of the largest record in the existing file.
- The VERIFY command found the program in memory was smaller than the program on the storage medium.

13 UNSUPPORTED COMMAND
- Attempted an operation not supported by the peripheral.

(continued)

(continued)

Code	Definition
14	DEVICE/FILE NOT OPENED FOR OUTPUT • Attempted to write to a file or device opened for input.
15	DEVICE/FILE NOT OPENED FOR INPUT • Attempted to read from a file or device opened for output or append.
16	CHECKSUM ERROR • The checksum calculated on the input record was incorrect.
17	RELATIVE FILES NOT SUPPORTED • Device specified in OPEN does not support relative record file organization.
19	APPEND MODE NOT SUPPORTED • Device specified in OPEN statement does not support append mode.
20	OUTPUT MODE NOT SUPPORTED • Device specified in OPEN statement does not support output mode.
21	INPUT MODE NOT SUPPORTED • Device specified in OPEN statement does not support input mode.
22	UPDATE MODE NOT SUPPORTED • Device specified in OPEN statement does not support update mode.
23	FILE TYPE ERROR • File type specified in OPEN statement is not supported by the specified device. • File type specified in OPEN statement does not match file type of existing file or device.
24	VERIFY ERROR • Program or data in memory does not match specified program or storage medium.
25	LOW BATTERIES IN PERIPHERAL • Attempted an I/O operation with a device whose batteries are low.

(continued)

(continued)

Code	Definition
26	UNINITIALIZED MEDIUM • Attempted to open a file on uninitialized storage medium. • Attempted to open a file on storage medium which has been accidentally erased or destroyed.
32	MEDIUM FULL • No available space on storage medium.
254	ILLEGAL IN SLAVE MODE • Attempted a normal (master) I/O bus operation while the computer was in peripheral (slave) mode. (This error occurs during some special applications and should not be encountered during normal execution of a BASIC program.) • **Note:** Improper modification of memory by the POKE, RELMEM, EXEC, or DEBUG subprograms can result in the computer being placed in peripheral (slave) mode.
255	TIME-OUT ERROR • Lost communication with the specified device. • Specified device is not connected to the I/O bus.

In Case of Difficulty

In the event that you have difficulty with your Compact Computer, the following instructions may help you diagnose and remedy the problem. Usually you can correct the problem without returning the unit to a service facility. If the suggested remedies are not successful, contact Texas Instruments' Consumer Relations Department by mail or telephone as described later in the section IF YOU HAVE QUESTIONS OR NEED ASSISTANCE.

Note: All peripherals attached to the CC-40 should be turned on for proper operation.

If one of the following symptoms appears, try the suggested remedy. If you are operating your computer with peripheral devices and the remedy does not correct the problem, remove the peripherals. If the symptom disappears, a peripheral is the most likely source of the difficulty. Refer to the appropriate peripheral or accessory manual for more information on the cause of the problem.

Symptom	Remedy/Cause
No display	Check that power is on. Move the display contrast control to see if the display becomes visible. If there is still no display, replace the batteries with fresh AA alkaline batteries.
No flashing cursor	Check the I/O display indicator to see if any I/O operations are in progress. If the indicator is on, wait for all peripheral activity to cease. If the indicator is still on several minutes later, disconnect the *HEX-BUS* interface cable from the computer. Then press the reset key. If the I/O indicator is not on, the system may be locked up. Press the **[BREAK]** key to try to halt the computer. If the word BREAK appears in the display, enter **CON** to continue executing the program in memory.

(continued)

(continued)

Symptom	Remedy/Cause
No flashing cursor	If the **[BREAK]** key is inoperable, press the reset key. The message `Memory contents may be lost` should be displayed. Press the **[CLR]** key to clear the display. You can check if your program is still in memory by entering **LIST**. If pressing the reset key does not cause the cursor to reappear, the batteries should be removed. Normally, the system is then initialized and any program in memory erased.

Returning Your Computer

When returning your Compact Computer for repair or replacement, also return any software cartridges that were being used when the difficulty occurred. For your protection, the CC-40 should be sent insured; Texas Instruments cannot assume any responsibility for loss of or damage to the CC-40 during shipment. It is recommended that the CC-40 be shipped in its original container to minimize the possibility of shipping damage. Otherwise, the CC-40 should be carefully packaged and adequately protected against shock and rough handling. Send shipments to the appropriate Texas Instruments Service Facility listed in the warranty. Please include information on the difficulty experienced with your computer as well as return address information including name, address, city, state, and zip code.

If the CC-40 is in warranty, it will be repaired or replaced under the terms of the Limited Warranty. Out-of-warranty units in need of service will be repaired or replaced with reconditioned units (at TI's option), and service rates in effect at the time will be charged. Because our Service Facility serves the entire United States, it is not feasible to hold units while providing service estimates. For advance information concerning our flat-rate service charges, please call our toll-free telephone number (800) 858-4565.

Exchange Centers

If your Compact Computer requires service and you do not wish to return the unit to your dealer or to a service facility for repair, you may elect to exchange the computer for a factory-reconditioned computer of the same model (or equivalent model specified by TI) by taking the computer to one of the exchange centers which have been established across the United States.

A handling fee will be charged by the exchange center for in-warranty exchange. Out-of-warranty exchanges will be charged at the rates in effect at the time of the exchange. To determine if there is an exchange center in your area, look for Texas Instruments Incorporated Exchange Center in the white pages of your telephone directory or look under the Calculating and Adding Machines and Supplies heading in the yellow pages. Please call the exchange center for the availability of your model. You can write or call Texas Instruments Consumer Relations Department for more information.

If You Have Questions or Need Assistance

For General Information

If you have questions concerning Compact Computer repair or peripheral, accessory, or software purchase, please call our Customer Relations Department at (800) 858-4565 (toll free within the contiguous United States). The operators at these numbers cannot provide technical assistance.

For Technical Assistance

For technical questions such as programming, specific computer applications, etc., you can call (806) 741-2663. We regret that this is not a toll-free number, and we cannot accept collect calls. As an alternative, you can write to:

Texas Instruments Consumer Relations
P.O. Box 53
Lubbock, Texas 79408

Because of the number of suggestions which come to Texas Instruments from many sources, containing both new and old ideas, Texas Instruments will consider such suggestions only if they are freely given to Texas Instruments. It is the policy of Texas Instruments to refuse to receive any suggestions in confidence. Therefore, if you wish to share your suggestions with Texas Instruments, or if you wish us to review any computer program which you have developed, please include the following in your letter:

"All of the information forwarded herewith is presented to Texas Instruments on a nonconfidential, nonobligatory basis; no relationship, confidential or otherwise, expressed or implied, is established with Texas Instruments by this presentation. Texas Instruments may use, copyright, distribute, publish, reproduce, or dispose of the information in any way without compensation to me."

90-Day Limited Warranty

THIS TEXAS INSTRUMENTS COMPACT COMPUTER WARRANTY EXTENDS TO THE ORIGINAL CONSUMER PURCHASER OF THE COMPUTER.

Warranty Duration:

This computer is warranted to the original consumer purchaser for a period of 90 days from the original purchase date.

Warranty Coverage:

This computer is warranted against defective materials or workmanship. **THIS WARRANTY DOES NOT COVER BATTERIES AND IS VOID IF THE PRODUCT HAS BEEN DAMAGED BY ACCIDENT, UNREASONABLE USE, NEGLECT, IMPROPER SERVICE OR OTHER CAUSE NOT ARISING OUT OF DEFECTS IN MATERIAL OR WORKMANSHIP.**

Warranty Disclaimers:

ANY IMPLIED WARRANTIES ARISING OUT OF THIS SALE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO THE ABOVE 90-DAY PERIOD. TEXAS INSTRUMENTS SHALL NOT BE LIABLE FOR LOSS OF USE OF THE COMPUTER OR OTHER INCIDENTAL OR CONSEQUENTIAL COSTS, EXPENSES, OR DAMAGES INCURRED BY THE CONSUMER OR ANY OTHER USER.

Some states do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusions may not apply to you.

Legal Remedies:

This warranty gives you specific legal rights, and you may also have other rights that vary from state to state.

Warranty Performance:

Please contact the retailer from whom you purchased the computer and determine the exchange policies of the retailer.

During the above 90-day warranty period, your TI Compact Computer will be repaired or replaced with a new or reconditioned comparable model (at TI's option) when the computer is returned either in person or by prepaid shipment to a Texas Instruments Service Facility listed below.

Texas Instruments strongly recommends that you insure the computer for value, prior to shipment.

The repaired or replacement computer will be warranted for 90 days from date of repair or replacement. Other than the cost of postage or shipping to Texas Instruments, no charge will be made for the repair or replacement of in-warranty computers.

Texas Instruments Consumer Service Facilities

U.S. Residents:
Texas Instruments Service Facility
2303 North University
Lubbock, Texas 79415

Canadian Customers Only:
Geophysical Services Incorporated
41 Shelley Road
Richmond Hill,
Ontario, Canada L4C5G4

Consumers in California and Oregon may contact the following Texas Instruments offices for additional assistance or information.

Texas Instruments Consumer Service
831 South Douglas Street
El Segundo, California 90245
(213) 973-1803

Texas Instruments Consumer Service
6700 Southwest 105th St.
Kristin Square
Suite 110
Beaverton, Oregon 97005
(503) 643-6758

Important Notice Regarding Programs and Book Materials

The following should be read and understood *before* purchasing and/or using TI's Compact Computer.

TI does not warrant that the programs contained in this computer and accompanying book materials will meet the specific requirements of the consumer, or that the programs and book materials will be free from error. The consumer assumes complete responsibility for any decision made or actions taken based on information obtained using these programs and book materials. Any statements made concerning the utility of TI's programs and book materials are not to be construed as express or implied warranties.

TEXAS INSTRUMENTS MAKES NO WARRANTY, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THESE PROGRAMS OR BOOK MATERIALS OR ANY PROGRAMS DERIVED THEREFROM AND MAKES SUCH MATERIALS AVAILABLE SOLELY ON AN "AS IS" BASIS.

IN NO EVENT SHALL TEXAS INSTRUMENTS BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE PURCHASE OR USE OF THESE PROGRAMS OR BOOK MATERIALS, AND THE SOLE AND EXCLUSIVE LIABILITY OF TEXAS INSTRUMENTS, REGARDLESS OF THE FORM OF ACTION, SHALL NOT EXCEED THE PURCHASE PRICE OF THIS COMPACT COMPUTER. MOREOVER, TEXAS INSTRUMENTS SHALL NOT BE LIABLE FOR ANY CLAIM OF ANY KIND WHATSOEVER AGAINST THE USER OF THESE PROGRAMS OR BOOK MATERIALS BY ANY OTHER PARTY.

Some states do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusions may not apply to you.

INDEX

INDEX