APPENDIX A

ASCII CHARACTER CODES

The 5 tables on the following pages list the American Standard Code for Information Interchange (ASCII) codes for the 256 characters in TI Extended BASIC II. The "ASCII CODE" column denotes the ASCII code for the characters. The "DISPLAY CHARACTER" column shows the character as it is displayed on the screen.

In tables 1, 3, 4, and 5, the "KEY-UNIT" column(s) shows the key pressed to generate the respective ASCII code. The key(s) listed depends upon the key-unit generated by the current, or last, CALL KEY statement. The valid ranges for the five key-units are listed below.

Key-unit	ASCII Code Range
0 (Uses last specified key-unit)	
1 (Split-left)	0-19
2 (Split-right)	0-19
3 (TI-99/4 Emulator)	1-15, 32-95
4 (Pascal)	0-127, 129-143, 176-198
5 (BASIC)	~~32-127, 128-159, 176-198~~ 1-15, 32-159, 176-198

Table 1:4ASCII Codes 0-31

ASCII		DISPLAY	KEY UNIT				
CODE	MNEMONIC	CHARACTER	1	2	3	4	5
0	NUL	(Space)	X	M		CTRL ,	
1	SOH	(Space)	A	H	FCTN 7	CTRL A	FCTN 7
2	STX	(Space)	S	J	FCTN 4	CTRL B	FCTN 4
3	ETX	(Space)	D	C	FCTN 1	CTRL C	FCTN 1
4	EOT	(Space)	W	U	FCTN 2	CTRL D	FCTN 2
5	ENQ	(Space)	E	I	FCTN =	CTRL E	FCTN =
6	ACK	(Space)	R	O	FCTN 8	CTRL F	FCTN 8
7	BEL	(Space)	2	7	FCTN 3	CTRL G	FCTN 3
8	BS	(Space)	3	8	FCTN S	CTRL H	FCTN S
9	HT	(Space)	4	9	FCTN D	CTRL I	FCTN D
10	LF	(Space)	5	0	FCTN X	CTRL J	FCTN X
11	VT	(Space)	T	P	FCTN E	CTRL K	FCTN E
12	FF	(Space)	F	L	FCTN 6	CTRL L	FCTN 6
13	CR	(Space)	V	.	ENTER	CTRL M	ENTER
14	SO	(Space)	C	,	FCTN 5	CTRL N	FCTN 5
15	SI	(Space)	Z	N	FCTN 9	CTRL O	FCTN 9
16	DLE	(Space)	B	/		CTRL P	
17	DC1	(Space)	G	;		CTRL Q	
18	DC2	(Space)	Q	Y		CTRL R	
19	DC3	(Space)	1	6		CTRL S	
20	DC4	(Space)				CTRL T	
21	NAK	(Space)				CTRL U	
22	SYN	(Space)				CTRL V	
23	ETB	(Space)				CTRL W	
24	CAN	(Space)				CTRL X	
25	EM	(Space)				CTRL Y	
26	SUB	(Space)				CTRL Z	
27	ESC	(Space)				CTRL .	
28	FS	(Space)				CTRL ;	
29	GS	(Space)				CTRL =	
30	RS	(Cursor)				CTRL 8	
31	US	(Space)				CTRL 9	

The "MNEMONIC" column is generally relevant when using the CHR$ function to generate ASCII codes to be output to an external device, such as a printer or modem.

The default characters on the TI Computer 99/8 are the standard ASCII characters for codes 32 through 127. The following table lists these characters and their codes.

The ASCII codes in the following table are accessible from key-units 4 and 5. Only ASCII codes 32-95 can be accessed from key-unit 3.

Table 2: ASCII Codes 32-127

ASCII	DISPLAY	ASCII	DISPLAY	ASCII	DISPLAY
CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER
32	(space)	65	A	97	a
33	! (exclamation point)	66	B	98	b
[illegible] mark 34	" (~~quote~~)	67	C	99	c
35	# (number or pound sign)	68	D	100	d
36	$ (dollar)	69	E	101	e
37	% (percent)	70	F	102	f
38	& (ampersand)	71	G	103	g
39	' (apostrophe)	72	H	104	h
40	((open parenthesis)	73	I	105	i
41	) (close parenthesis)	74	J	106	j
42	* (asterisk)	75	K	107	k
43	+ (plus)	76	L	108	l
44	, (comma)	77	M	109	m
45	- (minus, hyphen)	78	N	110	n
46	. (period)	79	O	111	o
47	/ (slant)	80	P	112	p
48	0	81	Q	113	q
49	1	82	R	114	r
50	2	83	S	115	s
51	3	84	T	116	t
52	4	85	U	117	u
53	5	86	V	118	v
54	6	87	W	119	w
55	7	88	X	120	x
56	8	89	Y	121	y
57	9	90	Z	122	z
58	: (colon)	91	[(open bracket)	123	;§(left brace)
59	; (semicolon)	92	(reverse slant)	124	| XX VERT BAR
60	(less than)	93	](close bracket)	125	§;(right brace)
61	= (equals)	94	^(caret)	126	(tilde)
62	| (greater than)	95	_(underline)	127	DEL(appears as a blank)
63	? (question mark)	96	` (grave)		
64	@ (at sign)				

The cursor is assigned to ASCII code 30. Character codes 128-223 are defined, respectively, to be the same as characters 32-127.

Table 3: ASCII Codes 128-160

ASCII CODE	DISPLAY CHARACTER	KEY-UNIT 4	KEY-UNIT 5
128	(Space)		CTRL ,
129	! (exclamation point)	FCTN 7	CTRL A
tion 130	" (~~quote~~)	FCTN 4	CTRL B
ark 131	# (number or pound sign)	FCTN 1	CTRL C
132	$ (dollar)	FCTN 2	CTRL D
133	% (percent)	FCTN =	CTRL E
134	& (ampersand)	FCTN 8	CTRL F
135	' (apostrophe)	FCTN 3	CTRL G
136	((open parenthesis)	FCTN S	CTRL H
137	) (close parenthesis)	FCTN D	CTRL I
138	* (asterisk)	FCTN X	CTRL J
139	+ (plus)	FCTN E	CTRL K
140	, (comma)	FCTN 6	CTRL L
141	- (minus, hyphen)	~~FCTN ENTER~~	CTRL M
142	. (period)	FCTN 5	CTRL N
143	/ (slant)	FCTN 9	CTRL O
144	0		CTRL P
145	1		CTRL Q
146	2		CTRL R
147	3		CTRL S
148	4		CTRL T
149	5		CTRL U
150	6		CTRL V
151	7		CTRL W
152	8		CTRL X
153	9		CTRL Y
154	: (colon)		CTRL Z
155	; (semicolon)		CTRL .
156	(less than)		CTRL ;
157	= (equals)		CTRL =
158	| (greater than)		CTRL 8
159	? (question mark)		CTRL 9
160	@ (at sign)		

Table 4: ASCII Codes 161-198

ASCII CODE	DISPLAY CHARACTER	KEY-UNITS 4 and 5
161	A	
162	B	
163	C	
164	D	
165	E	
166	F	
167	G	
168	H	
169	I	
170	J	
171	K	
172	L	
173	M	
174	N	
175	O	
176	P	CTRL 0
177	Q	CTRL 1
178	R	CTRL 2
179	S	CTRL 3
180	T	CTRL 4
181	U	CTRL 5
182	V	CTRL 6
183	W	CTRL 7
184	X	FCTN ,
185	Y	FCTN .
186	Z	FCTN /
187	[(open bracket)	CTRL /
188	(reverse slant)	FCTN 0
189	](close bracket)	FCTN ;
190	^(caret)	FCTN B
191	__(underline)	FCTN H
192	(grave)	FCTN J
193	a	FCTN K
194	b	FCTN L
195	c	FCTN M
196	d	FCTN N
197	e	FCTN Q
198	f	FCTN Y

Table 5: ASCII Codes 199-255

ASCII CODE	DISPLAY CHARACTER	ASCII CODE	DISPLAY CHARACTER
199	g	228	(Space)
200	h	229	(Space)
201	i	230	(Space)
202	j	231	(Space)
203	k	232	(Space)
204	l	233	(Space)
205	m	234	(Space)
206	n	235	(Space)
207	o	236	(Space)
208	p	237	(Space)
209	q	238	(Space)
210	r	239	(Space)
211	s	240	(Space)
212	t	241	(Space)
213	u	242	(Space)
214	v	243	(Space)
215	w	244	(Space)
216	x	245	(Space)
217	y	246	(Space)
218	z	247	(Space)
219	;§(left brace)	248	(Space)
220	| XX VERT BAR	249	(Space)
221	§;(right brace)	250	(Space)
222	(tilde)	251	(Space)
223	DEL(appears as a blank)	252	(Space)
224	(Space)	253	(Space)
225	(Space)	254	(Space)
226	(Space)	255	(Space)
227	(Space)		

APPENDIX B

FUNCTION KEY CODES

The function keys are assigned the following codes. These codes are returned by the CALL KEY subprogram when the corresponding keys are pressed.

KEY CODE		Function Name	Function Key
Pascal u	BASIC & 99/4		
129	1	AID	FCTN 7
130	2	CLEAR	FCTN 4
131	3	DELete	FCTN 1
132	4	INSert	FCTN 2
133	5	QUIT	FCTN =
134	6	REDO	FCTN 8
135	7	ERASE	FCTN 3
136	8	LEFT ARROW	FCTN S
137	9	RIGHT ARROW	FCTN D
138	10	DOWN ARROW	FCTN X
139	11	UP ARROW	FCTN E
140	12	PROC'D	FCTN 6
13	13	ENTER	ENTER
142	14	BEGIN	FCTN 5
143	15	BACK	FCTN 9

APPENDIX C

CONTROL KEY CODES

Key Codes BASIC Mode	 Pascal Mode	 Mnemonic ~~Code~~	 Press	 Comments
5	4 0	NUL	CTRL ,	Null character
129	1	SOH	CTRL A	Start of heading
130	2	STX	CTRL B	Start of text
131	3	ETX	CTRL C	End of text
132	4	EOT	CTRL D	End of transmission
133	5	ENQ	CTRL E	Enquiry
134	6	ACK	CTRL F	Acknowledge
135	7	BEL	CTRL G	Bell
136	8	BS	CTRL H	Backspace
137	9	HT	CTRL I	Horizontal tabulation
138	10	LF	CTRL J	Line feed
139	11	VT	CTRL K	Vertical tabulation
140	12	FF	CTRL L	Form feed
141	13	CR	CTRL M	Carriage return
142	14	SO	CTRL N	Shift out
143	15	SI	CTRL O	Shift in
144	16	DLE	CTRL P	Data link escape
145	17	DC1	CTRL Q	Device control 1 (X-ON)
146	18	DC2	CTRL R	Device control 2
147	19	DC3	CTRL S	Device control 3 (X-OFF)
148	20	DC4	CTRL T	Device control 4
149	21	NAK	CTRL U	Negative acknowledge
150	22	SYN	CTRL V	Synchronous idle
151	23	ETB	CTRL W	End of transmission block
152	24	CAN	CTRL X	Cancel
153	25	EM	CTRL Y	End of medium
154	26	SUB	CTRL Z	Substitute
155	27	ESC	CTRL .	Escape
156	28	FS	CTRL ;	File separator
157	29	GS	CTRL =	Group separator
158	30	RS	CTRL 8	Record separator
159	31	US	CTRL 9	Unit separator

APPENDIX D

KEYBOARD MAPPING

The following diagrams illustrate the key codes returned in the four keyboard modes specified by the key-unit value in the CALL KEY statement. The figures on the upper key face are function codes, and the lower figures are control codes.

Figure 1. Split Keyboard Scan

Key-units 1 and 2.

Codes returned: 0-19

Figure 2. TI-99/4 Emulator Keyboard Scan

Key-unit ~~0~~3. Both upper- and lower-case alphabetical characters returned ~~as~~ in uppercase.

Function Codes: 1-15, 32.
Control Codes: 13, 32

Figure 3. Pascal Keyboard Scan

Key-unit 4. Upper- and lower-case characters active‸

Function Codes: 13, 32, 127, 129-140, 142, 143, 184-186, 188-198.
Control Codes: 0-32, 176-183, 187.

Figure 4. BASIC Keyboard Scan

Key-unit 5. Upper- and lower-case characters active.

Function Codes: 1-15, 32, 127, 184-186, 188-198.
Control Codes: 13, 32, 128-159, 176-183, 187.

Key-Unit 1

The following table shows the key station assignments for split keyboard 1 and the corresponding return values.

Key	# Key Code
1	19
2	07
3	08
4	09
5	10
A	01
B	16
C	14
D	03
E	05
F	12
G	17
Q	18
R	06
S	02
T	11
V	13
W	04
X	00
Z	01

All other keys return a "No Key" condition.

The fire button on Joystick Controller 1 ~~one~~ is logically identical to the "Q" key with one exception: the fire button takes precedence over all other keys; the "Q" key does not.

Key-Unit 2

The following table shows the key station assignments for split keyboard 2 and the corresponding return values.

Key	Keycode
6	19
7	07
8	08
9	09
0	10
H	01
I	05
J	02
K	03
L	12
M	00
N	15
O	06
P	11
U	04
;	17
,	14
.	13
/	16

All other keys return a "No Key" condition.

The fire button on Joystick Controller 2 ~~two~~ is logically identical to the "Y" key with one exception: the fire button takes precedence over all other keys; the "Y" key does not.

Key-Unit 3--TI-99/4 Emulator Keyboard

The following table shows the key station assignments for key-unit 3 and the corresponding return values. The CAPS key has no effect. Note: NK de an "no k con

Key	Unmodified	Caps	Control	Function	Shift
1	49		NK	03	33
2	50		NK	04	64
3	51		NK	07	35
4	52		NK	02	36
5	53	C	NK	14	37
6	54	A	NK	12	94
7	55	P	NK	01	38
8	56	S	NK	06	42
9	57		NK	15	40
0	48	K	NK	NK	41
A	65	E	NK	NK	65
B	66	Y	NK	NK	66
C	67		NK	NK	67
D	68	H	NK	09	68
E	69	A	NK	NK	69
F	70	S	NK	NK	70
G	71		NK	NK	71
H	72	N	NK	NK	72
I	73	O	NK	NK	73
J	74		NK	NK	74
K	75	E	NK	NK	75
L	76	F	NK	NK	76
M	77	F	NK	NK	77
N	78	E	NK	NK	78
O	79	C	NK	NK	79
P	80	T	NK	NK	80
Q	81		NK	NK	81
R	82	I	NK	NK	82
S	83	N	NK	08	83
T	84		NK	NK	84
U	85	T	NK	NK	85
V	86	H	NK	NK	86
W	87	I	NK	NK	87
X	88	S	NK	NK	88
Y	89		NK	NK	89
Z	90	S	NK	NK	90
=	61	T	NK	05	43
-	45	A	NK	NK	95
/	92	T	NK	NK	NK
[	91	E	NK	NK	NK
]	93		NK	NK	NK
'	NK		NK	NK	NK
;	59		NK	NK	58
'	39		NK	NK	34
ENTER	13		13	13	13
,	44		NK	NK	60
.	46		NK	NK	62
/	47		NK	NK	63
SPACE	32		32	32	32

Key-Unit 4--PASCAL lc

The following table shows the key station assignments for key unit 4 and the corresponding return values. Note: NK denotes a "No Key" Condition.

Key	Unmodified	Caps	Control	Function	Shift
1	49	49	177	131	33
2	50	50	178	132	64
3	51	51	179	135	35
4	52	52	180	130	36
5	53	53	181	142	37
6	54	54	182	140	94
7	55	55	183	129	38
8	56	56	30	134	42
9	57	57	31	143	40
0	48	48	176	188	41
A	97	65	01	NK	65
B	98	66	02	190	66
C	99	67	03	NK	67
D	100	68	04	137	68
E	101	69	05	139	69
F	102	70	06	NK	70
G	103	71	07	NK	71
H	104	72	08	191	72
I	105	73	09	NK	73
J	106	74	10	192	74
K	107	75	11	193	75
L	108	76	12	194	76
M	109	77	13	195	77
N	110	78	14	196	78
O	111	79	15	NK	79
P	112	80	16	NK	80
Q	113	81	17	197	81
R	114	82	18	NK	82
S	115	83	19	136	83
T	116	84	20	NK	84
U	117	85	21	NK	85
V	118	86	22	127	86
W	119	87	23	NK	87
X	120	88	24	138	88
Y	121	89	25	198	89
Z	122	90	26	NK	90
=	61	61	29	133	43
-	45	45	NK	NK	95
\	92	92	NK	NK	124
[	91	91	NK	NK	123
]	93	93	NK	NK	125
\ `	96	96	NK	NK	126
;	59	59	28	189	58
'	39	39	NK	NK	34
ENTER	13	13	13	13	13
,	44	44	00	184	60
.	46	46	27	185	62
/	47	47	187	186	63
SPACE	32	32	32	32	32

Key-Unit 5--BASIC

The following table shows the key station assignments for key-unit 5 and the corresponding return values.

Key	Unmodified	Caps	Control	Function	Shift
1	49	49	177	03	33
2	50	50	178	04	64
3	51	51	179	07	35
4	52	52	180	02	36
5	53	53	181	14	37
6	54	54	182	12	94
7	55	55	183	01	38
8	56	56	158	06	42
9	57	57	159	15	40
0	48	48	176	188	41
A	97	65	129	NK	65
B	98	66	130	190	66
C	99	67	131	NK	67
D	100	68	132	09	68
E	101	69	133	11	69
F	102	70	134	NK	70
G	103	71	135	NK	71
H	104	72	136	191	72
I	105	73	137	NK	73
J	106	74	138	192	74
K	107	75	139	193	75
L	108	76	140	194	76
M	109	77	141	195	77
N	110	78	142	196	78
O	111	79	143	NK	79
P	112	80	144	NK	80
Q	113	81	145	197	81
R	114	82	146	NK	82
S	115	83	147	08	83
T	116	84	148	NK	84
U	117	85	149	NK	85
V	118	86	150	127	86
W	119	87	151	NK	87
X	120	88	152	10	88
Y	121	89	153	198	89
Z	122	90	154	NK	90
=	61	61	157	05	43
-	45	45	NK	NK	95
/	92	92	NK	NK	124
[	91	91	NK	NK	123
]	93	93	NK	NK	125
'	96	96	NK	NK	126
;	59	59	156	189	58
'	39	39	NK	NK	34
ENTER	13	13	13	13	13
,	44	44	128	184	60
.	46	46	155	185	62
/	47	47	187	186	63
SPACE	32	32	32	32	32

APPENDIX E

CHARACTER SETS

The ASCII character codes are grouped into 32 sets for use in color graphics programs (Pattern Mode only).

SET	ASCII CODES
29	0-7
30	8-15
31	16-23
0	24-31
1	32-39
2	40-47
3	48-55
4	56-63
5	64-71
6	72-79
7	80-87
8	88-95
9	96-103
10	104-111
11	112-119
12	120-127
13	128-135
14	136-143
15	144-151
16	152-159
17	160-167
18	168-175
19	176-183
20	184-191
21	192-199
22	200-207
23	208-215
24	216-223
25	224-231
26	232-239
27	240-247
28	248-255

APPENDIX F

ACCURACY INFORMATION

Displayed Results Versus Accuracy

The TI Computer 99/8, like all other 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 computer appears to use 10 digits as shown by the display, but actually uses more to perform all calculations. When rounded for display purposes, these extra digits help maintain the accuracy of the values presented. Example:

```
        1/3X3 = .9999999999 (inaccurate)
```

The example shows that 1/3 = .3333333333, when multiplied by 3, produces an inaccurate answer. However, a 13-digit string of nines, when rounded to 10 places, is 1.0000000000.

The higher-order mathematical functions use iterative and polynomial calculations. 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. With certain calculations, however, these digits may appear as an answer when not expected. The mathematical limits of a finite operation (word length, truncation, and rounding errors) do not allow these digits to be always completely accurate. Therefore, when subtracting two expressions that are mathematically equal, the computer may display a nonzero result.
Example:

```
        X=2/3-1/3-1/3
        PRINT X
        1E-14
```

The final result indicates a discrepancy in the fourteenth digit.

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. For the previous example, the statement shown below can be used to truncate the undisplayed digits of the variable X, leaving only the rounded display value.

```
     X=1E-10*(INT(X*1E10))
```

Internal Numeric Representation

The TI Computer 99/8 uses radix-100 format for internal calculations. A single radix-100 digit has a range of 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. Radix-100 exponents range 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 -9.9999999999999E127 through -1.0000000000000E-128; zero; and then +1.0000000000000E-128 through +9.9999999999999E127.

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). The correspondence between exponent values is shown below.

Biased hexadecimal value	00_{16}	to	4016	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 (2'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							
127_{10}	41	01	1B	00	00	00	00	00
0.5_{10}	3F	32	00	00	00	00	00	00
Ú/2	40	01	39	07	60	20	43	5F
-Ú/2	BF	FF	39	07	60	20	43	5F

APPENDIX G

RESERVED WORDS

The following is a complete list of all reserved words in TI Extended BASIC II. These are words that are reserved for use by TI Extended BASIC II and may not be used as variable names. However, you may use a reserved word as part of a variable name (for example, ALEN and LENGTH are allowed).

Subprogram names are not reserved words. Therefore, names of built-in subprograms (such as CLEAR) are valid variable names.

ABS
ACCEPT
ALL
ALPHA
AND
APPEND
ASC
AT
ATN
BASE
BEEP
BREAK
BYE
CALL
CHR$
CLOSE
CON
CONTINUE
COS
DATA
DEF
DELETE
DIGIT
DIM
DISPLAY
ELSE
END
EOF
ERASE
ERROR
EXP
FIXED
FREESPACE
FOR
GO
GOSUB
GOTO

IF
IMAGE
INPUT
INT
INTEGER
INTERNAL
LALPHA
LEN
LET
LINPUT
LIST
LOG
MAX
MERGE
MIN
NEW
NEXT
NOT
NUM
NUMBER
NUMERIC
OLD
ON
OPEN
OPTION
OR
OUTPUT
PERMANENT
PI
POS
PRINT
RANDOMIZE
READ
REAL
REC
RELATIVE
REM

RES
RESEQUENCE
RESTORE
RETURN
RND
RPT$
RUN
SAVE
SEG$
SEQUENTIAL
SGN
SIN
SIZE
SQR
STEP
STOP
STR$
SUB
SUBEND
SUBEXIT
TAB
TAN
TERMCHAR
THEN
TO
TRACE
UALPHA
UNBREAK
UNTRACE
UPDATE
USING
VAL
VALHEX
VALIDATE
VARIABLE
WARNING
XOR

APPENDIX H

MUSICAL TONE FREQUENCIES

The following table gives freauencies (rounded to integers) of four octaves of the tempered scale (one half-step between notes). Although this list does not represent the entire range of tones (or even of musical tones), it can be helpful for musical programming.

Frequency	Note	Frequency	Note
110	A	440	A (above middle C)
117	A#, Bb	466	A#, Bb
123	B	494	B
131	C (low C)	523	C (high C)
139	C#, Db	554	C#, Db
147	D	587	D
156	D#, Eb	622	D#, Eb
165	E	659	E
175	F	698	F
185	F#, Gb	740	F#, Gb
196	G	784	G
208	G#, Ab	831	G#, Ab
220	A (below middle C)	880	A (above high C)
220	A (below middle C)	880	A (above high C)
233	A#, Bb	932	A#, Bb
247	B	988	B
262	C(middle C)	1047	C
277	C#, Db	1109	C#, Db
294	D	1175	D
311	D#, Eb	1245	D#, Eb
330	E	1319	E
349	F	1397	F
370	F#,Gb	1480	F#,Gb
392	G	1568	G
415	G#,Ab	1661	G#,Ab
440	A(above middle C) *	1760	A

APPENDIX I

TRIGONOMETRIC CALCULATIONS AND RESTRICTIONS

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

Trigonometric Identities

The following trigonometric functions are not part of TI Extended BASIC II, but may be calculated by using the DEF function. (For more information on DEF, refer to page XX of the reference section.)

Function	TI Extended BASIC statement II/Cap
Secant	DEF SEC(X)=1/COS(X)
Cosecant	DEF CSC(X)=1/SIN(X)
Cotangent	DEF COT(X)=1/TAN(X)
Inverse Sine	DEF ARCSIN(X)=ATN(X/SQR(1-X*X))
Inverse Cosine	DEF ARCCOS(X)=-ATN(X/SQR(1-X*X))+PI/2
Inverse Secant	DEF ARCSEC(X)=ATN(SQR(X*X-1))+(SGN(X)-1)*PI/2
Inverse Cosecant	DEF ARCCSC(X)=ATN(1/SQR(X*X-1))+(SGN(X)-1)*PI/2
Inverse Cotangent	DEF ARCCOT(X)=PI/2-ATN(X) or =PI/2+ATN(-X)
Hyberbolic Sine	DEF SINH(X)=(EXP(X)-EXP(-X))/2
Hyberbolic Cosine	DEF COSH(X)=(EXP(X)+EXP(-X))/2
Hyperbolic Tangent	DEF TANH(X)=-2*EXP(-X)/(EXP(X)+EXP(-X))+1
Hyperbolic Secant	DEF SECH=2/(EXP(X)+EXP(-X))
Hyperbolic Cosecant	DEF CSCH=2/(EXP(X)-EXP(-X))
Hyperbolic Cotangent	DEF COTH(X)=2*EXP(-X)/(EXP(X)-EXP(-X))+1
Inverse Hyperbolic Sine	DEF ARCSINH(X)=LOG(X+SQR(X*X+1))
Inverse Hyperbolic Cosine	DEF ARCCOSH(X)=LOG(X+SQR(X*X-1))
Inverse Hyperbolic Tangent	DEF ARCTANH(X)=LOG((1+X)/(1-X))/2
Inverse Hyperbolic Secant	DEF ARCSECH(X)=LOG((1+SQR(1-X*X))/X)
Inverse Hyperbolic Cosecant	DEF ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
Inverse Hyperbolic Cotangent	DEF ARCCOTH(X)=LOG((X+1)/(X-1))/2
Base Ten Logarithm	DEF LOG10(X)=LOG(X)/LOG(10)

	Input Range	Output Range	
	|X|<=1E10	1<=|f(X)|<=1E128	
	|X|<=1E10	~~1~~<=|f(X)|<=1E128	1
t*	|X|<=1E10	|f(X)|<=1E128	
ne	|X|<~~=~~1	|f(X)|<=π/2	
	|X|<~~=~~1	0<=f(X)<=π	
	~~1~~<=|X|<1E~~128~~ 64	~~|f(X)|<=π/2~~ −π < f(x) < −π/2 , 0 < f(x) < π/2	
	~~1<=~~|X|<1E~~128~~ 64	−π<~~=~~f(X)<~~=~~π/2 , 0 < f(x) < π/2	number 1
nt	|X|<1E128	0<=f(X)<=π	
ic	|X|<295	|f(X)|<=5E127	
ic	|X|<295	1<=f(X)<=5E127	
ic	−294<=X<295	|f(X)|<=1	
lic	|X|<295	2E−128<=f(X)<=1	
lic nt	~~−294<X<=−5E−127~~ ~~5.1E−10<X<295~~ 5.1E−13<=|x|<295	1<=|f(X)|<=1E12	3
lic	~~1E127<=|X|<=294.7308919~~	2E−128<=|f(X)|<=2E14	~~[illegible]~~ 1E−12<=|x|<295
lic	−1176470<=X<1E64 −~~1180000~~<X<1E64	−14<f(X)<=148	−1176471
lic	1<=X<~~=~~1E64	0<=f(X)<=148	
lic	|X|<1	|f(X)|<=15	
lic	3E−128<=X<=1	0<=f(X)<295	
lic t	3E−128<=|X|<1E64	0<=f(X)<295	
lic nt	1<|X|<1E128	|f(X)|<295	

alues are invalid at multiples of π/2

Restrictions for Built-in Trigonometric Functions

Function	Restriction
SIN	|X| pi/2*10E10
COS	|X| pi/2*10E10
TAN	|X| pi/2*10E10 X#pi/2
ATN	-pi/2 =X =pi/2

Restrictions for User-Defined Trigonometric Functions

Function	Input Range	Output Range
Secant	-1E10 =X =1E10	-1E128 f(X) =1 1 =1E128
Cosecant	-1E10 =X =1E10	-1E128 f(X) =-1 1 =f(X) E128
Cotangent*	-1E10 =X =1E10	-1E128 f(X) 1E128
Inverse Sine	-1 =X =1	-1.570796327 =f(X) =1.570796327
Inverse Cosine	-1 =X =1	0 =f(X) =3.141592654
Inverse Secant	-1E128 X =-1 1 =X 1E128	-1.570796327 =f(X) =1.570727
Inverse Cosecant	-1E128 X =-1 1 =X 1E128	-3.141592654 =f(X) =1.570796327
Inverse Cotangent	-1E128 X 1E128	0 =f(X) =3.141592654
Hyperbolic Sine	-294.7308919 =X X =294.7308919	-5E127 =f(X) =5E127
Hyperbolic Cosine	-294.7308919 =X X =294.7308919	1 =f(X) =5E127
Hyperbolic Tangent	-294.037744 =X X =294.7308919	-1 =f(X) =1
Hyperbolic Secant	-294.7308919 =X X =294.7308919	2E-128 =f(X) =1
Hyperbolic Cotangent	-294.037744 =X =-5E-127 5.1E-13 X =294.7308919	-1E12 =f(X) =-1 1 =f(X) =1E12
Hyperbolic Cosecant	-294.7308919 =X =-1E-127 1E127 =X =294.7308919	-2E14 =f(X) =-2E-128 2E-128 =f(X) =2E14
Inverse Hyperbolic Sine	-1176470.008 =X 1E64	-13.81551056 =f(X) =148.0585931
Inverse Hyperbolic Cosine	1 =X =1E64	0 =f(X) =148.0585931
Inverse Hyperbolic Tangent	-1 X 1	-15.31337669 =f(X) =15.31337669
Inverse Hyperbolic Secant	3E-128 =X =1	0 =f(X) =294.7308919
Inverse Hyperbolic Cosecant	-1E64 X =-3E-128 3E-128 =X 1E64	0 =f(X) =294.7308919
Inverse Hyperbolic Cotangent	-1E128 X -1 1 X 1E128	-294.7308919 =f(X) =294.7308919

*Input values are invalid at multiples of pi/2

Radian, Degree, and Grad Conversions

~~Since~~ Because Extended BASIC II trigonometric functions ~~expect~~ require the arguments to be expressed as radians, it ~~may be~~ is necessary to convert values entered as degrees or grads into radians, and convert the answer back into degrees or grads after the trig calculations have been completed. The following table provides the factors needed to make the appropriate conversions.

From/To	Degrees	Radians	Grads
Degrees		xpi/180	/0.9
Radians	x180/pi		x200/pi
Grads	x0.9	xpi/200	

APPENDIX J

COLOR CODES

COLOR	CODE	COLOR	CODE
Transparent	1	Medium Red	9
Black	2	Light Red	10
Medium Green	3	Dark Yellow	11
Light Green	4	Light Yellow	12
Dark Blue	5	Dark Green	13
Light Blue	6	Magenta	14
Dark Red	7	Gray	15
Cyan	8	White	16

APPENDIX K

COLOR COMBINATIONS

The following color combinations produce the sharpest, clearest character resolution.

BEST

2, 8	Black on Cyan	2, 13	Black on Dark Green
2, 7	Black on Dark Red	2, 15	Black on Gray
2, 6	Black on Light Blue	2, 14	Black on Magenta
2, 3	Black on Medium Green	2, 9	Black on Medium Red
5, 8	Dark Blue on Cyan	5, 15	Dark Blue on Gray
5, 6	Dark Blue on Light Blue	5, 4	Dark Blue on Light Green
5, 14	Dark Blue on Magenta	5, 16	Dark Blue on White
13, 8	Dark Green on Cyan	13, 11	Dark Green on Dark Yellow
13, 15	Dark Green on Gray	13, 4	Dark Green on Light Green
13, 12	Dark Green on Light Yellow	13, 3	Dark Green on Medium Green
7, 15	Dark Red on Gray	7, 10	Dark Red on Light Red
7, 12	Dark Red on Light Yellow	14,10	Magenta on Light Red
3, 12	Medium Green on Light Yellow	3, 16	Medium Green on White

SECOND BEST

2, 5	Black on Dark Blue	2, 11	Black on Dark Yellow
2, 4	Black on Light Green	2, 10	Black on Light Red
2, 12	Black on Light Yellow	13, 10	Dark Green on Light Red
13, 16	Dark Green on White	7, 16	Dark Red on White
6, 15	Light Blue on Gray	6, 4	Light Blue on Light Green
6, 16	Light Blue on White	4, 16	Light Green on White

THIRD BEST

2, 16	Black on White	5, 12	Dark Blue on Light Yellow
7, 9	Dark Red on Medium Red	4, 12	Light Green on Light Yellow
14, 15	Magenta on Gray	14, 16	Magenta on White
3, 11	Medium Green on Dark Yellow	3, 15	Medium Green on Gray
9, 15	Medium Red on Gray	9, 10	Medium Red on Light Red
9, 12	Medium Red on Light Yellow	9, 16	Medium Red on White
16, 7	White on Dark Red		

FOURTH BEST

8, 2	Cyan on Black	8, 16	Cyan on White
7, 2	Dark Red on Black	7, 4	Dark Red on Light Green
15, 16	Gray on White	6, 2	Light Blue on Black
4, 2	Light Green on Black	10, 2	Light Red on Black
10, 16	Light Red on White	14, 12	Magenta on Light Yellow
9, 4	Medium Red on Light Green	16, 6	White on Light Blue

APPENDIX L

LIST OF SPEECH WORDS

The following is a list of all the letters, numbers, words, and phrases that can be accessed with CALL SAY and CALL SPGET. See Appendix M for instructions on adding suffixes to anything in this list.

NOTE: Multiple words must be enclosed with number signs when used with CALL SAY; for example,

```
|CALL SAY("#TEXAS INSTRUMENTS#")
```

- (NEGATIVE)
+ (POSITIVE)
. (POINT)
0
1
2
3
4
5
6
7
8
9

A (a)
A1 (_)
ABOUT
AFTER
AGAIN
ALL
AM
AN
AND
ANSWER
ANY
ARE
AS
ASSUME
AT

B
BACK
BASE
BE
BETWEEN
BLACK
BLUE
BOTH
BOTTOM
BUT
BUY
BY
BYE

C
CAN
CASSETTE
CENTER
CHECK
CHOICE
CLEAR
COLOR
COME
COMES
COMMA
COMMAND
COMPLETE
COMPLETED
COMPUTER
CONNECTED
CONSOLE
CORRECT
COURSE
CYAN

D
DATA
DECIDE
DEVICE
DID
DIFFERENT
DISKETTE
DO
DOES
DOING
DONE
DOUBLE
DOWN
DRAW
DRAWING

E
EACH
EIGHT
EIGHTY
ELEVEN
ELSE
END
ENDS
ENTER
ERROR
EXACTLY
EYE

F
FIFTEEN
FIFTY
FIGURE
FIND
FINE
FINISH
FINISHED
FIRST
FIT
FIVE
FOR
FORTY
FOUR
FOURTEEN
FOURTH
FROM
FRONT

G
GAMES
GET
GETTING
GIVE
GIVES
GO
GOES

GOING
GOOD
GOOD WORK
GOODBYE
GOT
GRAY
GREEN
GUESS

H
HAD
HAND
HANDHELD UNIT
HAS
HAVE
HEAD
HEAR
HELLO
HELP
HERE
HIGHER
HIT
HOME
HOW
HUNDRED
HURRY

I
I WIN
IF
IN
INCH
INCHES
INSTRUCTION
INSTRUCTIONS
IS
IT

J
JOYSTICK
JUST

K
KEY
KEYBOARD
KNOW

L
LARGE
LARGER
LARGEST
LAST
LEARN
LEFT
LESS
LET
LIKE
LIKES
LINE
LOAD
LONG
LOOK
LOOKS
LOWER

M
MADE
MAGENTA
MAKE
ME
MEAN
MEMORY
MESSAGE
MESSAGES
MIDDLE
MIGHT
MODULE
MORE
MOST
MOVE
MUST

N
NAME
NEAR
NEED
NEGATIVE
NEXT
NICE TRY
NINE
NINETY
NO
NOT
NOW
NUMBER

O
OF
OFF
OH
ON
ONE
ONLY
OR
ORDER
OTHER
OUT
OVER

P
PART
PARTNER
PARTS
PERIOD
PLAY
PLAYS
PLEASE
POINT
POSITION
POSITIVE
PRESS
PRINT
PRINTER
PROBLEM
PROBLEMS
PROGRAM
PUT
PUTTING

Q

R
RANDOMLY
READ (read)
READ1 (red)
READY TO START
RECORDER
RED
REFER
REMEMBER
RETURN
REWIND
RIGHT
ROUND

S
SAID
SAVE
SAY
SAYS

SCREEN
SECOND
SEE
SEES
SET
SEVEN
SEVENTY
SHAPE
SHAPES
SHIFT
SHORT
SHORTER
SHOULD
SIDE
SIDES
SIX
SIXTY
SMALL
SMALLER
SMALLEST
SO
SOME
SORRY
SPACE
SPACES
SPELL
SQUARE
START
STEP
STOP
SUM
SUPPOSED
SUPPOSED TO
SURE

T
TAKE
TEEN
TELL
TEN
TEXAS INSTRUMENTS
THAN
THAT
THAT IS INCORRECT
THAT IS RIGHT
THE (the)
THE1 (th_)
THEIR
THEN
THERE
THESE
THEY
THING
THINGS
THINK
THIRD
THIRTEEN
THIRTY
THIS
THREE
THREW
THROUGH
TIME
TO
TOGETHER
TONE
TOO
TOP
TRY
TRY AGAIN
TURN
TWELVE
TWENTY
TWO
TYPE

U
UHOH
UNDER
UNDERSTAND
UNTIL
UP
UPPER
USE

V
VARY
VERY

W
WAIT
WANT
WANTS
WAY
WE
WEIGH
WEIGHT
WELL
WERE
WHAT
WHAT WAS THAT
WHEN
WHERE
WHICH
WHITE
WHO
WHY
WILL
WITH
WON
WORD
WORDS
WORK
WORKING
WRITE

X

Y
YELLOW
YES
YET
YOU
YOU WIN
YOUR

Z
ZERO

APPENDIX M

ADDING SUFFIXES TO SPEECH WORDS

This appendix describes how to add ING, S‸and ED to any word available in the Solid State Speech™ Synthesizer resident vocabulary.

The code for a word is first read using SPGET. The code consists of a number of characters, one of which tells the speech unit the length of the word. Then, by means of the subprograms listed here, additional codes can be added to give the sound of a suffix.

Words often have trailing-off data that make the word sound more natural but prevent the easy addition of suffixes. To add suffixes, you must remove all trailing-off data.

The following program enables you to input a word and, by trying different truncation values, make the suffix sound like a natural part of the word. The subprograms DEFING (lines 1000 through 1130), DEFS1 (lines 2000 through 2100), DEFS2 (lines 3000 through 3090), DEFS3 (lines 4000 through 4120), DEFED1 (lines 5000 through 5070), DEFED2 (lines 6000 through 6110), DEFED3 (lines 7000 through 7130), and MENU (lines 10000 through 10120) should be input separately and saved with the MERGE option. (The subprogram MENU is the same one used in the illustrative program with SUB.) You may wish to use different line numbers. Each of these subprograms (except MENU) defines a suffix.

DEFING defines the ING sound. DEFS1 defines the S sound as it occurs at the end of "cats." DEFS2 defines the S sound as it occurs at the end of "cads." DEFS3 defines the S sound as it occurs at the end of "wishes." DEFED1 defines the ED sound as it occurs at the end of "passed." DEFED2 defines the ED sound as it occurs at the end of "caused." DEFED3 defines the ED sound as it occurs at the end of "heated."

In running the program, enter a 0 for the truncation value in order to leave the truncation sequence.

```
100 REM ******************
110 REM REQUIRES MERGE OF:
120 REM MENU (LINES 10000 THROUGH 10120)
130 REM DEFING (LINES 1000 THROUGH 1130)
140 REM DEFS1 (LINES 2000 THROUGH 2100)
150 REM DEFS2 (LINES 3000 THROUGH 3090)
160 REM DEFS3 (LINES 4000 THROUGH 4120)
170 REM DEFED1 (LINES 5000 THROUGH 5070)
180 REM DEFED2 (LINES 6000 THROUGH 6110)
190 REM DEFED3 (LINES 7000 THROUGH 7130)
200 REM ************************
210 CALL CLEAR
```

```
220 PRINT "THIS PROGRAM IS USED TO"
230 PRINT "FIND THE PROPER TRUNCATION"
240 PRINT "VALUE FOR ADDING SUFFIXES"
250 PRINT "TO SPEECH WORDS.": :
260 FOR DELAY=1 TO 800: :NEXT DELAY
270 PRINT "CHOOSE WHICH SUFFIX YOU"
280 PRINT "WISH TO ADD.": :
290 FOR DELAY=1 TO 800: :NEXT DELAY
300 CALL MENU (8,CHOICE)
310 DATA 'ING','S' AS IN CATS,'S' AS IN CADS,'S' AS IN WISHES,
'ED' AS IN PASSED,'ED' AS IN CAUSED,'ED' AS IN HEATED,END
320 IF CHOICE=0 OR CHOICE=8 THEN STOP
330 INPUT "WHAT IS THE WORD? ":WORD$
340 ON CHOICE GOTO 350,370,390,410,430,450,470
350 CALL DEFING(D$)
360 GOTO 480
370 CALL DEFS1(D$)!CATS
380 GOTO 480
390 CALL DEFS2(D$)!CADS
400 GOTO 480
410 CALL DEFS3(D$)!WISHES
420 GOTO 480
430 CALL DEFED1(D$)!PASSED
440 GOTO 480
450 CALL DEFED2(D$)!CAUSED
460 GOTO 480
470 CALL DEFED3(D$)!HEATED
480 REM TRY VALUES
490 CALL CLEAR
500 INPUT "TRUNCATE HOW MANY BYTES? ":L
510 IF L=0 THEN 300
520 CALL SPGET(WORD$,B$)
530 L=LEN(B$)-L-3
540 C$=SEG$(B$,1,2)&CHR$(L)&SEG$(B$,4,L)
550 CALL SAY(,C$&D$)
560 GOTO 500
```

The data items have been given in short DATA statements to make them as easy as possible to input. They may be consolidated to make the program shorter.

```
1000 SUB DEFING(A$)
1010 DATA 96,0,52,174,30,65
1020 DATA 21,186,90,247,122,214
1030 DATA 179,95,77,13,202,50
1040 DATA 153,120,117,57,40,248
1050 DATA 133,173,209,25,39,85
1060 DATA 225,54,75,167,29,77
1070 DATA 105,91,44,157,118,180
1080 DATA 169,97,161,117,218,25
1090 DATA 119,184,227,222,249,238,1
1100 RESTORE 1010
1110 A$=""
1120 FOR X=1 TO 55::READ A::A$=A$&CHR$(A)::NEXT X
1130 SUBEND

2000 SUB DEFS1(A$)!CATS
2010 DATA 96,0,26
2020 DATA 14,56,130,204,0
2030 DATA 223,177,26,224,103
2040 DATA 85,3,252,106,106
2050 DATA 128,95,44,4,240
2060 DATA 35,11,2,126,16,121
2070 RESTORE 2010
2080 A$=""
2090 FOR X=1 TO 29::READ A::A$=A$&CHR(A)::NEXT X     $/
                                      ^
2100 SUBEND

3000 SUB DEFS2(A$)!CADS
3010 DATA 96,0,17
3020 DATA 161,253,158,217
3030 DATA 168,213,198,86,0
3040 DATA 223,153,75,128,0
3050 DATA 95,139,62
3060 RESTORE 3010
3070 A$=""
3080 FOR X=1 TO 20::READ A::A$=A$&CHR$(A)::NEXT X
3090 SUBEND

4000 SUB DEFS3(A$)!WISHES
4010 DATA 96,0,34
4020 DATA 173,233,33,84,12
4030 DATA 242,205,166,55,173
4040 DATA 93,222,68,197,188
4050 DATA 134,238,123,102
4060 DATA 163,86,27,59,1,124
4070 DATA 103,46,1,2,124,45
4080 DATA 138,129,7
```

```
4090 RESTORE 4010
4100 A$=""
4110 FOR X=1 TO 37::READ A::A$=A$&CHR$(A)::NEXT X
4120 SUBEND

5000 SUB DEFED1(A$)!PASSED
5010 DATA 96,0,10
5020 DATA 0,224,128,37
5030 DATA 204,37,240,0,0,0
5040 RESTORE 5010
5050 A$=""
5060 FOR X=1 TO 13::READ A::A$=A$&CHR$(A)::NEXT X
5070 SUBEND

6000 SUB DEFED2(A$)!CAUSED
6010 DATA 96,0,26
6020 DATA 172,163,214,59,35
6030 DATA 109,170,174,68,21
6040 DATA 22,201,220,250,24
6050 DATA 69,148,162,166,234
6060 DATA 75,84,97,145,204
6070 DATA 15
6080 RESTORE 6010
6090 A$=""
6100 FOR X=1 TO 29::READ A::A$=A$&CHR$(A)::NEXT X
6110 SUBEND

7000 SUB DEFED3(A$)!HEATED
7010 DATA 96,0,36
7020 DATA 173,233,33,84,12
7030 DATA 242,205,166,183
7040 DATA 172,163,214,59,35
7050 DATA 109,170,174,68,21
7060 DATA 22,201,92,250,24
7070 DATA 69,148,162,38,235
7080 DATA 75,84,97,145,204
7090 DATA 178,127
7100 RESTORE 7010
7110 A$=""
7120 FOR X=1 TO 39::READ A::A$=A$&CHR$(A)::NEXT X
7130 SUBEND

10000 SUB MENU(COUNT,CHOICE)
10010 CALL CLEAR
10020 IF COUNT|22 THEN PRINT "TOO MANY ITEMS" :: CHOICE=0 :: SUBEXIT
10030 RESTORE
10040 FOR X=1 TO COUNT
10050 READ TEMP$
10060 TEMP$=SEG$(TEMP$,1,25)
10070 DISPLAY AT(X,1):X;TEMP$
10080 NEXT X
10090 DISPLAY AT(X+1,1):"YOUR CHOICE: 1"
```

```
10100 ACCEPT AT(X+1,14)BEEP VALIDATE(DIGIT)SIZE(-2):CHOICE
10110 IF CHOICE 1 OR CHOICE|COUNT THEN 10100
10120 SUBEND
```

You can use the subprograms in any program once you have determined the number of bytes to truncate. The following program uses the subprogram DEFING in lines 1000 through 1130 to have the computer say the word DRAWING using DRAW plus the suffix ING. Note that it was found that DRAW should be truncated by 41 characters to produce the most natural sounding DRAWING. The subprogram DEFING in lines 1000 through 1130 is the program you saved with the merge option.

cap

```
100 CALL DEFING(ING$)
110 CALL SPGET("DRAW",DRAW$)
120 L=LEN(DRAW$)-3-41! 3 BYTES OF SPEECH OVERHEAD, 41 BYTES TRUNCATED
130 DRAW$=SEG$(DRAW$,1,2)&CHR$(L)&SEG$(DRAW$,4,L)
140 CALL SAY("WE ARE",DRAW$&ING$,"A1 SCREEN")
150 GOTO 140
1000 SUB DEFING(A$)
1010 DATA 96,0,52,174,30,65
1020 DATA 21,186,90,247,122,214
1030 DATA 179,95,77,13,202,50
1040 DATA 153,120,117,57,40,248
1050 DATA 133,173,209,25,39,85
1060 DATA 225,54,75,167,29,77
1070 DATA 105,91,44,157,118,180
1080 DATA 169,97,161,117,218,25
1090 DATA 119,184,227,222,249,238,1
1100 RESTORE 1010
1110 A$=""
1120 FOR X=1 TO 55::READ A::A$=A$&CHR$(A)::NEXT X
1130 SUBEND
```

(Press CLEAR (FCTN 4) to stop the program.)

APPENDIX N

PATTERN-IDENTIFIER CONVERSION TABLE

Blocks	BINARY CODE (0=off; 1=on)	HEXADECIMAL
! ! ! ! !	0000	0
! ! ! !X!	0001	1
! ! !X! !	0010	2
! ! !X!X!	0011	3
! !X! ! !	0100	4
! !X! !X!	0101	5
! !X!X! !	0110	6
! !X!X!X!	0111	7
!X! ! ! !	1000	8
!X! ! !X!	1001	9
!X! !X! !	1010	A
!X! !X!X!	1011	B
!X!X! ! !	1100	C
!X!X! !X!	1101	D
!X!X!X! !	1110	E
!X!X!X!X!	1111	F

APPENDIX O

ASSEMBLY LANGUAGE SUPPORT ROUTINES

The TI Computer 99/8 provides several utilities that enable you to access special capabilities of the computer through TMS9900 assembly language. With /=/ these utilities, you can change the values in the Video Display Processor (VDP) chip, access the Device Service Routine (DSR) for peripheral devices, scan the keyboard, link a program to Graphics Programming Language (GPL) routines, and link to the Editor/Assembler loader. Remember that these can only be used in TI assembly-language programs.

The following list gives each of the utilities predefined in the REF/DEF table and describes briefly the use of each.

Name	Use
VSBW	Writes a single byte to VDP RAM.
VMBW	Writes multiple bytes to VDP RAM.
VSBR	Reads a single byte from VDP RAM.
VMBR	Reads multiple bytes from VDP RAM.
VWTR	Writes a single byte to a VDP Register.
KSCAN	Scans the keyboard.
GPLLNK	Links a program to GPL routines.
XMLLNK	Links a program to the assembly-language routines in the console ROM or in RAM.
DSRLNK	Links a program to DSRs.
LOADER	Links a program to the Loader to load TMS9995 tagged object code.

The TI Computer 99/8 has more utilities available through the Editor/Assembler than did the TI-99/4A Home Computer. As a result, the XMLLINK tables have changed, so that old assembly-language programs may need to be updated.

The XMLLINK utility uses the format

```
BLWP  @XMLLINK
DATA  |xxxx
```

where |xxxx defines the routine to be executed.

The following example uses this format to execute the Convert Floating Point to Integer routine:

```
BLWP  @XMLLINK
DATA  |1200
```

The table below lists the current XMLLINK routines.

(Note: FAC (the Floating Point Accumulator) begins at address |834A. ARG, which contains arguments, begins at address 835C. The STATUS byte is located at address |837C. The letters f.p. mean "floating point radix-100 format." See Appendix F under "Internal Numeric Representation" for a description of radix-100 format.)

|xxxx Description

|0100 Round Floating Point Value Using Guard Digits

INPUT: FAC contains the f.p. value.

OUTPUT: FAC contains the f.p. result after rounding by the contents of the most sig~~i~~nificant byte of FAC+8. |=|/
(FAC+8 contains guard digits that are maintained to guarantee the accuracy of the 14 most significant |=| /
digits of the results of f.p. operations.)

|0200 Round Floating Point Value to the Position Specified by FAC+10

INPUT: FAC contains the f.p. value.

OUTPUT: FAC contains the f.p. result after rounding by the contents of FAC+10.

|0300 Floating Point Value Status

INPUT: FAC contains the f.p. value.

OUTPUT: Sets the STATUS byte according to the f.p. value in FAC.

|0400 Test Floating Point Value for Overflow or Underflow

INPUT: FAC contains the f.p. value.

OUTPUT: FAC contains 0 if the f.p. value caused an underflow.
FAC contains the largest possible f.p. number (9.9999999999999E+128) if the f.p. value caused an overflow.
Otherwise, FAC contains the original f.p. value.

|0500 Test Floating Point Value for Overflow

INPUT: FAC contains the f.p. value.

OUTPUT: FAC contains the largest possible f.p. number if the f.p. value caused an overflow. Otherwise, FAC contains the original f.p. value.

|0600 Floating Point Addition--Adds two f.p. values

INPUT: FAC contains the first f.p. value and ARG contains the second f.p. value.

OUTPUT: FAC contains the f.p. result.

|0700 Floating Point Subtraction--Subtracts two f.p. values

INPUT: FAC contains the f.p. value to be subtracted. ARG contains the f.p. value from which FAC is subtracted.

OUTPUT: FAC contains the f.p. result.

|0800 Floating Point Multiplication--Multiplies two f.p. values

INPUT: FAC contains the f.p. multiplier. ARG contains the f.p. multiplicand.

OUTPUT: FAC contains the f.p. result.

|0900 Floating Point Division--Divides two f.p. values

INPUT: FAC contains the f.p. divisor. ARG contains the f.p. dividend.

OUTPUT: FAC contains the f.p. result.

|A000 Floating Point Compare--Compares two f.p. values

INPUT: ARG contains the first f.p. argument. FAC contains the second f.p. argument.

OUTPUT: Sets the STATUS byte. The high bit is set if ARG is logically higher than FAC. The greater than bit is set if ARG is arithmetically greater than FAC. The equals bit is set if ARG and FAC are equal.

Operations |0B00 through |0F00 use VSPTR (located at address |836E) as a pointer into an area of VDP RAM that is used as a stack. The stack grows toward high memory, and VSPTR points to the top element. Push is pre-increment; pop is post-decrement.

(Note: This is NOT the stack used by the BASIC interpreter. VSPTR should not be used as such while in the TI Extended BASIC II environment.)

|0B00 Value Stack Addition--Adds, using a stack in VDP RAM

INPUT: VSPTR contains the address in VDP RAM where the left f.p. value is located. FAC contains the right f.p. value.

OUTPUT: FAC contains the f.p. result.

|0C00 Value Stack Subtraction--Subtracts, using a stack in VDP RAM

INPUT: VSPTR contains the address in VDP RAM where the left f.p. value is located. FAC contains the f.p. value to be subtracted.

OUTPUT: FAC contains the f.p. results.

|0D00 Value Stack Multiplication--Multiplies, using a stack in VDP RAM

INPUT: VSPTR contains the address in VDP RAM where the f.p. multiplicand is located. FAC contains the f.p. multiplier.

OUTPUT: FAC contains the f.p. result.

|0E00 Value Stack Division--Divides, using a stack in VDP RAM

INPUT: VSPTR contains the address in VDP RAM where the f.p. dividend is located. FAC contains the f.p. divisor.

OUTPUT: FAC contains the f.p. result.

|0F00 Value Stack Compare--Compares an f.p. value in the VDP RAM stack to the f.p. value in FAC

INPUT: VSPTR contains the address in VDP RAM where the f.p. value to be compared is located. FAC contains the other f.p. value in the comparison.

OUTPUT: Sets the STATUS byte. The high bit is set if the f.p. value pointed to by VSPTR is logically higher than FAC. The greater than bit is set if the f.p. value pointed to by VSPTR is arithmetically greater than FAC. The equals bit is set if the f.p. value pointed to by VSPTR and FAC are equal. S

|1000 Convert String to Number (VDP RAM)--Converts an ASCII string in VDP RAM to an f.p. number

INPUT: FAC+12 is a pointer to the start of the string on input, and to the first unconverted character on output. The normal convention is to terminate the string with an ASCII null (|00) character.

OUTPUT: FAC contains the f.p. result.

|1100 Convert String to Number (CPU RAM)--Converts an ASCII string in CPU RAM to an f.p. number

INPUT: FAC+12 is a pointer to the start of the string on input, and to the first unconverted character on output. The normal convention is to terminate the string with an ASCII null (|00) character.

OUTPUT: FAC contains the f.p. result.

|1200 Convert Floating Point to Integer--Converts an f.p. value to an integer

INPUT: FAC contains the f.p. value to be converted.

OUTPUT: FAC contains the one-word integer value. The maximum value is |FFFF. If an error occurs, the byte at FAC+10 is set to a nonzero value.

|1700 VDP RAM Stack Push--Push the 8 bytes from FAC onto the VDP RAM stack, using VSPTR as the stack pointer.

|1800 VDP RAM Stack Pop--Pop 8 bytes into FAC from the VDP RA stack, M using VSPTR as the stack pointer.

|1001 Greatest Integer Function--Compare the greatest integer contained in the f.p. value

INPUT: FAC contains the f.p. value.

OUTPUT: FAC contains the result, which is the largest integer not greater than the original f.p. value.

|1101 Involution Routine--Raises a number to a specified power

INPUT: FAC contains the exponent value. ARG contains the base value.

OUTPUT: FAC contains the f.p. result.

|1201 Square Root Routine--Computes the square root of a number

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

|1301 Exponent Routine--Computes the inverse natural logarithm of a number

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

|1401 Natural Logarithm Routine--Computes the natural logarithm of a number

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

|1501 Cosine Routine--Computes the cosine of a number expressed in radians

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

|1601 Sine Routine--Computes the sine of a number expressed in radians

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

|1701 Tangent Routine--Computes the tangent of a number expressed in radians

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

|1801 Arctangent Routine--Computes the arctangent of a number expressed in radians

INPUT: FAC contains the input value.

OUTPUT: FAC contains the f.p. result.

|1901 Convert Number to String--Converts an f.p. number to an ASCII string

INPUT: FAC contains the input value.
FAC+11=0 for free format. (This causes all other inputs to be ignored.)
FAC+11|0 for width, excluding decimal point.
FAC+12=0 for underflow to 0, overflow to EEEEEEE.
FAC+12|0 for E-format on overflow or underflow.
FAC+13|=0 for number of digits to the right of the decimal point.
FAC+13 0 disables fixed mode.

OUTPUT: FAC is modified.
FAC+12 (byte) contains the length.
FAC+13 (byte) is the least significant byte of a pointer to the answer. The most significant byte is always |83. l=l / l=l /

|1A01 Convert Integer to Floating Point--Converts an integer to a floating point number.

INPUT: FAC contains the one-word integer value to be converted.

OUTPUT: FAC contains the 8-byte f.p. result.

|4001 Multicolor Mode--Set up VDP Pattern Name Table for multicolor mode (0-31 four times, 32-63 four time, etc.). ca s

|4101 High-Resolution Mode--Set up VDP Pattern Nmae Table for high-resolution mode (0-255 three times). cap

|4201 Draw Line--Draw a line in high-resolution mode. caps

INPUT: FAC : (byte) Graphics Mode (2, 3, or 4--one less than TI Extended BASIC II's CALL GRAPHICS mode).

FAC+1 : (byte) Color (foreground/background), each 0-15 (one less than TI Extended BASIC II's CALL COLOR values).

FAC+2 : (byte) Line Type (-1, 0, or 1).

FAC+4 : (word) Y1 (zero-based).

FAC+6 : (word) X1 (zero-based)

FAC+8 : (word) Y2 (zero-based)

FAC+10: (word) X2 (zero-based)

|4301 Fill--Fill screen area in high-resolution mode caps

INPUT: FAC : (byte) Graphics Mode (2, 3, or 4--one less than TI Extended BASIC II's CALL GRAPHICS mode).

FAC+1 : (byte) Color (foreground/background), each 0-15 (one less than TI Extended BASIC II's CALL COLOR values).

FAC+4 : (word) Y (zero-based).

FAC+6 : (word) X (zero-based).

FAC+10: (word) CPU RAM stack pointer.

FAC+12: (word) CPU RAM stack limit. The stack area is a scratch area used by the FILL routine. A 2K byte area is recommended.

Several general-use addresses are predefined with symbols. The following list gives the associated address and describes briefly each symbol.

Name	Address	Description
SCAN	|000E	Address of branch to the keyboard scan utility (KSCAN).
UTLTAB	|8600	Start of the utility variable table.
PAD	|8300	Start of CPU scratch-pad RAM.
GPLWS	|83E0	GPL interpreter workspace pointer.
SOUND	|8400	Sound chip register.
SPCHRD	|9000	Speech ~~R~~ead ~~D~~ata ~~R~~egister. /c
SPCHWT	|9400	Speech ~~W~~rite ~~D~~ata ~~R~~egister. /c

Some addresses useful for accessing memory-mapped devices are predefined with symbols. The following list gives the address and a brief description of each symbol.

Name	Address	Description
VDPWA	|8C02	VDP RAM Write Address Register.
VDPRD	|8800	VDP RAM Read Data Register.
VDPWD	|8C00	VDP RAM Write Data Register.
VDPSTA	|8802	VDP RAM Read Status Register.
GRMWA	|9C02	GROM/GRAM Write Address Register.
GRMRA	|9802	GROM/GRAM Read Address Register.
GRMRD	|9800	GROM/GRAM Read Data Register.
GRMWD	|9C00	GROM/GRAM Write Data Register.

/c

There are several TI Extended BASIC II support utilities that enable you to access variables and values passed in the parameter list of the subprogram LINK. In addition, ERR enables you to return an error to the calling TI Extended BASIC II program. Remember that these can be used only in TI assembly-language programs.

The following list gives the available utilities and describes briefly the use of each. tr

Name	Use
NUMASG	Makes a numeric assignment.
STRASG	Makes a string assignment.
NUMREF	Gets a numeric parameter.
STRREF	Gets a string parameter.
ERR	Reports errors.

The ERR utility transfers control to the error-reporting routine in TI Extended BASIC II. The assembly-language program can report any existing TI Extended BASIC II error or warning upon return to TI Extended ~~BASIC~~ II. Upon return, Workspace Register 0 contains the error code in the most-significant byte. The utility is accessed by BLWP @ERR st. Before reporting error 36 (I/O error), you must store the Input/Output ~~opcode~~ (see ~~next page~~) at >833E, and the error code in the most-significant nibble (4 bits) of >833F. The error messages that can be issued from your assembly-language program are given ~~on the next page~~ in Appendix P. Append TI

Also, six subprograms in TI Extended BASIC II can be used to interface with assembly-language programs. They are: INIT, LOAD, LINK, POKEV, PEEK, and PEEKV. These subprograms are described in the Reference section of this manual. lc

APPENDIX P

ASSEMBLY-LANGUAGE ERROR CODES

ERRTAB	ERROR CODE	MESSAGE	
0	100	* Integer overflow	
1	101	* Graphics mode error	
(2)	102	* Numeric overflow	UC
3	103	* Syntax error	
4	104	* Illegal after subprogram	
5	105	* Unmatched quotes	
6	106	* Name too long	
7	107	* String-number mismatch	
8	108	* Option-base error	
9	109	* Improperly used name	
10	10A	* Image error	
11	10B	* Memory full	
12	10C	* Stack overflow	
13	10D	* NEXT without FOR	
14	10E	* FOR-NEXT nesting	
15	10F	* Must be in subprogram	
16	110	* Recursive subprogram call	
17	111	* Missing SUBEND	
18	112	* RETURN without GOSUB	
(19)	113	* String truncated	UC
20	114	* Bad subscript	
21	115	* Speech string too long	
22	116	* Line not found	
23	117	* Bad line number	
24	118	* Line too long	
25	119	* Can't continue	
26	11A	* Command illegal in program	
27	11B	* Only legal in a program	
28	11C	* Bad argument	Jet
(29)	11D	* No program present	UC
30	11E	* Bad value	
31	11F	* Incorrect argument list	
(32)	120	* Input error	UC
33	121	* Data error	
34	122	* File error	
35	123	* I/O error	
36	124	* I/O error	
37	125	* Subprogram not found	
(38)	126	* Line not found	UC
40	128	* Unrecognized character	
41	129	* Input error	
42	12A	* Check program in memory	

(ERRTAB numbers enclosed in parentheses indicate a warning.)

APPENDIX P

ASSEMBLY-LANGUAGE ERROR CODES

ERRTAB	ERROR CODE	MESSAGE	
0	|00	* Integer overflow	
1	|01	* Graphics mode error	
(2)	|02	* Numeric overflow	UC
3	|03	* Syntax error	
4	|04	* Illegal after subprogram	
5	|05	* Unmatched quotes	
6	|06	* Name too long	
7	|07	* String-number mismatch	
8	|08	* Option-base error	
9	|09	* Improperly used name	
10	|0A	* Image error	
11	|0B	* Memory full	
12	|0C	* Stack overflow	
13	|0D	* NEXT without FOR	
14	|0E	* FOR-NEXT nesting	
15	|0F	* Must be in subprogram	
16	|10	* Recursive subprogram call	
17	|11	* Missing SUBEND	
18	|12	* RETURN without GOSUB	
(19)	|13	* String truncated	UC
20	|14	* Bad subscript	
21	|15	* Speech string too long	
22	|16	* Line not found	
23	|17	* Bad line number	
24	|18	* Line too long	
25	|19	* Can't continue	
26	|1A	* Command illegal in program	
27	|1B	* Only legal in a program	
28	|1C	* Bad argument	Jet
(29)	|1D	* No program present	UC
30	|1E	* Bad value	
31	|1F	* Incorrect argument list	
(32)	|20	* Input error	UC
33	|21	* Data error	
34	|22	* File error	
35	|23	* I/O error	
36	|24	* I/O error	
37	|25	* Subprogram not found	
(38)	|26	* Line not found	UC
40	|28	* Unrecognized character	
41	|29	* Input error	
42	|2A	* Check program in memory	

(ERRTAB numbers enclosed in parentheses indicate a warning.)

APPENDIX R

Transfer Raw Data

In certain special applications, you might need to ~~interface~~ use the Computer 99/8 with a peripheral that uses commands not supported by the 99/8. To do this, the 99/8 transfers information in bytes over the HEX-BUS™ Interface, /c enabling you to control the ~~HEX-BUS~~ Interface and the peripheral directly, /c without the ~~HEX-BUS~~ interface interpreting the signals.

When transferring raw data over the HEX-BUS™ Interface, you must construct /c and send a command message and then interpret the response from the peripheral. A command message must contain all of the following information, in the order in which it is presented:

Ital

Field Name	Number of Bytes
Device-number	1
Command code	1
File-number	1
Record-number	2 (least-significant byte first)
Buffer length	2 (least-significant byte first)
Data length	2 (least-significant byte first)
Data	number of bytes specified by data length

Response from peripheral:

Field Name	Number of Bytes
Data length	2 bytes (LSB/MSB)
Data	Number of bytes specified by data length
Status byte	1 byte

The command codes used by the HEX-BUS™ Interface are as follows: /c

Command Code	Command
0	Open
1	Close
3	Read data
4	Write data
5	Restore file
6	Delete file
7	Return status
8	Service request enable
9	Service request disable

10	Service request poll	
11	Take control of the HEX-BUS line	ital
12	Verify read/write operation	
13	Format and certify media	
14	Catalog directory	
15	Set null characteristics	
16	Transmit break	
254	Null operation	
255	BUS reset	lc

To transfer raw data between a peripheral and the Computer 99/8, use a PRINT statement followed immediately by a LINPUT statement. PRINT sends a command to the peripheral; LINPUT accepts a response message from the peripheral. Each program should contain at least three pairs of statements: one to open the peripheral; one or more pairs, as needed, to use the peripheral; and a pair to close the peripheral. Before the first pair of statements, the HEX-BUS™ Interface itself must be opened in the Transfer Raw Data mode. Be sure to close the ~~HEX-BUS~~ interface after closing the peripheral.

c

The following program illustrates the use of the Transfer Raw Data mode. It accepts input and ~~prints~~ sends it to a printer through the RS232.

Enter "END" to stop the program.

```
100 OPEN #1:"HEXBUS.TR" ! ENTER TRANSFER RAW DATA MODE
110 CALL CLEAR
120 Z$=CHR$(0)
130 ZZ$=Z$&Z$
140 REM MAKE SURE THE RS232 IS CLOSED
150 MSG$=CHR$(20)&CHR$(1)&Z$&ZZ$&ZZ$&ZZ$
160 PRINT #1:MSG$
170 LINPUT #1:RESPONSE$
180 DISPLAY AT(5,5):"ENTER BAUD RATE: "
190 ACCEPT AT(5,21)VALIDATE(DIGIT):BAUDRATE
200 MSG$=CHR$(20)&ZZ$&ZZ$&CHR$(4)&Z$&CHR$(6+LEN(STR$(BAUDRATE)))&Z$
210 MSG$=MSG$&ZZ$&CHR$(192)&"BA="&STR$(BAUDRATE)
220 REM OPEN RS232
230 PRINT #1:MSG$
240 GOSUB 400
250 REM BUILD OUTPUT STRING
260 MSG$=CHR$(20)&CHR$(4)&Z$&ZZ$&ZZ$
270 LINPUT "ENTER MESSAGE: ":MESSAGE$
280 IF LEN(MESSAGE$)|246 THEN PRINT "MESSAGE TOO LONG"::GOTO 270
290 MSG$=MSG$&CHR$(LEN(MESSAGE$))&Z$&MESSAGE$
300 REM OUTPUT TO RS232
310 PRINT #1:MSG$
320 GOSUB 400
330 IF MESSAGE$ |"END" THEN 260
340 REM CLOSE RS232
350 MSG$=CHR$(20)&CHR$(1)&Z$&ZZ$&ZZ$&ZZ$
360 PRINT #1:MSG$
370 GOSUB 400
380 CLOSE #1 ! EXIT TRANSFER RAW DATA MODE
390 STOP
400 REM CHECK FOR VALID RESPONSE
410 LINPUT #1:RESPONSE$
420 ERRCHK$=SEG$(RESPONSE$,LEN(RESPONSE$),1)
430 IF ERRCHK$=CHR$(0) THEN RETURN
440 PRINT "ERROR IN TR"
450 CLOSE #1 ! EXIT TRANSFER RAW DATA MODE
460 STOP
```

Line 100 opens the HEX-BUS Interface in the Transfer Raw Data mode. lc
Lines 140-170 ensure that the RS232 is closed. (This is a precautionary measure.)
Lines 180-190 prompt for the baud rate of the attached printer.
Lines 200-210 build the command message, which is then transmitted in line 230.
Line 240 calls a subroutine that checks for a valid response from the RS232.
Lines 260-290 build the command message, including the input message. Line 310 transmits this message to the RS232.
Line 320 again calls the subroutine that checks for a valid response from the RS232.
Line 330 tests for an input of ~~closes the RS232 when the input is~~ "END".
Lines 350-360 build and transmit a command message to close the RS232.
Line 370 calls the response-checking subroutine.
Line 380 closes the HEX-BUS Interface and ~~exits~~ leaves the Transfer Raw Data mode. lc

APPENDIX S

Using the Computer 99/8 as a Slave Device

Normally the Computer 99/8 acts as the controlling, or master, device. All the peripheral devices attached to the computer are typically slave devices that follow the commands given by the computer. If you have another HEX-BUS compatible computer, however, you can cause this second computer to control the 99/8 as a slave device. In this way you could, for example, display information from another computer on the screen attached to the 99/8.

To operate the Computer 99/8 as a slave device, you must first open the computer using the HEX-BUS subcommand .SL (SLAVE) in your OPEN statement. To ~~cause the Home Computer to~~ act upon instructions given it by a second computer, your ~~Home Computer~~ must obtain instructions and then reply to them; it does this by a pair of INPUT and PRINT statements. The structure of the command and response strings used in the INPUT and PRINT statements is the same as that used in the .TR subcommand (see the sample program in Appendix R).. You may take the 99/8 out of slave mode by closing the file in which the computer was declared a slave device.

The following partial program demonstrates how to use the Computer 99/8 as a slave device. Notice that the INPUT statement precedes the PRINT statement in an exchange of this type. Your computer reads the command string (INPUT) and sends a response (PRINT).

```
100 OPEN #1:"HEXBUS.SL.61"
.
.
.
290 REM PRINT EACH BYTE ON THE SCREEN
300 INPUT #1:A$
310 FOR J=1 TO LEN(A$)
320 PRINT ASC(SEG$(A$,1,J))
330 NEXT J
.
.
...(analyze A$ using the command message discussed in Appendix R)
   (packing responding message in B$)
.
.
400 PRINT #1:B$
.
.
.
(more INPUT and PRINT pairs)
.
.
.
500 CLOSE #1
```

In this example, the 99/8 acts as a slave peripheral with device ~~code~~ number 61. When controlling a computer with this subcommand, use only those device ~~codes~~ numb within the range specified for the slave mode (60-63).

APPENDIX V

HEX-BUSTM ERROR MESSAGES

The HEX-BUSTM peripherals generate many more error messages. Therefore, there will be several HEX-BUS error codes that can cause any one of the standard seven error codes. A summary of these is presented below. For a further explanation of the HEX-BUS error codes, refer to the appropriate manual for each HEX-BUS peripheral.

cap / 1:1

Second digit of Error Code	HEXBUS Error Code	Error Type
0	0	HEX-BUS peripheral not connected
1	9	Write protect error
	10	A peripheral device denies requesting service from the "master" device
2	1	Device/file options error
	2	Error in attributes
	17	Relative files not supported
	18	Sequential files not supported
	19	Append mode not supported
	20	Output mode not supported
	21	Input mode not supported
	22	Update mode not supported
	23	Incorrect file type
3	13	Unsupported command
	14	Device/file not open for input
	15	Device/file not open for output
4	11	Directory full
	12	Buffer size error
	32	Media full
5	7	EOF error
6	6	Device error
	16	Data error (checksum)
	24	Verify error
	25	Low batteries in peripheral device
	26	Uninitialized media
	27	Peripheral bus error
	255	Bus time-out
7	3	File not found
	4	Device/file not open
	5	Device/file already open
	8	Data/file too long
		All other errors

um

um

APPENDIX W

ERROR MESSAGES

The following lists all the error messages that TI Extended BASIC II gives. The first list is alphabetical by the message that is given, and the second list is numeric by the number of the error. The error messages, with the exception of "I/O ERROR," do not include the error number when displayed. Use CALL ERR to ascertain the error number.

An I/O error always gives a 2-digit code. For ~~a description~~ an explanation of that code, number/ refer to Appendix U, "I/O Error Messages," or see the manual that comes with the peripheral being used.

If the error occurs in the execution of a program, the error message is often followed by "in line-number".

Sorted by Message

#	Message	Descriptions of Possible Errors
74	Bad argument	!o! Bad value given in ASC, ATN, COS, EXP, INT, LOG, SIN, SOUND, SQR, TAN, or VAL. !o! An array element specified in a SUB statement. !o! Bad first parameter or too many parameters in LINK.
61	Bad line number	!o! Line number equals zero or is greater than 32767. !o! Omitted line number. !o! Line number greater than 32767 produced by RES.
57	Bad subscript	!o! Use of too large or too small subscript in an array. !o! Incorrect subscript in DIM.
79	Bad value	!o! Incorrect value given in AND, CHAR, CHR$, CLOSE, EOF, FOR, GOSUB, GOTO, HCHAR, INPUT, MOTION, NOT, OR, POS, PRINT, PRINT USING, REC, RESTORE, RPT$, SEG$, SIZE, VCHAR, or XOR. !o! Incorrect line number given after THEN or ELSE. !o! Array subscript value greater than 32767. !o! File number greater than 255 or less than zero. !o! More than three tones and one noise generator specified in SOUND. !o! An unacceptable value passed to a subprogram. For example, a sprite velocity value less than -128 or a character value greater than 143. !o! Value in ON GOTO or ON GOSUB greater than the number of lines given. !o! Incorrect position given after the AT clause in ACCEPT or DISPLAY.
67	Can't continue	!o! Program edited after being stopped by a breakpoint. !o! Program not stopped by a breakpoint.

APPENDIX W

ERROR MESSAGES

The following lists all the error messages that TI Extended BASIC II gives. The first list is alphabetical by the message that is given, and the second list is numeric by the number of the error. The error messages, with the exception of "I/O ERROR," do not include the error number when displayed. Use CALL ERR to ascertain the error number.

An I/O error always gives a 2-digit code. For an explanation ~~a description~~ of that code, number/ refer to Appendix U, "I/O Error Messages," or see the manual that comes with the peripheral being used.

If the error occurs in the execution of a program, the error message is often followed by "in line-number".

Sorted by Message

# Message Descriptions of Possible Errors

74 Bad argument
!o! Bad value given in ASC, ATN, COS, EXP, INT, LOG, SIN, SOUND, SQR, TAN, or VAL.
!o! An array element specified in a SUB statement.
!o! Bad first parameter or too many parameters in LINK.

61 Bad line number
!o! Line number equals zero or is greater than 32767.
!o! Omitted line number.
!o! Line number greater than 32767 produced by RES.

57 Bad subscript
!o! Use of too large or too small subscript in an array.
!o! Incorrect subscript in DIM.

79 Bad value
!o! Incorrect value given in AND, CHAR, CHR$, CLOSE, EOF, FOR, GOSUB, GOTO, HCHAR, INPUT, MOTION, NOT, OR, POS, PRINT, PRINT USING, REC, RESTORE, RPT$, SEG$, SIZE, VCHAR, or XOR.
!o! Incorrect line number given after THEN or ELSE.
!o! Array subscript value greater than 32767.
!o! File number greater than 255 or less than zero.
!o! More than three tones and one noise generator specified in SOUND.
!o! An unacceptable value passed to a subprogram. For example, a sprite velocity value less than -128 or a character value greater than 143.
!o! Value in ON GOTO or ON GOSUB greater than the number of lines given.
!o! Incorrect position given after the AT clause in ACCEPT or DISPLAY.

67 Can't continue
!o! Program edited after being stopped by a breakpoint.
!o! Program not stopped by a breakpoint.

!o! Putting a user defined function name on the left of the equals sign in an assignment statement.

!o! Using the same variable twice in the parameter list of a SUB statemtent.

81 Incorrect argument list

!o! CALL and SUB mismatch of arguments.

83 Input error

!o! An error detected in an INPUT.

60 Line not found

!o! Incorrect line number found in BREAK, GOSUB, GOTO, ON ERROR, RUN, or UNBREAK, or after THEN or ELSE.

!o! Line to be edited not found.

62 Line too long

!o! Line too long to be entered into a program.

39 Memory full

!o! Program too large to execute one of the following: DEF, DELETE, DIM, GOSUB, LET, LOAD, ON GOSUB, OPEN, or SUB.

!o! Program too large to add a new line, insert a line, replace a line, or evaluate an expression.

49 Missing SUBEND

!o! SUBEND missing in a subprogram.

47 Must be in subprogram

!o! SUBEND or SUBEXIT not in a subprogram.

19 Name too long

!o! More than 15 character variable or subprogram name (including $ with string variables).

43 NEXT without FOR

!o! FOR statement missing, NEXT before FOR, incorrect FOR-NEXT nesting, or branching into a FOR-NEXT loop.

78 No program present

!o! No program present when issuing a LIST, RESEQUENCE, RESTORE, RUN, or SAVE command or when entering Edit Mode.

!o! LINK called without first calling INIT.

10 Numeric overflow

!o! A number too large or too small resulting from a *, +, -, / operation or in ACCEPT, ATN, COS, EXP, INPUT, INT, LOG, SIN, SQR, TAN, or VAL.

!o! A number outside the range -32768 to 32767 inclusive in PEEK or LOAD.

70 Only legal in a program

!o! One of the following statements used as a command: DEF, GOSUB, GOTO, IF, IMAGE, INPUT, ON BREAK, ON ERROR, ON GOSUB, ON GOTO, ON WARNING, OPTION BASE, RETURN, SUB, SUBEND, or SUBEXIT

25 OPTION BASE error

!o! OPTION BASE executed more than once, or with a value other than 1 or ~~zero~~.

48 Recursive subprogram call

!o! Subprogram calls itself, directly or indirectly.

51 RETURN without GOSUB

!o! RETURN without a GOSUB or an error handled by the previous execution of an ON ERROR statement.

56 Speech string too long

!o! Speech string returned by SPGET longer than 255 characters.

40 Stack overflow

!o! Too many sets of parentheses.

!o! Not enough memory to evaluate an expression or assign a value.

54 String truncated

!o! A string created by RPT$, concatenation ("&" operator), or a user-defined function longer than 4090 characters.

!o! The length of a string expression in the VALIDATE ~~clause~~ option greater than 40~~89~~90 characters.

24 String-number mismatch

!o! A string given where a number was expected (or vice versa) in a TI Extended BASIC II function or subprogram.

!o! Assigning a string value to a numeric value or vice versa.

!o! Attempting to concatenate ("&" operator) a number.

!o! Using a string as a subscript.

135 Subprogram not found

!o! A nonexistant subprogram called or an assembly language subprogram named in LINK not loaded. e/1

!o! A assembly language routine has been loaded that REFers to labels not DEFined. n/1

14 Syntax error

!o! Missing or extra comma or parenthesis, parameters in the wrong order, missing parameters, missing keyword, misspelled keyword, keyword in the wrong order, reference to a negative line-number, or the like detected in a TI Extended BASIC II command, statement, function, or subprogram.

!o! DATA or IMAGE not ~~first and~~ only statement on a line. the

!o! Items after final ")".

!o! Missing "#" in SPR:TE.

!o! Missing ~~ENTER, tail comment symbol~~ symbol (!), ~~or~~ statement separator symbol (::) 9/ Trailing re , or ENTE keystroke

!o! Missing THEN after IF.

!o! Missing TO after FOR.

!o! Nothing after CALL, SUB, FOR, THEN, or ELSE.

!o! Two E's in a numeric constant. 9/

!o! Wrong parameter list in a ~~TI Extended BASIC II supplied~~ built-i subprogram.

!o! Going into or out of a subprogram with GOTO, GOSUB, ON ERROR, etc.

!o! Using a constant where a variable is required.

!o! More than seven dimensions in an array.

17 Unmatched quotes

!o! Odd number of quotes in an input line.

20 Unrecognized character

!o! An character such as ? or % not enclosed in quotation marks. 9/

!o! A bad field in an object file accessed by LOAD.

Sorted by number

#	Message
10	* Numeric overflow
12	* Integer overflow
14	* Syntax error
16	* Illegal after subprogram
17	* Unmatched quotes
19	* Name too long
20	* Unrecognized character
22	* Check program in memory
24	* String-number mismatch
25	* OPTION BASE error
28	* Improperly used name
30	* Graphics mode error
36	* IMAGE error
39	* Memory full
40	* Stack overflow
43	* NEXT without FOR
44	* FOR-NEXT nesting
47	* Must be in subprogram
48	* Recursive subprogram call
49	* Missing SUBEND
51	* RETURN without GOSUB
54	* String truncated
56	* Speech string too long
57	* Bad subscript
60	* Line not found
61	* Bad line number
62	* Line too long
67	* Can't continue
69	* Command illegal in program
70	* Only legal in a program
74	* Bad argument
78	* No program present
79	* Bad value
81	* Incorrect argument list
83	* Input error
84	* Data error
97	* Protection violation
109	* File error
130	* I/O error
135	* Subprogram not found

GLOSSARY

Accessory Devices--See Peripheral Devices.

Array--A collection of numeric or string variables arranged in a list or matrix for processing by the computer. Each element in an array is referenced by a subscript describing its position in the list.

ASCII--The American Standard Code for Information Interchange, the code structure used internally in most personal computers to represent letters, numbers, and special characters. Appendix A gives the ASCII codes as used by the Computer 99/8.

BASIC (Beginners All-purpose Symbolic Instruction Code)--An easy-to-use, popular programming language used in most personal computers. BASIC was developed at Dartmouth College in the 1960s.

Baud--The transmission rate, in bits per second, of data over a communication line, such as between a computer and a peripheral. A baud rate of 300 indicates that 300 bits of information are being transmitted serially every second.

Binary--The two-digit (bit) number system based on 0 and 1. Computers recognize the binary bits 0 and 1 by using gates. Gates are electronic circuits that are either off or on, representing 0 or 1, respectively.

Bit--A binary digit (0 or 1).

Branch--A departure from the sequential execution of program statements. An unconditional branch causes the computer to jump to a specified program line every time the branching statement is encountered. With a conditional branch, transfer of program control is contingent on the result of some arithmetic or logical operation.

Breakpoint--A point in a program specified by the BREAK command at which program execution is suspended. During a breakpoint, you can perform operations in the Command Mode to help you locate program errors. Program execution can be resumed with a CONTINUE command, unless the program was edited during the break.

Buffer--An area of computer memory used for temporary storage of an input or output record.

Bug--An error in the hardware or software of a computer that causes an operation to be performed incorrectly.

Byte--A group of binary digits (bits) treated as a unit, often representing one data character. With most microcomputers, eight bits are equal to one byte. The computer's memory capacity is often expressed as the number of bytes available. For example, a computer with "16K" has 16,384 bytes of memory available for storing programs and data. See K (kilo).

Cartridge--Preprogrammed ROM modules that are easily inserted into the 99/8 to extend its capabilities.

Cassette--A standard audio cassette tape used to store programs and other data; the same type of tape commonly used to record music. (Use of "metal" tapes is not recommended.)

Central Processing Unit (CPU)--The nerve center of a computer; the network of electronic circuits that interprets programs and tells a computer how to carry them out.

Character--A letter, number, punctuation symbol, or special graphics symbol, usually requiring one byte of memory storage.

Chip--Tiny silicon slices used to make electronic memories and other circuits. A single chip may have as many as 500,000 electronic parts.

Circuit Board--A rigid fiberglass or phenolic card on which various electronic parts are mounted. Printed or etched copper tracks connect the various parts to one another.

Command--An instruction that the computer performs immediately. Commands are not a part of a program and thus are entered with no preceding line number.

Examples: NEW, LIST, RUN, CALL CLEAR.

Command Mode--A computer mode in which commands are entered directly into the computer without a line number; such commands are executed immediately. Also called Immediate Mode.

Computer--A network of electronic circuits and memories that processes data.

Concatenation--The linking of two or more strings to make a longer string. The "&" is the concatenation operator.

Constant--A real number (such as 1.2 or -9054), an integer (such as 5 or 32767), or a string of characters (any combination of up to approximately 160 characters enclosed in quotes, such as "HELLO THERE" or "275 FIRST ST.").

CPU--See Central Processing Unit.

Crunched Line--A program line that has been reduced to an internal format by replacing reserved words with single special characters. The maximum length of a crunched line is 160 characters.

Cursor--A flashing underline or rectangle that indicates where the next typed character will appear.

Data--Basic elements of information that are processed or produced by the computer.

Default--A standard characteristic or value that the computer assumes if certain specifications are omitted within a statement or program.

device--See peripheral devices. caps

Diskette--A mass-storage medium used with a disk drive; also called "floppy disk." Diskettes can store both sequential and relative files.

Display--As a noun, the video screen; as a verb, to cause characters to appear on the screen.

Edit Mode--The mode used to change existing program lines. The Edit mode is entered by typing the line number followed by UP ARROW (FCTN E) or DOWN ARROW (FCTN X). The line specified is displayed on the screen and changes can be made to any character (including the line number) using the editing keys. caps

End-of-file (EOF)--The condition indicating that all data has been read from a file.

Execute--To perform the task specified by a statement or command; to cause a program to be performed by the computer.

Exponent--A number indicating the power to which a number or expression is raised, usually written to the right and above the number; for example, 28 means 2x2x2x2x2x2x2x2. In TI Extended BASIC II, the exponent is entered following the ^ symbol or following the letter "E" in scientific notation; for example, 28 is entered as 2^8, and 1.3 X 1025 is represented by 1.3E25 (or 1.3E+25).

Exponential Notation--See scientific notation. caps//

Expression--A combination of constants, variables, and operators that can be evaluated to a single result; expressions can be numeric, string, relational, or logical. stet

File--A collection of related data records stored on a peripheral device; also used interchangeably with "device" for input/output equipment that cannot use multiple files, such as a line printer. A file can also be a program.

Fixed-length Records--File records that are all the same length. If a file has fixed-length records of 95 characters, each record is allocated 95 bytes even if the data occupy only 76 bytes. The computer adds padding characters on the right to ensure that the record has the specified length.

Floppy--See Diskette.

Function--A feature that enables you to specify as "single" operations a variety of procedures, each of which actually contains a number of steps (for example, a procedure to calculate square roots via a simple reference name). The DEF statement can be used to define a function.

Gate--A very simple electronic circuit that is always either on or off. Clusters of gates can manipulate binary numbers (0 = off, 1 = on). They can also count, do arithmetic, make decisions, and store binary numbers. Gates are the basic building blocks of computers.

Graphics--A set of subprograms that enables you to create a representation of an object or objects on the monitor or television screen.

Hardware--The various devices that comprise a computer system, including the central processing unit, keyboard, screen, data storage/retrieval devices, line printers, etc.

Hertz (Hz)--A unit of frequency; 1 Hz = 1 cps (cycles per second).

Hexadecimal--A base-16 number system using 16 symbols, 0-9 and A-F. It is used as a convenient shorthand way to express binary code; for example, 1010 in binary is A in hexadecimal; 11111111 in binary is FF in hexadecimal.

Hierarchy--A series of expressions ranked according to priority of execution.

Immediate Mode--See Command Mode.

Increment--A positive or negative value that is used to modify a variable.

Input--As a noun, data entered into memory to be processed; as a verb, the process of transferring data into memory.

Input Line--The number of data items that can be entered at one time; see Crunched Line.

Input/Output (I/O)--Usually referring to a device function, I/O is used for communication between the computer and other devices (~~e.g.~~, keyboard, Program Recorder). for exa

Integer--A whole number, either positive, negative, or zero; also, a numeric data type that uses only whole number values. A variable of the integer data type must be within the range of -32768 and 32767, inclusive.

Internal Data Format--Data in the form used directly by the computer. Internal numeric data of the real data type are 9 bytes long. Internal numeric data of the integer data type are 3 bytes long. The length for internal string data is one byte per character in the string plus one length byte.

Interpreter--The program stored inside a computer that converts or translates TI Extended BASIC II statements into the computer's assembly language.

Iteration--The technique of repeating a group of program statements; one repetition of such a group. See Loop.

K--Short for "kilo," meaning "thousand"; 1K of memory is actually 1024 (210) bytes. Thus, a 4K memory has 4,096 bytes available for storage.

Line--See input line, print line, or program line. caps

Line number--A number identifying a statement in a program; line numbers determine the order in which a computer executes the commands of a program. cap

Loop--A group of consecutive program lines repeatedly performed, usually a specified number of times. tr

Mantissa--The base-number portion of a number expressed in scientific notation; in 3.264E+4, the mantissa is 3.264.

Mass-storage Device--A peripheral device (such as the Disk Drive/Controller or Program Recorder) that stores programs or data for later use by the computer.

Memory--See RAM, ROM, and mass-storage device. caps

Microprocessor--The central processing unit of a computer assembled on a single silicon chip.

Module--See cartridge. cap

Noise--Various frequencies that can be used to produce non-musical sound effects. A noise, rather than a musical tone, is generated by the CALL SOUND subprogram when a negative frequency value is specified (-1 through -8).

Null String--A string that contains no characters and has zero length.

Number Mode--The mode in which the computer automatically generates program line numbers for entering or changing statements.

Operator--A symbol used in calculations (arithmetic operators), in comparisons (relational operators), and string concatenation (linkage). The arithmetic operators are +, -, *, /, and ^. The relational operators are |, , =, |=, =, and |. The logical operators are NOT, XOR, AND, and OR. The string operator is &.

Output--As a noun, information supplied by the computer; as a verb, the process of transferring information from the computer's memory to a peripheral device, such as a screen, printer, or mass-storage device.

Overflow--The condition that occurs when a rounded numeric value greater than 9.9999999999999E127 or less than -9.9999999999999E127 is entered, computed, or assigned to a variable. An integer overflow occurs when a value outside the range of -32768 through +32767 is assigned to a variable of the integer data type. When an overflow occurs, the value is replaced by the computer's limit, a warning is displayed, and the program continues.

Parameter--A value that affects the output of a statement, function, subprogram, or subroutine.

Peripheral Devices--Equipment that attaches to the computer to extend its functions and capabilities; these units send, receive, or store data. They are often called simply peripherals or devices.

Precedence--The order in which expressions are graded or ranked for execution within a program.

Print Line--A line used by the PRINT and DISPLAY statements. When the Computer 99/8 is in Pattern Mode, the print line has 28 positions; in Text Mode, 40 positions.

Program--A sequence of instructions (statements) designed to be executed by a computer.

Program Line--A line that contains one statement or a series of statements separated by the statement separator symbol (::).

Prompt--A symbol (|) that marks the beginning of each command or program line; a symbol or phrase that requests input from the user.

Pseudo-random Number--A number produced by a set of calculations (an algorithm), sufficiently random for most applications. A truly random number is obtained entirely by chance.

Radix-100--A number system based on 100; see "Accuracy Information."

RAM--Random-Access Memory; the memory where program statements and data are stored during program execution. New programs and data can be read in, accessed, and changed in RAM. Data items stored in RAM are erased when the power is turned off or BASIC ~~is exited~~. the comput leaves

Real--A number that contains a fractional part, thus decimal places. Real numbers may be either positive or negative. Also, a numeric data type that can use either whole number values or real number values.

Record--A collection of related data, such as an individual's payroll information or a student's test scores; a group of similar records, such as a company's payroll records, is called a file.

Relative (Random Access)--A type of file organization in which records may be read or written in any order.

Reserved Word--A special word with a predefined meaning in programming languages. A reserved word must be spelled precisely, appear in its proper position in a statement or command, and must not be used as a variable name.

ROM--Read-Only Memory; the memory where certain instructions for the computer are permanently stored ROM can be read but cannot be changed. ROM is not erased when electrical power is turned off.

Run Mode--The mode in which the computer executes a program. Run Mode is terminated when program execution ends, either normally or abnormally. To leave Run Mode, press CLEAR during program execution (see Breakpoint).

Scientific Notation--A method of expressing very large or very small numbers by using a base number (mantissa) times 10 raised to some power (exponent). To represent scientific notation in TI Extended BASIC II, enter the mantissa (preceded by the minus sign if negative), the letter E, and the exponent (preceded by a minus sign if negative): for example, 3.264E4; -2.47E-17. This special format of scientific notation is called exponential notation.

Scroll--Movement of text on the screen to display additional information.

Sequential Access--A type of file organization in which records are read or written in order, from beginning to end.

Software--Programs that are executed by the computer, including programs built into the computer, programs on cartridges, diskettes, or cassettes, and programs entered by the user.

Statement--An instruction in a program that is preceded by a line number. In TI Extended BASIC II, more than one statement can be entered in one program line.

String--A series of letters, numbers, and/or symbols treated as a unit.

Subprogram--A general-purpose procedure that may be either predefined or defined by the user. In TI Extended BASIC II, predefined subprograms are accessed by the user through the CALL statement. User-defined subprograms are defined with the SUB statement and terminated with the SUBEND statement. Subprograms extend the capability of TI Extended BASIC II.

Subroutine--A program segment, written by the user, that can be used more than once during the execution of a program to perform a special task (~~e.g.~~., a set of calculations or a print routine). In TI Extended BASIC II, a subroutine is accessed by a GOSUB statement and terminated with a RETURN statement. for ex

Subscript--A numeric expression that specifies a particular item in an array; in TI Extended BASIC II, the subscript is written in parentheses immediately following the array name.

Trace--A command that lists the order in which the computer performs program statements; tracing line numbers can help you find errors in a program.

Underflow--The condition that occurs when the computer generates a nonzero numeric value greater than -1E-128 and less than 1E-128. When an underflow occurs, the value is replaced by zero.

Variable--A value that may vary during program execution. A variable is stored in a memory location and can be replaced by new values during program execution.

Variable-length records--Records in a file that vary in length depending on the number of data per record. Using variable-length records conserves space on a file. Variable-length records must be accessed sequentially. cap