TEXAS INSTRUMENTS

Bringing Affordable Electronics To Your Fingertips

Home Computer

Software Development System

Programmer's Guide

ORIGINAL ISSUE 15 AUGUST 1979
REVISED 6 NOVEMBER 1979

Personal Computer Division

November 6, 1979

Dear User,

The contents of this manual have been reviewed by members of the Personal Computer Division for clarity, correctness and completeness. If you find contradictions to any claims made in this manual, or have any suggestions or corrections, please let us know on the User's Response Sheet found in the back of this manual. Send your response to:

Texas Instruments Incorporated
Personal Computer Software Manager
RE: Personal Computer Division Manuals
P.O. Box 10508, M/S 5890
Lubbock, Texas 79408

Thank you.

Table of Contents

1.0	Introduction	1
2.0	Applicable Documents	2
3.0	The GPL Assembler	3
3.1	GPLASM Command	3
3.2	GPLXREF Command	3
4.0	Graphics Language Linker	5
4.1	Operation	5
4.2	Error Indications	5
5.0	990/10 Graphics Language Emulator	6
5.1	Operation	6
5.2	Debug Features	8
5.3	Keyboard Codes	10
5.4	Error Messages	11
6.0	990/10 Graphics Language Debugger	12
7.0	BASIC Program Conversion for GROM	13
7.1	Functional Description	13
7.2	Use	14
7.2.1	General Operation	14
7.2.2	Extended Features	15
7.2.2.1	Program Name	15
7.2.2.2	Duplicate Lines of Basic	15
7.2.2.3	Large Programs	16
7.2.2.4	Custom Start-Up Code	16
7.2.2.5	Disable Variable Name Replacement	17
7.3	Factors of Program Size	18

1.0 Introduction

The Graphics Language Assembler, Linker and Simulator are used to create Graphics Language programs and to test them. The Assembler is used to translate GPL source code into a machine executable object code and to produce a listing for the programmer. The Graphics Language Linker is used to concatenate two or more Graphics Language (GPL) object modules together. The linked object module or a single object module can be tested with the GPL simulator on the 990/10 to detect errors.

After the software has been prepared with the GPLASM or BASGROM command, the GPL debugger is used to load, execute, and debug the software on a GROM simulator. The BASGROM command is used to prepare a BASIC program to execute in a GROM simulator.

2.0 Applicable Documents

Graphics Programming Language
Programmer's Guide

Graphics Programming Language
Debugger Operation Guide

3.0 The GPL Assembler

The GPL Assembler is written in a mixture of FORTRAN and assembly language, and is available for use on the 990/10. The assembler translates a GPL source program into an executable object code and also produces a listing file. The assembler is executed via the GPLASM command. The prompts for the GPLASM command are described in the following section. A cross reference program is also available to provide a listing of all symbols defined in a GPL assembly and all references to those symbols. The GPL cross reference program is invoked by the GPLXREF command and is also described in a later section.

3.1 GPLASM Command

The GPLASM command, when entered, will display the following prompts:

```
GRAPHICS LANGUAGE ASSEMBLER
                SOURCE FILE: UNIT21
                OBJECT FILE: UNIT20
                  LIST FILE: UNIT22
PAST/CURRENT/FUTURE (P,C,F): C
```

The SOURCE FILE should be the Graphics Language source file created using the 990/10 text editor. The OBJECT FILE is the file which is to receive the object code from the assembler. The LIST FILE is the file which is to receive the listing generated by the assembler. If the object and/or listing files do not exist, they are created by the assembler. If either file exists, it is replaced by the new file generated by the assembler.

UNIT20, UNIT21, and UNIT22 are synonyms supplied by the GPLASM procedure file and will be changed when the source, object and listing files are specified.

3.2 GPLXREF Command

The GPLXREF command is used to produce a cross reference listing of the symbols used in a GPL program. When the GPLXREF command is entered into the 990/10 the following prompt is displayed:

```
GRAPHICS LANGUAGE CROSS REFERENCE
      LIST FILE: UNIT27
```

The LIST FILE specified is the file to which the cross reference is to be written. The cross reference produced is for the GPL program most recently assembled on the user's particular terminal. Therefore, if a cross reference is desired, it must be done before any other GPL assembly is done. It must also be

done before the station is signed off the system.

UNIT27 is a synonym supplied by the GPLXREF procedure and is changed when the cross reference listing file is specified.

4.0 Graphics Language Linker

The linker accepts a list of files containing Graphics Language object modules and concatenates those modules together to form one linked object module. This makes it possible to code, assemble and, debug a program in small or moderate sized parts, and then combine the parts afterward. Also, a program must be linked to the monitor before it can be run on the 990/10 Emulator.

4.1 Operation

The list of files to be concatenated is contained on a control file which is created with the text editor. Each record of the control file contains the name of one object file. The object file name starts in column 1 of the record and cannot use synonyms. A sample control file follows:

```
.HCDEV.CONSOL10
HC2.DEMO.OBJ.SPOTLOOP
HC2.DEMO.OBJ.SALESMSG
HC2.DEMO.OBJ.QUIKDEMO
HC2.DEMO.OBJ.SUBDATA
```

The linker is executed with the LINKGPL command. This command asks for the name of the control file and the name of an output object file.

```
LINK GRAPHICS LANGUAGE PROGRAMS
        CONTROL FILE:
         OBJECT FILE:
```

the output file need not be created before the linker is run. If it already exists, it will be replaced with the new output. Synonyms may be used in the names of these two files as in other SCI commands.

The GPL linker runs in the background mode to allow other terminal activity during processing. Successful execution is indicated with the message "GPL LINKER NORMAL COMPLETION". When completed the output object file may be used by the simulator.

4.2 Error Indications

Error conditions are reported with the termination message "XXXX ERROR ENCOUNTERED". A description of the cause of the error code "XXXX" can be found in Volume VI, Chapter 6 of the DX10 manuals.

4.0 Graphics Language Linker

The linker accepts a list of files containing Graphics Language object modules and concatenates those modules together to form one linked object module. This makes it possible to code, assemble and, debug a program in small or moderate sized parts, and then combine the parts afterward. Also, a program must be linked to the monitor before it can be run on the 990/10 Emulator.

4.1 Operation

The list of files to be concatenated is contained on a control file which is created with the text editor. Each record of the control file contains the name of one object file. The object file name starts in column 1 of the record and cannot use synonyms. A sample control file follows:

```
. HCDEV. CONSOL10
HC2. DEMO. OBJ. SPOTLOOP
HC2. DEMO. OBJ. SALESMSG
HC2. DEMO. OBJ. QUIKDEMO
HC2. DEMO. OBJ. SUBDATA
```

The linker is executed with the LINKGPL command. This command asks for the name of the control file and the name of an output object file.

```
LINK GRAPHICS LANGUAGE PROGRAMS
        CONTROL FILE:
         OBJECT FILE:
```

the output file need not be created before the linker is run. If it already exists, it will be replaced with the new output. Synonyms may be used in the names of these two files as in other SCI commands.

The GPL linker runs in the background mode to allow other terminal activity during processing. Successful execution is indicated with the message "GPL LINKER NORMAL COMPLETION". When completed the output object file may be used by the simulator.

4.2 Error Indications

Error conditions are reported with the termination message "XXXX ERROR ENCOUNTERED". A description of the cause of the error code "XXXX" can be found in Volume VI, Chapter 6 of the DX10 manuals.

5.0 990/10 Graphics Language Emulator

The GPL Emulator allows for the interactive debug of GPL programs on the 990/10 computer. Before a program can be run on the emulator, it must be linked with the file .HCDEV.CONSOL10 containing the console software. This file should be the first one listed in the control file.

5.1 Operation

The Graphics Language Emulator is executed with the GPL10 command. This command will request the name of the file containing the linked object code.

```
GRAPHICS LANGUAGE EMULATOR
        OBJECT FILE:
```

When execution begins, the first record of the object module is displayed on the bottom line of the screen. When the module is loaded, the screen will be formatted to simulate the TV screen. Two columns of twelve lines are used on the 913 terminals, and one column of 24 lines on the 911 terminals. In either case the right side of the screen is used for debug display.

The execution of GPL instructions is done as on the home computer with the following exceptions;

1) MOVEs to the VDP registers are ignored.
2) The defined character set consists of both the upper and lower case ASCII character set and cannot be changed (an attempt will change VDP RAM but not the display), although characters can be moved from one portion of the character set to another. See Table 5.1 for a list of corresponding hexadecimal values and ASCII characters. Note that the lower case characters are available only on the 990/10 emulator and cannot be used on the 99/4.
3) The SCAN instruction does not return with a no-key-down indication unless the debug mode is entered. This means that timer loops, etc. will not work.

HEX	CHAR	ASCII	HEX	CHAR	ASCII	HEX	CHAR	ASCII
00	40	A0	@	20	80	E0	space	60	C0	`
01	41	A1	A	21	81	E1	!	61	C1	a
02	42	A2	B	22	82	E2	"	62	C2	b
03	43	A3	C	23	83	E3	#	63	C3	c
04	44	A4	D	24	84	E4	$	64	C4	d
05	45	A5	E	25	85	E5	%	65	C5	e
06	46	A6	F	26	86	E6	&	66	C6	f
07	47	A7	G	27	87	E7	'	67	C7	g
08	48	A8	H	28	88	E8	(	68	C8	h
09	49	A9	I	29	89	E9	)	69	C9	i
0A	4A	AA	J	2A	8A	EA	*	6A	CA	j
0B	4B	AB	K	2B	8B	EB	+	6B	CB	k
0C	4C	AC	L	2C	8C	EC	,	6C	CC	l
0D	4D	AD	M	2D	8D	ED	-	6D	CD	m
0E	4E	AE	N	2E	8E	EE	.	6E	CE	n
0F	4F	AF	O	2F	8F	EF	/	6F	CF	o
10	50	B0	P	30	90	F0	0	70	D0	p
11	51	B1	Q	31	91	F1	1	71	D1	q
12	52	B2	R	32	92	F2	2	72	D2	r
13	53	B3	S	33	93	F3	3	73	D3	s
14	54	B4	T	34	94	F4	4	74	D4	t
15	55	B5	U	35	95	F5	5	75	D5	u
16	56	B6	V	36	96	F6	6	76	D6	v
17	57	B7	W	37	97	F7	7	77	D7	w
18	58	B8	X	38	98	F8	8	78	D8	x
19	59	B9	Y	39	99	F9	9	79	D9	y
1A	5A	BA	Z	3A	9A	FA	:	7A	DA	z
1B	5B	BB	[	3B	9B	FB	;	7B	DB	{
1C	5C	BC	\	3C	9C	FC	<	7C	DC	|
1D	5D	BD	]	3D	9D	FD	=	7D	DD	}
1E	5E	BE		3E	9E	FE	>	7E	DE	~
1F	5F	BF	_	3F	9F	FF	?	7F	DF	space

Table 5.1

5.2 Debug Features

The area on the right side of the CRT screen is used for debug interaction. The screen lines and their use is listed below.

LINE	USE
0	GPL program counter
1	program status
2	breakpoint address
3	step mode indication
4	second keyboard function indication
5-10	debug command interaction
11	where cursor is positioned when input is needed

The GPL program counter (on line 0) is updated every 100 GPL instructions, whenever a scan is done, when a breakpoint is taken, or when a program error is detected. Line 1 indicates the program status with the following messages:

PAUSED	waiting for input of debug key
BREAKPOINT	breakpoint has been taken
* ERR ADDR *	GPL address error at the instruction identified by the PC
* OP ERROR *	undefined GPL instruction at the address identified by the PC
* END VECT *	undetected GPL error - see 5.4
(blank)	program is running (if the cursor is at the lower right corner of a 913 or center right edge of a 911, a SCAN instruction is executing)

Certain keyboard keys are used for debug purposes. These keys are tested every 100 GPL instructions, whenever a SCAN instruction is executed, and when the debug mode is executing. The debug mode is executing whenever line 1 of the debug column is not blank. The debug mode may be entered before program execution begins by pressing one of the following debug keys while the GPL program is being loaded (as evidenced by the header record being displayed on the bottom line). The debug keys and their use are:

913	911	
F0	CTRL 1	Quit simulation
F1	F1	Resume simulation
F2	F2	Second keyboard function
F3	F3	Modify program counter
F4	F4	Set breakpoint
F5	F5	Step one GPL instruction
F6	F6	Inspect and modify memory

When hexadecimal numbers are requested, the user enters a string of hexadecimal characters terminated by a non-hexadecimal character (usually a space or new-line). The last 2 or 4 characters entered are used depending on whether one byte or two are needed. Thus if an error is made entering a number, more characters are entered to effectively throw away the previous ones.

The F0 key (CONTROL-1 on the 911) terminates the GPL simulator which is indicated by the message "GPL SIMULATOR - NORMAL TERMINATION".

The F1 key is used to resume the simulation when line 1 of the debug display shows the "PAUSED" message.

The F2 key sets the keyboard in the second function mode. This mode allows a character set extension for control characters and applies only to SCAN instructions to keyboard 0. The defined control characters are listed below (see 5.3). The second function mode is turned off when a character is entered on keyboard 0 or when the F2 key is entered again. Since the second function key causes GPL simulation to resume immediately, care should be taken that this key is used only when a SCAN is being done.

The F3 key prompts for a value for the GPL Program Counter. If no value is entered the PC is not changed. After changing the PC the emulator displays the "PAUSED" message and waits for another function key.

The F4 key prompts the user for a breakpoint address. When entered, the breakpoint address will be displayed on the screen. If no value is entered, the breakpoint is set to location 0000 which is indicated by a blank breakpoint address (line 2). After breakpoint entry the debugger will display the "PAUSED" message and wait for another function key. The breakpoint address (line 2) is compared to the GPL PC at each instruction execution. When a match is found, a breakpoint is taken and indicated in line 1. The match must be on the address of the first byte of an instruction.

The F5 key will cause immediate execution of one Graphics Language instruction after which a breakpoint will be taken independent of the current breakpoint address. The execution in step mode is indicated by a "STEP MODE" message on line 3 of the debug display. If this message does not immediately go away, the program is executing a SCAN instruction, and a key must be entered to the SCAN before the step mode will complete.

The F6 key will first ask the user whether (1)CPU (processor RAM), (2)VDP (RAM), or (3)GROM (ROM) is to be displayed. The user enters a 1, 2, or 3 accordingly. The user is then prompted for the address to be displayed. If an invalid

address is entered the user is re-prompted for the address. When the address is entered the data for that location is displayed. The user may change the data by entering new data, may proceed to the next location (+ key) or the previous location (- key). Entry of any other key will cause the debugger to enter the "PAUSE" mode.

5.3 Keyboard Codes

When keyboard 0 is SCANned, the ASCII characters on the terminal keyboard (including lower case on the 911) may be entered. When keyboards 1 or 2 are SCANned, only certain characters are used. These characters and the codes produced by them are listed in table 5.3 below.

When the second function flag is set (as indicated by line 4 of the debug display), certain keys on the keyboard are used to provide special key codes. The second function characters on keyboard 0 SCAN are as follows:

KEY	SECOND FUNCTION	CODE
A	Aid To User	>01
C	BASIC BREAK/Clear	>02
D	Forward Space	>09
E	Up Arrow	>0B
F	Delete Character	>03
G	Insert Character	>04
R	Redo Screen	>06
S	Back Space	>08
T	Clear Line	>07
V	Program Definable	>0C
W	Restart GROM Prog	>0E
X	Down Arrow	>0A
Z	Previous Screen	>0F

Table 5.2

To simulate the handheld keyboards, certain keys are used

or the terminal keyboard. For a SCAN instruction on other than keyboard 0, these keys used and the equivalent handheld functions are:

913 or 911 KEY	HANDHELD KEY	KEY CODE
0-9	0-9	0-9
/	divide	10 (>0A)
*	multiply	11 (>0B)
-	-	12 (>0C)
+	+	13 (>0D)
=	=	14 (>0E)
.	.	15 (>0F)
N	next	16 (>10)
S	set.	17 (>11)
G	go	18 (>12)
C	clear	19 (>13)

Table 5.3

If any other key is entered when the handheld is being SCANed, a no-key-down indication is returned and the key is ignored.

5.4 Error Messages

GPL execution error messages are described in 5.2. Some undetected GPL errors cause abnormal termination of the simulator and the displaying of the "* END VECT *" message. When this occurs, all of the debug features are still available; however, if the program is restarted by modifying the Program Counter and the error occurs again, the task will be aborted and control will be returned to the SCI of the 990/10.

Other error messages can be displayed if an error is encountered while the GPL program is being loaded into memory. The message "XXXX ERROR ENCOUNTERED" indicates a supervisor call error. The cause can be determined by reference to Volume VI, Chapter 5 of the DX10 manuals. The message "BAD DATA FORMAT" refers to an error reading the GPL object. This error occurs when the load file does not contain GPL object code.

6.0 990/10 Graphics Language Debugger

The GPL debugger provides the human interface necessary to LOAD, EXECUTE, and DEBUG GPL software in a GROM simulator. When the software has been prepared with the GPLASM (Section 3.1) or BASGROM (Section 7.0) command, the programmer is ready to start the debug session. To begin debug, the command DEBUG is entered. This provides the user with the following prompt screen:

```
GPL DEBUGGER
      USER MEMORY (K): 8
      COMMAND LIBRARY: .HCDEV.COMLIB
```

The default memory of 8K bytes is usually sufficient so that only the ENTER key is necessary. The Command Library Module file name is provided by default so you only need to press the ENTER key. The Command Library file is a source file of debugger PROCs designed to aid the programmer in the debug process. Each programmer may compose his own set of PROCs and enter that file name after the COMMAND LIBRARY prompt. It is not necessary to load a Command Library, but it is helpful to be able to use the PROCs it provides. If a Command Library file name is given, there will be a pause before execution of the debugger begins.

The debugger then starts execution and the user is prompted (with ?) for commands to:

- Connect the debugger to a particular system simulator.
- Load programs into simulated GROM.
- Debug programs with HALT, RUN, BREAK, UNBREAK, and inspect/modify memory.

More detailed instructions can be found in "Graphics Programming Language Debugger Operation Guide."

7.0 Conversion of BASIC Programs for GROM

This section describes the capabilities of and procedures for preparation of a BASIC program to execute in a GROM simulator on a Software Development System. This procedure applies to program development with a 990/10 computer and a Home Computer Simulator (which may consist of a Home Computer and a GRAM simulator). This section is specifically for use by a person developing a BASIC program for a solid state Command Module (GROM).

7.1 Functional Description

These procedures allow a programmer to enter a BASIC program onto a 990/10 disk file and convert it into a form which can be loaded into and executed from a GROM simulator. When testing is complete, the resulting code can be placed in GROM chips for mass distribution as a solid state Command Module.

Section 7.2 describes the use of the BASGROM command and options provided to customize the resulting GROM code and eliminate copies of duplicate lines. Programs are stored most efficiently if short variable names are used. The BASGROM program will convert variable names into short names so that the programmer can write readable code (see Section 7.2.2.6).

The BASGROM command produces four segments of code:

- GROM header
- BASIC text
- BASIC line number table
- Start-up code

The GROM header is always placed in GROM 3. The programmer has some control over the contents of the GROM header (Sections 7.2.2.4, 7.2.2.5)

The BASIC text consists of the converted lines of BASIC code prepared for the GROM. More detail of the contents of this text is given in Section 7.3. This BASIC text normally starts immediately after the GROM header but the placement in memory can be altered with the "REM +" statement described in Section 7.2.2.3.

The BASIC line number table follows the program text. This consists of four bytes for each BASIC program line and contains the line number and a pointer to the text for that line. The line number table must be contiguous and must not cross a GROM boundary.

When a GROM BASIC program is selected on the Home Computer menu, GPL code is executed before the BASIC Interpreter takes control. A default version of this start-up code is provided by

the BASGROM program unless an option is selected (Section 7.2.2.4).

7.2 Use

The use of the BASGROM utility program is described and various options are discussed in detail.

7.2.1 General Operation

The conversion of a BASIC program into GROM code is accomplished with the BASGROM command on the 990/10 computer. This command initiates a two phase process. The first phase is a conversion of the BASIC source into a form which can be assembled by the GPL assembler. The second phase is a GPL assembly which produces an object module which can be loaded into a simulator. This process is illustrated in Figure 7.2.1.

The BASGROM command prompts are as follows:

```
CONVERT BASIC FOR GROM
        BASIC SOURCE FILE:
GPL SOURCE (OUTPUT):
    OBJECT (OUTPUT):
GPL LISTING (OUTPUT):
```

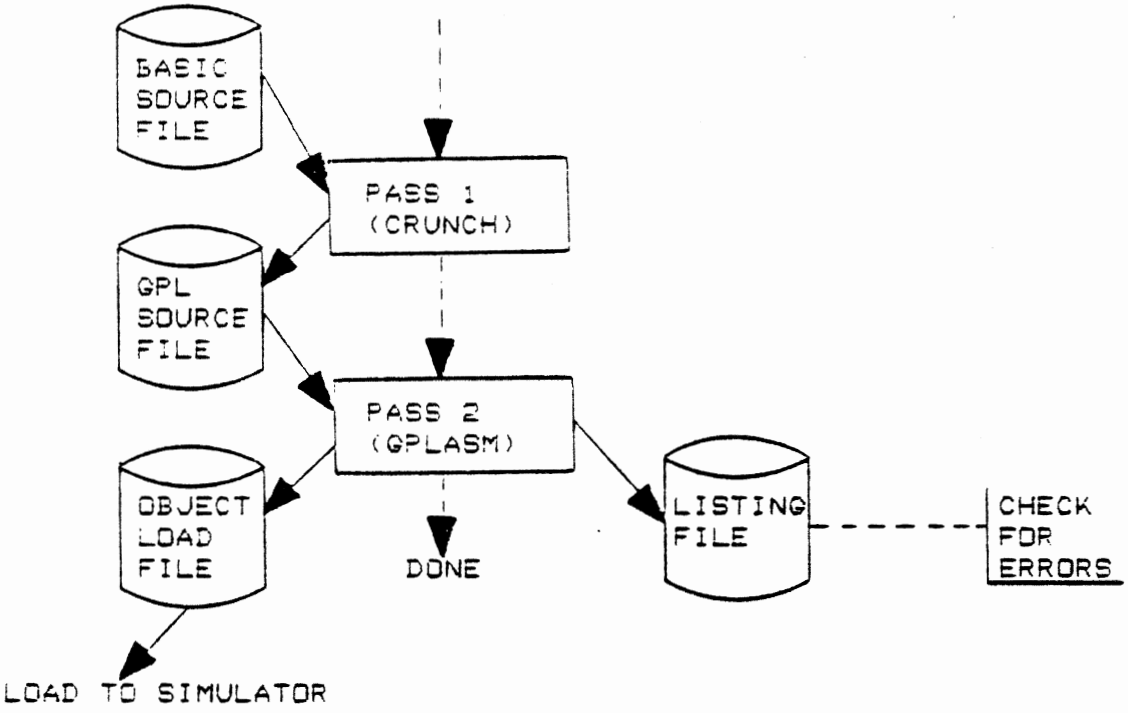
The BASIC SOURCE file must include line numbers starting in column 1 and must be ordered in ascending line number order. This is as a program listing would be. The END statement is not required. REM statements cannot be referenced in other statements such as GOTO, etc.

The GPL SOURCE file is a temporary file used to pass data from phase one to phase two.

The GPL OBJECT file is the data that can be loaded into a simulator for execution.

The GPL LISTING file contains a listing result of phase two. It is useful if any errors were encountered during phase two.

After entering all of the required file names, phase one of the crunch will execute in the foreground mode. The GPL assembly which follows executes in the background mode. The WAIT command is convenient to wait for phase two completion. When the GPL assembly is complete, a message is displayed on the CRT to indicate if any errors have been encountered.



Action of BASGROM Command
Figure 7.2.1

7.2.2 Extended Features

Certain features are provided in BASGROM through the use of REM statements. Certain types of REM statements control program name, duplicate lines, placement of code, start-up code, and variable name replacement. Each of these REM statements has a special character as the first character following the REM.
IMPORTANT NOTE: The ampersand (&) should never be used as the first symbol in a REM statement.

7.2.2.1 Program Name

ln REM *name (ln = line number)

The asterisk indicates that the next 22 characters are to be used as the program name for the Home Computer menu.

7.2.2.2 Duplicate Lines of Basic

ln REM/ln2

This statement specifies that the statement immediately following the REM/ statement is identical to the statement at

ln2. The BASGROM program will not produce text for the succeeding statement. The line number table entry for the succeeding line will point to the text for ln2. Refer to section 7.3 for a description of the memory representation of a GROM BASIC program.

7.2.2.3 Large Programs

ln REM +

The BASGROM command produces a GROM header in GROM 3. The BASIC program will follow that GROM header. The REM + statement will cause subsequent lines of the BASIC program to be placed in the next sequential GROM (the first REM + will start at GROM 4, ORG 0). The BASIC line number table (4 bytes for each BASIC statement) is produced at the end of the BASIC program.

7.2.2.4 Custom Start-Up Code

ln REM ^ address

The address is a five-character hex number (e.g., >8020). BASIC programs in GROM are initiated by a short segment of GPL code. Control returns to the GPL code after the program completes either normally or by an error. If an error has occurred, the condition bit will be set on return from BASIC. If the above statement is not included, BASGROM will include the following start-up code with the BASIC program.

```
START1 BR     START2
LOOP1  SCAN
       BR     LOOP1
       EXIT
START2 MOVE 1 FROM ROM(#H20) TO VDP (1)
       MOVE 21 FROM ROM (#OMPMES) TO RAM(>185)
       B      START3
OMPMES DATA :ONE MOMENT PLEASE...:
H20    DATA >20
```

If a REM ^ statement is included, only the following is generated:

```
START1 EQU address
```

Then the programmer must supply start-up code of the form:

```
START1   BR      START2      ENTRY FOR START UP
*                              ENTRY FOR END OF PROGRAM
         BR      END2        TEST PGM ERROR FLAG
LOOP     SCAN                IF EROR, WAIT
         BR      LOOP
END2     EXIT

START2   (START UP CODE)
         B   >6010           GO TO CALL BASIC
```

Execution will be at START1 before BASIC program execution and at START1+2 at BASIC program termination. Therefore, the BR instruction at START1 is required. The actual code located at START2 depends on the individual program requirements.

7.2.2.5 Disable Variable Name Replacement

In REM -

The REM - statement will disable the substitution of minimal length variable names for the user's variable names. This is valuable when using the program transfer feature described in Section 7.2.3. This statement must be entered previous to any use of a variable name in the BASIC program. If variables have been used before this statement occurs, an error will be produced and the statement will be ignored.

7.3 Factors of Program Size

The following items give an overview of the space necessary for various items of BASIC syntax. This is to assist in keeping BASIC programs as short as possible.

Each BASIC program line has 4 bytes of overhead in the line number table. Each program line which is not a duplicate of another has a text string representing the program line. This text string has one byte of overhead to represent the end of line. The lengths of other items are:

Key words - 1 byte (GO TO are two key words but GOTO is one).

Variables - 1 byte for each letter in the variable name. The BASGROM command will convert user variable names into short names.

Numeric constants - 2 bytes plus the number of characters in the constant. Because of this, it can be more efficient to use variables for often-used constants.

String Constants - 1 byte for each character including quotes.

Line numbers in GOSUB, GOTO, etc. - 3 bytes for each line number

HOME COMPUTER SOFTWARE DEVELOPMENT SYSTEM

User's Name _____ Telephone _____

Company _____ Date _____

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. Additional sheets may be attached. If there are any other suggestions that you wish to make, feel free to include them.

Thank you.

LOCATION IN MANUAL	COMMENT/SUGGESTION
_____	_____
	_____
	_____
_____	_____
	_____
	_____
_____	_____
	_____
	_____
_____	_____
	_____
	_____
_____	_____
	_____
	_____
_____	_____
	_____
	_____