(TI WRITER Version 4 Required)

INTRODUCTION

This manual and the RAGPATCH program described in it are PUBLIC DOMAIN. They are being distributed in order to provide a uniform means of distributing patches to GROM/GRAM modules and to Assembler Language program modules. The patch program also provides an organized way of keeping track of patches you have made to programs. The patches can be made from statements in a file. Even though the program modules being patched may be copyrighted and cannot be distributed, your patch file and the patch utility can be distributed as you see fit.

You are requested to distributed only complete copies of the Patch Utility. There are three files:

PATCHDOC	The documentation
RAGPATCH	The Patch Program
XBPATCH	Extended BASIC Loader

Any errors, omissions or contributions can be mailed to:

RAG SOFTWARE
R. A. Green.
1032 Chantenay Dr.
Gloucester, Ont.
CANADA
K1C 2K9

THE PATCH UTILITY

The Patch Utility, RAGPATCH, gives you an organized means of making patches to a GPL program in GRAM or to an Assembler Language program on disk. These patches can be modifications, bug fixes or installation options. The utility can read patches from the console keyboard or from a "Patch File". The patch file, which you create with a text editor, for input to the patch utility also serves as a permanent record of what you have patched.

The Patch Utility is a standard E/A Option 5 assembler language program that can be loaded and run using any E/A Option 5 loader. An Extended BASIC loader is supplied. The program is independent from the means of loading it. The patch utility will prompt you for the name of the patch file. If no patch file name is given, the patch statements will be read from the console.

The Patch File

The patch file, created with any text editor, can contain eight types of statements, three to identify the destination or target of the patches: GRAM, FILE and CFILE; three to specify the patches: EXPAND, PATCH and VERIFY; and two for notes: SAY and Comments. The entries in the patch file may be in either upper or lower case. Comments are identified by an asterisk in column one. The other statements have the following formats:

```
GRAM   type,page
FILE   filename
CFILE  filename,length
EXPAND amount.
PATCH  addr,data,data,..
VERIFY addr,data,data,...
SAY    anytext
```

The keyword identifying the statement must begin in column one and must be followed by at least one blank. The operand field is terminated by the end of the statement or by a blank.

The GRAM statement identifies the destination of the patches to be a GRAM device. The "type" specifies the type of GRAM device. It is specified as:

```
GK  Gram Kracker.
GU  Gramulator.
PG  P-Gram or P-Gram+ Card.
```

The GRAM type is used to enable the device for GRAM writes if no GRAM type is specified then no special action is taken. If "type" is PG (for the P-Gram+ Card) then the "page" operand is processed. It is the GRAM page to be patched, specified as 1 to 4.

Note that a GRAM/GROM program saved to disk is identical in form to a standard E/A Option 5 program and can be patched on disk using the FILE statement.

The FILE statement names the standard E/A Option 5 program file on disk to be patched. The three word header from the file will be displayed when the file is read. A complete Assembler Language program may consist of more than one segment, each in a separate file. Each file must be patched separately.

The CFILE statement names a non-standard program file on disk to be patched, and gives the length of the program segment contained in the file. Non-standard, or "custom" program files have no three word header. Addresses for custom program files always begin at zero. The length is

specified in hexadecimal. The ">" usually used to indicate hexadecimal notation is optional.

The EXPAND statement allows the length of the module to be changed by the "amount" specified. The amount is specified in hexadecimal. The ">" usually used to indicate hexadecimal notation is optional. The amount is specified in two's complement notation. A negative amount shortens the program (i.e. >FFFE is -2).

The PATCH and VERIFY statements are similar. The first operand, "addr", is the address at which the patch or verification is to begin. The address is specified in hexadecimal. The ">" usually used to indicate hexadecimal notation is optional.

The one or more "data" items may be specified as either an even number of hexadecimal digits (with or without the ">") or as text enclosed in single (') or double (") quotes. Within the text an occurrence of the enclosing quote is represented by two consecutive quotes.

The VERIFY statement causes the data in the program file to be compared to the specified data. If a comparison is not equal then all following PATCH statements are processed but no patches are made. Additionally, if any syntax errors are detected then all following statements are processed but no patches are made.

The SAY statement simply prints up to 28 characters of the text on the screen. This could be used to inform the console operator what is being patched.

.....