REIS-WARE COMPUTER PRODUCTS

REFERENCE MANUAL FOR THE GPL-ASSEMBLER AND GPL-DISASSEMBLER

IMPRINT

This manual was developed and written by

Michael Humbert

Important notice:

Author and editor are not liable for anyone for damages arising out of the purchase [illegible] use of these programs or book materiels

Publisher:

Reis-Ware
Computer Products GmbH
Postfach 36
D-5584 Bullay

Printed in Germany

TABLE OF CONTENTS

PART 1: GPL-ASSEMBLER

4 1. Source code

5 2. Valid instructions

5 2.1 List of instructions
5 2.2 Pseudo instructions

6 3. Addressing modes

6 3.1 General information
6 3.2 Direct addressing mode
6 3.3 Indirect addressing mode
6 3.4 Indexed addressing mode
7 3.5 Immediate operands

8 4. Instruction formats

8 4.1 Format I
8 4.2 Format II
8 4.3 Format III
8 4.4 Format IV
8 4.5 Format V
9 4.6 Format VI
9 4.7 Format VII
12 4.8 Format VIII
12 4.9 Format IX

13 5. How to load and start the GPL-Assembler

13 5.1 Required equipment
13 5.2 Loading the program
13 5.3 The main menue
14 5.4 Error reports
14 5.5 End of the program

PART 2: GPL-DISASSEMBLER

15 1. Required equipment

15 2. Loading the program

15 3. The function keys

16 4. Main menue

17 5. General information

PART 1: GPL-ASSEMBLER

1. Source code

The source code has to be in DIS/VAR 80 format on diskette.It must have been written under use of the instructions and/or pseudo instructions which are mentioned behind pos. 2. and must correspondend to the following line:

LABEL INSTRUCTION SOURCE OPERAND, DESTINATION OPERAND, OPERAND (if required)

The use of the LABEL field is optional. It begins in the first character postion of the source record and extends to the first blank. When the label is omitted, the 1st character position must contain a blank. The label field consists of a symbol containing up to six characters, the first of which must be from A-Z. The label must be followed by a blank. This makes the LABEL field a maximum length of 7 characters (incl. blank). Rows consisting only of a label aren't admited. Per program only up to 599 labels are allowed.

In the OPERATION field only the use of the instructions/ pseudo instructions named under pos.2 is valid. It must be followed by a minimum of 1 blank. It's maximum length is 6 characters incl. the following blank(s).

Source operand, destination operand and/or operand only can be used, according to the requirements of the particular instruction and addressing-modes (pos. 3).They must be followed by a minimum of 1 blank or an asterisk. The rest of the source record can be used for comments.

Putting an asterisk into the first character of a source record makes it become a comment record. Whole of the record then can be filled with commands or blanks.

Hexadecimal integer constants must be preceded by a greater than sign(>). Decimal integer constants don't need a special identification sign. The use of hexadecimal or decimal integer constants is optional if no special instruction t to use hexadecimal constants.

Using an 'END' instruction at the end of the source code is not valid and will release an error message.

When the use of a label during addressing is valid, this label may not be changed by any addition like, for example, '+1' or '-5'. During a program it's only allowed to use a label for addressing for up to 1300 times.

2. Valid instructions or pseudo instructions:

2.1 List of instructions

Inst.	Format	Inst.	Format	Inst.	Format	Inst.	Format	Inst.	Format
ABS	V	COINC	I	DEC	V	EXIT	III	RTN	III
ADD	I	CONT	III	DECT	V	FETCH	V	RTNB	III
ALL	II	CZ	V	DINC	V	FMT	VII	RTNC	III
AND	I	DABS	V	DINCT	V	GT	III	SCAN	III
B	II	DADD	I	DINV	V	H	III	SLL	I
BACK	II	DAND	I	DIV	I	I/O	I	SRA	I
BR	IV	DCASE	V	DMUL	I	INC	V	SRC	I
BS	IV	DCEQ	I	DNEG	V	INCT	V	SRL	I
CALL	II	DCGE	I	DOR	I	INV	V	ST	I
CARRY	III	DCGT	I	DSLL	I	MOVE	VI	SUB	I
CASE	V	DCH	I	DSRA	I	MUL	I	SWGR	I
CEQ	I	DCHE	I	DSRC	I	NEG	V	XGPL	VIII
CGE	I	DCLOG	I	DSRL	I	OR	I	XML	II
CGT	I	DCLR	V	DST	I	OVF	III	XOR	I
CH	I	DCZ	V	DSUB	I	PARSE	II		
CHE	I	DDEC	V	DXOR	I	PUSH	V		
CLOG	I	DDECT	V	EX	I	RAND	II		
CLR	V	DDIV	I	EXEC	III	RTGR	III		

2.2 Pseudo instructions:

BYTE	IX	TEXT	IX	COPY	IX	WRITE	VII	BIAS	VII

. Addressing modes:

.1 General information:

t's possible to address the VDP-RAM and CPU-RAM/ROM. When addressing the VDP-RAM, the
'DP-address must be preceded by a 'V' sign. CPU-RAM/ROM-addresses don't need special
dentification-sign.

.2 Direct addressing mode:

n this mode it's possible to address the VDP-RAM and the CPU-RAM/ROM by writing the '@'-sign
between the 'V'-sign and the VDP-address or in front of the CPU-address.

xample:

```
      LABEL  CLR   @>8354  sets address >8354 to 0 (1 Byte).
             INC   V@>1000 adds 1 to the value in VDP-RAM address >1000
```

ddressing by use of a label isn't valid, for it is not possible to address the GROM/GRAM memory
n this mode.

.3 Indirect addressing mode:

n this mode it's possible to address the VDP-RAM and the CPU-RAM/ROM by writing the '*'-sign
between the 'V'-sign and the VDP-address or in front of the CPU-address.

xample:

```
      LABEL  CLR   *>8354  sets the contents of the address which is found in
                           CPU-address >8354 to 0.
             DEC   V*>8354 substracts 1 from the contents of the VDP-RAM address
                           which is found in CPU-address >8354.
```

Addressing by use of a label isn't valid, for it is not possible to address the GROM/GRAM memory
in this mode.

.4 Indexed addressing mode:

In this mode it's possible to address the VDP-RAM and the CPU-RAM/ROM by using an index. The
direct addressing mode is as well allowed as the indirect.

Example:

```
     LABEL  CLR   @>0005(@>8354) sets the contents of the address you get by
                                 adding the index to >8354, to 0.
            INC   *5(@>8354)     adds 1 to contents of that address you
                                 found in the address you get by adding the
                                 index to >8354.
            DEC   V*>1(@>8354)   subtracts 1 from that VDP-RAM address you
                                 found in the address you get by adding the
                                 index to >8354.
```

Addressing by use of a label isn't valid, for it is not possible to address the GROM/GRAM memory
in this mode.

3.5 Immediate operands:

Depending of the instruction in front, the use of byte or word operands is valid. If the use of
more than 1 operand is allowed, they have to be seperated by a comma.

Example:

```
     LABEL  BYTE  >A,15,1,>2       or
            BACK  >2               or
            ALL   32               or
            DST   @>8354,>1234
```

Exceptions:

The instructions 'B', 'CALL', 'BR' and 'BS' are jump or branch instructions duri the GROM/GRAM
memory. For these instructions this immediate addressing is compar to the direct addressing
mode (pos. 3.2) of the VDP or CPU memory. For this reason the immediate operands must be
preceded by the 'G@'-signs.
The immediate operand also can be a label which is used in the program. In this case the
declarations in pos. 1 had to be mentioned.

Example:

```
     LABEL  CALL  G@>0010
            B     G@108

     or:

     LABEL  CALL  G@LABEL1
            CALL  G@>30
            BS    G@>6010
            BR    G@LABEL
     LABEL1 RTN
```

4. Instruction formats:

4.1 Format I:

Form: instruction destination operand, source operand

The addressing mode of the destination and source operand is optional the direct, indirect or indexed addressing mode.

Example:

```
LABEL  ST    @>8358,*2(@>8354) moves one byte of the contents of the
                               address you get by adding the index to
                               >8354 to location >8358.
```

4.2 Format II:

Form: instruction immediate operand (1 byte)

Exception: The instructions B and CALL (see pos. 3.5)

4.3 Format III:

Form: instruction

The use of a operand is not valid.

Example:

```
LABEL  SCAN
       RTN
```

4.4 Format IV:

see the exceptions on pos. 3.5

4.5 Format V:

Form: instruction destination operand

The addressing mode is optional the direct, indirect or indexed mode.

Example:

```
LABEL  CLR   *>8354     sets the contents of the address you found in >8354
                        to 0 (1 byte).
```

4.6 Format VI:

Form: instruction number BYTE TO destination operand FROM source operand

The number of bytes to move can be given in the direct, indirect or indexed addressing mode or as an immediate operand (1 word). Pos. 3.1 has to be mentioned. Destination and source operands can be addressed in direct, indirect or indexed addressing modes.

Example:

```
LABEL  MOVE  >0001 BYTE TO @>8354 FROM *>2(@>8300)
```

Further, the following addressing modes are possible:

Source and destination operand: It is possible to address the GROM/GRAM memory either in the direct or in the indexed addressing mode (indexed mode only over CPU-RAM from >8300 to >83FF). The 'G' character must precede the address. Naming this address by a label which must have been used in the program is possible. In this case, you had to mention the declarations in pos. 1.

Only the destination operand: Using this operand allows you to address the VDP-Registers too. In this case the characters 'VR' must be preceding the VDP-Register number.

Example:

```
LABEL  MOVE  >0001 BYTE TO VR>01 FROM G@LABEL1(@>8320)
       MOVE  @>8320 BYTE TO V@>100 FROM G@>6020
       RTN
LABEL1 BYTE  >E0
```

4.7 Format VII:

FMT is a special instruction to display charcters at the screen. It's interpreted by a special interpreter in the console. Therefore it's only possible to set the FMT instructions by using the instructions BYTE, BIAS, WRITE TEXT in the following way:

Horizontal string display at the screen:

In this case the instruction is BYTE 0

Further on, the string length of 1 must be added to this value. The maximum length of a string is 32 characters. In this case the instruction had to be BYTE 31 or >1F. This byte must be followed by the string, which shall be displayed at the screen. This string can either be defined by using the TEXT instruction in the next source record or by using the BYTE instruction in the same and/or the next record(s). When using the TEXT instruction, the string expression had to be in "" signs. For string display only the ASCII characters from 32 to 127 are valid. So there are two different ways to display the string 'TI-99/4A' horizontal at the screen:

```
either:  LABEL  BYTE  7
                TEXT  'TI-99/4A'
or:      LABEL  BYTE  7,84,73,45,57,57,47,52,65
```

Vertical string display at the screen:

This is comparable to the horizontal display. Only the value, to which the string length-1 has to be added is not 0 but 32 or >20. This means the last example to be as follows:

```
either:  LABEL  BYTE 39
                TEXT 'TI-99/4A'
or:      LABEL  BYTE 39,84,73,45,57,57,47,52,65
```

Horizontal or vertical character repetation at the screen:

These cases are also comparable to string display. Only the value to which the number of repetations-1 must be added is 64 or >40 for horizontal and 96 or >60 for vertical display. The character which shall be repeated can be defined by using the BYTE or TEXT instruction.

Examples:

```
either:  LABEL  BYTE 83       repeats the letter 'A' 20 times horizontally
                TEXT 'A'
or:      LABEL  BYTE >53,>41
and:
either:  LABEL  BYTE >73      repeats the letter 'A' 20 times vertically
                TEXT 'A'
or:             BYTE 115,65
```

Relative setting of that row, in which the display shall start:

In this case the starting row of screen display will be set dependent on that row, that had been the last. The screen has a maximum of 32 rows, of which only the first 24 are visible. Starting value for this setting is 128 or >80. To this value the following difference must be added: new row minus old row minus one. For example the actual row is the row 0 and you want to start in row 10. In this case you had to add the difference from 10-0-1=9 to >80.

The corresponding instruction would be like this:

```
LABEL BYTE >89 changes from row 0 to row 10
```

Relative setting of that column, in which the display shall start:

This case is comparable to the setting of a new row. There are only two differences. The starting value is not >80 but 160 or >A0 and it is possible to display whole of the 32 columns at the screen.

Example:

```
LABEL BYTE >B1 changes from column 0 to column 18
```

Setting loops:

Starting value to which the number of loops-1 must be added is 192 or >C0. The maximum number of loops is 32. A loop ends while setting a value of >FB or 251, followed by that GROM/GRAM address, at which the loop starts again.

```
Example:  LABEL  BYTE >C1           * 2 loops
                   :
                   :
                   :
                 BYTE >FB,>01,>12   *End this loop and start the repetition
                   :                *at GROM/GRAM address >0112. After the
                   :                *2nd loop further on in program.
```

Fixed setting of row and column in which the display shall start:

Starting value to set the row is >FE or 254, followed by the new row number. To set a new column, this value is >FF or 255, also followed by the new row number.

Example:

```
LABEL  BYTE >FE,>A              *sets display start to row 10
LABEL  BYTE >FF,>0              *sets display start to column 0
```

Setting of screen offset:

For setting of a new screen offset, there a two ways. One of this is the direct way. In this way, the new offset (max. 255 or >FF) must be set after a value of 252 or >FC.

Example:

```
LABEL  BYTE >FC,>60             *screen offset will become 96 or >60
```

The other way is to set a CPU-RAM address, in which the offset (1 byte) is to be found. Setting of this CPU address can be in the direct, indirect or indexed addressing mode. The pseudo instruction BIAS had to be used in front of that CPU address.

Example:

```
LABEL  BIAS @>8354              *the screen offset is locatet in
                                *CPU-RAM address >8354
```

Horizontal string display at the screen directly from CPU-RAM/ROM:

Further on it is possible to display a 1 to 27 character long string, which is to be found in the CPU-RAM/ROM, at that screen location, which is set. This is possible under use of the following instruction:

```
WRITE  x BYTE FROM source. x means the string length-1 (max. 26).
                           The source operand can be addressed in
                           indirect, direct or indexed mode.
```

Ending FMT:

The way back from FMT to GPL-interpreter requires the following instruction:

```
    LABEL BYTE >FB
```

Notice:

All of these format VII instructions/pseudo instructions only show the described results under the supposition of activating the FMT-Interpreter by using FMT.

Example for a small FMT-routine:

```
    LABEL  FMT              leads to the FMT-Interpreter
           BYTE  >FE,12     set row 12
           BYTE  >FF,>3     set column 3
           BYTE  4          horizontal display of a 5 character long string,
           TEXT  'TI-99'    starting at row 12, column 3
           BYTE  >FB        end with FMT and back to the GPL-Interpreter
```

4.8 Format VIII:

The XGPL instruction must be followed by a code. Valid codes are from >14 - >1F, >[illegible] - >9F, >F0 - >F4 and >FC - >FF.

Example:

```
    LABEL  XGPL   >14   or
           XGPL   >FF   or
           XGPL   254
```

4.9 Format IX:

This is the description of the pseudo instructions as far as they had not been described during the other formats.

BYTE:

With the help of this pseudo instruction it is possible to define DATA ranges, header etc. Notice position 3.5.

TEXT:

It is possible to use this pseudo instruction as described in pos. 4.7. Further on it is possible to define DATA ranges, which contents consists of text. In this cases it is not necessary to set a length byte in front of the string like described in pos. 4.7.

COPY:

In every source code program part one COPY pseudo instruction is allowed. So the particular source code parts can be linked during assembling. This pseudo instruction had to be in the last line of a source code part. None line may follow this pseudo instruction, or it will be ignored. The file name has to be in quotation marks.

Example: `LABEL COPY "DSK1.PART2"`

5. How to load and start the GPL-ASSEMBLER

5.1 Required equipment:

```
    TI-99/4A and Editor/Assembler or Extendet Basic module
    + 32K Memory expansion
    + min. of 1 Diskdrive
    + optional 1 printer with RS232/PIO interface.
```

5.2 Loading the program:

- insert the E/A or XB module into the mudule slot and turn on the equipment as described in the manual
- choose the Editor/Assembler option 5 ("RUN PROGRAM FILE")
- type in the filename DSK1.GPLASS and press ENTER
- under use of the XB-module the programm loads automatically when the GPL-Disk is in drive 1 and XB has been selected for the first time. Another way to load the program from XB is to type "RUN DSK1.LOAD" and then pressing ENTER
- The programm starts automatically. Pressing ENTER selects the main menue

5.3 The main menue:

The main menue looks as following:

```
    Source code name
    Object code name
    Saving format
        1=DIS/FIX 80; 2=PROGFILE
    Start addresse
```

The 1st question requires the file name and the name of the source code

Example: DSK1.PART1

The 2nd question requires the same answers for the object code. The object code can be saved either on diskette or printer. The maximum length for the name and options of a printer is 22 characters.

Example:

```
DSK1.OBJECTCODE       or
PIO                   or
RS232.BA=9600.DA=8
```

Only if a disk drive is selected for saving the object code, the question for the saving format is interesting. When RS232 or PIO is chosen, the output is allways in FIX 80 format. Nevertheless, one of the both options must be chosen in that case. If another value then "1" or "2" had been used, the program switches automatically back to the start of the main menue, when all menue questions had been answered.

Finally there is the question for the start address of the object code in GROM/GRAM. The maximum length for one GROM/GRAM is >2000 bytes. This length will be found out depending of the start address. If a program starts e. g. at start address >1FF0, the maximum free memory space for this GROM/GRAM is >10 or 16 bytes. If the object code will become longer than this, there will be an error report and assembling stops. Further there will be a display of the actual program length.

5.4 Error reports:

If the assembler finds an error, there will be displayed an error message at the screen. It consists of the source record number in which the mistake had been found, and an error code. These codes have the following meanings:

```
0 600 label had been set or one of the instructions 'B', 'CALL', 'BS' or 'B
  had been used in connection with a label for more then 1300 times
1 a mistake in the COPY inst. (filename too long, no quotation marks etc.)
2 incorrect instruction (too many blanks, a blank line has been used,
  incorrect addressing mode, mistake by typing an instruction etc)
3 branching to a not existing label
4 wrong number (too big, no seperation by comma etc.)
5 not valid label or branching address (too long, wrong starting character
  etc.)
6 branching to an address which is outside of the used GROM/GRAM range
  (only when using BS or BR)
7 GPL-program leaves boundry of a GROM/GRAM (>1FFF, >3FFF, >5FFF, >7FFF,
  >9FFF, >BFFF, >DFFF or >FFFF) (assembling stops)
8 a label has been defined twice
```

5.5 End of the program:

At the end of the assembling, the length of the assembled program will be displayed at the screen. If using a printer, there will be a question for the sort of paper in use(single sheet or endless), if not, it's possible to switch back to the beginning of the main menue by pressing ENTER. Pressing FCTN 4 switches back to the E/A module and FCTN = returns to the master title screen.

PART 2: GPL-DISASSEMBLER

1. Required equipment:

TI-99/4A
+ Editor/Assembler or Extended Basic module
+ 32K RAM memory expansion
+ 1 disk drive
+ 1 printer with RS232 or PIO interface (optional)

Using a printer, it's recommended to select the American character set and to switch off the paper end signal, before loading the GPL-disassembler. These funktions are not implemented to get the best compatibility between the different printer models.

2. Loading the program:

- turn on equipment and console as discribed in the manual
- insert Editor/Assembler or Extended Basic module into the module slot
- select the Editor/Assembler option 3 'LOAD AND RUN'
- type in the filename DSK1.@GPLDIS and press ENTER
- using the XB module, the program loads automatically if the GPL diskette is in drive 1 and XB is selected. The program also may be loaded by typing "RUN DSK1.LOAD" and pressing ENTER.

3. Function keys:

```
- FCTN 4 = back to E/A module (not with XB)
- FCTN 8 = back to 2nd sub menue (only possible at the end of disassembling).
           Both, the options which had been selected in the main- and in
           the 1st sub menue, and those DATA ranges and subroutines containing
           FETCH routines will be preserved. Disk file names will be incremented
           by one.
- FCTN 9 = back to main menue
- FCTN = = back to the master title screen
- disassembling can be stoped/ continued by pressing any key
```

4. Main menue:

Output device:

1) DSK1
2) V24
3) Screen

Pressing one of the keys 1, 2 or 3 selects the corresponding output device.
Selecting "1" displays the question "NAME ?" at the screen, expecting the input of a filename, such as "SOURCE CODE" for example. For this disassembler only runs under diskdrive 1, you needn't type "DSK1." in front of this name.
Selecting "2" displays the question "PRINTER OPTIONS" at the screen, expecting the input these options, like "PIO" or "RS232.BA=9600.DA=8" for example. The next question asks, whether single sheet or endless paper is in use.

1st sub menue:

Read from:

1) CPU RAM
2) VDP RAM
3) GROM

Pressing one of these keys selects that memory range, in which the GPL-programm that shall be disassembled, is to be found. Normaly this is the GROM range. Nevertheless it is possible to disassemble GPL-programs, which had been loadet into the VDP or CPU RAM. In this reason the next question asks, in which GROM the GPL-program had been, before reading it out. GROMs 0 to 2 are built into the console and GROMs 3 to 7 are built into modules. The GPL-disassembler needs to know this, for this is the only way to calculate some jump addresses correctly.

2nd sub menue:

This is the question for start and end address (hexadecimal) of the program, which shall be disassembled.

3rd sub menue:

This is the question for the disassembler options, you want to get (not when diskdrive 1 had been selected). Possible options are:

1) displays each byte, found in memory, as hexadecimal integer constant

2) displays mnemonic code

3) quick-overview of memory contents. Displays the ASCII-characters that correspond to the memory contents. If they are not displayable, an "." will be shown. This option is interesting, for example, to found some special strings etc. in memory.

If option 2 had been selected it is possible to define up to 8 subroutines, containing a FETCH instruction (see pos. 5). Further on it is possible to define up to 8 ranges containing DATAs (see pos. 5 too). There will be no disassembling of these DATA ranges.

5. General information:

If a "G" character is preceding an address, a GROM address is meant. If the character is a "V", the address is in VDP-RAM, and if "VR" is preceding an address, a VDP-register is meant. None preceding character means a CPU-RAM/ROM address.

To get best results, the mnemonic code had to be checked step by step, whether the disassembled program parts really had been instructions, or if they only had been DATA ranges. For example, if an instruction is "MOVE >0008 BYTE TO G>6500 FROM G>6600", then the GROM addresses >6600 >6607 only contain 8 bytes of datas. To disassemble these 8 bytes would produce incorrect mnemonic instructions.

Further, subroutines had to be evaluated, whether they contain a FETCH instruction or not. If they do, the byte following the subroutine call, or if there is more than one FETCH instruction in the subroutine, the number of bytes corresponding to the number of FETCH instructions in the subroutine, only contain datas. Only the byte following now is an "instruction byte" again.