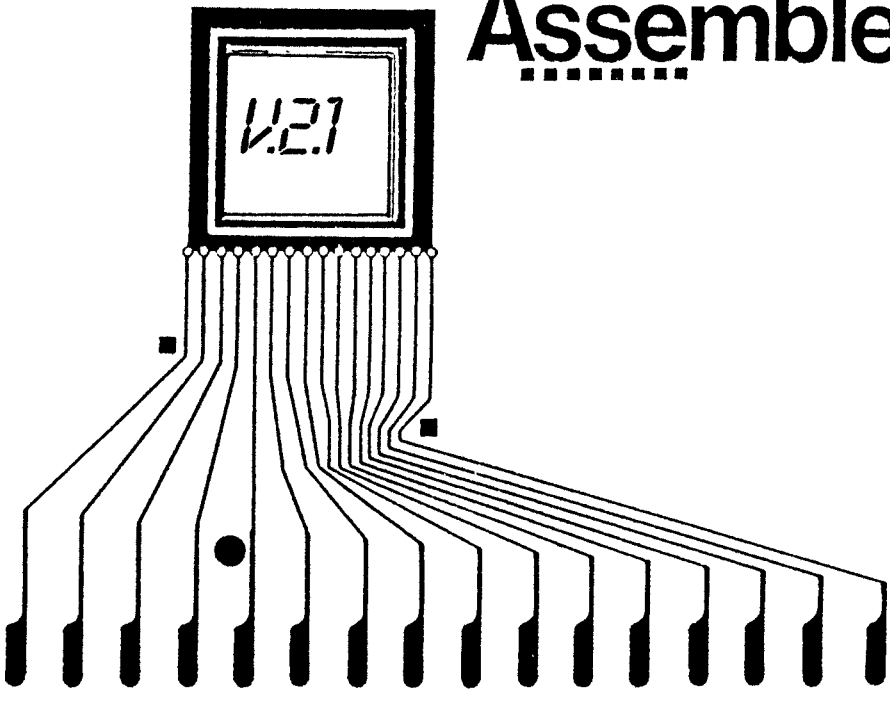
GPL Assembler



Program Manual

GPL-ASSEMBLER
Version 2.1
(Weiand 1985)

Content

Program description :

GPL-ASSEMBLER Version 2.1 , Copyright 1985 by Michael Weiand
Distributed by : Elektronik Service, Linning 37, D-4044 Kaarst 2

Purpose : Assembling of GPL Programs

The program 'ASSM1/2/3' - the GPL-Assembler - reads DIS/VAR 80 input files from diskette which have been created by means of any editor program, for instance with the EDIT1 program of the Editor-Assembler (named E/A later on) package.

Source files are translated into a list file and an object file. The DIS/VAR 80 list file will contain all necessary information for documentation of your GPL program, like source code, generated object code, symbol tables and error messages. The list file resembles very much that created by the E/A-Assembler.

The object file (output file) is a tagged object code, DIS/FIX 80 file like the object files created by the E/A-Assembler, but will contain only absolute data. Relocatable code as well as REFs/DEFs/Entries cannot be generated. This is due to the fact that GPL-Loaders will not be able to use these features.

The object files will later on have to be LOADed by means of a special loader into EPROMs or GRAMs. The loader is not part of this package.

Hard- and software requirements :

TI99/4A-console with Editor-Assembler command module. This implies the need for a memory expansion and at least one disk drive. A printer together with its peripheral (RS232/PIO etc.) is strongly recommended.

Operating instructions :

Copy the EDIT1 program of your E/A package to the GPL diskette. Then insert the diskette labelled 'GPL-ASSEM' into disk drive 1 (instead of the 'E/A-#PARTA' diskette) and plug in the Editor-Assembler command module. The module will behave normally in all aspects.

When you select the option '2=Assemble', the new GPL-Assembler will be loaded. The questions concerning files are answered as usual, but there is a slight difference for the 'Options' question:

```
C  = generation of compressed object code,
     otherwise : not compressed' (cp.'COMP')
L  = Generation of a list file, otherwise start
     without list file (cp. 'LIST','UNL')
Sn = Generation of 'n' symbol tables
     otherwise no symbol tables (cp. 'SYMB')
Pm = maximum number of assembler passes 'm'
     otherwise 'm'=4  (cp.'PASS')
```

In the above codes 'n' and 'm' is a one-digit hexadecimal number ranging from 0 to F=15.

GPL source code

The GPL source code is of course different from the TMS9900 machine language. The E/A-manual will insofar be of little use, although it is important for an overview and description of the TI99/4A architecture. For a description of the GPL language please refer to special books, for instance the german booklet 'TI99/4A intern' by H.Martin (1985, published by 'Verlag fuer Technik und Handwerk', Baden-Baden, Germany).

The following tables are only meant as an overview of GPL commands and GPL syntax. This is not sufficient for writing GPL programs.

Syntax :

<SYMBOL> <OPCODE> <DATA>,<DATA>,...

<SYMBOL> : unique name of up to 32 characters
e.g. POINTER
a <SYMBOL> must start in column 1 !
<OPCODE> : xxxx = 4 significant characters as given in the following table, e.g. ADD
or for 'Double'-Opcodes :
Dxxxx = 5 significant characters including 'D'
e.g. DADD
an <OPCODE> must not start in column 1 !
<DATA> : Syntax depends on <OPCODE>, but can also be void.
excess <DATA> are ignored.

If any field - also the <SYMBOL>-field - begins with an asterisk '*', the rest of the line is treated as comment.

possible <DATA>-formats :

<STRING> : Text in single quotes, e.g. 'abcdefg'
For a quote, insert two quotes! e.g. 'Sam''s'

<NAME> : <STRING> which represents a valid filename
e.g. 'DSK1.TEST-COPY'

<IMM> : a value consisting of (NUMBER)s, (SYMBOL)s and (OPER)ators :
e.g. POINTER+1
(NUMBER) : integer value, e.g. 13
: Hexadecimal value, e.g. >AA01
: Binary value, e.g. &10100101
(SYMBOL) : name like <SYMBOL>, e.g. POINTER
: % = current GROM base address (e.g. >6000)
: $ = current address in the GROM (cp.'AORG')
(OPER) : + = Plus
: - = Minus
: * = Multiply
: / = Integer Division
: ! = Modulo
Note: Brackets are not allowed! No operator precedence, but strict left-to-right calculation.
e.g. 1+3*4 gives 16 !

<SP> : like <IMM>, but the result must be within >8300 to >83FF (ScratchPad-RAM addresses)

```
------------------------------------------------------------
<S>/<D>  : Source / Drain ("General address"="gas")
           can be one of the follwing forms,
           indicating the type of memory (RAM/ROM,VDP,GROM)
           and address mode (direct, indexed, indirect)
IMM      : Immediate value (cp. <IMM>)
           NOTE : ONLY VALID IN <S> = Source !!!!
           e.g. BACK >20
@IMM     : direct RAM/ROM, e.g. CLR @>8300
*IMM     : indirect to/from RAM/ROM into RAM/ROM
           e.g. CZ *FAC (the value from FAC is read as an
           address in RAM/ROM where the value is read)
V@IMM    : VDP direct, e.g. V@0
V*IMM    : VDP indirect by RAM/ROM, e.g. V*PABPTR
           (the value from PABPTR in RAM/ROM is read as an
           address in VDP)
@IMM(@SP) : direct in RAM/ROM, indexed by a scratchpad value
           e.g. @>0001(@FAC)
           (the value at FAC is read, >0001 is added ans the
           result is an address in RAM/ROM)
*IMM(@SP) : indirect in RAM/ROM, indexed by a scratchpad value
           e.g. *>0001(@FAC)
           (like above, but the result is an address in RAM/ROM
           where the final address is fetched.)
V@IMM(@SP) : like @IMM(@SP), but the result is in VDP memory!
V*IMM(@SP) : like *IMM(@SP), but the result is in VDP memory!
------------------------------------------------------------
<MOVES>  : the MOVE command accepts as source data the <D>
           values and the following :
G@IMM    : direct GROM/GRAM
           e.g. G@>6000
G@IMM(@SP) : like @IMM(@SP), but the result is in GROM memory!
------------------------------------------------------------
<MOVED>  : the MOVE drain data is similar to <MOVEQ>, but
           permits VDP registers as well:
#IMM     : VDP register direct (number IMM = 0 to 7)
           e.g. #7
```

GPL commands in alphabetical order

Note: (D) means: the command is available as 'word' or
'Double' command. Otherwise, bytes are accessed.
e.g. ADD 1,@FAC adds a byte value to @FAC
DADD 1,@FAC adds a word value to @FAC,@FAC+1

```
(D)ABS  <D>
(D)ADD  <S>,<D>
   ALL  <IMM>
(D)AND  <S>,<D>
   B    <IMM>     * IMM is a GROM address
   BACK <IMM>
   BR   <IMM>     * IMM is a GROM address in the current GROM
   BS   <IMM>     * IMM is a GROM address im the current GROM
   CALL <IMM>     * IMM is a GROM address
   CARR
(D)CASE <D>
(D)CEQ  <S>,<D>
(D)CGE  <S>,<D>
(D)CGT  <S>,<D>
(D)CH   <S>,<D>
(D)CHE  <S>,<D>
(D)CLOG <S>,<D>
(D)CLR  <D>
   COIN <S>,<D>
   COL  <IMM>           * FMT command
   COL+ <IMM>           * FMT command
   CONT
(D)CZ   <D>
(D)DEC  <D>
(D)DECT <D>
(D)DIV  <S>,<D>
   EX   <S>,<D>
   EXEC
   EXIT
   FEND                 * FMT command
   FETC <D>
   FOR  <IMM>           * FMT command
   FMT
   GT
   H
   HCHA <IMM>,<IMM>     * FMT command
   HOME                 * = DCLR @YPT
   HSTR <IMM>,<D>       * FMT command
   HTEX <IMM>/<STRING>  * FMT command
(D)INC  <D>
(D)INCT <D>
   I/O  <S>,<D>
(D)INV  <D>
   MOVE <S>,<MOVES>,<MOVED>
(D)MUL  <S>,<D>
(D)NEG  <D>
(D)OR   <S>,<D>
   OVF
   PARS <IMM>
   POP  <D>
   PUSH <D>
   RAND <IMM>
   ROW  <IMM>           * FMT command
   ROW+ <IMM>           * FMT command
   RTN
```

```
      RTNB
      RTNC
      RTGR
      SCAN
      SCRO <S>              * FMT command
    (D)SLL  <S>,<D>
    (D)SRA  <S>,<D>
    (D)SRC  <S>,<D>
    (D)SRL  <S>,<D>
    (D)ST   <S>,<D>
    (D)SUB  <S>,<D>
      SWGR <S>,<D>
      VCHA <IMM>,<IMM>     * FMT command
      VTEX <IMM>/<STRING> * FMT command
      XML  <IMM>
    (D)XOR  <S>,<D>
```

Pseudo commands

```
      BYTE <IMM>,<IMM>,...
      COMP <IMM>
      COPY <NAME>
      DATA <IMM>,<IMM>,...
      END
      EQU  <IMM>
      IDT  <STRING>
      LENG <IMM>
      LIST
      LIST <NAME>
      OBJE <IMM>
      OFFS <IMM>
      PAGE
      PASS <IMM>
      STRI <STRING>
      SYMB <IMM>
      TEXT <STRING>
      TITL <STRING>
      UNL
      VAR  <IMM>
```

FMT commands

Some GPL commands are applicable only within the 'formatted display'. This mode ist entered with the FMT command. FEND, if not used for finishing a FOR-command, will switch to 'normal' GPL mode again. Normal commands are not allowed within the FMT block !

Note : <CNT> is an <IMM>-value of 1 to 32

```
COL  <CNT>     * Column <CNT>
COL+ <CNT>     * advance column by <CNT>
FEND           * finishes previous FOR-Loop
FEND           * finishes FMT, if no FOR-Loop is pendant
FOR  <CNT>     * repeats the following commands up to
               * FEND <CNT> times
HCHA <CNT>,<IMM> * display CHAR(IMM) <CNT> times
               * horizontally
HSTR <CNT>,<D> * Display string of length <CNT>, which
               * is stored at address <D>
HTEX <STRING> * display <STRING> horizontally
               * max. length is 32 characters
HTEX <IMM>,<IMM>,....
               * display CHAR(IMM) ... horizontally
ROW  <CNT>     * Row <CNT>
ROW+ <CNT>     * advance row by <CNT>
SCRO <S>       * get new screen offset at <S>
VCHA <CNT>,<IMM> * display CHAR(IMM) <CNT> times
               * vertically
VTEX <STRING> * display <STRING> vertically
               * max. length is 32 characters
VTEX <IMM>,<IMM>,....
               * display CHAR(IMM) ... vertically
```

Example of an FMT command block:

```
FMT            * Start FMT
FOR  3         * Repeat 3 times
HTEX 'HELLO'   * display 'HELLO' horizontally
ROW+ 2         * advance row by 2
FEND           * finish FOR-Loop
FEND           * finish FMT
```

Another example can be found on the diskette (TEST-COPY).

Pseudo commands

The following assembler commands will create data structures, not executable GPL code:

```
BYTE <IMM>,<IMM>,... * (max.16 values)
                     * stores values <IMM> as bytes
                     e.g. BYTE 1,2,3,4,5
DATA <IMM>,<IMM>,...
                     * like BYTE, but words are stored.
                     e.g. DATA >AA01,0,0,POINTER
TEXT <STRING>        * stores <STRING> of ASCII-values.
                     e.g. TEXT 'ASSEMBLER'
STRI <STRING>        * like TEXT, but the generated code
                     * will be preceded by the length byte.
                     e.g. STRI 'GPL HELP'
```

A symbol value can be assigned not only by giving it the address value of the instruction which it precedes. The following statements will assign constants to any <SYMBOL>:

```
SYMBOL1    EQU <IMM>      e.g. ID    EQU >AA01
SYMBOL2    EQU <STRING>   e.g. BLANK EQU '  '
SYMBOL3    VAR <IMM>      e.g. STEP  VAR 2
SYMBOL3    VAR <STRING>   e.g. STEP  VAR '0'
```

SYMBOL1 will get the value of <IMM>. SYMBOL2 will be assigned the value of the first two bytes of <STRING> or of >00xx if the string length is 1. SYMBOL3 is the same as SYMBOL1, but the VAR statement disables error messages for 'Multiple Symbols' and 'New Symbol value'. Thus you can assign different values to a symbol which is important if you want to use them as input or output variables from 'Macro'-type COPY-files.

Be cautious using recursive <IMM>-values since these may result in unstable values which will cause the assembler to retry endlessly.

e.g. : START EQU START+1 (!!!)

This condition can be avoided using the PASS pseudo command or the P-option of the assembler.

Control of assembler operation

Input file :

```
COPY <NAME>   * insert file <NAME> into the source
              * code. Be cautious not to create loops!
END           * Stop assembly of source code.
PASS <IMM>    * new value <IMM> for the maximum pass
              * count. Default value = 4
     ---------> This command overrides the assembler
              * option 'P' with the same meaning !
```

Output file :

```
COMP <IMM>    * if <IMM> is 0, the object code is
              * 'not compressed', else compressed.
              * Default : compressed
     ---------> This command overrides the assembler
              * option 'C' with the same meaning !
OBJE <NAME>   * Open new object file <NAME>
     ---------> This command overrides the assembler
              * input 'OBJECT FILE' !
OFFS <IMM>    * <IMM> is a 'load offset' value, i.e.
              * the generated addresses are offset
              * from the calculated values by <IMM>
              * This can be used by special loaders.
              * Default = 0
IDT <STRING>  * <STRING> is a new ID-field within
              * the object file. Default = 'GPL-ASSM'
```

List file :

```
LIST ''       * (Re)starts list file generation
              * same meaning as the 'L'-option
LIST <NAME>   * Open new list file <NAME> and start
              * list file generation.
     ---------> This command overrides the assembler
              * input 'LIST FILE' !
LENG <IMM>    * <IMM> is the lines/page value
              * Default = 66
PAGE          * generate page break
SYMB <IMM>    * Indicate symbol tables to be generated
              * the <IMM> value is coded bit by bit
              * i.e. the following values have to be
              * added for each table desired:
              * 1 = new symbols, in alphabetical order
              * 2 = new symbols, sorted by value
              * 4 = predefined symb., in alphabetical order
              * 8 = predefined symb., sorted by value
     ---------> This command overrides the assembler
              * option 'S' with the same meaning !
TITL <STRING> * <STRING> is the new listing header line
UNL           * Stop list file generation.
     ---------> This command overrides the assembler
              * option 'L' !
```

Assembler principles

The GPL ASSEMBLER had to be designed to cope with varying code length depending on the symbol values. This means that addresses can change from pass to pass, making a 2-pass assembly impossible.

Therefore an 'endless-pass' assembler had to be written which will stop assembly only if no more changes in symbol values are encountered. The last pass will generate object and list files.

Any error is therefore classifed as follows:
- warnings (no influence on generated code)
- Retry (may be better next time)
- Fatal (no meaningful code can ever be expected)

After fatal errors no object file will be generated !

During each pass the following information is displayed :

- Pass number and current error state (correct, retry, aborted)
< Current Input and Copy file(s)
> Current Output file(s)
- Error messages togehter with the row number in the current input or copy file (restarts with 1 in each copy file!) the list of error messages is given below.
- total error state in case of aborted assembly.

The messages are only informative till the last pass. You don't have to note all the messages passing by for the first passes!

List file generation is postponed till the last pass. An object file s i will also be generated in this pass if the error state is 'correct'.

Example

On the GPL ASSEMBLER diskette you will find a working example for a command module type GPL program. The files TEST... are accessed in the following way:

```
INPUT FILE  : DSK1.TEST-INPUT
OBJECT FILE : DSK1.TEST-OUTPT
LIST FILE   : DSK1.TEST-LIST
OPTIONS     : L S5 P4
```

Note: the input file TEST-INPUT will COPY the file TEST-COPY twice. You can of course use other filenames, like PIO for you list file, if you wish to.

Error messages

Adress Error (Fatal)
Invalid address syntax

Adress Mode Error (Fatal)
Invalid address mode (not allowed for this command)

Adress not even (Warning)
Warning: an uneven load address has been generated which your loader might not accept!

COMP ignored !! (Warning)
the COMP command cannot change the COMP mode of an object file when data have been written to it.

Copy File Error (Fatal)
any file error on the Copy file

Default changed (Warning)
Warning: the value of a predefined symbol has been altered. see list below!

Fatal Error (Fatal)
error state which means 'abort assemblation'

File Error (Fatal)
any input file error

FMT missing (Fatal)
you have used FMT commands without previous FMT

FMT not complete (Fatal)
you have used normal commands within FMT

Format Error (Fatal)
address mode error within FMT

"gas" error (Fatal)
general error in <S>/<D> data ("general address")

Invalid Label (Warning)
a label syntax is incorrect

Invalid Mnemonic (Fatal)
a command could not be recognized

Invalid Numeric (Fatal)
a number could not be decoded

Multiple Symbols (Fatal)
you tried to redefine a symbol

New Symbol value (Retry)
a symbol value has changed since the last pass

No Object File (Warning)
Warning: there is no object file for your data

Object File Error (Fatal)
any file error on the object file

Out of Range (Fatal)
a value is out of range (in most cases <SP>)

Parameter Error (Fatal)
missing parameter = data

Print File Error (Warning)
any list file error

String Error (Fatal)
invalid string syntax (missing ')

Symbol Table full (Fatal)
no more room for new symbols

Syntax Error (Fatal)
any other syntax error

Too many ! (FMT) (Fatal)
wrong <CNT> value

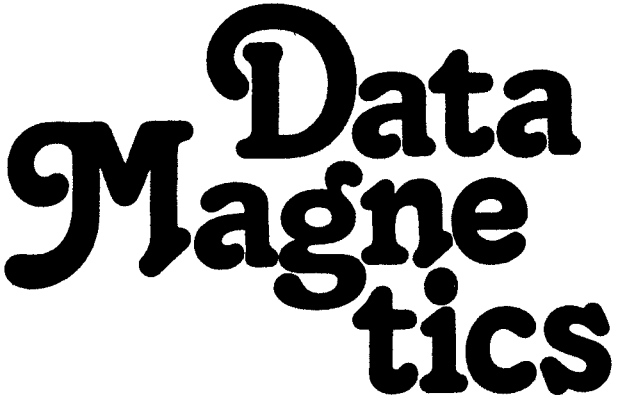
Too many passes (Fatal)
the maximum number of passes has been exceeded

Undefined Symbol (Fatal)
you referenced an undefined symbol

Predefined symbols

Symbol Table #4 (Def,alpha)

0034 ACCTON	835C ARG	0032 ATN	0036 BADTON	003B BITREV
0012 CFI	0014 CNS	002C COS	0010 CSN	8372 DATSTK
0001 DIVZER	0003 ERRIOV	0006 ERRLOG	0005 ERRNIP	0002 ERRSNN
0004 ERRSQR	0028 EXP	834A FAC	0006 FADD	000A FCOMP
0009 FDIV	0008 FMUL	836C FPERAD	0007 FSUB	0038 GETSPACE
0022 INT	0010 LINK	0018 LOCASE	002A LOG	8370 MEMSIZ
003D NAMLNK	8300 PAD	0024 PWR	0012 RETURN	000B SADD
000F SCOMP	000E SDIV	8375 SGN	002E SIN	000D SMUL
8400 SOUND	0026 SQR	000C SSUB	837C STATUS	0016 STCASE
8373 SUBSTK	0030 TAN	0007 TRIGER	004A UPCASE	836E VSPTR
0001 WRNOV	837F XPT	837E YPT		


Data
Magne
tics