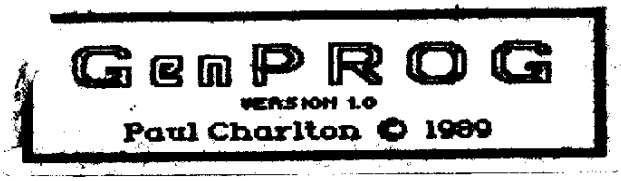
GenASM
v1.00

Reference guide.



NOTE: GENASM .E ASM CAN ONLY BE RAN FROM
A BATCH FILE - IT DOESN'T LIKE
BEING RAN FROM A COMMAND PROMPT.

CONTENTS

INTRODUCTION

The GenASM assembler is a powerful one-pass macro assembler for 9995 assembly source code.

The primary purpose of this manual is to describe the differences between the GenASM assembler which runs in MDOS mode and the TI-99/4 assembler which runs in GPL mode. This is intended to be a supplement to the TI manual for the TI-99/4 assembler. The TI manual (TI part #1035984-2) and reference card (TI part #1035988-1) can still be ordered from Texas Instruments, Inc. at the phone number 1-800-TI-CARES.

Features unique to the GenASM assembler:

1)	one-pass,	increases speed
2)	macros,	can increase productivity
3)	conditional assembly,	can increase productivity
4)	symbolic debugger support,	can save endless headaches
5)	9995 opcodes,	allows full use of the Geneve hardware
6)	detailed error messages,	save much time, good learning tool

OVERVIEW

GenASM uses four types of files during the course of its execution, all of these will be described briefly below.

"ASM" and "ASN" are the MDOS program files which read the assembly language source files and translate them into the resulting tagged object file.

Assembly source files are files in a Dis/Var [1..254] file format (Dis/Var 80 is most common) which contain assembly language source code. These files can be read by GenASM under either of two conditions: 1) the source file was specified on the MDOS command line, 2) the source file was referenced with a COPY directive within the initial source file.

Tagged object files are files created by GenASM in Dis/Fix 80 file format. More information on these files can be found in the GenLINK reference guide.

List files are files created by GenASM in Dis/Var 80 format which contain error messages as well as any other information you directed GenASM to list out (cross-reference symbol lists, etc.)

USING GenASM

GenASM is executed from an MDOS command line or from within an MDOS batch file.

You must perform the following actions before using GenASM:

First, MDOS must be able to find the files "ASM" and "ASN" in the same directory somewhere in your current search path (set with the "PATH" command in MDOS.)

Second, you must create the primary source file and any source files included with a COPY directive during the assembly process.

Third, GenASM must be able to write to the tagged object file specified on the command line. This means that your destination disk must not be write protected, and that it must have enough free sectors to allow the tagged object file to be complètely written.

Continuing with the assumption that the three previous conditions have been met, GenASM is invoked from MDOS with the following command:

GENASM source_file,object_file,[list_file],[flags][=macro_string]

Items between square brackets are optional.

CONTROL FLAGS

GenASM recognizes several flags which you can use to modify the behavior of GenASM.

COMMONLY USED FLAGS

"R" or "r" This flag causes GenASM to define symbols R0..R15 with values 0..15 as "register symbols".

"C" or "c" This flag causes GenASM to generate compressed object code (These files take less disk space. For more information on compressed object code, see the GenLINK reference guide.)

LESS COMMON FLAGS

"O" or "o" This flag causes GenASM to produce tagged object files compatible with the TI-99/4 object code loader. This option has the following implications:

1) REF and DEF symbols are limited to 6 characters in length.
2) 8-bit values may be expanded to 16-bit words when object code is generated.
3) the value used in a BYTE or TEXT directive must be defined at a point earlier during the assembly process.
4) debugger information is not written to the object file.

"G" or "g" This flag causes GenASM to write debugger symbol and file records into the object file. This flag is ignored if the "O" flag was specified.

"S" or "s" This flag causes GenASM to write information about each symbol in the global symbol table to the list file.

"U" or "u" This flag causes GenASM to print the names of all symbols which were defined in the source file, but not used anywhere withing the source file.

"X" or "x" This flag causes GenASM to write a cross reference for all symbols in the global symbol table, except register symbols, to the list file.

"Y" or "y" This flag causes GenASM to write a cross reference for all symbols in the global symbol table, including register symbols, to the list file. (It is highly recommended that you use the "X" flag or the XREF directive to create a usable cross reference listing.)

SOURCE CODE FORMAT

Each source line is either a comment or a statement. Comment lines must begin with an asterisk "*" or semicolon ";" or must be completely blank.

Statements can have the following forms:

"symbol"
"symbol mnemonic"
"symbol mnemonic operands"
" mnemonic"
" mnemonic operands"

A sequence of two or more spaces in a source line is treated as a single space. A semicolon always represents the end of a statement when it is not enclosed as part of a character string within quote delimiters.

SYMBOLS

Symbols recognized by GenASM are allowed to have has many as 31 characters from the following characters:

--- !"#$'.0123456789:< =>?ABCDEFGHIJKLMNOPQRSTUVWXYZ[]_ ---
--- 'abcdefghijklmnopqrstuvwxyz{}~ ---

The first character of a symbol may not be among the following three characters:

--- >#' ---

There are four classes of symbols used by GenASM:

"$" This single character symbol returns the current address.

local: Any symbol beginning with the ":" character is a local symbol. Local symbols are not listed in symbol table or cross-reference listings. The names of any local symbols are erased whenever GenASM defines a new global symbol. These are used to reduce the number of symbols in list files and debug tables. In general, one would use these to avoid listing unimportant symbol names, such as labels used inside of subroutines for jump instructions.

global: Symbols which will show up in listing files, and in debug tables. You generally will only want to define the entry points of subroutines and data addresses with global symbols.

register: Use of the "R" flag on the command line defines 16 register symbols with the names R0..R15. The only difference between register symbols and other global symbols is that they are not listed when a cross reference listing is obtained with the "X" flag.

MNEMONICS

9900 OPCODES

GenASM reconizes all 9900 opcodes with the syntax described in the TI manual.

9995 OPCODES

GenASM recognizes the 9995 microprocessor enhancements to the 9900 instruction set. These are described in the following paragraphs.

LST

syntax: LST register
object format: >008r

The Load_STatus instruction causes the contents of the specified register to be copied into the processor status register. All 16 bits of the processor status register are modified.

LWP

syntax: LWP register
object format: >009r

The Load_Workspace_Pointer causes the contents of the specified register to be copied into the workspace pointer register on the processor.

MPYS

syntax: MPYS general_address
object format: #0000 0001 11tt rrrr

The MultiPlY_Signed instruction takes one signed 16 bit number from R0 and another signed 16 bit number from the address specified in the operand field of the instruction, then places the 32 bit product into R0 (most significant 16 bits) and R1 (least significant 16 bits.) The results of this instruction are compared to zero and L>, A>, and EQ status bits are set accordingly.

DIVS

syntax: DIVS general_address
object format: #0000 0001 10tt rrrr

The DIVide_Signed instruction uses R0 (most significant 16 bits) and R1 (least significant 16 bits) as a 32 bit signed dividend for the divide operation, and the 16 bit signed number from the address specified in the operand as the divisor. If the quotient can't be expressed as a signed 16 bit number, the overflow status bit is set and the divide is aborted. Otherwise, R0 is loaded with the 16 bit signed quotient from the division and R1 is loaded with a signed remainder. The sign of the remainder is the same as the sign of the initial 32 bit dividend, and the magnitude of the remainder is less than the magnitude of the divisor.

PSUEDO OPCODES

GenASM recognizes the following two mnemonics as psuedo opcodes:

NOP	same as "JMP $+2"
RT	same as "B *R11"

USER DEFINED MACROS

Any symbol which appears after a MAC directive may be used in the mnemonic field of any subsequent line in the source code. User defined macros are invoked with the following format:

[!0] macro_name [!1]...[,!9]

A macro invocation can have one label and up to nine operands (separated by commas.)

Further information on macros can be found in the section of this manual entitled "USING MACROS".

ASSEMBLER DIRECTIVES

Macro directives:

These are discussed in the section of this manual entitled "USING MACROS"

Conditional assembly directives:

These are discussed in the section of this manual entitled "CONDITIONAL ASSEMBLY"

Directives which affect the current address:

AORG	sets PC to previous AORG value
AORG expression	PC = new AORG value
DORG	sets PC to previous DORG value
DORG expression	PC = new DORG value
RORG	sets PC to previous RORG value
RORG expression	PC = new RORG value

Any label on the line with AORG,DORG,RORG is assigned the new value of the program counter.

	BES expression	New PC = Old PC + expression
	BSS expression	New PC = Old PC + expression
symbol	BES expression	New PC = Old PC + expression; symbol = New PC
symbol	BSS expression	New PC = Old PC + expression; symbol = Old PC
	EVEN	New PC = (Old PC + 1) & >fffe
symbol	EVEN	New PC = (Old PC + 1) & >fffe; symbol = New PC

Miscellaneous directives:

	DEF symbol_list	DEFines the symbols into object code for linking.
	REF symbol_list	Places symbols into object code for the linker to resolve.
	END	Discontinues assembly process.
	END symbol	Discontinues assembly process, "symbol" is defined as an auto-start entry point for the object code.
symbol	EQU expression	The symbol is assigned the expression's value.

Unused directives:

UNL	ignored
LIST	ignored
TITL	ignored
PAGE	ignored

The following directives use text strings. The first non-blank character in the string is used as a delimiter, using it twice in a row will place it into the string. (Most people will use double quote marks, but any non-blank character is allowed by GenASM.)

COPY	text_string	include another file
TEXT	text_string	place string into object code
IDT	text_string	place 1st 8 characters of the string into the identifier field in the first record of the object file. (the default is the current time of day.)
ORIT	text_string	place the string into the object file as a comment

Examples:

```
COPY  "include1"
COPY  +include1+
```

XREF DIRECTIVE

syntax: XREF symbol_list

The XREF directive causes all symbols in the list to be cross-referenced if a list file was specified (this is more selective than the "X" or "Y" flags.)

This is VERY useful when you want to find out where only a few symbols are used in the source code without taking the time to print an entire cross-reference listing.

DXOP DIRECTIVE

syntax: DXOP xop_name,xop_number

The DXOP directive is not recognized by GenASM. If you wish to use the DXOP directive for compatibility with older source code, please refer to the DXOP macro described in the example section of this manual.

CONDITIONAL ASSEMBLY

You will want to use conditional assembly directives when you are developing several slightly different versions of the same program. An example you may want to use if you are developing a program to run in both MDOS and GPL mode would be:

```
IF      MDOS
TEXT    "MDOS version"
ELSE
TEXT    "GPL version"
FI
```

Conditional assembly is also useful for including extra code for help in debugging a new application. By simply changing one equate (or defining a macro to look at the command-line option string) before re-assembling the file, you can turn off all of the debugging code, and not have it appear in the resulting object code.

Debugging code can be included as follows:

```
IF      DEBUG
PRINT   "routine #1 ok"
FI
```

Conditional assembly directives:

```
IF      expression          ;expr != 0
IFEQ    expression          ;expr == 0
IFGE    expression          ;expr >= 0
IFGT    expression          ;expr >  0
IFLE    expression          ;expr <= 0
IFLT    expression          ;expr <  0
IFNE    expression          ;expr != 0
IFS     string1,string2     ;string1 == string2
IFNS    string1,string2     ;string1 != string2
IFGS    string1,string2     ;string1 >  string2
IFLS    string1,string2     ;string1 <  string2

ELSE
FI
```

Two forms of conditional directives are allowed:

```
IFxx
...condition true lines
FI

IFxx
...condition true lines
ELSE
...condition false lines
FI
```

Conditionals may be nested to a depth of 32,764 levels (who'd want to?)

A label on an ELSE or FI directive is treated as if it were on the previous line, and is treated as one of the conditional lines of code.

ie:

```
symbol      ELSE
```

is the same as:

```
symbol      equ     $
            ELSE
```

USING MACROS

GenASM supports the use of user defined macros in your source code. User defined macros allow you to define a single identifier (the "macro name") which represents many lines of code (the "macro body"). When GenASM later encounters the macro name in your source code, it will assemble all of the lines of code found in the macro body.

A good set of macro definitions can be used to enhance your productivity on assembly language programming projects.

You can pass parameters to the macro body whenever the macro is invoked in your source code. Within the body of the macro, these parameters can be referenced by their position in the line during the macro call. The label on a macro call is referred to as "!0", the first and subsequent operands are referred to as "!<digit[1..9]>".

Defining a macro:

A macro is defined with the following sequence of lines in your source code:

```
MAC   macro_symbol     "macro_symbol" is defined as a macro
...macro body          these lines are the macro definition
MEND                   exit the macro definition
```

Example:

```
MAC   ASCIIZ           ;ascii string, null terminated
TEXT  !1               ;first parameter is the string
BYTE  0                ;NULL termination
MEND
```

The line

```
ASCIIZ "Hi there!"
```

in your source code would generate object code for the following lines of code:

```
TEXT  "Hi there!"
BYTE  0
```

Macro strings

The GenASM macro processor recognizes several special macro strings which can appear at any point within the body of a macro. If GenASM detects the name of any of these macro strings anywhere (even inside of quoted strings) in a macro body when it is expanding the macro, it will substitute any text associated with the string into the macro body at that point. The names of all macro strings have the form "!<char>".

Names of macro strings:

!0..!9 These ten names represent positional parameters provided when the macro is invoked. Any of these parameters not defined in a call to the macro will be a NULL string. Any parameters in the macro call which aren't referenced in the macro body are ignored.

!N This is a 4 character numeric string with the current macro invocation number. <used to generate unique labels>.

ie: "m!Nx" would be a unique string such as "m0001x"
or "m0010x", if this was the 10th macro invoked.

!D 8 character date string "mm-dd-yy"

!T 8 character time string "hh:mm:ss"

!S The string of all characters following an "=" in the command flags passed to the assembler from the command line.

!- This is a NULL string.

!! A single character string "!".
--used to obtain a "!" inside of a string in a macro body.

All other characters preceeded by a "!" are discarded.

The following lines of code would generate an ascii string with the date and time of assembly if they were encountered during the expansion of a macro:

```
MAC   EXAMPLE
TEXT  "Assembled on !D,"
TEXT  "At !T."
MEND
```

Macro substring operations:

You may break macro strings into substrings with an index operation of the form:

!<char> [start,len]

IE:

!D[4,2]

Will return a 2 character string which contains the current day of the month, extracted from the 8 character macro string containing the current date.

Any character positions specified for the substring which weren't defined in the initial parameter string are filled with spaces.

IE:

!D[4,8]

Will return the string "dd-yy ", with three trailing spaces, since the date string didn't contain all of the characters specified with the substring operation.

Macro nesting

Any macro may be called from within the body of another macro during the expansion process. Macros may be nested to a depth of 16 levels.

Any macro may call itself, recursively, though you must provide a means to terminate the nesting.

Example:

```
MAC         RECURSIVE
IF          !1
DATA        !1
RECURSIVE   (!1)-1
FI
MEND
```

Try "RECURSIVE 10" to see what results you get...

EXPRESSIONS

The following table is a list of all expression constructs recognized by GenASM, note that parens may be used to change the evaluation order of an expression.

Syntax

expression:		term1(term2)	
term1:		term	
term2:		aterm	value must be 1..15, no forward references allowed
term:		rterm	RORG value
		aterm	AORG value
rterm:		rsymbol	symbol with relative (RORG) value
	or	(rterm)	up to 16 nested levels of "()" are allowed
	or	aterm + rterm	the two terms are added
	or	rterm + aterm	the two terms are added
	or	rterm - aterm	the 2nd term is subtracted from the first term
	or	aterm - -rterm	the 2nd term is added to the first term.
aterm:		constant	
	or	asymbol	symbol with absolute value
	or	-aterm	
	or	(aterm)	up to 16 nested levels of "()" are allowed
	or	aterm * aterm	the two terms are multiplied
	or	aterm / aterm	the 1st term is divided by the 2nd term
	or	aterm + aterm	the two terms are added
	or	rterm + -rterm	the 2nd term is subtracted from the first term
	or	rterm - rterm	the 2nd term is subtracted from the first term
	or	aterm - aterm	the 2nd term is subtracted from the first term
	or	term % term	remainder of 1st term divided by 2nd term
	or	term ^ term	the 1st term is left-shifted 2nd term bits
	or	term & term	bitwise "and" of the two terms
	or	term | term	bitwise inclusive "or" of the two terms

All terms in parens are evaluated before other terms, normal evaluation proceeds from left-to-right unless parens are used.

Constants

GenASM reconizes four different types of constants in your source code. They are allowed to have the following forms:

binary:	#binary_digits	01
decimal:	decimal_digits	0123456789
hex:	>hex_digits	0123456789ABCDEFabcdef
character:	'chars'	>01..>ff, "'" produces "'" within the constant

ERROR REPORTING

Error mesages from GenASM are written to your screen and to any list file you may have specified.

GenASM provides a highly descriptive message for each error it encounters in your source code. For each error, GenASM provides you with the statement number, the filename, and the line within the file on which the error occurred. It also tells you exactly what was wrong with the line (no generic message like "SYNTAX ERROR".)

You should find GenASM's error reports to be helpful and instructive. They should provide you with a good aid for learning the proper syntax of assembly language programs.

EXAMPLE: PRINT macro

```
*
* generally useful PRINT macro
* (for debugging)
*
            MAC             PRINT
*
* define the label first, so that we
* can jump to the print statement
*
            EVEN
!0          EQU             $
*
* MDOS print routine
*
            IF              MDOS
*
            LI              R0,WRTTTY
            LI              R1,:PSTR!N
            CLR             R2
            XOP             @VIDEO,0
            JMP             :PSJ!N
*
:PSTR!N     TEXT            !1
            BYTE            >0D,>0A,0
*
* make sure that VIDEO is defined
*
            IFEQ            VIDEO
VIDEO       DATA            6
            FI
*
            EVEN
:PSJ!N      EQU             $
```

```
*
* GPL mode simple-minded print,
* only to top screen line, for debugging
*
            ELSE
*
            CLR         R0
            LI          R1,:PSTR!N
            LI          R2,:PLEN!N
            BLWP        @VMBW
            JMP         :PSJ!N
*
:PSTR!N     TEXT        !1
:PLEN!N     EQU         $-:PSTR!N
*
            EVEN
:PSJ!N      EQU         $
*
* end of print macro
*
            FI
            MEND
*
```

EXAMPLE: DXOP macro

```
*
* DXOP macro,
* defines another macro with appropriate name
*
* format:     DXOP            name,#
*
             MAC             DXOP
*
* define a macro with the user's names
*
             MAC             !1
*
* items with a double "!" are hidden until the
* inner macro is called, since "!!" becomes "!"
* when the DXOP macro itself is called
*
!!0          XOP             !!1,!2
*
* end the user's new macro
* the "!-" is a null string which hides
* the MEND from the DXOP macro.
*
             !-MEND
*
* end the DXOP macro expansion
*
             MEND
*
```