DECUS
PROGRAM LIBRARY

DECUS NO.	8-503
TITLE	MACRO-8X: 8K EXTENDED MACRO-8 ASSEMBLER
AUTHOR	David J. Waks
COMPANY	Submitted by: Robert M. Supnik Applied Data Research Cambridge, Massachusetts
DATE	1966
SOURCE LANGUAGE	No Sources Available

ATTENTION

This is a USER program. Other than requiring that it conform to submittal and review standards, no quality control has been imposed upon this program by DECUS.

The DECUS Program Library is a clearing house only; it does not generate or test programs. No warranty, express or implied, is made by the contributor, Digital Equipment Computer Users Society or Digital Equipment Corporation as to the accuracy or functioning of the program or related material, and no responsibility is assumed by these parties in connection therewith.

MACRO-8X: 8K EXTENDED MACRO-8 ASSEMBLER

DECUS Program Library Write-up DECUS NO. 8-503

INTRODUCTION

MACRO8X is an improved and expanded version of the MACRO8 assembler. It is a two-pass assembler which can run on a PDP-5, 8, 8/S, 8/I, or LINC-8 with 8,192 (or more) words of memory and a high-speed paper tape reader and punch. MACRO8X is fully compatible with both PAL-III and MACRO8, and will assemble any program that can be assembled by either of them. This manual describes the differences between MACRO8X and MACRO8 and gives operating instructions for MACRO8X.

GENERAL ENHANCEMENTS

Larger Symbol Table. The permanent symbol table of MACRO8X contains all the symbols that are contained in the symbol table of MACRO8. In addition, there is room for over 750_{10} user-defined symbols, minus that part of the user symbol area that is used to store macro definitions. This is more than five times the storage provided by MACRO8.

Improved Literal and Link Processing. The processing of literals and the generation of links have been improved in two significant ways. First, in MACRO8, the buffer in which literals are stored in the assembler is fairly small. When many literals are used on a page (or when many links are generated) the buffer overflows and is dumped. The accumulation of literals then begins again as if no literals had been assigned on the page. This results in duplication of literals on the same page and sometimes causes a page to overflow that should not overflow. This will not happen when a program is assembled with MACRO8X. Literals will not be dumped in the middle of a page and there will be no duplication of literals. Second, MACRO8X will, when assigning any literal defined by parentheses or link, determine whether that literal has earlier in the source program been assigned to page 0. If the value has already been assigned a location on page 0, the page 0 literal will be used and no literal will be assigned on the current page.

Paginated Output. Output listings produced during pass 2 of the assembler are divided into 8-1/2 X 11 inch pages. At the beginning of the assembly, at every occurrence of a form-feed character in the source program, and after every fifty-six lines of output, a page-break sequence is printed consisting of ten blank lines with five minus signs in the middle. The minus signs serve as cut marks for dividing the listing into eleven-inch pages. Pagination does not occur during the printing of the symbol table, which is printed so as to produce a DDT-compatible paper tape on the low-speed punch. In addition to automatic pagination, the MACRO8X listing routines do tabulation, converting tab characters into the number of spaces needed to produce a tabular listing.

Memory Allocation Table. At the end of each assembly (or, in the case of a multi-field assembly, at each FIELD or BANK command), a table is printed showing what parts of memory were not used by the program. For each page of memory, MACRO8 prints the address of the lowest numbered location into which no instruction was assembled and the address of the location that is 1 "below" the last-assembled literal. This table is intended for use as a guide in making patches and corrections to an assembled program.

Improved Functioning of FIELD Pseudo-op. When a FIELD (or BANK) command is encountered in the course of a MACRO8X assembly, all page zero literals are dumped on the output tape (during pass 2), and assignment of page zero literals begins for the new field.

ADDED PSEUDO-OPS

BANK. The BANK pseudo-op is synonymous with the FIELD pseudo-op (modified as described above) and is used in the same way.

UNLIST. The UNLIST pseudo-op suspends listing of the source and object programs on the teleprinter during pass 2 of an assembly. It does not suppress the printing of error messages, allocation tables, or the symbol table. This pseudo-op is not itself listed on the teleprinter.

LIST. The LIST pseudo-op resumes listing of the source and object programs on the teleprinter during pass 2 of an

assembly if such listing has been suspended by use of UNLIST. Otherwise, LIST has no effect. This pseudo-op is not itself listed on the teleprinter.

NOLGM. The NOLGM pseudo-op suspends printing of the diagnostic message "LG" during pass 2 of an assembly.

LGM. The LGM pseudo-op resumes printing of the diagnostic message "LG" where applicable during pass 2 of an assembly if such printing has been suspended by use of NOLGM. Otherwise, LGM has no effect.

LIT. The LIT pseudo-op causes the current-page-literal-buffer to be printed on the teleprinter (unless an UNLIST command has been given) and punched on paper tape during pass 2 of an assembly. Any literals used on the same page thereafter will not be compared with those already dumped, but will be assigned again even if they are duplicates. The intended use of this pseudo-op is as follows:

```
(last instruction on page) (cr) (lf)
LIT (cr) (lf)
(ff)
/(comments if desired)
PAGE
(first instruction of next page)
```

The above sequence will cause literals to be dumped at the end of a physical page of the listing, which will correspond to the end of a logical page of computer memory, rather than at the top of the next page of the listing. This improves readability of listings.

LITBAS. The LITBAS pseudo-op permits the user to specify an origin for the generation of literals which is other than location 177 of the page for which they are generated. The format of the LITBAS command is LITBAS n, where n is any number of symbolic expression, except one which contains an as yet undefined symbol. If n is greater than 177, it is interpreted as an absolute address in the current field. Thus, the statement "LITBAS 1357" means "Set the literal base for page 5 to location 1357". If n is less than 200 and the current address indicator is greater than 177, it

is interpreted as the twelve bit address formed by the concatenation of the high-order 5 bits of the current address indicator with n. Thus, the sequence

```
*4020
LITBAS 147
```

means "Set the current address indicator to 4020 and set the literal base for the page that begins with location 4000 to 4147". Use of LITBAS when n and the current address indicator are both less than 200 is illegal. The LITBAS command may be used at any time and becomes effective when it is given. If it affects the literal base of a page on which literals are being assigned (as it will if it is used in the middle of a sequence of instructions), those literals already assigned will be dumped and literals used thereafter will be assigned starting at the new literal base. The user must not permit the current address indicator to exceed the address into which the next literal will be assembled, lest "PE" errors result. The sequence:

```
*4020
LITBAS 33
TYASC,   0
         CLA
         TAD I TYASC
         ISZ TYASC
         TAD (260)
         TLS
         TSF
         JMP .-1
         JMP 1 TYASC
*4034
SAM,     0
         CLA
```

is illegal. In order to accomplish what is obviously intended, it is necessary to precede the statement "*4034" with "LITBAS b" where b is sufficiently large to allow room for the sequence of code that starts with "SAM".

VFD. The VFD pseudo-op permits the assembly of a word con-

sisting of the concatenation of bit-patterns representing several numbers or symbolic expressions. Its format is:

VFD A:B,C:D,E:F...

It is terminated by the occurrence of any punctuation or expression which does not fit the VFD syntax. The meaning of the expression above is "Assemble a word consisting of A bits of B followed by C bits of D followed by E bits of F, etc." A, B, C, D, etc. may be numbers or symbolic expressions. If numbers are used in A, C, E, they are interpreted in decimal radix. If numbers are used in B, D, F, they are interpreted in the radix prevailing at the time the VFD was encountered. The sum of A, C, etc. must not exceed 12. If the values of B, D, etc. are too large to be represented in the indicated number of bits, the appropriate number of bits at the low-order end of the value will be used. VFD may be used in any context in which a symbolic expression is legal.

OPERATING PROCEDURES

The operating instructions for MACRO8X do not resemble the operating instructions for MACRO8. There are no switch options in MACRO8X. Also, an assembly consists of only two passes, with a listing produced during pass 2. To assemble a program, follow the procedure below.

1. Load the MACRO8X binary tape via the binary loader.

2. Place the first (or only) source tape in the high speed reader. (Each tape except the last must end with a PAUSE pseudo-op. The last source tape must end with a dollar sign.) Put 200 in the switch register. Depress the Load Address key. Depress the Start key.

3. If running a multi-tape assembly, depress the Continue key after placing each subsequent tape in the reader.

4. After pass 1 has been terminated, place the first (or only) source tape in the reader and depress the Continue key.

5. If running a multi-tape assembly, depress the Continue key after placing each subsequent tape in the reader.

6. After pass 2 has been completed, and before the symbol table has been printed, the computer will halt. Turn on the low-speed paper-tape punch and depress the Continue key. The symbol table will be printed and punched in a format acceptable to DDT.

7. To run another assembly, restart from step 1 above. The assembler must be reloaded, as it does not initialize itself when it begins an assembly.

AMC:dnr