The Macro Processing System STAGE2:
Transfer of Comments to the Generated Text

by

Odd Pettersen

Research sponsored by

Advanced Research Projects Agency
ARPA Order No. 2494

COMPUTER SCIENCE DEPARTMENT
Stanford University
The Macro Processing System STAGE2:
Transfer of Comments to the Generated Text
by
Odd Pettersen

ABSTRACT

This paper is a short description of a small extension of STAGE2, providing possibilities to copy comments etc. from the source text to the generated text. The description presupposes familiarity with the STAGE2 system: its purpose, use and descriptions, like [1] to [9]. Only section 3 of this paper requires knowledge of the internal structures and working of the system, and that section is unnecessary for the plain use of the described feature. The extension, if not used, is completely invisible to the user: No rules, as described in the original literature, are changed. A user, unaware of the extension, will see no difference from the original version.

This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC 15-73-C-0435. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, ARPA, or the U. S. Government.

Reproduced in the U.S.A. Available from the National Technical Information Service, Springfield, Virginia 22161.
The Macro processing system STAGE2:
Transfer of comments to the generated text.

by

Odd Pettersen

SINTEF, Div. of Automatic Control,
The Technical University of Norway
(presently with
Stanford University
Artificial Intelligence Lab.
Computer Science Dept.)

December 1974

1. INTRODUCTION

The following is a short description of a small extension of STAGE2, providing possibilities to copy comments etc. from the source text to the generated text. The description presupposes familiarity with the STAGE2 system; its purpose, use and descriptions, like [1] to [9]. Only section 3 of this paper requires knowledge of the internal structures and working of the system, and that section is unnecessary for the plain use of the described feature.

The extension, if not used, is completely invisible to the user: No rules, as described in the original literature, are changed. A user, unaware of the extension, will see no difference from the original version.

2. THE FLAG LINE

As described in [1], the input material for any translation by STAGE2 starts with a collection of macros, defining the correspondence between the source language of input (program) text following the macros, and the target language. In the very beginning of the input material, however, and preceding the macros, is a so-called FLAG LINE. This line, terminated by carriage return, defines the character set used.

2.1. The original format of the Flag Line.

As also stated in [1], the flag line consists of the following characters, in the order shown:

<table>
<thead>
<tr>
<th>Pos. in Flag line</th>
<th>Function of Character</th>
<th>Usual character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Source end-of-line flag</td>
<td>$</td>
</tr>
<tr>
<td>2</td>
<td>Source parameter flag</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>Target end-of-line flag</td>
<td>@</td>
</tr>
<tr>
<td>4</td>
<td>Target parameter flag</td>
<td>#</td>
</tr>
<tr>
<td>5</td>
<td>Zero, defines all digits</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>Space, Also used as padding character</td>
<td>space</td>
</tr>
<tr>
<td>7</td>
<td>Left parenthesis</td>
<td>(</td>
</tr>
<tr>
<td>8</td>
<td>Addition operator</td>
<td>+</td>
</tr>
<tr>
<td>9</td>
<td>Subtraction operator</td>
<td>-</td>
</tr>
</tbody>
</table>
Multiplication operator
Division operator
Right parenthesis

If the character following immediately after }, i.e., in position 13, is not a carriage return or space, the flag line will be considered "extended", affecting the copying of comments, as explained in section 2.2. Contrary, if the character immediately following } is carriage return or space (the latter relevant for punched cards), the flag line is considered "normal", and nothing is changed, with respect to use or the appearance of the generated text.

2.2. Extended Flag Line.

With a simple extension of the flag line, one can specify that comments in the source text shall be copied over to the generated text lines. By "comments" is here meant strings of characters, other than spaces, following the source end-of-line flag, before carriage return or end of line. As required by assembler or other systems program, to be used for later processing of the generated text, comments in the generated text should usually begin with a special delimiter, after which the remainder of the line will be considered as comments and neglected by the assembler (or post-processor). One often used symbol for this purpose is ; but any single character can be used, since this is to be specified on the extended flag-line. An example of an extended flag-line can look like:

```
*0..(*-*/)B   ; tab
```

where _ here symbolizes one space character, and tab symbolizes one "tab".

The first 12 characters are unchanged. Further along the line is typed the comment delimiter, which here is ; in the position where it is to appear in the generated lines. The comments will be inserted in the generated first line, immediately following the delimiter, such that any tabulator or space characters in front of the comments in the source text will be suppressed, and substituted by tabulator and space characters as necessary to place the delimiter and comments in the proper place on the line. Also, if the source line comments begin with the delimiter character (following possible leading tabs and spaces), this delimiter is suppressed, to give only one delimiter character. Delimiter characters later on the line will be copied normally, however. By "first line" is here meant the first of the the group of lines generated by one macro, i.e., the group of lines corresponding to the source line.

The first character following } (i.e., in position 13) in the flag line should be an integer, in the range 1 to 9, specifying $W$ = the number of spaces equivalent to one tabulator. The next character should be the tabulator character ("tab"). Following this, comes any number of more "tabs" (may be zero), followed by any number of spaces, which can also be zero. Tabs and spaces can not be mixed.

More concisely: If the number of consecutive tabs in the flag line is $k$, followed by $j$ spaces, the comment delimiter will be placed in position number:

$$ D = [((13 : W) + k) \ast W + j] $$

where : symbolizes integer division, discarding remainder.

The integer 13 originates from the thirteen leading characters on the line, before the first tab. As a matter of fact, one can simply forget the formula and just put the comment delimiter in the proper position along the line.

If the generated string, before comments, extends beyond the position specified for start of
To summarize, the character positions along an extended flag line have the following significance:

<table>
<thead>
<tr>
<th>Pos. in Flag line</th>
<th>Function of Character</th>
<th>Usual character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Source end-of-line flag</td>
<td>Ø</td>
</tr>
<tr>
<td>2</td>
<td>Source parameter flag</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>Target end-of-line flag</td>
<td>§</td>
</tr>
<tr>
<td>4</td>
<td>Target parameter flag</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Zero, defines all digits</td>
<td>O</td>
</tr>
<tr>
<td>6</td>
<td>Space, Also used as padding character</td>
<td>space</td>
</tr>
<tr>
<td>7</td>
<td>Left parenthesis</td>
<td>(</td>
</tr>
<tr>
<td>8</td>
<td>Addition operator</td>
<td>+</td>
</tr>
<tr>
<td>9</td>
<td>Subtraction operator</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>Multiplication operator</td>
<td>*</td>
</tr>
<tr>
<td>11</td>
<td>Division operator</td>
<td>/</td>
</tr>
<tr>
<td>12</td>
<td>Right parenthesis</td>
<td>)</td>
</tr>
<tr>
<td>13</td>
<td>W, number of spaces between tabulator positions</td>
<td>8</td>
</tr>
<tr>
<td>14+i</td>
<td>Tabulator character value</td>
<td>tab</td>
</tr>
</tbody>
</table>

Possible further characters, up to carriage return, are ignored.

3. DESCRIPTION OF MODIFICATIONS IN STAGE2

The modification of STAGE2, necessary to record extended flag-line and provide copying of comments, consists of 3 parts:

1. Extension of the first part of STAGE2, reading the flag line.
2. Modification of the line input routine.
3. Modification and extension of the routine that outputs characters to the line buffer.

Part 1 is close to the beginning of STAGE2, part 2 is around the label LOC 03, and part 3 is at LOC 57. The modifications are shown in the enclosed listings, on the following pages. The modifications are distinguished from the original parts of the program, by the use of small letters for comments. Also, the new or modified lines are not finished with the word STG2, which indicates original program text. The semicolon, introducing each comment, is superfluous here, it is merely included due to a habit of the author. The listings included here are only extracts of the program, showing the modifications and their surroundings. By comparison with a complete listing of the original text, it should be fairly simple to spot the places where the modifications are made.

The modified program partly explains itself, through the comments included. A couple of further details to note are, however:

- The set of registers of the simulated FLUB machine is extended with 6 more triples: FLG, VAL, and
PTR, with suffixes AA, AB, AD, AE, AF, and ZC. This involves that the modified version of STAGE2 no longer can be translated by SIMCMP, since one more character is used in these variable names. For the bootstrapping implementation is therefore recommended, that the original version is used, until a primitive version of STAGE2 is running. Then, this one can be used to translate the modified version.

The variables mentioned are used in the following applications:

<table>
<thead>
<tr>
<th>VAL</th>
<th>PTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>value of char</td>
</tr>
<tr>
<td>AB</td>
<td>not used</td>
</tr>
<tr>
<td>AD</td>
<td>working variable</td>
</tr>
<tr>
<td>AE</td>
<td>tab character value</td>
</tr>
<tr>
<td>AF</td>
<td>comment delimiter value</td>
</tr>
<tr>
<td>ZC</td>
<td>not used</td>
</tr>
</tbody>
</table>

One more remark is important, concerning PTR ZC: This variable is also manipulated, and changed, by the internal mechanisms of the I/O-package:

PTR ZC is set to 0 at each call of READ NEXT = and of WRITE NEXT =. It is incremented by each statement CHAR = VAL =. Here, = signifies any valid parameter, according to macro notations.

Only PTR ZC is affected by internal operations. All other FLUB registers are only modified by the STAGE2 program, as positively expressed by FLUB statements.

4. REFERENCES


APPENDIX

The following pages comprise:
1. Extracts from listings of STAGE2, in FLUB.
2. Example: Macros for translation from FLUB to assembly for PDP-10
3. Extracts from PDP-10 assembly version of STAGE2, extracts corresponding to item 1 above, as translated with macros, item 2.
APPENDIX 1: Extracts from listings of STAGE2, in FLUB.

(Section containing routines for input of Flag line and for input of normal lines)

PTR J = 0 + 0.
FLG L = 1.       END-OF-LINE INDICATOR.
VAL L = 0 - 1.   CARRIAGE RETURN IS -1.
PTR L = 0 + 0.   LOCATION COUNTER.
VAL N = CHAR.   LEFT PARENTHESIS.
PTR N = 0 + 0.   RESET THE SKIP COUNT.
FLG N = CHAR.   SET EXPRESSION SIGN POSITIVE.
VAL N = CHAR.   ADDITION OPERATOR.
FLG O = 0.      SUBTRACTION OPERATOR.
VAL O = CHAR.   MULTIPLICATION OPERATOR.
VAL P = CHAR.   DIVISION OPERATOR.
VAL R = CHAR.   RIGHT PARENTHESIS.

VAL AF = 0 + 0.   : read extension of FLAG LINE
VAL AE = 0 + 0.   : initialize
PTR AF = 0.       : remains=0 if no extension
PTR AE = 0.       : no extension of FLAG LINE
PTR AD = 0.       : corresp. no. positions for TAB
VAL AD = CHAR.   : into tab.-parameter
TO 1E IF VAL AD LT 0. : POSNO(No of pos.1)=13
TO 1F IF VAL AD = F. : NMTAB = POSNO/TABPOS
PTR AF = AF / AD. : initialize POSNO
PTR AE = AE + 1. : TAB-character value
LOC 1A.
 PTR AE = AE + 1.
LOC 1B.
 TO 1C IF VAL AD NE F. : read new if space
PTR AF = AF + 1. : count no. of spaces
VAL AD = CHAR. : read new if space
TO 1B.
LOC 1C.
TO 1D IF VAL AD LT 0. : terminate extension
TO 1A IF VAL AD = AE. : read new if tab
VAL AF = AD. : comment-delimiter
LOC 1D.
PTR AE = AE * AD. : calculate extension-parameters
PTR AF = AE + AF. : pos.no. correspond to integer tabs
LOC 1E.
 PTR R = 0 + 0. : continue original STAGE2
PTR 4 = 7 + 7.
PTR 8 = F + 7.
TO 01 BY 0. : LENGTH OF TWO DESCRIPTORS.
LOC 01.
GET I = A. : POINT TO THE FIRST AVAILABLE SPACE.
READ NEXT I. : START WORKING IN EARNEST.
TO 98 IF FLG I NE 0. : ROUTINE TO READ FROM THE INPUT.
PTR I = C + 8. : RECALL THE CHANNEL SPEC.
VAL Y = 0 + 0. : GET OUT UNLESS ALL IS OK.
PTR Y = C + 8. :
TO 82 IF PTR M = 0.  
PTR M = M - 1.  
TO 01.  
LOC 02.  
PTR 9 = 1 \times 0.  
VAL 1 = CHAR.  
PTR 1 = 0 - 7.  
TO 97 IF PTR 8 GE 1.  
STO 9 = 1.  
TO 04 IF VAL 1 = L.  
TO 03 IF VAL 1 = A.  
VAL Y = Y + 1.  
TO 92 IF VAL 1 NE B.  
PTR B = 1 + 0.  
STO 9 = B.  
TO 02.  
LOC 03.  
PTR AA = 9.  
LOC 0A.  
TO 0B IF VAL 1 = L.  
PTR 9 = 1 + 0.  
LOC 06.  
VAL 1 = CHAR.  
TO 01 IF VAL 1 NE A.  
VAL 1 = CHAR.  
LOC 01.  
TO 06 IF VAL 1 = F.  
TO 06 IF VAL 1 = AE.  
TO 01 IF VAL 1 NE AF.  
VAL 1 = CHAR.  
TO 04.  
LOC 0J.  
PTR AA = 9.  
LOC 0H.  
PTR 1 = 9 - 7.  
STO 9 = 1.  
TO 97 IF PTR 8 GE 1.  
TO 0B IF VAL 1 = L.  
PTR 9 = 1 + 0.  
VAL 1 = CHAR.  
TO 04.  
LOC 0B.  
PTR AB = 9.  
LOC 04.  
PTR U = 9 - 7.  
STO U = 3.  
PTR U = U - 7.  
STO U = 3.  
PTR U = U - 7.  
STO U = 3.  
PTR U = U - 7.  
STO U = 3.  
PTR U = U - 7.  
STO U = 3.  
PTR U = U - 7.  
STO U = 3.  
PTR U = U - 7.  
STO U = 3.  
PTR U = U - 7.  
STO U = 3.  
PTR U = U - 7.  
STO U = 3.  

SHOULD THIS LINE BE SKIPPED, NO.  
YES. DROP THE SKIP COUNT  
TRY AGAIN.  
READING LOOP.  
ADVANCE THE SPACE POINTER.  
READ THE NEXT CHARACTER.  
POINT TO THE NEXT CHARACTER SPACE.  
HAVE WE OVERRUN THE AREA, YES.  
PUT AWAY THE CHARACTER.  
WAS THIS A CARRIAGE RETURN, YES.  
HAVE WE COMPLETED THE READ, YES.  
BUMP THE INPUT STRING LENGTH.  
NO, IS THIS A PARAMETER FLAG, NO.  
YES, SET THE PARAMETER POINTER AND  
STORE IT WITH THE PHASE FLAG.  
READ THE REMAINDER OF THE LINE.  
; remark limitmark  
; loop to read comments etc.  
; carret., i.e. no remainder  
; read new if first was "source EOL-flag"  
; eliminate leading spaces  
; eliminate leading tabs  
; continue if no comment-delimiter  
; eliminate comment-delimiter  
; adjust startpointer  
; normal read/store loop  
; error if full  
; terminate when carret.  
; read next  
; and continue in loop  
; remainder of line finished  
; set terminate-pointer  
SCANNER.  
SET ALL PARAMETERS UNDEFINED.
(Section containing the Modified output routine)

STG 8 = 1.
PTR 8 = 8 + 7.
TO 97 IF PTR 8 GE 9.
VAL 1 = CHAR.
TO 55 IF VAL 1 NE C.
FLG H = 0.
LOC 55.
RETURN BY D.
LOC 56.
VAL W = 3 + 0.
PTR X = C + 0.
PTR AA = AB.
LOC 57.
GET X = X.
TO 80 IF VAL X NE L.
TO 80 IF PTR AD = 0.
TO 80 IF PTR AA = AB.
LOC 2A.
TO 2B IF PTR ZC GE AE.
CHAR = VAL AE.
TO 2A.
LOC 2B.
TO 2C IF PTR ZC GE AF.
CHAR = VAL F.
TO 2B.
LOC 2C.
TO 8C IF VAL AF = 0.
CHAR = VAL AF.
LOC 8C.
GET AA = AA.
TO 2E IF VAL AA NE A.
LOC 2D.
GET AA = AA.
LOC 2E.
CHAR = VAL AA.
TO 8F IF FLG AA = 1.
TO 20 IF PTR AA NE AB.
LOC 8D.
CHAR = VAL X.
TO 8E IF VAL X NE L.
TO 57 IF FLG X NE 1.
LOC 8F.
PTR AA = AB.
LOC 8E.
WRITE NEXT W.
TO 98 IF FLG W NE 0.
TO 55 IF VAL X = L.
CHAR = VAL X.
TO 57.
LOC 58.
PTR Z = W + Z.
TO 60 IF PTR W NE 0.
TO 71 IF FLG B = 2.
LOC 59.
TO 70 IF PTR V GE 9.
GET Z = V.
GET Y = Q.
GET X = Y.
TO 63 IF FLG Z = 2.
TO 64 IF FLG Z = 3.
APPENDIX 2: Example: Macros for translation from FLUB to assembly for PDP-10

```
# # # 0 (+-/*) 8
GET # = #.
IF AC2 = 'PTDR#28' SKIP 1$
    MOVE 2,PTDR#28#F1$
SET AC2 TO 0$
    MOVIE 15,FLG#18#F1$
    JSR UNPACK#F1$
$
STO # = #.
IF AC2 = 'PTDR#18' SKIP 2$
    MOVE 2,PTDR#18#F1$
SET AC2 TO 'PTDR#18$
    MOVIE 15,FLG#20#F1$
    JSR PACK#F1$
$
FLG # = #.
IF #28 NE 0 SKIP 2$
    SETZM FLG#18#F1$
SKIP 4$
IF AC2 = 'FLG#28' SKIP 1$
    MOVE 2,FLG#28#F1$
SET AC2 TO 'FLG#18$
    MOVEM 2,FLG#18#F1$
$
VAL # = PTR #.
IF AC2 = 'PTDR#28' SKIP 1$
    MOVE 2,PTDR#28#F1$
SET AC2 TO 'VAL#18$
    MOVEM 2,VAL#18#F1$
$
PTR # = VAL #.
IF AC2 = 'VAL#28' SKIP 1$
    MOVE 2,VAL#28#F1$
SET AC2 TO 'PTR#18$
    MOVEM 2,PTR#18#F1$
$
PTR # = 0.
    SETZM PTR#18#F1$
$
VAL # = 0.
    SETZM VAL#18#F1$
$
# # = 0 + 0.
    SETZM #18#20#F1$
$
# # = # + 0.
IF AC2 = '#10#30' SKIP 1$
    MOVE 2,#10#30#F1$
SET AC2 TO '#10#20$
```
```
# # = # + 1.
IF #20 NE #30 SKP 3$
IF AC2 = '#10#30' SKIP 5$
  AOS  #10#20#F1$
  SKIP 5$
  IF AC2 = '#10#30' SKIP 1$
   MOVE 2,#10#30#F1$
   SET AC2 TO '#10#20'$
   ADJ 2,#F1$
   MOVEM 2,#10#20#F1$
$
# # = # - 1.$
IF #30 NE #30 SKP 3$
IF AC2 = '#10#30' SKIP 5$
  SOS  #10#20#F1$
  SKIP 5$
  IF AC2 = '#10#30' SKIP 1$
   MOVE 2,#10#30#F1$
   SET AC2 TO '#10#20'$
   SOJ 2,#F1$
   MOVEM 2,#10#20#F1$
$
# # = # + 1.$
IF AC2 = '#10#30' SKIP 8$
IF AC2 = '#10#40' SKIP 8$
  MOVE 2,#10#30#F1$
  AOD 2,#10#40#F1$
  SKIP 1$
  ADO 2,#10#30#F1$
  SET AC2 TO '#10#20'$
  MOVEM 2,#10#20#F1$
$
# # = # - 1.$
IF AC2 = '#10#30' SKIP 1$
  MOVE 2,#10#30#F1$
  SUB 2,#10#40#F1$
  SET AC2 TO '#10#20'$
  MOVEM 2,#10#20#F1$
$
# # = # - 1.$
IF #20 = #30 SKP 4$
IF AC2 = '#10#30' SKIP 1$
  MOVE 2,#10#30#F1$
  SET AC2 TO '#10#20'$
  MOVEM 2,#10#20#F1$
```
PTR # = # * #.
IF #30 NE 7 SKP 3$
PTR #10 = #20$
SKIP 5$
IF AC2 = 'PTR#20' SKIP 2$
IF AC2 = 'PTR#30' SKIP 3$
MOVE 2,PTR#20#F1$
IMUL 2,PTR#30#F1$
SKIP 1$
IMUL 2,PTR#20#F1$
NOVEM 2,PTR#30#F1$
SET AC2 TO 'PTR#10'$
$ 
PTR # = # / #.
IF #30 NE 7 SKP 3$
PTR #10 = #20$
SKIP 5$
IF AC2 = 'PTR#20' SKIP 1$
MOVE 2,PTR#20#F1$
DIV 2,PTR#30#F1$
NOVEM 2,PTR#10#F1$
SET AC2 TO 'PTR#10'$
$ 
TO # IF # = = #.
IF AC2 = '#20#30' SKIP 2$
MOVE 2,#20#30#F1$
SET AC2 TO '#20#30'$
CAIN 2,#20#40#F1$
JRST LOC#10#F1$
$ 
TO # IF # NE 0.
IF AC2 = '#20#30' SKIP 2$
MOVE 2,#20#30#F1$
SET AC2 TO '#20#30'$
JUMP 2,LOC#10#F1$
$ 
TO # IF # GE 0.
IF AC2 = '#20#30' SKIP 2$
MOVE 2,#20#30#F1$
SET AC2 TO '#20#30'$
JUMPE 2,LOC#10#F1$
$ 
TO # IF # = 0.
IF AC2 = '#20#30' SKIP 2$
MOVE 2,#20#30#F1$
SET AC2 TO '#20#30'$
JUMPE 2,LOC#10#F1$
$ 
TO # IF # NE #.
IF AC2 = '#20#30' SKIP 3$
IF AC2 = '#20#40' SKIP 4$
MOVE 2,#20#30#F1$
SET AC2 TO '#20#30'$
CAIN 2,#20#40#F1$
SKIP 1$
CAIN 2,#20#30#F1$
JRST LOC#10#F1$
TO # IF # # GE #.
IF AC2 = '20#30' SKIP 3$  
IF AC2 = '20#40' SKIP 4$  
  MOVE 2,#20#30#F1$  
SET AC2 TO '20#30'  
  CML 2,#20#40#F1$  
  SKIP 1$  
  CAMG 2,#20#30#F1$  
  JRST LOC#10#F1$  
$  
TO # IF # # #.  
IF AC2 = '20#30' SKIP 2$  
  MOVE 2,#20#30#F1$  
SET AC2 TO '20#30'  
IF AC13 = 'BOL#40' SKIP 2$  
SET AC13 TO 'BOL#40'  
  MOVE 13,BOL#40#F1$  
  MOVE 15,#20#50#F1$  
  JSR 80L#F1$  
  JRST LOC#10#F1$  
$  
TO #.  
  JRST LOC#10#F1$  
$  
TO # BY #.  
  MOVE 14,PTR#20#F1$  
  MOVE 15,LOC#10#F1$  
  JSP 13,SUBRT#F1$  
SET AC13 TO 6$  
RETURN BY #.  
  MOVE 13,PTR#10#F1$  
  JRST (13)#F1$  
  SET AC13 TO 0$  
$  
LOC #.  
LOC#10:#F1$  
SET AC2 TO 0$  
SET AC13 TO 0$  
$  
STOP.  
SETA$  
EXITS  
$  
END PROGRAM.  
LOWEND-.#F1$  
BLOCK MASS12#F1$  
HIGHEND-.#F1$  
END START#F1$  
#F0$  
$
READ NEXT #.
  MOVE 5,VAL#10#F1$
  JSR READIN#F1$
  MOVEM 2,FLG#10#F1$
  SET AC2 TO 'FLG#10'
$  VAL # = CHAR.
  JSR GET1C#F1$
  MOVEM 2,VAL#10#F1$
  SET AC2 TO 'VAL#10'
$  CHAR = VAL #.
  SET AC2 TO 'VAL#10'
  MOVE 2,VAL#10#F1$
  JSR UTCVAR#F1$
  MOVEM 3,FLG#10#F1$
$  WRITE NEXT #.
  MOVE 5,VAL#10#F1$
  JSR WRTLIN#F1$
  MOVEM 2,FLG#10#F1$
  SET AC2 TO 'FLG#10'
$  REWIND #.
  MOVE 5,VAL#10#F1$
  JSR REWIND#F1$
  MOVEM 3,FLG#10#F1$
$  MESSAGE # TO #.
  MOVE 13, (ASCII/#10/#) #F1$
  MOVE 15,VAL#20#F1$
  JSR MSGOUT#F1$
  MOVEM 3,FLG#20#F1$
$  SET # TO #.
  #F3$
$  IF # = # SKIP #.
  IF #11 = #20 SKP #30$
$  IF # = # SKIP #.
  #F50$
$  IF # NE # SKIP #.
  #F51$
$  SKIP #.
  #F4$
$$
Appendix 3: Extracts from PDP-10 assembly version of STAGE2, extracts corresponding to Appendix 1, as translated with macros, Appendix 2.

(Section containing routines for input of Flag line and for input of normal lines)

```assembly
SETZM PTRJ  :STG2
MOVE 2,FLG1 :END-OF-LINE INDICATOR.
MOVE 2,FLG0
MOVE 2,VAL0  :CARRIAGE RETURN IS -1.
CUD 2,
MOVE 2,VALL
SETZM PTRL  :LOCATION COUNTER.
JSR GETIC  :LEFT PARENTHESIS.
MOVE 2,VALM
SETZM PTRM  :RESET THE SKIP COUNT.
SETZM FLGN  :SET EXPRESSION SIGN POSITIVE.
JSR GETIC  :ADDITION OPERATOR.
MOVE 2,VALN
SETZM FILD  :STG2
JSR GETIC  :SUBTRACTION OPERATOR.
MOVE 2,VALP
JSR GETIC  :MULTIPLICATION OPERATOR.
JSR GETIC  :DIVISION OPERATOR.
JSR GETIC  :RIGHT PARENTHESIS.
MOVE 2,VALR

SETZM VALAF  : read extension of FLAG LINE
SETZM VAFAE  : initialize
SETZM PTRAE
SETZM PTRAF
SETZM PTRAD  : remains=0 if no extension
JSR GETIC
MOVE 2,VALAD
MOVE 13,B0L1L
MOVE 15,VAL0
JSR BOOL
JRST LOC1E
CAMN 2,VALF  : no extension of FLAG LINE
JRST LOC1E
SUB 2,VALE
MOVE 2,VALAD
MOVE 2,PTRAD  : into tab.-parameter
MOVE 2,PTR5  : POSNO(no of pos.)==13
ADD 2,PTR3
MOVE 2,PTRAF
MOVE 2,PTRAF
MOVE 2,PTRAD
MOVE 2,PTRAE
SETZM PTRAE  : initialize POSNO
JSR GETIC  : TAB-character value
MOVE 2,VALE

LOC1A:
AOS  PTRAE  : count no. of tabs
JSR GETIC
MOVE 2,VALAD
MOVE 13,B0L1L
MOVE 15,VAL0
JSR BOOL
JRST LOC1D
CAMN 2,VAFAE
JRST LOC1A  : read new if tab
```
LOC1B:
MOVE 2,VALAO : jump if not space
CAME 2,VALF
JIRST LOC1C
AOS PTRA F
JSR GETIC
MOVEN 2,VALAO
JIRST LOC1B
LOC1C:
MOVE 2,VALAO : terminate extension
MOVE 13,BOLLT
MOVE 15,VALO
JSR BOOL
JIRST LOC1O
CANN 2,VALAE
JIRST LOC1A
MOVEN 2,VALAF : read new if space
LOC10:
MOVE 2,PTRA E
INUL 2,PTRA O
MOVEN 2,PTRA E
ADD 2,PTRAF
MOVEN 2,PTRAF
LOC1E:
SETZM PTRR : continue original STAGE2
MOVE 2,PTR7 ;SET NO REPETITION IN PROGRESS.
ADJ
MOVEN 2,PTR4 ;LENGTH OF TWO DESCRIPTORS.
ADJ 2.
MOVEN 2,PTR8 ;POINT TO THE FIRST AVAILABLE SPACE.
ADJ
MOVEI 14,PRO
MOVEI 15,LOC01 STG2
JSP 13,SUBRT STG2
LOC01:
MOVE 2,PTRA ;ROUTINE TO READ FROM THE INPUT.
MOVE 15,FLGI ;RECALL THE CHANNEL SPEC.
JSR UNPACK STG2
MOVE 5,VALI STG2
JSR READIN
MOVE 2,FLGI
JUMPN 2,LOC38 ;GET OUT UNLESS ALL IS OK.
MOVE 2,PTRC
MOVEI 2,PTR1 STG2
SETZM VALY STG2
MOVE 2,PTRC STG2
MOVE 2,PTRY STG2
MOVE 2,PTRM ;SHOULD THIS LINE BE SKIPPED, NO.
JUMPE 2,LOC02 STG2
SOJ 2, STG2
MOVEN 2,PTRM JIRST LOC01
STG2
LOC02:
MOVE 2,PTRI STG2
MOVEI 2,PTR9 STG2
JSR GETIC STG2
MOVEI 2,VAL1 STG2
MOVE 2,PTR9 ;POINT TO THE NEXT CHARACTER SPACE.
SOJ 2, STG2
MOVEI 2,PTRI STG2
LOC08:
MOVE 2, PTR9
MOVEM 2, PTRAB
JSR GETIC
JST LOC08

LOC04:
MOVE 2, PTR9
MOVEM 2, PTRU
MOVEI 15, FLG3
JSR PACK
SOJ 2.

: read next
: and continue in loop
: remainder of line finished
: set terminate-pointer

; scanner.
; set all parameters undefined.

STG2
STG2
(Section containing the Modified output routine)

MOVE 2, PTR8
MOVEV 15, FLG1
JSR UNPACK
ADVANCE THE SPACE POINTER.

ADJ 2,
MOVEM 2, PTR8
CAML 2, PTR9
JRST LOC97
GET THE NEXT CHARACTER.

MOVEM 2, VALI
CAME 2, VALC
JRST LOC55
;jio that close the definition phase, no.

SETZM FLGB
;YES, RESET THE PHASE FLAG.

LOC55:
MOVE 13, PTR0
JRST (13)
;COMMON SYSTEM RETURN POINT.
REGISTER C IS THE RETURN ADDRESS.

LOC56:
MOVE 2, VAL3
;PUNCH AN UNRECOGNIZED LINE.

LOC57:
MOVE 2, VAL4
;ADDRESS THE FIRST CHARACTER.

LOC57:
MOVE 2, PTRX
;eliminate double output of comments

MOVE 15, FLGX
JSR UNPACK
;output characters

LOC57:
MOVE 2, VALX
;norm. loop, get character

CAME 2, VALL
JRST LOC00

MOVEM 2,PTRX
;continue in normal loop if not CR

CAML 2, PTRC
JRST LOC00
;or if "main extension switch" off

MOVEM 2,PTRX
;or if no remainder in this line

LOC2A:
MOVE 2, PTRZC
CAML 2, PTRAE
JRST LOC2B
MOVEM 3, FLGAE
JRST LOC2A
;output remainder of line:

LOC2B:
MOVE 2, PTRZC
CAML 2, PTRAF
JRST LOC2C
MOVEM 3, FLGF
JRST LOC2B
;loop for spaces

LOC2C:
MOVE 2, VALAF
JRST LOC00
MOVEM 3, FLGAF
;put out space(s)

LOC0C:
MOVE 2, PTRAA
MOVEM 15, FLGAA
JSR UNPACK
;loop for output of remaining string

;jio that close the definition phase

YES, SET THE TERMINATOR.

YES, ADVANCE THE SPACE POINT.

YES, HAVE WE OVERRUN THE AREA, YES.

YES THAT CLOSE THE DEFINITION PHASE, NO.

;COMMON SYSTEM RETURN POINT.
REGISTER C IS THE RETURN ADDRESS.

;PUNCH AN UNRECOGNIZED LINE.
;ADDRESS THE FIRST CHARACTER.
;eliminate double output of comments
;output characters
;norm. loop, get character
;continue in normal loop if not CR
;or if "main extension switch" off
;or if no remainder in this line
;output remainder of line:
;if pos.counter less integr. tab.pos
;then output tab and loop
;loop for spaces
;put out space(s)
;loop for output of remaining string
;get first char.
MOVE 2, VALAA ; eliminate possible "source EOL-flag"
CAME 2, VALA
JAST LOC2E

LOC2D:
MOVE 2, PTRAA ; loop for output remainder (comments)
MOVEI 15, FLGAA ; get next character
JSR UNPACK

LOC2E:
MOVE 2, VALAA ; put out the character
JSR UCHAR
MOVEM 3, FLGAA
MOVE 2, FLGAA ; possible terminate
CAMN 2, FLG1
JAST LOC0F
MOVE 2, PTRAA ; go loop if not exhausted
CAME 2, PTRAB
JAST LOC2D

LOC0D:
MOVE 2, VALX ; normal output-loop
JSR UCHAR
MOVEM 3, FLGX
CAMN 2, VALL
JAST LOC0E
MOVE 2, FLGX ; HAVE WE REACHED THE END, NO.
CAME 2, FLG1
JAST LOC57

LOC0F:
MOVE 2, PTRAB ; squeeze possible remaining of "remainder"
MOVEM 2, PTRAA

LOC0E:
MOVE 5, VALW ; end of line reached:
JSR WRATLN
MOVEM 2, FLGW
JUMPN 2, LOC98 ; YES, PUT IT OUT ON THE DESIGNATED CHANNEL. STG2
MOVE 2, VALX ; TREAT ANY ERROR AS FATAL.
CAMN 2, VALL ; ELSE IF THE LINE IS COMPLETE, RETURN. STG2
JAST LOC55
MOVE 2, VALX ; ELSE REPRINT THE LAST CHARACTER
JSR UCHAR
MOVEM 3, FLGX
JAST LOC57

LOC58:
MOVE 2, PTRW ; AND CONTINUE.
ADD 2, PTRZ
MOVEM 2, PTRY
JUMPN 2, LOC60 ; TRY FOR AN ALTERNATIVE MATCH.
MOVE 2, FLGB ; GET THE POINTER TO THE ALTERNATIVE.
CAMN 2, FLG2
JAST LOC71

LOC59:
MOVE 2, PTRY ; TRY EXTENDING THE PREVIOUS PARAMETER.
CAML 2, PTR9
JAST LOC70
MOVEI 15, FLGZ ; IS THERE ONE TO EXTEND, NO.
JSR UNPACK

MOVE 2, PTRQ ; RECALL THE MACRO POINTER.
MOVEI 15, FLGY
JSR UNPACK

MOVE 2, PTRY ; YES, RECALL THE INPUT POINTER
MOVEI 15, FLGX
JSR UNPACK

MOVE 2, PTRY ; AND THE CURRENT CHARACTER
MOVEI 15, FLGY
JSR UNPACK
MOVE 2,FLGZ  :IS THIS THE FIRST TIME FOR A PARAMETER, YES STG2
CAMN 2,FLG2
JRST L0C63
CAMN 2,FLG3  :NO, IS IT A PARAMETER EXTENSION, YES STG2
JRST L0C64
THE MACRO PROCESSING SYSTEM STAGE 2: TRANSFER OF COMMENTS TO THE GENERATED TEXT.
This paper is a short description of a small extension of STAGE2, providing possibilities to copy comments etc. from the source text to the generated text. The description presupposes familiarity with the STAGE2 system: its purpose, use and descriptions, like [1] to [9]. Only section 3 of this paper requires knowledge of the internal structures and working of the system, and that section is unnecessary for the plain use of the described feature. The extension, if not used, is completely invisible to the user: No rules, as described in the original literature, are changed. A user, unaware of the extension, will see no difference from the original version.