The Macro Processing System \textsc{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 PeterSEN

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>0</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>s p a c e</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>
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”, effecting 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:

```plaintext
.*$0,(_+*/)8 tab 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:

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

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>0</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>$</td>
</tr>
<tr>
<td>4</td>
<td>Target parameter flag</td>
<td>#</td>
</tr>
<tr>
<td>5</td>
<td>Zero, defines all digits</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>Space. Also used as padding character</td>
<td>space</td>
</tr>
<tr>
<td>7</td>
<td>Left parenthesis</td>
<td>(</td>
</tr>
<tr>
<td>8</td>
<td>Addition operator</td>
<td>+</td>
</tr>
<tr>
<td>9</td>
<td>Subtraction operator</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>Multiplication operator</td>
<td>*</td>
</tr>
<tr>
<td>11</td>
<td>Division operator</td>
<td>/</td>
</tr>
<tr>
<td>12</td>
<td>Right parenthesis</td>
<td>)</td>
</tr>
<tr>
<td>13</td>
<td>$W_i$, number of spaces between tabulator positions</td>
<td>8</td>
</tr>
<tr>
<td>i=0,1,...,k-1</td>
<td>Tabulator character value</td>
<td>tab</td>
</tr>
<tr>
<td>next j</td>
<td>Space, as padding character, used to fix the start of the comment</td>
<td>space</td>
</tr>
<tr>
<td>(j=0 permitted)</td>
<td></td>
<td>section between two tab positions</td>
</tr>
<tr>
<td>14+i</td>
<td>Comment delimiter</td>
<td>;</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-text 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 suffix: 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>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAL</td>
<td>value of char</td>
</tr>
<tr>
<td>AA</td>
<td>working variable</td>
</tr>
<tr>
<td>AB</td>
<td>tab character value</td>
</tr>
<tr>
<td>AD</td>
<td>comment delimiter value</td>
</tr>
<tr>
<td>AE</td>
<td>not used</td>
</tr>
<tr>
<td>AF</td>
<td>not used</td>
</tr>
<tr>
<td>ZC</td>
<td>value of char not used</td>
</tr>
<tr>
<td>PTR</td>
<td>pointer, current character of comment text</td>
</tr>
<tr>
<td>PTR</td>
<td>pointer, end of comment text</td>
</tr>
<tr>
<td>PTR</td>
<td>number of positions between each tab position</td>
</tr>
<tr>
<td>PTR</td>
<td>pos. no. corresponding to integer tabs before comments</td>
</tr>
<tr>
<td>PTR</td>
<td>pos. no. of beginning comments</td>
</tr>
<tr>
<td>PTR</td>
<td>pointer for current character during output</td>
</tr>
</tbody>
</table>

Onemore 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.
VAL L = 0 + 1.
PTR L = 0 + 0.
VAL1 = CHAR.
FTR N = 0 + 0.
FLG N = 0.
VAL N = CHAR.
FLG O = 0.
VAL O = CHAR.
VAL R = CHAR.
VAL AF = 0 + 0.
VAL AE = 0 + 0.
PTR AE = 0.
PTR AD = 0.

; read extension of FLAG LINE
VAL AF = 0 + 0.
VAL AE = 0 + 0.
PTR AE = 0.
PTR AD = 0.

; remains=0 if no extension
VAL AD = CHAR.
TO 1E IF VAL AD LT 0.
TO 1E.
VAL AD = AD + 0.

; no extension of FLAG LINE
TO 1E IF VAL AD LT 0.
TO 1E.
VAL AD = AD + 0.

; no extension of FLAG LINE
TO 1E IF VAL AD LT 0.
TO 1E.
VAL AD = AD + 0.

; corresponds no. positions for TAB
PTR AD = VAL AD.
PTR AF = AF + 0.
PTR AE = AE + 0.

; into tab. -parameter
PTR AD = 5 + 3.
PTR AE = AE + 0.

; initialize POSNO
PTR AD = 0.

; terminate extension
TO 1C.
TO 1D.

; no extension of FLAG LINE
TO 1E.
TO 1E.
TO 1E.

; read new if tab
TO 1D IF VAL AD LT 0.
TO 1D.
VAL AD = AE + 0.

; read new if tab
TO 1E.
TO 1D.

; read new if tab
TO 1E.
TO 1D.

; terminate extension
TO 1E.
TO 1D.

; read new if tab
TO 1E.
TO 1D.

; read new if tab
TO 1E.
TO 1D.

; read comment-delimiter
LOC 1D.
PTR AE = AE * AD.

; pos.no. corresp; to integer tabs
PTR AF = AE + AF.

; pos.no. of beginning comments
LOC 1E.
PTR R = 0 + 0.

; continue original STAGE2
LOC 1E.
PTR R = 0 + 0.

; read new if tab
TO 1D.

; set no repetition in progress.
PTR R = 0 + 0.

; set no repetition in progress.
PTR R = 0 + 0.

; set no repetition in progress.
PTR R = 0 + 0.

; set no repetition in progress.
PTR R = 0 + 0.

; set no repetition in progress.
PTR R = 0 + 0.

; set no repetition in progress.
PTR R = 0 + 0.
TO $2 IF PTR M = 0.
PTR M = M - 1.
TO 01.
LOC 02.
PTR 3 = I + 0.
VAL I = CHAR.
PTR I = 9 - 7.
TO 97 IF PTR 8 GE I.
STO 9 = I.
TO 04 IF VAL I = L.
TO 03 IF VAL I = A.
VAL Y = Y + 1.
TO 02 IF VAL I NE B.
PTR B = I + 0.
STO 9 = B.
TO 02.
LOC 03.
PTR AA = 3.
LOC 0A.
TO OB IF VAL I = L.
PTR 3 = I + 0.
LOC 0G.
VAL I = CHAR.
TO 01 IF VAL I NE A.
VAL I = CHAR.
LOC 01.
TO 0G IF VAL I = F.
TO 0G IF VAL I = A E.
TO 0J IF VAL I NE AF.
VAL I = CHAR.
TO 0H.
LOC 0J.
PTR AA = 3.
LOC 0H.
PTR I = 9 - 7.
STO 9 = I.
TO 97 IF PTR 8 GE I.
TO OB IF VAL I = L.
PTR 3 = I + 0.
VAL I = CHAR.
TO 0H.
LOC OB.
PTR AB = 3.
LOC 04.
PTR U = 3 - 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.
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.
}
Section containing the Modified output routine

ST0 8 = 1.
PTR 8 = 8 + 7.
TO 97 IF PTR 8 GE 9.
VAL I = CHAR.
TO 55 IF VAL I NE C.
FL-G H = 0.
LOC 55.
RETURN BY D.
LOC 56.
VAL J = 3 + 0.
PTR X = C + 0.
PTR AA = AB.
LOC 57.
GET X = X.
TO 0D IF VAL X NE L.
TO 0D IF PTR AD = 0.
TO 0D 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 AF.
TO 2B.
LOC 2C.
TO 0C IF VAL AF = 0.
CHAR = VAL AF.
LOC 0C.
GET AA = AA.
TO 2E IF VAL AA NE A.
LOC 2D.
GET AA = AA.
LOC 2E.
CHAR = VAL AA.
TO 0F IF FLG AA = 1.
TO 2D IF PTR AA NE AB.
LOC 0D.
CHAR = VAL X.
TO 0E IF VAL X = L.
TO 57 IF FLG X NE 1.
LOC 0F.
PTR AA = AB.
LOC 0E.
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 Z NE 0.
TO 71 IF FLG B = 2.
LOC 59.
TO 70 IF PTR V GE 3.
GET Z = V.
GET Y = CL.
GET X = Y.
TO 63 IF FLG Z = 2.
TO 64 IF FLG Z = 3.

; 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:
; loop for spaces
; put out space(s)

; put out comment-de1 imitter
; loop for output of remaining string
; get first char.
; eliminate possible "source EOL-flag"
; loop for output remainder (comments)
; get next character

; put out the character
; possible terminate
; go loop if not exhausted
; normal output-loop

; terminate if CR

HAVE WE REACHED THE END, NO.

; squeeze possible remaining of "remainder"

END OF LINE REACHED:

YES, PUT I T OUt ON THE DESIGNATED CHANNEL. STG2
TREAT ANY ERROR AS FATAL. STG2
ELSE IF THE LINE IS COMPLETE, RETURN. STG2
ELSE REPRINT THE LAST CHARACTER STG2
AND CONTINUE. STG2
TRY FOR AN ALTERNATIVE MATCH. STG2
GET THE POINTER TO THE ALTERNATIVE. STG2
WAS THERE ONE AFTER ALL, YES. STG2
ND, ARE WE DEFINING, YES. STG2
TRY EXTENDING THE PREVIOUS PARAMETER. STG2
IS THERE ONE TO EXTEND, NO. STG2
RECALL THE MACRO POINTER. STG2
YES, RECALL THE INPUT POINTER STG2
AND THE CURRENT CHARACTER STG2
IS THIS THE FIRST TIME FOR A PARAMETER, YES? STG2
NO, IS IT A PARAMETER EXTENSION, YES. STG2
APPENDIX 2: Example: Macros for translation from FLUB to assembly for PDP-10

```
.#$0(+-*/) 8
GET  # = #.
IF AC2 = 'P#20' SKIP 1$ 
    MOVE .2,P#20#F1$
SET AC2 TO 0$ 
    MOVE1 1S,FLG#10#F1$
    JSR UNPACK#F1$
STO # = #.
IF AC2 = 'P#10' SKIP 2%
    MOVE 2,P#10#F1$
SET AC2 TO 'P#10'$
    MOVE1 1S,FLG#20#F1$
    JSR PACK#F1$
FLG # = #.
IF #20 NE 0 SKIP 2$ 
    SETZM FLG#10#F1$
SKIP 4$
IF AC2 = 'F#20' SKIP 1$
    MOVE 2,FLG#20#F1$
SET AC2 TO 'F#10'$
    MOVE2 2S,FLG#10#F1$
VAL # = PTR #.
IF AC2 = 'P#20' SKIP 1$
    MOVE 2,P#20#F1$
SET AC2 TO 'V#10'$
    MOVEM 2,VAL#10#F1$
PTR # = VAL #.
IF AC2 = 'V#10' SKIP 1$
    MOVE 2,VAL#10#F1$
SET AC2 TO 'P#10'$
    MOVEM 2,P#10#F1$
PTR # = 0.
    SETZM P#10#F1$
VAL # = 0.
    SETZM V#10#F1$
# # = 0 + 0.
    SETZM #10#20#F1$
# # = # + 0.
IF AC2 = '#10#30' SKIP 1$
    MOVE 2,#10#30#F1$
    MOVEM 2,#10#20#F1$
SET AC2 TO '#10#20' $ 
```

I $\# = \# + 1.$
IF $\#20 \neq \#30$ SKIP $3$
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#F1$
AOS $\#10#20#F1$

$\# = \# - \#1$.
IF $\#20 \neq \#30$ SKIP $3$
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#F1$
SOS $\#10#20#F1$

$\# = \# + 7.$
$\#10#20 = \#30 + 1$
$\# = \# - 7.$
$\#10#20 = \#30 - 1$
$\# = \# + \#.$
IF AC2 =:'#10#30' SKIP $2$
IF AC2 =:'#10#40' SKIP $3$
MOVE 2,$\#10#30#F1$
ADD 2,$\#10#40#F1$
SKIP $1$
ADO 2,$\#10#30#F1$
SET AC2 TO:'#10#20#F1$
MOVE 2,$\#10#20#F1$

$\# = \# - \#.$
IF AC2 =:'#10#30' SKIP $1$
MOVE 2,$\#10#30#F1$
SUB 2,$\#10#40#F1$
SET AC2 TO:'#10#20#F1$
MOVE 2,$\#10#20#F1$

$\# = \#.$
IF $\#20 = \#30$ SKIP $4$
IF AC2 =:'#10#30' SKIP $1$
MOVE 2,$\#10#30#F1$
SET AC2 TO:'#10#20#F1$
MOVE 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$
MOVEM 2,PTR#10#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$
IDIV 2,PTR#30#F1$
MOVEM 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$
CAMN 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$
JUMPG 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 38
IF AC2 = '#20#40' SKIP 4$
MOVE 2,#20#30#F1$
SET AC2 TO '#20#30$
CAM 2,#20#40#F1$

'SKIP 1$
CAM 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#Fl$
SET AC2 TO '#20#30'$
CAML 2,#20#40#Fl$
Skip 1$
CAML 2,#20#30#Fl$
JRST LOC#10#Fl$

TO # IF # # # # #.
IF AC2 = '#20#30' SKIP 2%
MOVE 2,#20#30#Fl$
SET AC2 TO '#20#30'$
IF AC13 = 'BOL#40' SKIP 2%
SET AC13 TO 'BOL#40'$
MOVE 13,BOL#40#Fl$
MOVE 15,#20#50#Fl$
JSR BOOL#Fl$
JRST LOC#10#Fl$

TO #.
JRST LOC#10#Fl$

TO # BY #.
MOVEI 14,PTR#20#Fl$
MOVEI 15,LOC#10#Fl$
JSR 13,SUBRT#Fl$
SET AC13 TO 0$
RETURN BY #.
MOVE 13,PTR#10#Fl$
JRST (13)#Fl$
SET AC13 TO 0$
SET LOC #.
LOC#10;#Fl$
SET AC2 TO 0$
SET AC13 TO 0%
STOP.
EXITS$

EN0 PROGRAM.$
LOWEND-.#Fl$
BLOCK MASS1Z#Fl$
HIGEND-.#Fl$
END START#F1$
#F8$
READ NEXT #.
  MOVE 5, VAL#10#F1$
  JSR READIN#F1$
  MOVEM 2, FLG#10#F1$

SET AC2 TO 'FLG#10'$

VAL# = CHAR.
  JSR GETIC#F1$
  MOVEM 2, VAL#10#F1$

SET AC2 TO 'VAL#10'$

CHAR = VAL#.
  SET AC2 TO 'VAL#10'$
  JSR UTCHAR#F1$
  MOVEM 3, FLG#10#F1$

WRITE NEXT #.
  MOVE 5, VAL#10#F1$
  JSR WRTLIN#F1$
  MOVEM 2, FLG#10#F1$

REWIND #.
  MOVE 5, VAL#10#F1$
  JSR REWIND#F1$
  MOVEM 3, FLG#10#F1$

MESSAGE # TO #.
  MOVE1 13, [ASCIZ/#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 # = # SKP #.
  #F50$
  IF # NE # SKP #.
  #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;ENO-OF-LINE INDICATOR.
MOVE 2,FLGL
MOVE 2,VAL0;CARRIAGE RETURN IS -1,
SOJ 2,
MOVEM 2,VALL
SETZM PTRL;LOCATION COUNTER.
JSR GETIC;LEFT PARENTHESIS.
MOVEM 2,VALM
SETZM PTRM;RESET THE SKIP COUNT.
SETZM FLGN;SET EXPRESSION SIGN POSITIVE.
JSR GETIC;ADDITION OPERATOR.
MOVEM 2,VALN
SETZM FLGO;SUBTRACTION OPERATOR.
JSR GETIC;MULTIPLICATION OPERATOR.
MOVEM 2,VALP
JSR GETIC;DIVISION OPERATOR.
MOVEM 2,VALQ
JSR GETIC;RIGHT PARENTHESIS.
MOVEM 2,VALR

SETZM VALAF
SETZM VALAE
SETZM PTRAE
SETZM PTRAFL
SETZM PTRAD
JSR GETIC
MOVEM 2,VALAD
MOVE 13,BOLLT
MOVE 15,VAL0
JSR BOOL
JRST LOC1E
CAMN 2,VALF
JRST LOC1E
SUB 2,VALE
MOVEM 2,VALAD
MOVEM 2,PTRAD
MOVE 2,PTR5
ADD 2,PTR3
MOVEM 2,PTRAFL
MOVE 2,PTRAFL
DIV 2,PTRAD
MOVEM 2,PTRAE
SETZM PTRAFL
JSR GETIC
MOVEM 2,VALAE

LOC1A:
AOS PTRAFL
JSR GETIC
MOVEM 2,VALAD
MOVE 13,BOLLT
MOVE 15,VAL0
JSR BOOL
JRST LOC1D
CAMN 2,VALAE
JRST LOC1A

;read extension of FLAG LINE
;initial ire
: remains=0 if no extension
;no extension of FLAG LINE
;no extension of FLAG LINE
;corresp. no. positions for TAB
;into tab. parameter
;POSNO(no of pos.)+=13
;NMTAB = POSNO/TABPOS
;initialize POSNO
;TAB-character value
;count no. of tabs
;terminate extension
;read new if tab
```
LOC1B:
MOVE 2,VALAD
CAME 2,VALF
JRST LOC1C
AOS PTRA F
JSR GETIC
MOVEM 2,VALAD
JRST LOC1B

LOC1C:
MOVE 2,VALAD
MOVE 13,BOLL T
MOVE 15,VAL0
JSR BOOL
JRST LOC1D
CAMA 2,VALAE
JRST LOC1A
MOVEM 2,VALAF

LOC1D:
MOVE 2,PTRAE
IMUL 2,PTRAD
MOVEM 2,PTRAE
ADO 2,PTRAF
MOVEM 2,PTRAF

LOC1E:
SETZM PTRR
MOVE 2,PTR7
A0J 2
MOVEH 2,PTR4
MOVE 2,PTRF
A0J 2
MOVEH 2,PTR8
MOVE1 14,PTRD
MOVE1 15,LOCO1
JSR 13, SUBRT

LOC01:
HOVE 2,PTRA
HOVE1 15,FLGI
JSR UNPACK
HOVE 5,VALI
JSR READIN
HOVEH 2,FLGI
JUPRN 2,LOC98
MOVE 2,PTRC
MOVEH 2,PTRI
SETZH VALY
MOVEH 2,PTRY
MOVE 2,PTRM
JUHPE 2,LOC02

LOC02:
MOVE 2,PTR9
MOVEH 2,PTR9
JSR GETIC
MOVE 2,VALI
HOVE 2,PTR9
SOJ 2
JRST LOC01

LOC02:
MOVE 2,PTR9
MOVEH 2,PTR9
JSR GETIC
MOVE 2,VALI
HOVE 2,PTR9
SOJ 2
JRST LOC01

; execute code here
CAMG 2, PTR8
JRST LOC97
MOVE 2, PTR9
MOVEI 15, FLGI
JSR PACK
MOVE 2, VAL1
CAHN 2, VALL
JRST LOC04
CAMN 2, VALA
JRST LOC03
AOS VALY
CAME 2, VALB
JRST LOC02
MOVE 2, PTRI
MOVEH 2, PTRB
MOVE 2, PTR9
MOVEI 15, FLGB
JSR PACK
JRST LOC02

LOC03:
MOVE 2, PTR9
MOVEH 2, PTRRA
LOC0A:
MOVE 2, VAL1
CAHN 2, VALL
JRST LOC0B
MOVE 2, PTRI
MOVEH 2, PTR9
LOCBG:
JSR GETIC
MOVEM 2, VAL1
CAME 2, VALA
JRST LOC01
JSR GETIC
MOVEM 2, VAL1
JRST LOC0H

LOC0I:
MOVE 2, VAL1
CAHN 2, VALF
JRST LOC0G
CAHN 2, VALAE
JRST LOC0G
CAME 2, VALAF
JRST LOC0J
JSR GETIC
MOVEM 2, VALI
JRST LOC0H

LOC0J:
MOVE 2, PTR9
MOVEM 2, PTRAA
LOC0H:
MOVE 2, PTR9
SOJ 2,
MOVEH 2, PTRI
MOVE 2, PTR9
MOVEI 15, FLGI
JSR PACK
MOVE 2, PTR8
CAHN 2, PTRI
JRST LOC97
MOVE 2, VAL1
CAHN 2, VALL
JRST LOC0B

; 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.
; STG2

; READ THE REMAINDER OF THE LINE.
; remark limit mark
; loop to read comments etc.
; car.ret., i.e. no remainder
; read new if first was "source EOL-f lag"
; 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 car.ret.
LOC04:

MOVE 2, PTR9
SOJ 2.

MOVEH 2, PTRU
HOVE 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.

MOVM 2, PTRU
MOVEH 15, FLG3
JSR PACK

SOJ 2.
(Section containing the Modified output routine)

MOVE 2, PTR8  ; YES, SET THE TERMINATOR.
MOVEI 15, FLG1  ; STG2
JSR PACK
AOJ 2  ; ADVANCE THE SPACE 'POINTER.
MOVEH 2, PTR8
CAML 2, PTR9
J R S T LOC97
JSR GET1C
MOVEH 2, VAL1
CAME 2, VALC
J R S T LOC55
SETZM FLGB

LOC55:  
HOVE 13, PTR0  ; HAVE WE OVERRUN THE AREA, YES.
J R S T (13)

LOC56:  
MOVE 2, VAL3  ; GET THE NEXT CHARACTER.
MOVE2, VALW
MOVE 2, PTRC
MOVE 2, PTRX
MOVE 2, PTRA
MOVE 2, PTRA1

LOC57:  
MOVE 2, PTRX  ; DID THAT- CLOSE THE DEFINITION PHASE, NO.
MOVEI 15, FLGX
JSR UNPACK
MOVE 2, VALX
CAME 2, VALL
J R S T LOC60
MOVE 2, PTRA1
J U H P E 2, LOC60
HOVE 2, PTRA2
CAHN 2, PTRAB
J R S T LOC50

LOC58:  
MOVE 2, PTRZC  ; YES, RESET THE PHASE FLAG.
MOVE2, VALF
CAML 2, PTRA1
J R S T LOC2B
HOVE 2, VALAE
JSR UTCHAR
MOVE 3, FLGAE
J R S T LOC2A

LOC2A:  
MOVE 2, PTRZC  ; COMMON SYSTEM RETURN POINT.
CAML 2, PTRAE
J R S T LOC2B
HOVE 2, VALAE
JSR UTCHAR
MOVE 3, FLGAE
J R S T LOC2B

LOC2B:  
MOVE 2, PTRZC  ; REGISTER 0 IS THE RETURN ADDRESS.
CAML 2, PTRAE
J R S T LOC2C
MOVE 2, VALF
JSR UTCHAR
MOVE 3, FLGAF
J R S T LOC2B

LOC2C:  
MOVE 2, VALF  ; PUNCH AN UNRECOGNIZED LINE.
J U M P E 2, LOC93
MOVE 2, VALF
JSR UTCHAR
MOVE 3, FLGAF
J R S T LOC2A

LOC93:  
MOVE 2, PTRA1  ; CHANNEL 3 USED WHEN A LINE IS NOT HATCHED.
HOVE 2, PTRA2
MOVE 2, VALF
JSR UTCHAR
MOVE 3, FLGAF
J R S T LOC2B

LOC94:  
MOVE 2, VALAF  ; elminate double output of comments
J U M P E 2, LOC93
MOVE 2, VALAF
JSR UTCHAR
MOVEM 3, FLGAF
J R S T LOC2B

LOC95:  
MOVE 2, VALAF  ; ADDRESS THE FIRST CHARACTER.
J U M P E 3, LOC2A

LOC2A:  
MOVE 2, PTRZC  ; output characters
CAML 2, PTRAE
J R S T LOC2B
HOVE 2, VALAE
JSR UTCHAR
MOVE 3, FLGAE
J R S T LOC2A

LOC2B:  
MOVE 2, PTRZC  ; output remainder of line:
CAML 2, PTRAF
J R S T LOC2C
MOVE 2, VALF
JSR UTCHAR
MOVE 3, FLGAF
J R S T LOC2B

LOC2C:  
MOVE 2, VALF  ; output remainder of line:
J U M P E 2, LOC93
MOVE 2, VALF
JSR UTCHAR
MOVE 3, FLGAF
J R S T LOC2A

LOC93:  
MOVE 2, PTRA1  ; if pos. counter < integr. tab. pos

LOC94:  
MOVE 2, VALAF  ; then output tab and loop
J U M P E 2, LOC93
MOVE 2, VALAF
JSR UTCHAR
MOVE 3, FLGAF
J R S T LOC2B

LOC95:  
MOVE 2, VALAF  ; loop for spaces
J U M P E 3, LOC2A

LOC2A:  
MOVE 2, PTRZC  ; put out space(s)
CAML 2, PTRAF
J R S T LOC2C
MOVE 2, VALF
JSR UTCHAR
MOVE 3, FLGAF
J R S T LOC2B

LOC2C:  
MOVE 2, VALF  ; put out comment-delimiter
J U M P E 3, LOC2A

LOC93:  
MOVE 2, PTRA1  ; loop for output of remaining string
H O V E 1 15, FLGAA
J S R UNPACK

LOC94:  
MOVE 2, VALAF  ; get first char.
MOVE 2, VALAA
CAVE 2, VALA
JRST LOC2E

; eliminate possible “source EOL-f lag”

LOC2D:
MOVE 2, PTRAA
MOVE1 15, FLGAA
JSR UNPACK

; loop for output remainder (comments)
; get next character

LOC2E:
MOVE 2, VALAA
JSR UCHAR
MOVEM 3, FLGAA
MOVE 2, FLGAA
CANN 2, FLG1
JRST LOC08
MOVE 2, PTRAA
CAME 2, PTRAB
JRST LOCZD

; put out the character
; possible terminate
; go loop if not exhausted

LOC0D:
MOVE 2, VALX
JSR UTCHAR
MOVEM 3, FLGX
CANN 2, VALL
JRST LOC0E
MOVE 2, FLGX
CAME 2, FLG1
JRST LOC2D

; normal output-loop

LOC0F:
MOVE 2, PTRAB
MOVEM 2, PTRAA

; squeeze possible remaining of “remainder”

LOC0E:
MOVE 5, VALW
JSR WRLIN
MOVEM 2, FLGW
JUMPN 2, LOC98
MOVE 2, VALX
CANN 2, VALL
JRST LOC0E
MOVE 2, VALX
JSR UTCHAR
MOVEM 3, FLGX
JRST LOC57

; end of line reached:
; HAVE WE REACHED THE END, NO.
; HAVE WE REACHED THE END, YES, PUT IT OUT ON THE DESIGNATE0 CHANNEL.

LOC58:
MOVE 2, PTRW
ADD 2, PTRZ
MOVEM 2, PTRZ
MOVE 2, PTRW
JUMPN 2, LOC60
MOVE 2, FLGB
CANN 2, FLG2
JRST LOC71

; AND CONTINUE.
; TRY FOR AN ALTERNATIVE MATCH.
; GET THE POINTER TO THE ALTERNATIVE.

LOC59:
MOVE 2, PTRV
CAML 2, PTR9
JRST LOC70
MOVEM 15, FLGZ
JSR UNPACK
MOVE 2, PTR0
MOVEM 15, FLGY
JSR UNPACK
MOVE 2, PTRY
MOVEM 15, FLGX
JSR UNPACK

; TRY EXTENDING THE PREVIOUS PARAMETER.
; IS THERE ONE TO EXTEND, NO.
; IS THERE ONE TO EXTEND, YES.

LOC70:
MOVE 2, PTRV
AFTM 2, PTR9
JRST LOC70
MOVEM 15, FLGZ
JSR UNPACK
MOVE 2, PTR0
MOVEM 15, FLGY
JSR UNPACK
MOVE 2, PTRY
MOVEM 15, FLGX
JSR UNPACK

; RECALL THE MACRO POINTER.
; YES, RECALL THE INPUT POINTER
; AND THE CURRENT CHARACTER
MOVE 2,FLGZ
CAMN 2,FLG2
JRST LOC63
CAMN 2,FLG3
JRST LOC64

; IS THIS THE FIRST TIME FOR A PARAMETER, YES STG2

; NO, IS IT A PARAMETER EXTENSION, YES. STG2