stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IW 3/73 MARCH

G. TEN VELDEN

MIXAL ASSEMBLY

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS (MOS) subject classification scheme: 68A10, 68A15

ACM-Computing Reviews-category 4.1, 4.2

Abstract

MIXAL is the assembly language for the (hypothetical) MIX computer, introduced by D.E. Knuth. In the present report, an assembly program for this language is described, based upon a slightly modified syntactic definition of MIXAL.

The assembler has been written first as an ALGOL 60 program. Care has been taken to provide it with a clear structure; in particular, the whole (12 pages) program contains only one goto statement. Next, the same algorithm has been implemented in MIXAL on the MIX computer. The latter version of the MIXAL assembler has been used as a MIXAL program to be assembled by the ALGOL 60 version. Both versions of the assembler have been reproduced in this report.

Contents

1. Introduction.

In this report, an one-pass assembler for MIXAL is described, MIXAL being the assembly language for the (hypothetical) MIX computer,designed and described by Knuth [1].
The assembler has been written in ALGOL 60 as a boolean procedure, called "mixal assembler".
The assembler has been tested in the environment of a MIX simulator, written in ALGOL 60 by R.P. van de Riet [2].
The programs have been run on the EL X8 computer of the Mathematical Centre at Amsterdam, under supervision of the ALGOL 60 system, called MILLI [3].

The MIXAL system, consisting of assembler and simulator, accepts several MIXAL programs, textually being separated from each other by a job separator (a dollar symbol: $), and which are translated and executed consecutively.
The interface of assembler and simulator consists of two parameters to procedure "mixal assembler", in addition to the boolean value, delivered by its procedure identifier. The first parameter "m" represents the memory of the MIX computer; actually it has to be an array, being declared with the bound pair 0:3999. All locations of this memory have to be set equal to zero, before calling procedure "mixal assembler"; those locations for which this is not the case, cannot be filled by the assembler (see section 5.3).
The second parameter "start address" is an output parameter of type integer, passing to the simulator the location at which the execution of the program has to be started.
At last, the boolean value, delivered by the procedure identifier of "mixal assembler", is the complement of the boolean variable "erroneous", which is local to procedure "mixal assembler"; "erroneous" indicates whether or not errors have been detected during assembly of the MIXAL program.
The assembler has to take into account that the simulator is a base ten machine (the bytesize being one hundred), with a special internal code for the characters, and a special operation code (a field specification (3:5) is coded as 35, being the value of 3*10+5, instead of 29=3*8+5).
Besides the ALGOL 60 version of the MIXAL assembler, we also have developed a version written in MIXAL, the algorithm being the same, except for some

very little details.
This MIXAL version has been reproduced in chapter 8, in the way it has been printed out by the MIXAL assembler in ALGOL 60, reproduced in chapter 7; thus we use this MIXAL version as a demonstration and a test case of the ALGOL 60 version. Finally, in chapter 9, we give a loading program, -appropriate to the MIX simulator, mentioned before- serving as a demonstration of the MIXAL version. As a thorough test for the MIXAL version, we applied the assembler to the MIXAL assembler in MIXAL itself, using however another MIX simulator, written in ELAN (being the assembly language of the EL X8 computer). This simulator satisfies the same specifications as the ALGOL 60 version (except for the memory size, which has been increased from 4000 to 10000), but it is much faster and can be operated by hand. The results of this test, however, have not been reproduced in this report.

2. Definition of MIXAL.

2.1 The syntax of MIXAL.

<MIX character> ::= <any available character on the MIX computer>
<number> ::= <digit> | <number><digit>
<identifier> ::= <letter> | <identifier> <letter> | <identifier> <digit>
<symbol> ::= <identifier> | <digit><symbol>
<comment> ::= <empty> | <comment><MIX character>
<separator> ::= ; | <nlcr> | " <comment><nlcr>
<local forward> ::= <digit> F
<local label> ::= <digit> H
<local backward> ::= <digit> B
<local symbol> ::= <local forward> | <local label> | <local backward>
<undefined symbol> ::= <symbol>
<defined symbol> ::= <symbol>

```
<unary operator> ::= + | -
<binary operator> ::= + | - | * | / | // | :
<atomic expression> ::= <local backward> | <defined symbol> | <number> | *
<expression> ::= <atomic expression> | <unary operator><atomic expression> |
                 <expression><binary operator><atomic expression>
<field specification> ::= (<expression>) | <empty>
<word value> ::= <expression><field specification> | <word value>,
                 <expression><field specification>

<literal constant> ::= =<word value>=
<future reference> ::= <local forward> | <undefined symbol> |
                       <literal constant>

<address part> ::= <future reference> | <expression> | <empty>
<index part> ::= ,<expression> | <empty>
<address value> ::= <address part><index part>

<MIX instruction symbol> ::= ADD |...| SUB
<pseudo instruction symbol> ::= EQU | ORIG
<constant instruction symbol> ::= CON | ALF
<END instruction symbol> ::= END
<MIX machine instruction> ::= <MIX instruction symbol>
                              <address value><field specification>
<pseudo instruction> ::= <pseudo instruction symbol><word value>
<char> ::= <MIX character>
<character code> ::= .<char><char><char><char><char>
<MIX constant instruction> ::= CON <word value> |
                               ALF <character code>
<proper instruction> ::= <MIX machine instruction> |
                         <pseudo instruction> |
                         <MIX constant instruction>
<proper END instruction> ::= <END instruction symbol> <word value>
```

<label> ::= <undefined symbol> | <local label>
<MIXAL instruction> ::= <proper instruction> | < label > : < MIXAL instruction > | < separator > < MIXAL instruction >
< END instruction > ::= < proper END instruction > | < label > : < END instruction > | < separator > < END instruction >
<MIXAL body> ::= < empty > | <MIXAL instruction> <separator> <MIXAL body>
<MIXAL program> ::= <MIXAL body> <END instruction> <separator>

2.2 Remarks.

The language MIXAL as defined in the preceding section, differs from the language defined by Knuth on a few points.
As one of the purposes has been to design a version of MIXAL being layout independent as much as possible (i.e. the position of any syntactical unit on the card or papertape should be of no significance), some additional punctuations have been introduced to prevent ambiguities; the main case being the colon following each label, the other cases being the quotation mark preceding comment, and the point in character code preceding the five MIX characters.
Layout independence means that spaces freely may be inserted into the MIXAL text; however, there are some exceptions:
a) in character code, a space certainly is not layout, but stands for a space as MIX character,
b) in numbers and symbols spaces may also be inserted as layout, but not two (or more) spaces consecutively.
The latter exception has been made as (in some situations) two symbols may succeed each other (or a symbol and a number), without being separated by another syntactical unit. Thus we use two (or more) spaces as separator between symbols and/or numbers.
A second difference is concerned with labels. The layout independence has made it possible to allow more than one label, preceding a proper instruction, each of these labels being equivalent to the same value.
As these changes do not essentially affect MIXAL as defined by Knuth, it suffices to refer to [1] for the meaning and use of the assembly language MIXAL.

3. Reading, printing and recognition of basic syntactical units.

We distinguish between two levels of basic reading procedures, the first one being concerned with the reading and printing of single MIX characters, and the second one combining these MIX characters to basic syntactical units, which are analyzed more closely.

3.1 The low level reading procedure "read char".

This procedure reads the next character of the MIXAL program from the input medium. Both the procedure identifier of "read char" and the global variable "char" get as value the internal code of this MIX character.
Furthermore, the listing of the MIXAL program -each line generally consisting of the address and code, corresponding to the MIXAL instruction, the line number, and the text of the MIXAL instruction- is arranged by "read char", storing the characters read into a buffer (the array "line"), and printing out this buffer at the right moment, i.e. generally after another part of the assembler has produced the code corresponding to the instruction of this line, and has placed it also in the buffer. The boolean variables "text" and "code" make it possible to indicate this moment outside procedure "read char". They indicate whether some text has been completed for printing, and whether address and code have been produced. Thus, "read char" builds up a line in the buffer, organized with two pointers "pos" and "last pos",running from zero upwards, the first one pointing at the first free position of the buffer, the latter one pointing at the lastly occupied position.
It will be clear that reading a <nlcr> (new line carriage return, or new card) needs a special treatment, as, at the reading of this separator, the buffer generally may not be printed out (address and code may have to be determined yet). The <nlcr> itself is not stored into the buffer, as it is not a MIX character actually, but, as an indication that a <nlcr> just has been read, the pointer "pos" gets the value "-1", and the boolean variable "text" gets the value "true" (the latter being explained already).

When "read char" is called again, we may assume that address and code have been produced, if this were necessary. The value "true" for "text" (always) means that we may print this line before continuing, and the value "-1" for "pos" means that we have to start a new line at position zero, and that we also can store the new linenumber into the buffer.
During the initialisation of the assembler, when all MIX instruction symbols have to be read and stored into the symbol table, the high level reading procedures are used, which call "read char" in their turn. The text for the initialisation, however, is not read from an input medium, but is contained in the ALGOL 60 program by means of a string. Giving the boolean variable "from string" the value "true", the reading from this string is effectuated by means of the standard procedure "STRINGSYMBOL", called in "read from string", which delivers the representations of characters of a string [3].
remark:
When a dollar $ (the job separator, mentioned in the introduction) is read, obviously the END instruction of the MIXAL program is missing. Having reported this error, we jump out of all nested procedure calls, back to the first statement of the assembler, in order to treat the next MIXAL program. We mention this jump explicitly here, as this is the only goto statement, appearing in the whole program.

3.2 Some auxiliary procedures, concerned with the output buffer.

The printing of the characters, contained in the buffer, is done by procedure "print line". This procedure prints the contents of the buffer (using the pointer "last pos"), and fills the buffer with spaces again.
In the first place, procedure "read char" is responsible for the filling of the buffer with text.
Furthermore, a procedure "buffer" is available, used for storing numbers into the buffer (used for address, code and linenumber). Procedure "buffer" has four formal parameters: "p", "n", "m" and "value". The first parameter "p" indicates the position in the buffer at which the units of the number, contained in "value", are placed. There is a possibility to insert spaces between the digits of the number, by means of the parameters "n" and "m".

The digits of the number are divided into "abs(n)" groups of "abs(m)" digits; between each two successive groups one space is inserted. The sign of "n" indicates whether or not (positive or negative, respectively) a sign has to be placed preceding the number; the sign of "m" indicates whether leading zero's are printed, or replaced by spaces (positive or negative, respectively).
Finally, a procedure "buffer symb" may be called, storing symbols from the symbol table into the buffer. Its first parameter indicates the place of the symbol in the symbol table (see section 3.4.), the second parameter indicates the position in the buffer at which the first character of the symbol is placed, and the third parameter is an output parameter, delivering the first free place in the buffer, following the last character of the symbol. Procedure "buffer symb" actually has been used for special purposes concerned with the completion of the assembly process (see section 6).

3.3 The high level reading procedure "read synt unit".

Procedure "read synt unit" combines a number of MIX characters to one basic syntactical unit. Most of the basic syntactical units consist of one MIX character only (*,:,,,,.,(,),=,+,-,*,/ and :); they need no further analysis. The basic syntactical units, consisting of more than one character are:
a) number, b) symbol, c) separator and d) special divisor (//).

a) Reading a number, we don't know whether we are actually reading a number or a symbol after all, before we have read it completely. That is why we treat a number as if it were a symbol; this means that the digits temporarily are stored into the symbol table (see b.). In the boolean variable "contains letter" we keep up whether at least one letter has been read; if this is the case, we are concerned with a symbol (discussed below), otherwise we have read a number, of which the value is calculated by means of the digits stored in the symbol table. This value is assigned to the global variable "value of number", and the procedure identifier of "read synt unit", as well as the global variable "synt unit", get the constant value of "number" (value 102).

b) Reading a symbol, the letters and digits are stored meanwhile into the symbol table, in a way to be considered later on (3.4.). When all letters and digits have been read, procedure "look for symbol" looks up the symbol just read among the symbols which have been stored already in the symbol table. If this symbol is not found, a new information cell in the symbol table is created, containing the letters and digits of the symbol already. Thus, the new symbol has been joined to the symbol table by procedure "look for symbol". By means of the function designator "look for symbol", the place in the symbol table is passed on to the variable "last symbol". Besides, the variable "type" gets more information about the kind of symbol just read. After all, the procedure identifier of "read synt unit" and "synt unit" get the constant value of "symbol" (value 103).

c) The separator is read entirely, which means that comment is skipped, up to the first <nlcr>, by means of procedure "read char", and the procedure identifier of "read synt unit" (as well as "synt unit") gets the constant value of "separator" (value 101). Having read a semicolon as separator, the variable "text" is set on "true", as we expect that one MIXAL instruction has been completed, and the assembler will produce address and code, before the next character is read (at which moment the buffer will be printed out). The variable "pos" is left unchanged in this case; so succeeding characters will stay at the right position of the line, however, this will be a next line.

d) The reading of a special divisor (//) does not lead to any difficulty.

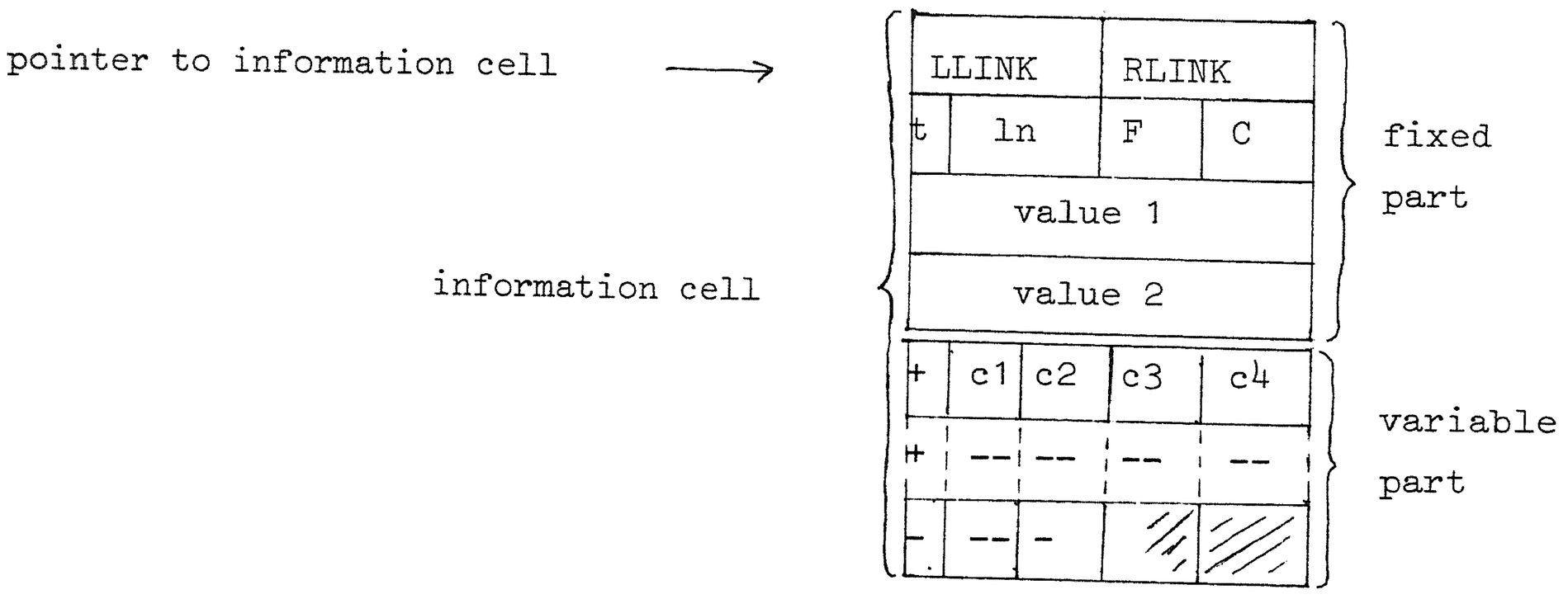
In most cases, the reading of one syntactical unit consists of the reading of one character, but, in some cases, we know that we have read one whole syntactical unit, just when we have read the first character of the following syntactical unit (this is the case when reading numbers, symbols and the division operater /).
Thus we arrive at the impossible situation that we sometimes have read already the first character of a following syntactical unit, and in other cases we have not. This can be resolved by consequently reading one

character too many in all cases, but this has an unpleasant effect on the synchronisation of possible error messages. We have chosen the following solution:
Instead of unnecessarily reading one character too many, we insert one extra character (a space as layout character), by simply giving the variable "char" the constant value of "space" (value 66) in those cases where we do not need to read the first character of the following syntactical unit. As, at the reading of the next syntactical unit, spaces are skipped beforehand by procedure "read synt unit", there will be no harmfull effect of this insertion.
The main program of the assembler, and the expression reading procedures, mainly read the MIXAL program by means of this procedure "read synt unit", which means that one need not worry, at that level, about the listing of the MIXAL program, the insertion of, and search for symbols in the symbol table, and the reading of numbers.

3.4 The organisation of the symbol table.

The symbol table is realised as an integer array "t", and consists of a set of information cells, each of which containing all information about one symbol. An information cell consists of a number of contiguous elements of "t", the address of the first element being used as the reference (pointer) of this information cell; the variable "last symbol" is used as a pointer to it. Local symbols,however, are not stored into this symbol table as other symbols are, but are detected explicitly in procedure "read synt unit", indicated by "type", while in these cases the variable "last symbol" gets as value the value of the digit, contained in the local symbol; this value can be seen as a pointer to an element of the arrays "h", "b", "f" and "defined local backward", into which the information concerning local symbols is stored.
The set of information cells may be divided into three contiguous parts, the first two being filled during initialisation with the MIX instruction symbols, and with the constant instruction symbols, the END instruction symbol, and the pseudo instruction symbols. Into the third part, all other symbols, introduced by the MIXAL programmer, are stored. The boundary between the first

two parts is given by means of the variable "last inst" being a pointer to the information cell of the lastly stored MIX instruction symbol.
For the symbols contained in the second part, explicit pointers are used, viz.: "alf", "con", "end", "equ" and "orig".
Concerning the internal construction of information cells, there exists no difference between the distinct parts, as each instruction symbol (e.g. ADD, CON or END) may be used by the MIXAL programmer as its own symbol. For such a symbol there exists only one information cell (in the first or second part of the symbol table) into which both the information with respect to the MIX instruction, and the information concerning the symbol, used by the programmer as its own, are stored.
An information cell consists of two parts, a fixed part and a variable part, the latter being used to contain the internal codes of the letters and digits, the particular symbol consists of.



The last element of the variable part is made negative, as opposed to the other elements, indicating the end of the alpha numerical information. This element may contain fewer than four characters (the others always contain four of them), in which case it is filled up by dummy characters.
Next, the fixed part of the information cell is dealt with closely. The fields F and C are concerned with MIX instruction symbols only (the first part of the symbol table). F contains the standard field, and C the code of

the corresponding MIX instruction. These fields are filled during initialisation. The field ln is used to store the linenumber of the line, on which the symbol has occurred as label (has got a value). The field t (type) indicates whether the symbol in question is defined or undefined (negative or positive value, respectively), i.e. whether or not the symbol has become equivalent to some value by means of an occurrence as label.

For the fields value 1 and value 2, we have to distinguish between undefined and defined symbols, the status being indicated by the value of the field t as described before. At information cells of undefined symbols, the field value 1 may be used to hold a pointer to the information cell of the symbol, which directly precedes the symbol in question, both symbols being used as label preceding the same MIXAL instruction. If this preceding label happens to be a local label, the pointer to it has been inverted and decreased by one, forcing a negative value, indicating in this case a reference out of the symbol table.

In case the symbol in question is not preceded by another label, the initial value of zero (stored in procedure "look for symbol") is replaced by the value 9999, indicating an empty pointer (N.B. this value neither may be a reference into the symbol table, nor a reference to a local symbol). The field value 2 may hold (if the symbol has occurred as future reference without being defined already) a pointer to that memory address of the MIX computer, at which the instruction has been placed, the address part of which contained the textually last future reference of the symbol in question. When the symbol has not (yet) occurred as future reference, value 2 holds the empty pointer (N.B. in this case the value 9999 may not be a valid MIX address).

For local symbols, the fields value 1 and value 2 also exist, but they are recorded in the respective elements of the arrays "h" and "f".

Concerning defined symbols, both pointers in the fields value 1 and value 2 have become irrelevant, but merely the value, the symbol recently has become equivalent to, is important. This value is split up into two parts, called the head and the tail, as the numerical capacity of one element of "t" would not be sufficient to store a word value.The head, stored into the field value 1, has the value "abs (word value) ÷10↑6", and the tail, stored into the

field value 2, has the value "sign (word value)*(abs(word value)-head *10↑6)".
The remarks concerning defined symbols do not pertain to defined local symbols, as the fields value 1 and value 2 may be used again at a next occurrence of the local symbol as label. So, a distinct field is used for storing the value of the local symbol. This field is recorded in the array "b" , and in an auxiliary array "defined local backward" we keep up in a boolean value whether the local symbol in question has occurred as local label at least once.

The search technique in the symbol table is a binary search technique in a tree-structured symbol table.To make a binary search applicable, we first have to define an order relation between symbols. The alphabetical ordering turned out to be an appropriate order relation for our purpose. As we also have to deal with digits, we have extended this ordering, defining the digits ordered naturally, preceding the letters (e.g. MIX, MIX 1, MIX 10, MIX 9, MIXAL, in this order).The tree structure of the symbol table is defined as follows:
1) Each node (actually an information cell) has at most two subtrees (called the left and right subtree, which are trees satisfying this definition), which are referred to by the pointers, contained in the fields LLINK and RLINK; if a subtree is not present, the corresponding field contains an empty reference (0).
2) All nodes in the left subtree of some node contain information cells of symbols, alphabetically preceding the symbol of this node, and all nodes of the right subtree contain information cells of symbols, succeeding the symbol of this node.
Due to this organisation, it is possible, after one comparison of an arbitrary symbol with the symbol contained in the root of the tree, to decide, in which of the two subtrees we have to search for this arbitrary symbol (if the symbols, just compared happen to be not the same). At each next comparison, we may exclude a whole class of symbols from the searching process. Trying to search for a symbol, which not yet has been inserted into the tree, we eventually come out at an empty pointer. At this point we know that an

information cell of this symbol does not occur in the symbol table, and we easily may join the new symbol to the symbol table, by replacing the empty pointer by a real pointer to the information cell (the variable part of which existing already) of the new symbol. Having filled in two empty pointers in the fields LLINK and RLINK of the newly created information cell, and initial values for the other fields (all within procedure "look for symbol"), we have extended the symbol table by one symbol, while the conditions, the tree structure satisfies, are still fulfilled. For, if we have to search for the recently inserted symbol, starting at the root of the tree, we do follow the same path through the tree as before, up to the node which recently has contained the empty pointer (nothing has been changed until so far). As this empty pointer has been replaced by the real pointer to the information cell of the symbol, we are searching for, we do not end in the blind alley now, but do find the appropriate information cell.

Finally, we direct our attention to the advantages gained by using the binary search. It may be clear that, offering new symbols to be inserted in alphabetical order, only branches of the tree to the right will originate, in which case we certainly do not gain any advantage over the linear search method. The order in which the MIXAL programmer introduces his symbols cannot be predicted, but, fortunately, we are able to introduce the MIX instruction symbols during initialisation, ordered in such a way, that the base of the tree will be filled homogeneously. The number of MIX instruction symbols amounts to 154 (including ALF,CON, etc.). Given a tree of seven levels,we can store into it $\sum_{i=0}^{6} 2\uparrow i = 127$ symbols. The remaining 27 symbols easily can be stored into the eighth level. This means that each of these instruction symbols can be found after eight comparisons at the most; the average number of comparisons amounting to:

$$1/154 * \left(\sum_{i=0}^{6} 2\uparrow i * (i+1) + 27 * 8\right) = 1001/154 = 6.5.$$

Thus we have made a considerable profit on the linear search method, which needs an avarage number of 154/2 = 77 comparisons for an instruction symbol.

3.5 The initialisation of the symbol table and the global variables.

During initialisation, characters are read from the string, contained in procedure "read from string", as discussed already in section 3.1. This string satisfies the following syntactical structure.

<MIX characters> ::= |*;:,.( ) = +-*/: azbfh
<standard field and code> ::= <number (2.1)>
<MIX instruction definition> ::= <empty>|
<MIX instruction symbol (2.1)><standard field and code>
<MIX instruction definitions> ::= <empty> | <MIX instruction definition >
< separator > < MIX instruction definitions >
<constant instruction symbols> ::= ALF CON
<pseudo instruction symbols> ::= EQU ORIG

<initial string> ::= <MIX characters>
<MIX instruction definitions>
(< constant instruction symbols >
<END instruction symbol (2.1)>
<pseudo instruction symbols>)

First, we read the MIX characters by means of procedure "read char", and assign their internal code to the variables:
"underlining", "bar", "space", "asterisk", etc. These variables, strictly speaking, are constants, introduced to make the program more readable. Having assigned to the other constants and variables their (initial) values, and having filled the outputbuffer "line" with spaces, we continue with filling the first part of the symbol table with the MIX instruction symbols. Having constructed explicitly the root of the tree, the symbol table consists of (viz. an information cell with initialized fields LLINK, RLINK, t, value 1 and value 2), the MIX instruction definitions are read by means of procedure "read synt unit ", the insertion to the symbol table being automatically now. Following each symbol, we read the standard field and code, the value of this number being: (standard field * 100 + code), and record both quantities in the fields F and C respectively of the information

cell in question. Having read all MIX instruction definitions, terminated by a paren, the first part of the symbol table is completed by assigning to "last inst" the pointer to the last information cell.
In addition, the next five symbols are read, and pointers to their information cells are recorded in the variables "alf", "con", "end", "equ" and "orig", respectively.
Finally, having checked whether we have reached the closing parenthesis, we assign to the boolean variable "from string" the value "false" (the effect of which being discussed already in section 3.1), and finish the initialisation process.

3.6 Errormessages.

When during the assembly process an error is detected, procedure "error" is called. The actual parameter of "error" is a string , describing (in the English language) the kind of error detected. In procedure "error", first all characters contained in the outputbuffer are printed out by means of a call of "print line". Next, at a new line, the actual parameter of error is printed out. The pointer "pos" (3.2), holding the position in the buffer, remains unchanged; thus, following characters (when reading on) will be printed out at their correct position, some lines lower however.
We already have mentioned the synchronisation of errormessages (3.3). Synchronisation means that the message is printed, textually immediately following those MIX characters, the errormessage is concerned with. In some cases however, we have read one character too many (see 3.3) when detecting an error, or even worse, a whole syntactical unit (including another following character). The latter phenomenon we meet at the reading of expressions (section 4). Reading too much means, in this context, that some text, belonging to following syntactical units, has been read already (while not being considered), merely to determine whether or not a whole unit (e.g. an expression) has been completed. In these cases, a possible errormessage will not be given in the right context.
Having detected and reported an error, the assembly process generally can be continued, and other errors may be detected and reported in the same way.

4. Expressions and word values.

4.1 Expressions.

As appears from the definitions in section 2.1, no priority rules exist for arithmetic operators in expressions; this means that operations are applied in order from left to right. Together with the fact that parenthesis are not allowed within expressions, this makes expressions very easy to evaluate; we do not need a stack for storing intermediate results, as these results immediately are used as left operand of an operator, the right operand being a single atomic expression, the value of which easily and directly is determined by means of procedure "read atomic expression". Of the atomic expressions, only symbols give rise to some further investigation. Symbols in expressions only may occur as defined symbol, or as defined local symbol, the values of which are denoted in the combined field value 1 and value 2, and in the array "b", respectively. The other possible occurrences of symbols are not admitted here, so they cause an errormessage, viz. "undefined symbol in expression", or "undefined local backward". How we can determine whether or not a symbol has been defined, has been explained already in section 3.4.
The other possibilities for atomic expressions are very easy to deal with.
When we are concerned with a number, we find its equivalent value in the global variable "value of number"; at this point it is important to observe that we have not read too much, as the following syntactical unit may be a number, the value of which would destroy the value of the global variable "value of number".
When we are concerned with an asterisk (called "self"), we only have to deliver the value of the location counter "lc" (see section 5).
For atomic expressions, we have to check, within procedure "read atomic expression", whether the equivalent value is not too large (exceeding the MIX word value). When executing arithmetical operations, by means of procedure "arithm", also capacity overflow may occur. In these cases an errormessage is given by the procedures concerned, viz. "overflow (atomic expression)", and "overflow (+)", "overflow (*)", etc., respectively, and, as resulting value, zero is delivered. The consequence of these tests,

within "read atomic expression" and "arithm", is that in procedure "read expression" only valid intermediate results occur, and no test on capacity overflow is necessary.
It is clear now that an expression can be evaluated in a straightforward manner by means of the procedures "read atomic expression" and "arithm". For reasons, later on to be explained, the first syntactical unit of an expression always has been read already when calling "read expression". Within this procedure we read the next syntactical unit and whenever this is an arithmetic operator, we go on reading an atomic expression, and executing the operation; otherwise, when no operator is read, we have done with the expression, and then, we did read one syntactical unit too much. This syntactical unit cannot be a syntactical unit belonging to the following MIXAL instruction, provided that two distinct MIXAL instructions are separated by a separator.

4.2 Word values.

A word value is used to denote the contents of one MIX word. According to the definition, we can describe a word value as:
E1 (F1), E2 (F2), ..., En (Fn), $n \geq 1$ and Ei and Fi being expressions for i=1, ..., n.
The value Ei is meant to be stored into the fields of a certain MIX word, specified by (Fi), for i=1, ..., n consecutively. The field specification (Fi) may be omitted, in which case a standard field (0:5) applies. The desired result, delivered by procedure "read w value", is the eventual value which would remain at the memory address CON when executing the next fragment of a MIXAL program:
STZ CON; LDA C1; STA CON(F1); ...; LDA Cn; STA CON(Fn), where C1, ..., Cn denote the addresses at which the values of E1, ..., En are stored respectively. Using literal constants, the fragment also can read:
STZ CON; LDA = E1 =; STA CON (F1); ...; LDA = En =; STA CON (Fn).
The Fi (i=1, ..., n), being the value of a field specification, must be of the form Li*10 + Ri (denoted in MIXAL as Li:Ri), where Li and Ri have to satisfy: $0 \leq Li \leq Ri \leq 5$.

Reading a word value, one syntactical unit too many is read (as it was the case in "read expression"), when testing whether or not we have done with the word value.
Also the word value (just as an expression) is followed by another syntactical unit which does not belong to the following MIXAL instruction. This means that we never incorrectly read (even when errors are detected) syntactical units belonging to following MIXAL instructions, at the level of the procedures "read expression" and "read w value".

5. The MIXAL instruction.

5.1 General outline.

An interesting feature of the MIXAL instruction is the fact that more than one label may precede the proper instruction. The first syntactical unit of an instruction has to be a symbol, being a label or an instruction symbol, the latter being the first syntactical unit of the proper instruction.
Finding out which of these two kinds of symbols we are concerned with, we have to consider the context in which the symbol occurs, as each instruction symbol also may be used as label. Checking whether the next syntactical unit is a colon, we are able to recognize a label as such.
However, if no label occurs, we thus have read the first syntactical unit belonging to the address value (if not empty), to the word value, or to succeeding character code.
When this last syntactical unit happens to be a symbol as well (this is very well possible), the global variables "last symbol" and "type", originally referring to the instruction symbol, have been overwritten; unless we have saved the original value of "last symbol" by means of a local variable of procedure "read mixal program" (actually called "inst"), we would have lost all information about the instruction symbol.
Another consequence of this looking for a colon is that the first syntactical unit of an expression, possibly succeeding the instruction symbol, has been read already, when we eventually come to the processing of the expression. (see section 4.1).

Now, the question arises, whether it is possible to give a symbol a value (make the symbol a defined symbol), immediately following the recognition of the symbol occurring as label. Unfortunately this value depends on the kind of proper instruction following. If this proper instruction happens to be the EQU pseudo instruction,the equivalent value for the label(s) is (are) denoted by the word value which follows EQU, otherwise the label(s) will become equivalent to the value which the location counter "lc" has at the moment

Consequently, we first have to read all labels preceding the proper instruction, including the instruction symbol (and the next syntactical unit to recognize the instruction symbol as such), before determining the equivalent value for these labels. Meanwhile, we retain all labels by linking them; each label is made to refer to its predecessor, and the first one refers to nothing, i.e. contains an empty pointer.

To hold this reference, the field value 1 of the concerning information cells in the symbol table is used. We already have seen that, for local labels, an element of the array "h" corresponds to the field value 1, and that a negative reference stands for a reference to "h" and a positive reference stands for a reference into the symbol table; this holds both for references contained in the symbol table and in the array "h" as well. Thus we have chained all labels, and a pointer to the last occurring - label is kept in the variable "last label". However, we carefully have to avoid circular lists, originating when one symbol occurs more than once as label preceding the same instruction.

Symbols, which have been inserted in a chain as described above, can be recognized by the following two properties for their information cells:

1) The field t is positive (irrelevant in the case of local labels), which means that the symbol has not yet become a defined one;

2) The field value 1 has a value, unequal to the initial value zero, which means that the symbol has occurred as label, as value 1 contains a reference (possibly an empty reference).

Subsequently, we treat the proper instruction entirely, and finally we update the chained list of labels by following the references, while storing into the information cells the equivalent value of the symbols, and making

the fields t negative, to indicate the defined status of the symbols (by means of procedure "update").
If one of the chained symbols occurs in some expression, contained in the proper instruction, it has not yet the defined status, and an error message follows. In the case of the EQU pseudo instruction, this error message is appropriate , as the value of a symbol may not depend on itself. In the other cases, the value of such a symbol could have been known. For congruence reasons with the previous case, we decided however, to maintain the general scheme of treatment; the advantage being that one rule can be given, stating, at which moment symbols get a value, viz. after the entire instruction has been dealt with.
Some examples may serve as illustration:

1) F: EQU F+10; In this case, the value of F would depend on itself; we rightly get an error message
2) F: ADD F+10; An error message is given, as the value of F has not yet been assigned to F, when evaluating the expression F+10, although it is known.
3) F: ADD *+10; The alternative construction of 2), using an asterisk which has the same value as F gets afterwards.

At the moment the equivalent value for the labels is known, we keep this value in the variable "equiv". When ultimately the procedure "update" is called, we pass on the value of "equiv" as actual parameter of "update".

5.2 Address value and field specification.

The following two facts, mentioned before, are recalled. Firstly, the pointer to the information cell, belonging to the instruction symbol, is hold in the variable "instr", local to procedure "read mixal program". This pointer is passed on to procedure "read mix machine inst" as actual parameter. The first action of procedure "read mix machine inst" will be then to fetch the standard field and code of the instruction from the fields F and C of the information cell concerned, and to assign these values to the variables "f part" and "c part",respectively.

Secondly, the first syntactical unit of address value and field specification has been read already at this moment. We start the actual processing by testing whether the address part contains a future reference, analyzing the present syntactical unit. When a future reference does occur, the variable "a part" will get the value of a reference, being discussed in the following section; otherwise, if the address part doesn't happen to be empty -i.e. if the present syntactical unit is not a comma (indicating an index part), nor a left parenthesis (indicating a field specification), nor a separator (indicating the absence of all quantities)-, we may expect an expression, treated by means of procedure "read expression", the value being assigned to the variable "a part". Of the quantities indexpart and field specification, easily can be determined now whether or not they are empty.
An empty index part gives rise to the assignment of the value zero to the variable "i part", and an empty field specification causes the standard field remaining in the variable "f part".
After all, the word value which has to be stored at the address indicated by "lc", can be assembled easily by means of the variables "a part", "i part", "f part" and "c part", and actually it is stored into the MIX memory by means of procedure "produce" (which also produces the output of address and code by means of the outputbuffer).

5.3 Future references.

Syntactically, a future reference may occur only as address part of a MIX machine instruction. The value for this future reference will become known in the future, when treating following instructions. The intention is, that, for the instruction in which the future reference occurs, code is produced as usual, except for the address part, which will be filled in later on: at the moment that the corresponding (local) label gets a value by means of procedure "update"; for literal constants it means at the end of the program (see section 6).
It is evident, that a symbol may occur as future reference more than once. This means that we have to update a number of MIX addresses, when the symbol in question gets its value. As for all these addresses it holds, that the

address part of the code has not yet been filled in, we can use these two bytes for linking the MIX addresses; thus we chain all addresses corresponding to a future reference of the same symbol. The beginning of this chain is kept in the symbol table, in the field value 2 of the information cell (for local forwards corresponding with an element of array "f"), and the chain is ended by means of an empty reference.

In the same way, a chain of literal constants is created, the access to it being kept in the variable "last constant". The difference between the chain of literal constants, and a chain of symbolic future references, is, that each MIX word in the first chain ultimately will obtain an unique address part (viz. the address at which the constant word value is stored by the assembler), while each MIX word of one other chain gets the same value in its address part. Chaining the MIX addresses, at which literal constants occur, is not sufficient; also the corresponding word values have to be retained somewhere until the end of the assembly process, when they are treated. We use the end of the array "t" for this purpose. For each word value to be stored, two consecutive elements of array "t" are used, identically to the way of coding a word value in the fields value 1 and value 2.

Describing the chaining of labels, we have seen that overwriting of references carefully has been avoided. Also in the case of future references, overwriting may take place, possibly leading to disastrous effects (such as an infinite loop of the assembler).

As we are able indeed, by means of the ORIG pseudo instruction, to start writing at each address we want, we necessarily have to protect ourselves against overwriting any address which contains a reference. As these addresses are hardly distinguishable from other ones - we would have to follow all references, starting at many places in the symbol table, and at the variable "last constant", to determine whether the address in question forms part of one of the chains -we decided to forbid each attempt to write on an address, which does not contain the value zero, the initial value of all addresses (see introduction).

We are aware of the fact that we unfortunately also forbid innocent

overwriting, but that is the price we have to pay, when we decide to desist from the very expensive algorithm, described above.
When the assembler detects an address, containing a value, unequal to the initial value zero, the errormessage "address already used" is given. Meeting an address, which does contain the value zero, necessarily one of the four following situations occurs:

1) the address never has been used,
2) the address has been filled by means of CON 0
3) the address has been filled by means of ALF .00000
4) the address has been filled by means of NOP 0

In the first three cases, the address does not contain a future reference, so there exist no danger (although it is a little inconsistent to do permit the overwriting in some cases). The address part of the instruction NOP may contain a future reference, previously occurred at address zero (in the address part of the NOP instruction, a reference to address zero is filled in). However, this may occur only once in a MIXAL program; thus, when we retain this address in a special variable "dangerous", we protect ourselves against overwriting of references, when testing whether a certain address contains the value zero, and this address is not the dangerous one.

6. The completion of the assembly process.

Having detected the END instruction, and having read the following word value, and assigned its value to the variable "start address", we have to deal with those symbolic future references, which have not yet been treated, and with the literal constants, by means of procedure "insert constants". At first, all information cells of local symbols and ordinary symbols are examined, the latter ones by means of procedure "traverse tree", which recursively walks through the tree, investigating the symbols in alphabetical order, and which calls at each node of the tree the procedure "define" as actual parameter for"investigate" (local symbols have been treated explicitly, but analoguously to "define"). Only those (local) symbols, which have been used as future reference, without having occurred as (local)

label, need some special treatment. The information cells of these symbols, satisfy the following properties:

1) The field t is positive (irrelevant in the case of local labels), indicating that the symbol has not yet become a defined one;

2) The field value 2 contains a reference, unequal to the empty pointer, indicating that the symbol has been used as future reference;

3) The field value 1 still contains the initial value zero, indicating that the symbol has not occurred as label preceding the END instruction (this chain of symbols has not yet been updated).

For each of these symbols, consecutively one MIX word is reserved, by means of procedure "prcduce" (overwriting of addresses, which have been used already, still is impossible), and the addresses of these words are filled in at the locations at which the future references occurred, by means of procedure "update" (also procedure "produce" is called, for updating an address; now the overwriting test mechanism is eliminated, by testing whether the location counter does not point at this address).

To inform the MIXAL programmer, for which symbols a new MIX word is reserved, output is delivered, immediately following the END instruction, consisting of the symbol in question, followed by a colon and the pseudo instruction CON 0 illustrating what happened.

Next, a number of words have to be reserved for the literal constants. Walking through the chain of occurrences of literal constants, we reserve for each one a consecutive MIX word and put the ccrrresponding word value in it by means of procedure "produce", and we update the address part with the actual address of the corresponding word value. Having dealt with the insertions, the possible chain of labels, preceding the END instruction is updated.

Here we remark that property 3) for symbols to be inserted, is necessary, as we don't want that the labels preceding the END instruction, get their value by means of "insert constants", but by means of the call of "update", at the end of each instruction.

After all, for the convenience of the MIXAL programmer, a dump of the symbol table is given by means of procedure "dump symbol tabel". Thus, an alphabetical list of symbols used is printed out, each symbol preceded

by the value it has become equivalent to, and the linenumber of the line at which the symbol occurred as label. This information easily is fetched from the symbol table, which is walked through, in alphabetical order of symbols, by means of procedure "traverse tree" with the actual parameter "dump" for "investigate".

7. The ALGOL 60 procedure "mixal assembler".

Next follows the complete text of the MIXAL assembler in ALGOL 60 without further comment.

```
boolean procedure mixal assembler(m, start0); array m; integer start0;
begin

boolean erroneous, from string, future, code, text;
integer underlining, bar, space, asterisk, semicolon, colon, comma,
  point, paren, thesis, equal sign, plus, minus, times, div1, dec, quote,
  letter a, letter z, letter b, letter f, letter h, nlcr, tab,
  div2, symbol, number, sep,
  undefined, defined, local label, local backward, local forward,
  equ, orig, con, alf, end, last inst,
  last symbol, type, char, last constant, synt unit,
  read ptr, pos, last pos, line number, table ptr1,
  lit cons ptr,  u1, b1, aux;
real start,lc, dangerous, value of number, time 0;
boolean array defined local backward[0 : 9];
integer array line[-40 : 99], h,f[0 : 9], t[1 : 2000];
real array b[0 : 9];


integer procedure read char;
if from string
then begin char:= read from string;
           read char:=char:= if char = 127 then 69
                             else if char > 35 then char - 27
                                  else char
     end
else begin integer s;
          if text ∨ code then print line;
          if pos = -1
          then begin if line number < 6708
                     then line number:= line number + 1;
                     buffer(-4,-1,-4, line number); pos:=0
               end;
          s:= resym; for s:=s  while s = 134 do s:= resym;
          if s = 135 then s:= 119;
          char:= if s = 127 then 69
                 else if s > 35 then s - 27 else s;
          if char = underlining then u1:=1000
          else if char = bar then b1:=1000
               else begin if pos > 99 ∧ char ≠ nlcr
                          then begin print  line; pos:=0 end;
                          if char = tab ∨ char = nlcr
                          then s:=ext(space);
                          s:=2 × b1 + u1 + s; u1:=b1:=0;
                          if s = 2028 ∨ s = 2055
                          then begin errorm(≠end of input≠);
                                     for aux:=0 step 1 until 3999 do
                                     m[aux]:=0; new page; goto begin
                               end;
                          line[pos]:=s; last pos:=pos;
                          if char = tab then pos:=(pos+9):8×8
                          else if char = nlcr
                               then begin text:=true; pos:= -1 end
                               else pos:=pos + 1
end;
```

```
        if char > 99 then errorm(‹unknown character›);
        read char:=char
    end read char;


real procedure read 5 char;
if synt unit = point
then begin real value; integer i, c; value:=0;
        for i:=1 step 1 until 5 do
        begin c:=if char = nlcr then space else read char;
            value:=value × 100 + (if c = nlcr then space else c)
        end;
        if char ≠ nlcr then char:= space;
        read synt unit; read 5 char:= value
    end
else begin errorm(‹. required›); read 5 char:=0 end;


procedure print line;
begin integer i, s; boolean u, b;
    for i:= -40 step 1 until last pos do
    begin s:=line[i]; u:=b:=false;
        if s ≥ 2000
        then begin b:=true; s:=s - 2000 end;
        if s > 1000
        then begin u:=true; s:=s - 1000 end;
        if u then prsym(ext(underlining));
        if b then prsym(ext(bar));
        prsym(s); line[i]:=ext(space)
    end;
    prsym(ext(nlcr)); code:=text:=false; last pos:=0
end print line;


procedure errorm(s); string s;
begin print line; printtext(s); prsym(ext(nlcr)); erroneous:=true end;


integer procedure read synt unit;
begin for char:=char while char = space ∨ char = tab do read char;
    if char < letter z
    then begin integer p, n, q1, q, i, value;
            boolean contains letter,compl;
            contains letter:= letter a < char ∧ char < letter z;
p:=table ptr1 + 4; q1:=char; q:=read char; n:=1;
for value:=q1 + 1, value while ¬ compl do
begin if char = space then read char;
    if q = space then q:=char;
    compl:= letter z < char;
    if compl then value:= - value × 100↑(4-n);
    if compl ∨ n = 4
    then begin t[p]:=value; value:=0;
            p:=p + 1; test on overflow(p)
        end;
    if ¬ compl
    then begin n:=if n < 4 then n + 1 else 1;
            value:=value × 100 + char + 1;
```

```
                    if letter a ≤ char ∧ char ≤ letter z
                    then contains letter:= true;
                    read char
                end
        end;
        if p = table ptr1 + 5 ∧ n = 2 ∧ q1 < letter a ∧
           (q = letter b ∨ q = letter f ∨ q = letter h)
        then begin synt unit:=symbol;
                type:=if q = letter b then local backward
                      else if q = letter f then local forward
                           else local label;
                last symbol:=q1
             end
        else if contains letter
             then begin synt unit:=symbol;
                        last symbol:=look for symbol(p);
                        type:=if t[last symbol+1] < 0
                              then defined else undefined
                  end
             else begin synt unit:=number; value of number:=0;
                        for p:=table ptr1 + 4, p+1 while t[p-1] > 0 do
                        begin value:=abs(t[p]);
                            for i:=3, i-1 while value ≠ 0 do
                            begin aux:=100↑i; q:=value : aux;
                                value:=value - q × aux;
                                value of number:=value of number ×10+q-1
                            end
                        end
                  end
             end
        else if char = quote ∨ char = nlcr
             then begin synt unit:=sep;
                        for char:=char while char ≠ nlcr do read char;
                        char:=space
                  end
             else if char = semicolon
                  then begin synt unit:=sep; char:=space; text:=true end
                  else if char ≠ div1
                       then begin synt unit:=char; char:=space end
                       else if read char = div1
                            then begin synt unit:=div2; char:=space end
                            else synt unit:=div1;
read synt unit:=synt unit
end read synt unit;


integer procedure look for symbol(ptr); value ptr; integer ptr;
begin integer p, next, i, wp, w, comp, llink, rlink;
    for p:=1, next while p ≠ 0 do
    begin for i:=4, i+1 while comp = 0 ∧ wp > 0 ∧ w > 0 do
        begin wp:=t[p + i]; w:=t[table ptr1 + i];
            comp:=abs(w) - abs(wp)
        end;
        if comp = 0 then comp:=if wp>0 then -1 else
                               if  w>0 then 1 else 0;
        if comp = 0
```

```
then begin look for symbol:=p; next:=0;
          if table ptr1 = 1 then table ptr1:=ptr
     end
else begin aux:=t[p]; llink:=aux : 10↑4; rlink:= aux - llink × 10↑4;
          next:=if comp < 0 then llink else rlink;
          if next = 0
          then begin look for symbol:=table ptr1;
                     if comp < 0
                     then llink:=table ptr1
                     else rlink:=table ptr1;
                     t[p]:=llink × 10↑4 + rlink;
                     t[table ptr1    ]:=0;
                     t[table ptr1 + 1]:=10↑4;
                     t[table ptr1 + 2]:=0;
                     t[table ptr1 + 3]:=9999;
                     table ptr1:=ptr
                end
          end
     end
end look for symbol;


procedure test on overflow(ptr); value ptr; integer ptr;
if ptr > lit cons ptr
then begin errorm(‹overflow symbol table›);
          for char:=char while true do read char
end test on overflow;


real procedure read atomic expression;
begin real at expr; read atomic expression:=at expr:=0;
     if synt unit = symbol
     then begin if type = defined
                then begin aux:=t[last symbol + 3];
                           at expr:=sgn(aux)×(t[last symbol+2]×10↑6+abs(aux))
                     end
                else if type = local backward
                     then begin if defined local backward[last symbol]
                                then at expr:=b[last symbol]
                                else errorm(‹undefined local backward›)
                          end
                     else if type = local forward
                          then errorm(‹local forward in expression›)
                          else if type = local label
                               then errorm(‹local label in expression›)
                               else errorm(‹undefined symbol in expression›);
                read synt unit
          end
     else if synt unit = number
          then begin at expr:=value of number; read synt unit end
          else if synt unit = asterisk
               then begin at expr:=lc; read synt unit end
               else errorm(‹atomic expression required›);
     if abs(at expr) < 10↑10
     then read atomic expression:=at expr
     else errorm(‹overflow(atomic expression)›)
end read atomic expression;
```

```
real procedure read expression;
begin real expr; integer operator; read expression:=0;
    expr:=if ¬ (synt unit = plus ∨ synt unit = minus)
          then read atomic expression else 0;
    for synt unit:=synt unit
    while synt unit = plus ∨ synt unit = minus ∨ synt unit = times ∨
            synt unit = div1 ∨ synt unit = div2 ∨ synt unit = dec
    do begin operator:=synt unit; read synt unit;
          expr:=arithm(expr, operator, read atomic expression)
       end;
    read expression:=expr
end read expression;


real procedure arithm(a, op, b); value a,op,b; real a,b; integer op;
if op = plus
then begin real value; value:=a + b; arithm:=0;
          if abs(value) < 10↑10 then arithm:=value
          else errorm({overflow (+)})
     end
else if op = minus
    then begin real value; value:=a - b; arithm:=0;
               if abs(value) < 10↑10 then arithm:=value
               else errorm({overflow (-)})
          end
    else if op = times
     then begin integer s,h1,t1,h2,t2,c; real r0,r1; arithm:=0;
               s:=sgn(a × b); a:=abs(a); b:=abs(b);
               h1:=entier(a/10↑6); t1:=a - h1 × 10↑6;
               h2:=entier(b/10↑6); t2:=b - h2 × 10↑6;
               r0:=t1 × t2; c:=entier(r0/10↑6); r0:=r0 - c × 10↑6;
               r1:=t1×h2 + h1×t2 + c; c:=entier(r1/10↑4); r1:=r1 - c×10↑4;
               if h1 × h2 + c = 0 then arithm:=s × (r1 × 10↑6 + r0)
               else errorm({overflow (×)})
     end
     else if op = div1
         then begin  arithm:=0; if b ≠ 0
                                then arithm:=sgn(a × b) × entier(abs(a/b))
                                else errorm({overflow (/)})
              end
         else if op = div2
             then begin arithm:=0; if b ≠ 0 ∧ abs(a) < abs(b)
                                   then arithm:=sgn(a×b)×entier(abs(a/b)×10↑10)
                                   else errorm({overflow (//)})
                  end
             else begin real value; value:=a × 10 + b; arithm:=0;
                       if abs(value) < 10↑10 then arithm:=value
                       else errorm({overflow (:)})
                  end arithm;


real procedure read w value;
begin real con, a, field, aux1, aux2, aux3;
    integer l,r,s; boolean comma read;
    con:=0;
    for synt unit:=synt unit, synt unit while comma read do
```

```
begin a:=read expression;
    if synt unit = paren
    then begin read synt unit; field:= read expression;
            if field = 5 then con:= a
            else if field = 0 then con:= sgn(a) × abs(con)
            else if field = 15 then con:= sgn(con) × abs(a)
            else begin l:=entier(field/10); r:=field - l × 10;
                if 0 ≤ l ∧ l < r ∧ r ≤ 5
                then begin if l ≠ 0 then s:=sgn(con)
                        else begin s:=sgn(a); l:=1 end;
                        a:=abs(a); con:=abs(con);
                        aux1:=100↑(6-l); aux2:=100↑(5-r);
                        aux3:=aux1/aux2;
                        con:=s × (entier(con/aux1) × aux1 +
                            (a - entier(a/aux3)×aux3) × aux2 +
                            con - entier(con/aux2) × aux2)
                    end
                else errorm(‹illegal field›)
            end;
        if synt unit = thesis then read synt unit
        else errorm(‹) required›)
    end
    else con:=a;
    if synt unit = comma
    then begin comma read:=true; read synt unit end
    else comma read:=false
end;
    read w value:=con
end read w value;


procedure read mixal program;
begin integer inst, last label; real equiv; last label:= 9999;
    for inst:= 0, inst while inst ≠ end do
    for synt unit:=colon, synt unit while synt unit = colon do
    if read synt unit ≠ symbol
    then begin if synt unit ≠ sep then errorm(‹symbol required›);
            synt unit:= colon
        end
    else begin inst:=last symbol;
            if read synt unit = colon
            then begin if type = undefined
                    then begin if t[last symbol+2] = 0
                            then begin t[last symbol+2]:=last label;
                                last label:=last symbol
                            end
                        else errorm(‹label used twice›)
                end
            else if type = local label
                then begin if h[last symbol] = 0
                        then begin h[last symbol]:=last label;
                                last label:=-last symbol-1
                            end
                        else errorm(‹local label used twice›)
                    end
                else errorm(‹illegal label›)
end
```

```
else begin equiv:=lc;
          if inst < last inst
          then produce(lc,read mix machine inst(inst))
          else if inst = equ
               then equiv:=read w value
               else if inst = orig
                    then lc:=read w value
                    else if inst = con
                         then produce(lc,read w value)
                         else if inst = alf
                              then produce(lc,read 5 char)
                              else if inst = end
                                   then begin start:=read w value;
                                              insert constants;
                                              equiv:=lc
                                        end
                                   else
                                   errorm(‹unknown instruction›);
               if synt unit ≠ sep
               then errorm(‹separator required›);
               for synt unit:=synt unit while synt unit ≠ sep do
               read synt unit;
               update(last label, equiv); last label:=9999
          end
     end
end read mixal program;
```

```
boolean procedure possible(address); value address; real address;
possible:=if 0 < address ∧ address < 3999
          then m[address] = 0 ∧ address ≠ dangerous
          else false;
```

```
real procedure read mix machine inst(instruction); value instruction;
integer instruction;
begin real a part,i part,f part,c part;
    a part:=i part:=0; aux:=abs(t[instruction + 1]); f part:=aux : 10↑2;
    c part:=aux - f part × 10↑2; f part:=f part - f part : 10↑2 × 10↑2;
    if synt unit = symbol
    then begin if type = undefined
               then begin a part:=t[last symbol + 3]; future:=true;
                          if possible(lc) then t[last symbol + 3]:=lc;
                          read synt unit
                    end
               else if type = local forward
                    then begin a part:=f[last symbol]; future:=true;
                               if possible(lc) then f[last symbol]:=lc;
                               read synt unit
                         end
                    else a part:=read expression
         end
    else if synt unit = equal sign
         then begin real value;
                    a part:=last constant; future:=true;
                    if possible(lc) then last constant:=lc;
                    read synt unit; value:=read w value;
```

```
                lit cons ptr:=lit cons ptr - 2;
                test on overflow(table ptr1);
                t[lit cons ptr + 1]:=aux:=entier(abs(value)/10^6);
                t[lit cons ptr + 2]:=sgn(value)×(abs(value) - aux×10^6);
                if synt unit = equal sign then read synt unit
                else errorm(‹= required›)
            end
        else if ¬ (synt unit=comma ∨ synt unit=paren ∨ synt unit=sep)
                then a part:=read expression;
            if synt unit = comma
            then begin read synt unit; i part:=read expression end;
            if synt unit = paren
            then begin read synt unit; f part:=read expression;
                    if synt unit = thesis then read synt unit
                    else errorm(‹) required›)
                end;
            if abs(a part) > 9999
            then begin errorm(‹abs(a part) > 9999›); a part:=0 end;
            if i part < 0 ∨ i part > 99
            then begin errorm(‹i part < 0 ∨ i part > 99›);  i part:=0 end;
            if f part < 0 ∨ f part > 99
            then begin errorm(‹f part < 0 ∨ f part > 99›);  f part:=0 end;
            read mix machine inst:=sgn(a part) ×
                    (abs(a part) × 10^6 + i part × 10^4 + f part × 10^2 + c part)
end read mix machine inst;
```

```
procedure produce(address,value);value address,value;real address,value;
begin if code then print line;
    if ¬ (0 ≤ address ∧ address ≤ 3999)
    then begin errorm(‹wrong address›);buffer(-28,1,-10,address) end
    else begin if address = lc ∧ (address = dangerous ∨ m[address] ≠ 0)
            then errorm(‹address already used›)
            else begin if future ∧ value = 0 then dangerous:=address;
                    m[address]:=value;
                    buffer(-14, if future then -3 else 5, 2, value)
                end;
            buffer(-35, -2, 2, address);
            if address = lc then lc:=lc+1 else line[-40]:=ext(asterisk)
        end;
        future:=false; code:= true
end produce;
```

```
procedure update(label,equiv);value label,equiv;integer label;real equiv;
for label:=label while label ≠ 9999 do
begin integer symb, link, address;
    if label > 0
    then begin symb:=label; label:=t[symb + 2];
            link:=t[symb + 3]; aux:=t[symb + 1];
            t[symb + 1]:= -(aux + (line number+1 - aux:10^4)×10^4);
            t[symb + 2]:=aux:=entier(abs(equiv)/10^6);
            t[symb + 3]:=sgn(equiv) × (abs(equiv) - aux × 10^6)
        end
    else begin symb:=-label-1; label:=h[symb]; h[symb]:=0;
            link:=f[symb]; f[symb]:=9999; b[symb]:=equiv;
```

```
            defined local backward[symb]:=true
        end;
    for link:=link while link ≠ 9999 do
    if abs(equiv) > 9999
    then begin link:=9999; errorm(‹abs(future ref) > 9999›) end
    else begin address:=link; link:=entier(m[address]/10^6);
            produce(address,sgn(equiv)×(m[address]+(abs(equiv)-link)×10^6))
        end
    end update;


    procedure buffer(p,n,m,value);value p,n,m,value;integer p,n,m;real value;
    begin integer i, j, q; real aux1;
        if n > 0
        then line[p - (abs(m)+1)×n - 1]:=ext(if value<0 then minus else plus);
        aux1:=10↑abs(n×m); value:=abs(value) - entier(abs(value)/aux1) × aux1;
        for i:=abs(n)-1 step -1 until 0 do
        for j:=abs(m)-1 step -1 until 0 do
        begin aux1:=10↑(m × i + j); q:=entier(value/aux1);
            value:=value - q × aux1;
            if q > 0 ∨ (i = 0 ∧ j = 0) then m:=abs(m);
            if m > 0 then line[p - ((m+1) × i + j)]:=ext(q)
        end
    end buffer;


    procedure traverse tree(investigate, symbol); value symbol;
    integer symbol; procedure investigate;
    if symbol ≠ 0
    then begin integer llink, rlink; aux:=t[symbol];
            llink:=aux : 10^4; rlink:=aux - llink × 10^4;
            traverse tree(investigate, llink);
            investigate(symbol);
            traverse tree(investigate, rlink)
        end traverse tree;


    procedure buffer symb(s, lb, ub); value s, lb; integer s, lb, ub;
    if s < 0
    then begin line[lb]:= -s-1; line[lb+1]:=ext(letter h); ub:=lb+2 end
    else begin integer p, q, value, i; ub:=lb;
            for p:=s + 4, p + 1 while t[p - 1] > 0 do
            begin value:=abs(t[p]);
                for i:=3, i - 1 while value ≠ 0 do
                begin aux:=100↑i; q:=value : aux;
                    value:=value - q × aux;
                    if ub = 99
                    then begin line[99]:=2000 + ext(space);
                            last pos:=99; print line; ub:=0
                        end;
                    line[ub]:=q - 1; ub:=ub + 1
                end
            end
        end
    end buffer symb;
```

```
procedure buffer constant(s); value s; integer s;
begin integer ptr;
    if code ∨ text then print line;
    buffer symb(s,0,ptr); line[ptr]:=ext(colon);
    if ptr > 15 then begin last pos:=ptr; print line end;
    buffer symb(con,16,ptr); last pos:=ptr + 5; line[last pos]:=ext(0)
end buffer constant;
```

```
procedure insert constants;
begin procedure define(symbol); value symbol; integer symbol;
      if t[symbol+1] > 0 ∧ t[symbol+2] = 0 ∧ t[symbol+3] ≠ 9999
      then begin integer equiv; equiv:=lc;
                 buffer constant(symbol); produce(lc, 0);
                 t[symbol+2]:=9999; update(symbol, equiv)
           end define;
    integer symb, address, ln;
    code:=true; ln:=line number; line number:=6709;
    for symb:= -1 step -1 until -10 do
   if f[-symb-1] ≠ 9999
    then begin integer equiv; equiv:=lc;
               buffer constant(symb); produce(lc, 0);
               h[-symb-1]:=9999; update(symb, equiv)
         end;
    traverse tree(define, 1);
    line number:=ln;
    for address:=last constant while address ≠ 9999 do
    begin real value; last constant:=entier(m[address]/10^6);
        value:=m[address] + (lc - last constant) × 10^6;
        aux:=t[lit cons ptr + 2];
        produce(lc, sgn(aux)×(t[lit cons ptr + 1] × 10^6 + abs(aux)));
        lit cons ptr:=lit cons ptr + 2;
        produce(address, value)
    end
end insert constants;
```

```
procedure dump(symbol); value symbol; integer symbol;
if t[symbol+1] < 0
then begin integer ptr; real value; aux:=t[symbol+3];
          value:=sgn(aux)×(t[symbol+2]×10^6+abs(aux));
          line number:=abs(t[symbol+1]) : 10^4 - 1;
          buffer(-14, 5, 2, value);
          if line number > 6708 then line[-3]:= ext(plus)
          else buffer(-4, -1, -4, line number);
          buffer symb(symbol, 0, ptr); last pos:= ptr - 1;
          print line
     end dump;
```

```
procedure dump symbol table;
begin new page;
    printtext(‡value and line number of defined symbols:‡);
    prsym(ext(nlcr)); prsym(ext(nlcr));
    traverse tree(dump, 1); new page
end dump symbol table;
```

```
procedure initialize;
begin integer i;
    procedure rc(s); integer s; s:=read char;
    erroneous:=false; from string:=true; read ptr:=0;
    rc(underlining); rc(bar); rc(space); rc(asterisk); rc(semicolon);
    rc(colon); rc(comma); rc(point); rc(paren); rc(thesis); rc(equal sign);
    rc(plus); rc(minus); rc(times); rc(div1); rc(dec); rc(letter a);
    rc(letter z); rc(letter b); rc(letter f); rc(letter h); read char;
    quote:=94; nlcr:=92; tab:=91;
    sep:=101; number:=102; symbol:=103; div2:=104;
    undefined:=0; defined:=1; local label:=2; local backward:=3;
    local forward:=4;
    table ptr1:=1; lit cons ptr:=2000; t[1]:=t[3]:=0; t[4]:=9999;
    for i:= -40 step 1 until 99 do line[i]:=ext(space);
    for i:=0 step 1 until 9 do
    begin h[i]:=0; f[i]:=9999; defined local backward[i]:=false end;
    last constant:=9999; lc:=0; dangerous:= -1;
    future:=code:=text:=false;
    pos:=last pos:=line number:=u1:=b1:=0;
    for synt unit:=sep, synt unit while synt unit = sep do
    if read synt unit = symbol
    then begin read synt unit;
            t[last symbol + 1]:=10 4 + read atomic expression
        end
        else if synt unit ≠ sep ∧ synt unit ≠ paren then errorm(⟨er 01⟩);
        if synt unit = paren
        then begin last inst:=last symbol;
                read synt unit; alf:=last symbol;
               read synt unit; con:=last symbol;
                read synt unit; end:=last symbol;
                read synt unit; equ:=last symbol;
                read synt unit;orig:=last symbol;
                if read synt unit ≠ thesis then errorm(⟨er 03⟩)
            end
        else errorm(⟨er 02⟩);
        if erroneous then
        begin errorm(⟨error detected during initialisation of assembler⟩);
                exit
        end;
        from string:=false
end initialize;
```

```
integer procedure read from string;
begin read from string:=stringsymbol(read ptr, ⟨|×;:,.()=+-×/:azbfh

    j3np  553;   ent5  265;   ld4   514;   dec6  166;   inc6  066
    jl    447;   slax  206;   cmp6  576;   enn5  365;   fsub  602
    j1p   251;   j6n   056;   jxp   257;   ldx   517;   st4   534
    cmp2  572;   dec2  162;   enn1  361;   ent1  261;   fadd  601
    inc2  062;   j1n   051;   j2np  552;   j4p   254;   janp  550
```

```
jxn   057;   ld2   512;   ld6   516;   nop   000;   src   506
stj   240;   char  105;   cmp4  574;   cmpx  577;   dec4  164
decx  167;   enn3  363;   enna  360;   ent3  263;   enta  260
fdiv  604;   in    044;   inc4  064;   incx  067;   j1np  551
j2n   052;   j2p   252;   j4n   054;   j5np  555;   j6p   256

jbus  042;   jnov  347;   jxnp  557;   ld1   511;   ld3   513
ld5   515;   lda   510;   move  107;   out   045;   sra   106
st2   532;   st6   536;   stz   541;   add   501;   cmp1  571
cmp3  573;   cmp5  575;   cmpa  570;   dec1  161;   dec3  163
dec5  165;   deca  160;   div   504;   enn2  362;   enn4  364

enn6  366;   ennx  367;   ent2  262;   ent4  264;   ent6  266
entx  267;   fcmp  670;   fmul  603;   hlt   205;   inc1  061
inc3  063;   inc5  065;   inca  060;   ioc   043;   j1nn  351
j1nz  451;   j1z   151;   j2nn  352;   j2nz  452;   j3n   053
j3p   253;   j4np  554;   j5n   055;   j5p   255;   j6np  556

jan   050;   jap   250;   jg    647;   jmp   047;   jred  046
jxnn  357;   jxnz  457;   jxz   157;   ld1n  521;   ld2n  522
ld3n  523;   ld4n  524;   ld5n  525;   ld6n  526;   ldan  520
ldxn  527;   mul   503;   num   005;   sla   006;   slc   406
srax  306;   st1   531;   st3   533;   st5   535;   sta   530
stx   537;   sub   502;   j2z   152;   j3nn  353;   j3nz  453
j3z   153;   j4nn  354;   j4nz  454;   j4z   154;   j5nn  355
j5nz  455;   j5z   155;   j6nn  356;   j6nz  456;   j6z   156
jann  350;   janz  450;   jaz   150;   je    547;   jge   747
jle   947;   jne   847;   jov   247;   jsj   147

      (      alf   con    end    equ    orig )                    ¢);
    read ptr:=read ptr + 1
end read from string;

integer procedure sgn(a); value a; real a;
sgn:=if a < 0 then -1 else +1;

integer procedure ext(c); value c; integer c;
ext:=if c = 69 then 127
     else if c > 35 then c + 27
          else c;

begin:  initialize; time 0:= time;
        read mixal program; print line;
prsym(ext(nlcr)); printtext(¢start address: ¢); fixt(10,0,start);
prsym(ext(nlcr));
if ¬ (0 ≤ start ∧ start ≤ 3999)
then errorm(¢wrong start address¢)
```

```
else start0:=start;
if erroneous then errorm(‹mixal program incorrect›);
prsym(ext(nlcr)); printtext(‹translation time:›);
absfixt(4,2,time - time 0); printtext(‹sec.›);
dump symbol table;
mixal assembler:= ¬ erroneous

end mixal assembler;
```

```
                    "CHAPTER 8      MIXAL ASSEMBLER DESCRIBED IN MIXAL.
                    
                    
                    
                    " UNIT NUMBERS FOR APPARATUS
                    READER:         EQU     16              " CARD READER
                    CPUNCH:         EQU     17              " CARD PUNCH
                    PRINTER:        EQU     18              " LINE PRINTER
                    
                    
                    " INTERNAL CODE OF CHARACTERS
                    LETTER B:       EQU     11              " B
                    LETTER F:       EQU     15              " F
                    LETTER H:       EQU     17              " H
                    LETTER Z:       EQU     35              " Z
                    PLUS:           EQU     37              " +
                    MINUS:          EQU     38              " -
                    TIMES:                                  " *
                        ASTERISK:   EQU     39              " *
                    DIV1:           EQU     40              " /
                    EQUAL SIGN:     EQU     43              " =
                    COMMA:          EQU     60              " ,
                    POINT:          EQU     61              " .
                    DEC:                                    " :
                        COLON:      EQU     63              " :
                    SEMICOLON:      EQU     64              " ;
                    SPACE:          EQU     66              "
                    PAREN:          EQU     71              " (
                    THESIS:         EQU     72              " )
                    NLCR:           EQU     92              "
                    QUOTE:          EQU     94              " "
                    
                    
                    " CONSTANTS USED BY ASSEMBLER
                    SEP:            EQU     101
                    NUMBER:         EQU     102
                    SYMBOL:         EQU     103
                    DIV2:           EQU     104             " //
                    UNDEFINED SYMB: EQU     0
                    DEFINED SYMB:   EQU     1
                    LOCAL LABEL:    EQU     2
                    LOCAL BACKWARD: EQU     3
                    LOCAL FORWARD:  EQU     4
                    VALUE1:         EQU     0 : 2           " SPECIFICATION OF FIELD VALUE1
                    VALUE2:         EQU     4 : 5           " SPECIFICATION OF FIELD VALUE2
                    MOD:            EQU     0               " CONTENTS TO BE MODIFIED
                    N:              EQU     - 1             " USED TO NEGATE EXPRESSIONS
                    LENGTH OF TABLE:EQU     1500
                    
                    
                    " WORKING SPACE OF ASSEMBLER
                                    ORIG    1200
                    BEGIN OF ASS:
                    
                    
12 00    +  00 00 00 00 00  CHAR:           CON     0
12 01    +  00 00 00 00 00  5 CHAR:         CON     0
12 02    +  00 00 00 00 00  LINE NUMBER:    CON     0
```

```
12 03    +  00 00 00 00 00        59  CODE:              CON     0
12 04    +  00 00 00 00 00        60  TEXT:              CON     0
12 05    +  00 00 00 00 00        61  SYNT UNIT:         CON     0
12 06    +  00 00 00 00 00        62  LAST SYMBOL:       CON     0
12 07    +  00 00 00 00 00        63  TYPE:              CON     0
12 08    +  00 00 00 00 00        64  VALUE OF NUMBER:   CON     0
12 09    +  00 00 00 00 00        65  ERRONEOUS:         CON     0
12 10    +  00 00 00 00 00        66  AT EXPR:           CON     0
12 11    +  00 00 00 00 00        67  EXPR:              CON     0
12 12    +  00 00 00 00 00        68  VALUE:             CON     0
12 13    +  00 00 00 00 00        69  CONSTANT:          CON     0
12 14    +  00 00 00 00 00        70  ADDRESS:           CON     0
12 15    +  00 00 00 00 00        71  A PART:            CON     0
12 16    +  00 00 00 00 00        72  I PART:            CON     0
12 17    +  00 00 00 00 00        73  F PART:            CON     0
12 18    +  00 00 00 00 00        74  INSTRUCTION:       CON     0
12 19    +  00 00 00 00 00        75  LC:                CON     0
12 20    +  00 00 00 00 00        76  FUTURE:            CON     0
12 21    +  00 00 00 00 00        77  DANGEROUS:         CON     0
12 22    +  00 00 00 00 00        78  LAST LABEL:        CON     0
12 23    +  00 00 00 00 00        79  EQUIV:             CON     0
12 24    +  00 00 00 00 00        80  INST:              CON     0
12 25    +  00 00 00 00 00        81  LAST CONSTANT:     CON     0
12 26    +  00 00 00 00 00        82  LIT CONS PTR:      CON     0
12 27    +  00 00 00 00 00        83  START ADDRESS:     CON     0
                                  84
                                  85
                                  86                     ORIG    * + 10
                                  87  BACKWARD:                  " SPACE FOR ARRAY "B"
                                  88                     ORIG    * + 10
                                  89  HERE:                      " SPACE FOR ARRAYS 'H' AND 'F'
                                  90  INBUF X:           ORIG    * + 16          " WORK SPACE FOR CARD READER
                                  91  INBUF:             ORIG    * + 16          " INPUT BUFFER FOR CARD READER
                                  92  OUTBUF X:          ORIG    * + 24          " WORK SPACE FOR LINE PRINTER
                                  93  OUTBUF:            ORIG    * + 24          " OUTPUT BUFFER FOR LINE PRINTER
                                  94  TABLE:             ORIG    * + LENGTH OF TABLE     " SPACE FOR SYMBOL TABLE
                                  95
                                  96
                                  97  " BEGIN OF PERMANENT PART OF ASSEMBLER
                                  98
                                  99
28 28    +  00 00 00 14 99       100  TABLE PTR1:       CON     LENGTH OF TABLE - 1
28 29    +  00 00 00 00 00       101  LAST INST:         CON     0
28 30    +  00 00 00 00 00       102  ALF:               CON     0
28 31    +  00 00 00 00 00       103  CON:               CON     0
28 32    +  00 00 00 00 00       104  END:               CON     0
28 33    +  00 00 00 00 00       105  EQU:               CON     0
28 34    +  00 00 00 00 00       106  ORIG:              CON     0
28 35    +  00 00 00 00 10       107  BASE:              CON     10
                                 108  BYTESIZE: S:       EQU     1 (4 : 4)
28 36    +  00 00 00 00 99       109  MAX BYTE:          CON     S-1
                                 110  END MARKER:
28 37    +  00 00 00 99 99       111  MAX ADDRESS:       CON     S-1 (4:4), S-1 (5:5)
28 38    +  99 99 99 99 99       112  MAX WORD:          CON     S-1 (1:1), S-1 (2:2), S-1 (3:3), S-1 (4:4), S-1 (5:5)
                                 113  LENGTH OF MEMORY:
28 39    +  00 00 00 39 99       114                     CON     3999
28 40    +  00 00 10 00 00       115  HUNDREDTHOUSAND:   CON     100000
28 41    +  66 66 66 66 66       116  SPACES:            ALF     .
28 42    +  66 66 66 94 94       117  JOB SEPARATOR:     ALF     .   ""          " TO BE PUNCHED IN COLUMNS 79 AND 80
28 43    +  14 27 27 24 27       118  ERROR:             ALF     .ERROR
```

```
28 44    +  01 01 01 01 01        119  CORRECTION:    CON     1(1:1), 1(2:2), 1(3:3), 1(4:4), 1(5:5)
                                   120
                                   121
                                   122  " SUBROUTINES OF ASSEMBLER
                                   123
                                   124
                                   125  READ CHAR:     " RA IS AN OUTPUT PARAMETER FOR THE CODE OF THE CHARACTER,
                                   126                 " R11 AND R12 ARE PERMANENT READING POINTERS.
28 45                 00 02 40     127                 STJ     9F              " SAVE RETURN ADDRESS
28 46    +  12 04 00 05 10        128                 LDA     TEXT            ") IF TEXT
28 47    +  12 03 00 05 01        129                 ADD     CODE            ") OR CODE
28 48                 00 02 50     130                 JAP     PRINT LINE      ") THEN PRINT THIS LINE
28 49                 00 05 52     131                 J2NP    READC           " JUMP IF BUFFER NOT EXHAUSTED
28 50    +  12 79 00 05 10        132                 LDA     INBUF + 15      " POSSIBLE JOB SEPARATOR
28 51    +  28 42 00 45 70        133                 CMPA    JOB SEPARATOR (4 : 5)   " IS IT A JOB SEPARATOR ?
28 52                 00 08 47     134                 JNE     1F              " ELSE CONTINUE THE READING
28 53    +  00 98 00 02 66        135                 ENT6    98              " REPORT ERROR:
28 54                 00 00 47     136                 JMP     ERRORM          " END OF INPUT
28 55                 00 00 47     137                 JMP     RESET TABLE     " RESET THE INITIAL SYMBOL TABLE
28 56                 00 00 47     138                 JMP     NEXT PROGRAM    " TREAT THE NEXT PROGRAM
28 57    +  12 02 00 05 10        139  1H:             LDA     LINE NUMBER     ") INCREASE
*28 58   +  28 57 00 08 47
28 58    +  00 01 00 00 60        140                 INCA    1               ") LINE NUMBER
28 59    +  12 02 00 05 30        141                 STA     LINE NUMBER     ") BY ONE
28 60                 00 00 47     142                 JMP     BUFFER LN       " 'PRINT' THE LINE NUMBER
28 61    +  00 16 00 02 62        143                 ENT2    16              " ONE AFTER LAST WORD OF BUFFER
28 62    +  28 62 00 16 42        144                 JBUS    * (READER)      " WAIT FOR CARD READER
28 63    +  12 64 00 02 61        145                 ENT1    INBUF           ") MOVE NEXT CARD FROM APPARATUS
28 64    +  12 48 00 16 07        146                 MOVE    INBUF X (16)    ") BUFFER TO INPUT BUFFER
28 65    +  12 79 00 05 10        147                 LDA     INBUF + 15      " POSSIBLE JOB SEPARATOR
28 66    +  28 42 00 45 70        148                 CMPA    JOB SEPARATOR (4 : 5)   " IS IT A JOB SEPARATOR ?
28 67    +  28 69 00 05 47        149                 JE      * + 2           " THEN SKIP NEXT INSTRUCTION
28 68    +  12 48 00 16 44        150                 IN      INBUF X (READER) " READ NEXT CARD
28 69    +  00 01 00 01 62        151  2H:            DEC2    1               " PREVIOUS WORD OF BUFFER
28 70                 00 00 52     152                 J2N     3F              " JUMP IF NOT AVAILABLE
28 71    +  12 64 02 05 20        153                 LDAN    INBUF, 2        " CONTENTS OF THE WORD (INVERTED)
28 72    +  28 41 00 45 70        154                 CMPA    SPACES (1 : 5)  ") IF SPACES
28 73    +  28 69 00 05 47        155                 JE      2B              ") THEN LOOP
28 74    +  12 65 02 00 30        156  3H:            STA     INBUF + 1, 2 (0 : 0)    " END OF BUFFER INDICATOR
*28 75   +  28 74 00 00 52
28 75    -  00 16 00 02 62        157                 ENT2    -16             " INITIALIZE READING PTR (TO WORDS)
28 76    -  00 55 03 03 63        158  READC:         ENN3    5 : 5 /N ,3     " PREVIOUS CHARACTER IN BYTE 5 ?
*28 77   +  28 76 00 05 52
28 77                 00 02 53     159                 J3P     3F              " ELSE JUMP
28 78    +  00 55 00 02 63        160                 ENT3    5 : 5           " INITIALIZE READING PTR (TO BYTES)
28 79    +  00 01 00 10 62        161                 INC2    1               " NEXT WORD
28 80                 00 05 52     162                 J2NP    2F              " JUMP IF WORD IS AVAILABLE
28 81    +  00 01 00 02 62        163  1H:            ENT2    1               " POSITION = - 1
28 82    +  00 92 00 02 60        164                 ENTA    NLCR            " END OF LINE READ
28 83    +  12 04 00 05 30        165                 STA     TEXT            " TEXT OF ONE LINE COMPLETED
28 84                 00 00 47     166                 JMP     8F              " READY
28 85    +  12 79 00 05 10        167  2H:            LDA     INBUF + 15, 2   " END OF BUFFER INDICATOR ?
*28 86   +  28 85 00 05 52
28 86    +  28 81 00 00 50        168                 JAN     1B              " THEN SKIP REMAINING SPACES
28 87    -  00 66 03 03 63        169  3H:            ENN3    6 : 6 /N ,3     " POINTER  TO NEXT BYTE
*28 87   +  28 87 00 02 53
28 88    +  28 89 00 44 33        170                 ST3     * + 1 (4 : 4)   " MODIFY NEXT INSTRUCTION
28 89    +  12 79 02 00 10        171                 LDA     INBUF + 15 ,2 (MOD)     " 'READ' THE CHARACTER
28 90    +  28 91 00 44 33        172                 ST3     * + 1 (4 : 4)   " MODIFY NEXT INSTRUCTION
28 91    +  13 27 02 00 30        173                 STA     OUTBUF+8 + 15 ,2 (MOD)  " 'PRINT' THE CHARACTER
```

```
 28 92     +  12 00 00 05 30      174   8H:          STA     CHAR            " INTERNAL CODE OF CHARACTER
*28 84     +  28 92 00 00 47
 28 93     +  00 00 00 00 47      175   9H:          JMP     MOD             " RETURN
*28 45     +  28 93 00 02 40
                                  176
                                  177
                                  178   READ 5 CHAR: " RA IS AN OUTPUT PARAMETER FOR THE CODE OF THE CHARACTERS
 28 94                00 02 40    179                STJ     9F              " SAVE RETURN ADDRESS
 28 95     +  12 05 00 05 10      180                LDA     SYNT UNIT       ") IS SYNTACTICAL UNIT
 28 96     +  00 61 00 01 60      181                DECA    POINT           ") A POINT ?
 28 97                00 04 50    182                JANZ    8F              ") ELSE ERROR
 28 98     +  00 55 00 02 65      183                ENT5    5 : 5           " INITIALIZE BYTE POINTER
 28 99     +  28 45 00 00 47      184                JMP     READ CHAR       " READ THE FIRST CHARACTER
 29 00     -  00 66 05 03 65      185   2H:          ENN6    6 : 6 /N ,5     " POINTER TO NEXT BYTE
 29 01     +  00 92 00 02 67      186                ENTX    NLCR            ") REPLACE POSSIBLE
 29 02     +  12 00 00 05 77      187                CMPX    CHAR            ") END OF CARD (NLCR)
 29 03     +  29 05 00 08 47      188                JNE     * + 2
 29 04     +  00 66 00 02 60      189                ENTA    SPACE           ") BY SPACES
 29 05     +  29 06 00 44 35      190                ST5     * + 1 (4 : 4)   " MODIFY NEXT INSTRUCTION
 29 06     +  12 01 00 00 30      191                STA     5 CHAR (MOD)    " STORE THE CHARACTER
 29 07     -  00 55 05 03 65      192                ENN5    5 : 5 /N ,5     " IS IT THE FIFTH CHARACTER ?
 29 08     +  28 45 00 08 47      193                JNE     READ CHAR       " READ NEXT CHARACTER
 29 09     +  29 00 00 02 55      194                J5P     2B              " ELSE CONTINUE THE READING
 29 10                00 00 47    195                JMP     READ SYNT UNIT  " NEXT SYNTACTICAL UNIT
 29 11     +  12 01 00 05 10      196                LDA     5 CHAR          " RESULTING VALUE OF CHARACTERS
 29 12     +  00 00 00 00 47      197   9H:          JMP     MOD             " RETURN
*28 94     +  29 12 00 02 40
 29 13     +  00 10 00 02 66      198   8H:          ENT6    10              " REPORT ERROR:
*28 97     +  29 13 00 04 50
 29 14                00 00 47    199                JMP     ERRORM          " . REQUIRED
 29 15     +  29 12 00 00 47      200                JMP     9B              " RETURN
                                  201
                                  202
                                  203   PRINT LINE:  " NO PARAMETERS
 29 15                00 02 40    204                STJ     9F              " SAVE RETURN ADDRESS
*28 48     +  29 16 00 02 50
 29 17     +  29 17 00 18 42      205                JBUS    * (PRINTER)     " WAIT FOR LINE PRINTER
 29 18     +  12 80 00 02 61      206                ENT1    OUTBUF X        ") MOVE NEXT LINE FROM OUTPUT
 29 19     +  13 04 00 24 07      207                MOVE    OUTBUF (24)     ") BUFFER TO APPARATUS BUFFER
 29 20     +  12 80 00 18 45      208                OUT     OUTBUF X (PRINTER)      " PRINT THE LINE
 29 21     +  28 41 00 01 07      209                MOVE    SPACES (1)      ") REFILL BUFFER
 29 22     +  13 04 00 23 07      210                MOVE    OUTBUF (23)     ") WITH SPACES
 29 23     +  12 03 00 05 41      211                STZ     CODE            " NO CODE IN BUFFER
 29 24     +  12 04 00 05 41      212                STZ     TEXT            " NO TEXT IN BUFFER
 29 25     +  00 00 00 00 47      213   9H:          JMP     MOD             " RETURN
*29 16     +  29 25 00 02 40
                                  214
                                  215
                                  216   ERRORM:      " R16 CONTAINS THE ERROR NUMBER
 29 26                00 02 40    217                STJ     9F              " SAVE RETURN ADDRESS
*29 14     +  29 26 00 00 47
*28 54     +  29 26 00 00 47
 29 27     +  29 16 00 00 47      218                JMP     PRINT LINE      " PRINT THE (INCOMPLETE) LINE
 29 28     +  28 43 00 05 10      219                LDA     ERROR           ") 'PRINT' THE
 29 29     +  13 05 00 05 30      220                STA     OUTBUF + 1      ") CHARACTERS: E R R O R
 29 30     +  00 00 06 02 60      221                ENTA    0 ,6            " ERROR NUMBER
 29 31     +  00 00 00 01 05      222                CHAR                    ") 'PRINT' IT
 29 32     +  13 06 00 45 37      223                STX     OUTBUF + 2 (4 : 5)
 29 33     +  12 05 00 05 10      224                LDA     SYNT UNIT       " SYNTACTICAL UNIT
 29 34     +  00 00 00 01 05      225                CHAR                    ") 'PRINT' IT
```

```
29 35     +  13 08 00 05 37      226                 STX     OUTBUF + 4 (3 : 5)
29 36     +  12 11 00 05 10      227                 LDA     EXPR            " VALUE OF EXPRESSION
29 37     +  00 00 00 01 05      228                 CHAR
29 38     +  13 10 00 05 30      229                 STA     OUTBUF + 6      ") 'PRINT'
29 39     +  13 11 00 05 37      230                 STX     OUTBUF + 7      ") THE VALUE
29 40     +  29 16 00 00 47      231                 JMP     PRINT LINE      " PRINT THIS LINE
29 41     +  00 01 00 02 66      232                 ENT6    1               ") NOTE
29 42     +  12 09 00 05 36      233                 ST6     ERRONEOUS       ") THE ERROR
29 43     +  00 00 00 00 47      234 9H!             JMP     MOD             " RETURN
*29 26    +  29 43 00 02 40
                                 235
                                 236
                                 237 READ SYNT UNIT: " RA IS AN OUTPUT PARAMETER FOR THE CODE OF THE SYNTACTICAL UNIT
                                 238                 " R:1 DELIVERS A POINTER TO A SYMBOL READ
29 44                 00 02 40  239                 STJ     9F              " SAVE RETURN ADDRESS
*29 10    +  29 44 00 00 47
29 45     +  12 00 00 05 10      240                 LDA     CHAR            " LAST CHARACTER
29 46     +  00 66 00 01 60      241 1H!             DECA    SPACE           " IS IT A SPACE ?
29 47                 00 04 50  242                 JANZ    LETGIT          " ELSE JUMP
29 48     +  28 45 00 00 47      243                 JMP     READ CHAR       ") SKIP
29 49     +  29 46 00 00 47      244                 JMP     1B              ") SPACES
29 50     -  00 31 00 01 60      245 LETGIT:         DECA    LETTER Z - SPACE
29 47     +  29 50 00 04 50
29 51                 00 02 50  246                 JAP     SEPARATOR       " JUMP IF NOT A LETTER OR DIGIT
29 52     -  00 26 00 01 60      247                 DECA    9 - LETTER Z    " IS IT A LETTER ?
29 53                 00 02 30  248                 STA     COMPL (0 : 2)   " NOTE THIS (MODIFY INSTRUCTION)
29 54     +  28 28 00 05 14      249                 LD4     TABLE PTR1      ") FREE SPACE
29 55     +  00 03 00 01 64      250                 DEC4    3               ") IN SYMBOL TABLE
29 56     +  12 26 00 05 16      251                 LD6     LIT CONS PTR    " END OF SPACE FOR SYMBOLS
29 57     +  00 00 04 01 66      252                 DEC6    0 ,4            " SPACE AVAILABLE ?
29 58                 00 02 56  253                 J6P     EXIT            " ELSE EXIT
29 59     +  13 28 04 05 41      254                 STZ     TABLE ,4        " CLEAN SPACE FOR CHARACTERS
29 60     +  00 10 00 00 60      255                 INCA    9 + 1           " MODIFY THE CHARACTER (ADD 1)
29 61     +  13 28 04 11 30      256                 STA     TABLE ,4 (1 : 1) " STORE THE FIRST CHARACTER
29 62     +  00 11 00 02 65      257                 ENT5    1 : 1           " INITIALIZE BYTE POINTER
29 63     +  28 45 00 00 47      258 READ!           JMP     READ CHAR       " READ NEXT CHARACTER
29 64     +  00 66 00 01 60      259                 DECA    SPACE           " IS IT A SPACE ?
29 65     +  28 45 00 01 50      260                 JAZ     READ CHAR       " THEN SKIP ONE
29 66     +  12 00 00 05 10      261                 LDA     CHAR            " CHARACTER
29 67     +  00 35 00 01 60      262                 DECA    LETTER Z        " IS IT A LETTER OR A DIGIT ?
29 68                 00 05 50  263                 JANP    2F              " THEN JUMP
29 69     +  00 01 00 03 61      264                 ENN1    1               ") INDICATOR FOR END OF
29 70     +  13 28 04 00 31      265                 ST1     TABLE ,4 (0 : 0)") ALPHA NUMERICAL INFORMATION
29 71                 00 00 47  266                 JMP     COMPL           " SYNTACTICAL UNIT COMPLETED
29 72     -  00 26 00 01 60      267 2H!             DECA    9 - LETTER Z    " IS THE CHARACTER A LETTER ?
*29 68    +  29 72 00 05 50
29 73     +  29 75 00 05 50      268                 JANP    * + 2           " ELSE JUMP
29 74                 00 02 34  269                 ST4     COMPL (0 : 2)   " NOTE OCCURRENCE OF LETTER
29 75     -  00 55 05 03 65      270                 ENN5    5 : 5 /N ,5     " PREVIOUS CHARACTER IN BYTE 5 ?
29 76                 00 02 55  271                 J5P     3F              " ELSE JUMP
29 77     +  00 55 00 02 65      272                 ENT5    5 : 5           " INITIALIZE BYTE POINTER
29 78     +  00 01 00 01 64      273                 DEC4    1               " NEXT WORD WITH ALPHA NUMERICALS
29 79     +  12 26 00 05 16      274                 LD6     LIT CONS PTR    " END OF SPACE FOR SYMBOLS
29 80     +  00 00 04 01 66      275                 DEC6    0 ,4            " SPACE AVAILABLE ?
29 81                 00 02 56  276                 J6P     EXIT            " ELSE EXIT
29 82     +  13 28 04 05 41      277                 STZ     TABLE ,4        " CLEAN SPACE FOR CHARACTERS
29 83     -  00 66 05 03 65      278 3H!             ENN5    6 : 6 /N ,5     " POINTER TO NEXT BYTE
*29 76    +  29 83 00 02 55
29 84     +  00 10 00 00 60      279                 INCA    9 + 1           " MODIFY THE CHARACTER (ADD 1)
29 85     +  29 86 00 44 35      280                 ST5     * + 1 (4 : 4)   " MODIFY NEXT INSTRUCTION
```

```
 29 86     +  13 28 04 00 30        281                 STA    TABLE ,4 (MOD)  " STORE THE LETTER OR DIGIT
 29 87     +  29 63 00 00 47        282                 JMP    READ            " PROCEED WITH NEXT CHARACTERS
 29 88     +  00 00 00 02 60        283   COMPL:        ENTA   MOD             " AT LEAST ONE LETTER READ ?
*29 74     +  29 88 00 02 34
*29 71     +  29 88 00 00 47
*29 53     +  29 88 00 02 30
 29 89                 00 02 50     284                 JAP    CONTAINS LETTER " THEN JUMP, AND TREAT SYMBOL
 29 90     +  28 28 00 05 11        285                 LD1    TABLE PTR1      " POINTER TO NEW INFO CELL
 29 91     +  12 08 00 05 41        286                 STZ    VALUE OF NUMBER " INITIALIZE VALUE OF NUMBER
 29 92     +  00 00 04 02 66        287   1H:           ENT6   0 ,4            " END OF INFO JUST STORED
 29 93     -  00 04 01 01 66        288                 DEC6   - 4 ,1          " LENGTH OF INFO
 29 94                 00 02 56     289                 J6P    3F              " JUMP IF ONLY ONE WORD WITH DIGITS
 29 95                 00 01 56     290                 J6Z    2F              " JUMP IF ONLY TWO WORDS WITH DIGITS
 29 96     +  13 24 01 05 10        291                 LDA    TABLE - 4 ,1    " FIFTH TILL TENTH DIGIT
 29 97     +  28 44 00 05 02        292                 SUB    CORRECTION      " CANCEL MODIFICATION OF CHARACTERS
 29 98     +  00 05 00 03 06        293                 SRAX   5               " SHIFT DIGITS INTO RX
 29 99     +  13 25 01 05 10        294                 LDA    TABLE - 3 ,1    " FIRST TILL FIFTH DIGIT
 30 00     +  28 44 00 05 02        295                 SUB    CORRECTION      " CANCEL MODIFICATION OF CHARACTERS
 30 01     +  00 00 00 00 05        296                 NUM                    " NUMERICAL VALUE OF TEN DIGITS
 30 02     +  12 08 00 05 17        297                 LDX    VALUE OF NUMBER " MORE DIGITS TREATED ALREADY ?
 30 03                 00 02 57     298                 JXP    OVERFLOW        " THEN OVERFLOW NOW
 30 04     +  12 08 00 05 30        299                 STA    VALUE OF NUMBER " STORE VALUE OF THE TEN DIGITS
 30 05     +  00 02 00 01 61        300                 DEC1   2               " POINTER TO NEXT DIGITS
 30 06     +  29 92 00 00 47        301                 JMP    1B              " CONTINUE THE PROCESS
 30 07     +  12 08 00 05 10        302   2H:           LDA    VALUE OF NUMBER ") SHIFT VALUE OBTAINED
*29 95     +  30 07 00 01 56
 30 08     +  28 40 00 05 03        303                 MUL    HUNDREDTHOUSAND ") OVER FIVE DECIMAL POSITIONS
 30 09                 00 04 50     304                 JANZ   OVERFLOW        " JUMP IF OVERFLOW OCCURS
 30 10     +  12 08 00 05 37        305                 STX    VALUE OF NUMBER " STORE INTERMEDIATE VALUE
 30 11     +  13 29 04 05 10        306                 LDA    TABLE + 1 ,4    " FIVE LAST BUT FIVE DIGITS
 30 12     +  28 44 00 05 02        307                 SUB    CORRECTION      " CANCEL MODIFICATION OF CHARACTERS
 30 13     +  00 05 00 03 06        308                 SRAX   5               " SHIFT DIGITS INTO RX, RA = 0
 30 14     +  00 00 00 00 05        309                 NUM                    " NUMERICAL VALUE OF FIVE DIGITS
 30 15     +  12 08 00 05 01        310                 ADD    VALUE OF NUMBER " ADD TO INTERMEDIATE VALUE
 30 16     +  12 08 00 05 30        311                 STA    VALUE OF NUMBER " RESULTING VALUE (NO OVERFLOW)
 30 17     +  00 00 00 02 60        312   3H:           ENTA   0               " RA = 0
*29 94     +  30 17 00 02 56
 30 18     -  00 55 05 03 67        313                 ENNX   5 : 5 /N ,5     ") NUMBER OF BYTES IN LAST WORD
 30 19                 00 05 04     314                 DIV    = 1 : 1 =       ") WHICH HAVE NOT BEEN FILLED IN
 30 20     +  30 21 00 02 30        315                 STA    * + 1 (0 : 2)   " MODIFY NEXT INSTRUCTION
 30 21     +  00 00 00 02 66        316                 ENT6   MOD             " NUMBER OF BYTES NOT FILLED IN
 30 22     +  28 40 00 05 10        317                 LDA    HUNDREDTHOUSAND ") ONE, FOLLOWED BY
 30 23     +  00 00 00 01 05        318                 CHAR                   ") FIVE DECIMAL ZEROS
 30 24     +  00 00 06 03 06        319                 SRAX   0 ,6            " SHIFT OUT THE BYTES NOT FILLED IN
 30 25     +  00 00 00 00 05        320                 NUM                    ") NUMERICAL VALUE INDICATING
 30 26                 00 05 30     321                 STA    AUX             ") THE DECIMAL SHIFT OF VALUE OF NUMBER
 30 27     +  12 08 00 05 10        322                 LDA    VALUE OF NUMBER " VALUE OF NUMBER
 30 28                 00 05 03     323                 MUL    AUX             " SHIFT OVER DECIMAL POSITIONS
 30 29                 00 04 50     324                 JANZ   OVERFLOW        " JUMP IF OVERFLOW OCCURS
 30 30     +  12 08 00 05 37        325                 STX    VALUE OF NUMBER " STORE INTERMEDIATE VALUE
 30 31     +  13 28 04 05 17        326                 LDX    TABLE ,4        " LAST DIGITS IN RX
 30 32     +  28 44 00 05 10        327                 LDA    CORRECTION      " (MODIFIED) ZEROS IN RA
 30 33     -  00 05 06 03 66        328                 ENN6   - 5 ,6          " NUMBER OF DIGITS IN LAST WORD
 30 34     +  00 00 06 02 06        329                 SLAX   0 ,6            " SHIFT DIGITS INTO RA
 30 35     +  28 44 00 05 02        330                 SUB    CORRECTION      " CANCEL MODIFICATION OF CHARACTERS
 30 36     +  00 05 00 03 06        331                 SRAX   5               " SHIFT DIGITS INTO RX, RA = 0
 30 37     +  00 00 00 00 05        332                 NUM                    " NUMERICAL VALUE OF LAST DIGITS
 30 38     +  12 08 00 05 01        333                 ADD    VALUE OF NUMBER " ADD TO INTERMEDIATE VALUE
 30 39     +  12 08 00 05 30        334                 STA    VALUE OF NUMBER " RESULTING VALUE (NO OVERFLOW)
 30 40     +  01 02 00 02 60        335   1H:           ENTA   NUMBER          ") THIS SYNTACTICAL UNIT
```

```
 30 41    +  12 05 00 05 30      336                  STA   SYNT UNIT          ") IS A NUMBER
 30 42          00 00 47         337                  JMP   9F                 " RETURN
 30 43    +  12 08 00 05 41      338  OVERFLOW:       STZ   VALUE OF NUMBER    " RESULTING VALUE
*30 29    +  30 43 00 04 50
*30 09    +  30 43 00 04 50
*30 03    +  30 43 00 02 57
 30 44    +  00 11 00 02 66      339                  ENT6  11                 " REPORT ERROR:
 30 45    +  29 26 00 00 47      340                  JMP   ERRORM             " VALUE OF NUMBER TOO LARGE
 30 46    +  30 40 00 00 47      341                  JMP   1B                 " READY
                                 342  CONTAINS LETTER:
 30 47    +  28 28 00 05 11      343                  LD1   TABLE PTR1         " POINTER TO NEW INFO CELL
*29 89    +  30 47 00 02 50
 30 48    +  00 03 04 01 61      344                  DEC1  3 ,4               " ONLY ONE WORD WITH CHARACTERS ?
 30 49          00 04 51         345                  J1NZ  LOOK FOR SYMBOL    " ELSE ORDINARY SYMBOL
 30 50    +  00 22 00 01 65      346                  DEC5  2 : 2              " TWO CHARACTERS ?
 30 51          00 04 55         347                  J5NZ  LOOK FOR SYMBOL    " ELSE ORDINARY SYMBOL
 30 52    +  13 28 04 11 11      348                  LD1   TABLE ,4 (1 : 1)   " FIRST CHARACTER
 30 53    +  00 10 00 01 61      349                  DEC1  9 + 1              " IS IT A DIGIT ?
 30 54          00 02 51         350                  J1P   LOOK FOR SYMBOL    " ELSE ORDINARY SYMBOL
 30 55    +  13 28 04 22 17      351                  LDX   TABLE ,4 (2 : 2)   " SECOND CHARACTER
 30 56    +  00 12 00 01 67      352                  DECX  LETTER B + 1       " IS IT LETTER B ?
 30 57          00 01 57         353                  JXZ   B                  " THEN JUMP
 30 58    +  00 04 00 01 67      354                  DECX  LETTER F - LETTER B
 30 59          00 01 57         355                  JXZ   F                  " JUMP IF IT IS LETTER F
 30 60    +  00 02 00 01 67      356                  DECX  LETTER H - LETTER F
 30 61          00 04 57         357                  JXNZ  LOOK FOR SYMBOL    " JUMP IF IT IS NOT LETTER H
 30 62    +  00 02 00 02 67      358                  ENTX  LOCAL LABEL        " TYPE LOCAL LABEL
 30 63          00 00 47         359                  JMP   1F                 " READY
 30 64    +  00 04 00 02 67      360  F:              ENTX  LOCAL FORWARD      " TYPE LOCAL FORWARD
*30 59    +  30 64 00 01 57
 30 65          00 00 47         361                  JMP   1F                 " READY
 30 66    +  00 03 00 02 67      362  B:              ENTX  LOCAL BACKWARD     " TYPE LOCAL BACKWARD
*30 57    +  30 66 00 01 57
 30 67    +  12 07 00 05 37      363  1H:             STX   TYPE               " STORE THE TYPE
*30 65    +  30 67 00 00 47
*30 63    +  30 67 00 00 47
 30 68    +  00 10 01 03 61      364                  ENN1  9 + 1 ,1           " POINTER (NEGATIVE-1) TO LOCAL SYMBOL
 30 69          00 00 47         365                  JMP   7F                 " LOCAL SYMBOL FOUND
                                 366  LOOK FOR SYMBOL:
 30 70    +  14 99 00 02 61      367                  ENT1  LENGTH OF TABLE + 1    " ROOT OF BINARY TREE
*30 61    +  30 70 00 04 57
*30 54    +  30 70 00 02 51
*30 51    +  30 70 00 04 55
*30 49    +  30 70 00 04 51
 30 71    -  00 03 01 02 65      368  NEXT:          ENT5  - 3 ,1             " POINTER TO LETTERS IN TREE
 30 72    +  28 28 00 05 16      369                  LD6   TABLE PTR1         ") POINTER TO LETTERS OF SYMBOL
 30 73    +  00 03 00 01 66      370                  DEC6  3                  ") TO BE LOOKED FOR
 30 74    +  13 28 06 05 10      371  1H:             LDA   TABLE ,6           " COMPARE THE NEW SYMBOL
 30 75    +  13 28 05 25 70      372                  CMPA  TABLE ,5 (1 : 5)   " WITH THE SYMBOL IN TREE
 30 76          00 08 47         373                  JNE   3F                 " SELECT NEXT SYMBOL , IF NOT EQUAL
 30 77    +  13 28 05 05 17      374                  LDX   TABLE ,5           ") JUMP IF THESE ARE THE FINAL
 30 78          00 05 57         375                  JXNP  2F                 ") CHARACTERS OF THE SYMBOL IN TREE
 30 79          00 05 50         376                  JANP  LEFT               " JUMP IF NEW SYMBOL EXHAUSTED
 30 80    +  00 01 00 01 65      377                  DEC5  1                  ") NEXT
 30 81    +  00 01 00 01 66      378                  DEC6  1                  ") FIVE LETTERS
 30 82    +  30 74 00 00 47      379                  JMP   1B                 ") TO BE COMPARED
 30 83          00 05 50         380  2H:             JANP  FOUND              " JUMP IF NEW SYMBOL ALSO EXHAUSTED
*30 78    +  30 83 00 05 57
 30 84          00 00 47         381                  JMP   RIGHT              " NEW SYMBOL IS LONGER
 30 85          00 06 47         382  3H:             JG    RIGHT              " JUMP IF NEW SYMBOL SUCCEEDS OLD ONE
```

```
*30 76    +  30 85 00 08 47
 30 85    +  13 28 01 23 10      383   LEFT:     LDA    TABLE ,1 (2 : 3) " LEFT BRANCH
*30 79    +  30 86 00 05 50
 30 87             00 04 50      384             JANZ   1F              " JUMP IF NOT EMPTY
 30 88    +  00 00 01 03 67      385             ENNX   0 ,1            " NEW THREAD
 30 89    +  28 28 00 05 10      386             LDA    TABLE PTR1      " NEW LEFT BRANCH
 30 90    +  13 28 01 23 30      387             STA    TABLE ,1 (2 : 3) " STORE NEW LEFT BRANCH
 30 91             00 00 47      388             JMP    NEW             " CREATE NEW INFO CELL
 30 92    +  30 93 00 02 30      389   1H:       STA    * + 1 (0 : 2)   " MODIFY NEXT INSTRUCTION
*30 87    +  30 92 00 04 50
 30 93    +  00 00 00 02 61      390             ENT1   MOD             " POINTER TO NEXT INFO CELL
 30 94    +  30 71 00 00 47      391             JMP    NEXT            " CONTINUE THE SEARCH
 30 95    +  13 28 01 05 10      392   RIGHT:    LDA    TABLE ,1        " RIGHT BRANCH
*30 89    +  30 95 00 06 47
*30 84    +  30 95 00 00 47
 30 95    +  30 92 00 02 50      393             JAP    1B              " JUMP IF IT IS NOT A THREAD
 30 97    +  13 28 01 45 27      394             LDXN   TABLE ,1 (4 : 5)" TAKE THE THREAD (NEGATIVE)
 30 98    +  28 28 00 05 10      395             LDA    TABLE PTR1      " NEW RIGHT BRANCH
 30 99    +  13 28 01 00 30      396             STA    TABLE ,1 (0 : 0) " NOT A THREAD
 31 00    +  13 28 01 45 30      397             STA    TABLE ,1 (4 : 5) " STORE RIGHT BRANCH
 31 01    +  28 28 00 05 11      398   NEW:      LD1    TABLE PTR1      " POINTER TO NEW INFO CELL
*30 91    +  31 01 00 00 47
 31 02    +  13 28 01 05 37      399             STX    TABLE ,1        " STORE A THREAD
 31 03    +  13 27 01 05 41      400             STZ    TABLE - 1 ,1    ") INITIALIZE
 31 04    +  28 37 00 05 10      401             LDA    END MARKER      ") NEW
 31 05    +  13 26 01 05 30      402             STA    TABLE - 2 ,1    ") INFO CELL
 31 06    +  00 01 00 01 64      403             DEC4   1
 31 07    +  28 28 00 05 34      404             ST4    TABLE PTR1      " INCREASE TABLE POINTER
 31 08             00 00 47      405             JMP    6F              " ORDINARY SYMBOL STORED
 31 09    +  28 28 00 05 10      406   FOUND:    LDA    TABLE PTR1      " POINTER TO NEW SYMBOL
*31 06    +  31 09 00 05 50
 31 10    +  14 99 00 01 60      407             DECA   LENGTH OF TABLE - 1
 31 11    +  28 37 00 05 27      408             LDXN   END MARKER      " FINAL THREAD
 31 12    +  31 01 00 01 50      409             JAZ    NEW             " JUMP IF ONLY ONE SYMBOL IN TREE
 31 13    +  13 27 01 01 10      410   6H:       LDA    TABLE - 1 ,1 (0 : 1)
*31 08    +  31 13 00 00 47
 31 14    +  12 07 00 05 30      411             STA    TYPE            " SAVE TYPE OF SYMBOL
 31 15    +  12 06 00 05 31      412   7H:       ST1    LAST SYMBOL     " POINTER TO THIS SYMBOL
*30 69    +  31 15 00 00 47
 31 16    +  01 03 00 02 60      413             ENTA   SYMBOL          ") THIS SYNTACTICAL UNIT
 31 17    +  12 05 00 05 30      414             STA    SYNT UNIT       ") IS A SYMBOL
 31 18             00 00 47      415             JMP    9F              " RETURN
 31 19    +  00 59 00 01 60      416   SEPARATOR: DECA   QUOTE - LETTER Z " IS THE CHARACTER A QUOTE ?
*29 93    +  31 19 00 02 50
 31 20             00 04 50      417             JANZ   2F              " ELSE JUMP
 31 21    +  28 45 00 00 47      418   1H:       JMP    READ CHAR       " READ NEXT CHARACTER
 31 22    +  00 92 00 01 60      419             DECA   NLCR            " IS IT A NLCR ?
 31 23    +  31 21 00 04 50      420             JANZ   1B              " ELSE SKIP UNTIL NLCR
 31 24             00 00 47      421             JMP    3F              " NLCR READ
 31 25    -  00 02 00 01 60      422   2H:       DECA   NLCR - QUOTE    " IS IT A NLCR ?
*31 20    +  31 25 00 04 50
 31 26             00 04 50      423             JANZ   4F              " ELSE JUMP
 31 27    +  01 01 00 02 60      424   3H:       ENTA   SEP             ") THIS SYNTACTICAL UNIT
*31 24    +  31 27 00 00 47
 31 28             00 00 47      425             JMP    8F              ") IS A SEPARATOR
 31 29    -  00 28 00 01 60      426   4H:       DECA   SEMICOLON - NLCR " IS THE CHARACTER A SEMICOLON ?
*31 25    +  31 29 00 04 50
 31 30             00 04 50      427             JANZ   DIV             " ELSE JUMP
 31 31    +  00 01 00 02 61      428             ENT1   1               ") TEXT OF ONE INSTRUCTION
 31 32    +  12 04 00 05 31      429             ST1    TEXT            ") COMPLETED NOW
```

```
 31 33     +  31 27 00 00 47      430                    JMP     3B              " SEPARATOR READ
 31 34     -  00 24 00 01 60      431   DIV:             DECA    DIV1 - SEMICOLON " IS THE CHARACTER A DIVISION SIGN ?
*31 36     +  31 34 00 04 50
 31 35                00 04 50   432                    JANZ    OTHER           " ELSE JUMP
 31 36     +  28 45 00 00 47      433                    JMP     READ CHAR       " READ NEXT CHARACTER
 31 37     +  00 40 00 01 60      434                    DECA    DIV1            " IS IT ALSO A DIVISION SIGN ?
 31 35                00 04 50   435                    JANZ    5F              " ELSE JUMP
 31 39     +  01 04 00 02 60      436                    ENTA    DIV2            ") THIS SYNTACTICAL UNIT
 31 40                00 00 47   437                    JMP     8F              ") IS A SPECIAL DIVISION OPERATOR
 31 41     +  00 40 00 02 60      438   5H:              ENTA    DIV1            " COMMON DIVISION OPERATOR
*31 38     +  31 41 00 04 50
 31 42     +  12 05 00 05 30      439                    STA     SYNT UNIT       " THIS SYNTACTICAL UNIT
 31 43                00 00 47   440                    JMP     9F              " RETURN
 31 44     +  00 40 00 00 60      441   OTHER:           INCA    DIV1            " OTHER SINGLE CHARACTER
*31 35     +  31 44 00 04 50
 31 45     +  12 05 00 05 30      442   8H:              STA     SYNT UNIT       " THIS SYNTACTICAL UNIT
*31 40     +  31 45 00 00 47
*31 28     +  31 45 00 00 47
 31 46     +  00 66 00 02 67      443                    ENTX    SPACE           " SIMULATE A SPACE
 31 47     +  12 00 00 05 37      444                    STX     CHAR            " INSERT IT
 31 48     -  00 00 00 00 47      445   9H:              JMP     MOD             " RETURN
*31 43     +  31 48 00 00 47
*31 18     +  31 48 00 00 47
*30 42     +  31 48 00 00 47
*29 44     +  31 48 00 02 40
                                  446
                                  447
                                  448   READ AT EXPRESSION:    " RA IS AN OUTPUT PARAMETER FOR THE VALUE OF THE AT EXPR
 31 49                00 02 40   449                    STJ     9F              " SAVE RETURN ADDRESS
 31 50     +  29 44 00 00 47      450                    JMP     READ SYNT UNIT  " READ ONE SYNTACTICAL UNIT
 31 51     +  31 54 00 00 47      451                    JMP     * + 3
 31 52                00 02 40   452   AT EXPRESSION:   STJ     9F              " SAVE RETURN ADDRESS
 31 53     +  12 05 00 05 10      453                    LDA     SYNT UNIT       " LAST SYNTACTICAL UNIT
 31 54     +  01 03 00 01 60      454                    DECA    SYMBOL          " IS IT A SYMBOL ?
 31 55                00 01 50   455                    JAZ     SYMB            " THEN JUMP
 31 56     -  00 01 00 01 60      456                    DECA    NUMBER - SYMBOL " OR IS IT A NUMBER ?
 31 57                00 01 50   457                    JAZ     NUMB            " THEN JUMP
 31 58     -  00 63 00 01 60      458                    DECA    ASTERISK - NUMBER
 31 59                00 01 50   459                    JAZ     ASTSK           " JUMP IF IT IS AN ASTERISK
 31 60     +  00 21 00 02 66      460                    ENT6    21              " REPORT ERROR:
 31 61     +  29 26 00 00 47      461   1H:              JMP     ERRORM          " ATOMIC EXPRESSION REQUIRED
 31 62     +  00 00 00 02 60      462                    ENTA    0               " RESULTING VALUE
 31 63                00 00 47   463                    JMP     8F              " READY
 31 64     +  12 06 00 05 11      464   SYMB:            LD1     LAST SYMBOL     " POINTER TO INFO CELL
*31 55     +  31 64 00 01 50
 31 65     +  12 07 00 05 10      465                    LDA     TYPE            " TYPE OF SYMBOL
 31 66     +  00 01 00 01 60      466                    DECA    DEFINED SYMB    " IS IT A DEFINED SYMBOL ?
 31 67                00 01 50   467                    JAZ     DEFINED         " THEN JUMP
 31 68     +  00 02 00 01 60      468                    DECA    LOCAL BACKWARD - DEFINED SYMB
 31 69                00 01 50   469                    JAZ     LOC DEFINED     " JUMP IF IT IS A LOCAL BACKWARD
 31 70     +  00 23 00 02 66      470                    ENT6    23              " REPORT ERROR:
 31 71     +  31 61 00 00 47      471                    JMP     1B              " UNDEFINED SYMBOL IN EXPRESSION
 31 72     +  13 26 01 05 10      472   DEFINED:         LDA     TABLE - 2 ,1    " VALUE OF SYMBOL
*31 67     +  31 72 00 01 50
 31 73                00 00 47   473                    JMP     8F              " READY
 31 74     +  12 48 01 33 10      474   LOC DEFINED:     LDA     HERE ,1 (3 : 3) " CORRESPONDING LABEL OCCURRED ?
*31 69     +  31 74 00 01 50
 31 75                00 02 50   475                    JAP     2F              " THEN JUMP
 31 76     +  00 22 00 02 66      476                    ENT6    22              " REPORT ERROR:
 31 77     +  31 61 00 00 47      477                    JMP     1B              " UNDEF. LOCAL BACKWARD IN EXPRESSION
```

```
 31 76    +  12 38 01 05 10      478  2H:            LDA   BACKWARD ,1      " VALUE OF LOCAL BACKWARD
*31 75    +  31 78 00 02 50
 31 77                 00 00 47  479                  JMP   8F               " READY
 31 80    +  12 08 00 05 10      480  NUMB:           LDA   VALUE OF NUMBER  " VALUE OF NUMBER
*31 57    +  31 80 00 01 50
 31 81                 00 00 47  481                  JMP   8F               " READY
 31 82    +  12 19 00 05 10      482  ASTSK:          LDA   LC               " VALUE OF LOCATION COUNTER
*31 59    +  31 82 00 01 50
 31 83    +  12 10 00 05 30      483  8H:             STA   AT EXPR          " VALUE OF ATOMIC EXPRESSION
*31 81    +  31 83 00 00 47
*31 79    +  31 83 00 00 47
*31 75    +  31 83 00 00 47
*31 63    +  31 83 00 00 47
 31 84    +  00 00 00 00 47      484  9H:             JMP   MOD              " RETURN
*31 52    +  31 84 00 02 40
*31 49    +  31 84 00 02 40
                                 485
                                 486
                                 487  READ EXPRESSION:" RA IS AN OUTPUT PARAMETER FOR THE VALUE OF THE EXPRESSION
 31 85                 00 02 40  488                  STJ   9F               " SAVE RETURN ADDRESS
 31 86    +  12 11 00 05 41      489                  STZ   EXPR             " INITIALIZE EXPR
 31 87    +  12 05 00 05 10      490                  LDA   SYNT UNIT        " LAST SYNTACTICAL UNIT
 31 88    +  00 37 00 01 60      491                  DECA  PLUS             " IS IT AN ADDING OPERATOR ?
 31 89                 00 01 50  492                  JAZ   ADD              " THEN JUMP
 31 90    +  00 01 00 01 60      493                  DECA  MINUS - PLUS     " IS IT A SUBTRACTING OPERATOR ?
 31 91                 00 01 50  494                  JAZ   SUB              " THEN JUMP
 31 92    +  31 52 00 00 47      495                  JMP   AT EXPRESSION    " EXPECT AN ATOMIC EXPRESSION
 31 93    +  12 11 00 05 30      496  NEXT OPERAND:   STA   EXPR             " SAVE THE VALUE OF IT
 31 94                 00 02 47  497                  JOV   ARITHM OV        " IF OVERFLOW THEN REPORT IT
 31 95    +  29 44 00 00 47      498                  JMP   READ SYNT UNIT   " READ NEXT SYNTACTICAL UNIT
 31 96    +  00 37 00 01 60      499                  DECA  PLUS             " IS IT AN ADDING OPERATOR ?
 31 97                 00 01 50  500                  JAZ   ADD              " THEN JUMP
 31 98    +  00 01 00 01 60      501                  DECA  MINUS - PLUS     " IS IT A SUBTRACTING OPERATOR ?
 31 99                 00 01 50  502                  JAZ   SUB              " THEN JUMP
 32 00    +  00 01 00 01 60      503                  DECA  TIMES - MINUS    " IS IT A MULTIPLICATION OPERATOR ?
 32 01                 00 01 50  504                  JAZ   MUL              " THEN JUMP
 32 02    +  00 01 00 01 60      505                  DECA  DIV1 - TIMES     " IS IT A COMMON DIVISION OPERATOR ?
 32 03                 00 01 50  506                  JAZ   DIV X            " THEN JUMP
 32 04    +  00 64 00 01 60      507                  DECA  DIV2 - DIV1      " IS IT A SPECIAL DIVISION OPERATOR ?
 32 05                 00 01 50  508                  JAZ   DIV A            " THEN JUMP
 32 06    -  00 41 00 01 60      509                  DECA  DEC - DIV2       " IS IT A DECIMAL OPERATOR ?
 32 07                 00 01 50  510                  JAZ   TIMES BASE PLUS  " THEN JUMP
 32 08    +  12 11 00 05 10      511                  LDA   EXPR             " RESULTING VALUE OF EXPRESSION
 32 09    +  00 00 00 00 47      512  9H:             JMP   MOD              " RETURN
*31 83    +  32 09 00 02 40
                                 513  TIMES BASE PLUS:
 32 10    +  12 11 00 15 10      514                  LDA   EXPR (1 : 5)     " ABSOLUTE VALUE OF EXPRESSION
*32 07    +  32 10 00 01 50
 32 11    +  28 35 00 05 03      515                  MUL   BASE             " MULTIPLY BY TEN (BASE)
 32 12    +  12 11 00 15 37      516                  STX   EXPR (1 : 5)     " RESTORE THE VALUE
 32 13    +  28 38 00 05 01      517                  ADD   MAX WORD         " FORCE OVERFLOW IF RA # 0
 32 14    +  31 49 00 00 47      518  ADD:            JMP   READ AT EXPRESSION
*31 97    +  32 14 00 01 50
*31 89    +  32 14 00 01 50
 32 15    +  12 11 00 05 01      519                  ADD   EXPR             " EXECUTE ADDING OPERATION
 32 16    +  31 93 00 00 47      520                  JMP   NEXT OPERAND     " PROCEED WITH NEXT OPERANDS
 32 17    +  31 49 00 00 47      521  SUB:            JMP   READ AT EXPRESSION
*31 99    +  32 17 00 01 50
*31 91    +  32 17 00 01 50
 32 18    +  12 11 00 05 10      522                  LDA   EXPR             ") EXECUTE
```

```
 32 19    +  12 10 00 05 02      523                    SUB     AT EXPR          ") SUBTRACTING OPERATION
 32 20    +  31 93 00 00 47      524                    JMP     NEXT OPERAND     " PROCEED WITH NEXT OPERANDS
 32 21    +  31 49 00 00 47      525  MUL:              JMP     READ AT EXPRESSION
*32 01    +  32 21 00 01 50
 32 22    +  12 11 00 05 03      526                    MUL     EXPR             ") EXECUTE
 32 23    +  12 11 00 05 37      527                    STX     EXPR             ") MULTIPLICATION OPERATION
 32 24    +  31 95 00 01 50      528                    JAZ     NEXT OPERAND + 2 " JUMP IF NO OVERFLOW
 32 25    +  00 31 00 02 66      529  ARITHM OV:        ENT6    31               " REPORT ERROR:
*32 94    +  32 25 00 02 47
 32 26    +  29 26 00 00 47      530                    JMP     ERRORM           " OVERFLOW BY ARITHMETIC OPERATION
 32 27    +  00 00 00 02 60      531                    ENTA    0                " RESULTING VALUE
 32 28    +  31 93 00 00 47      532                    JMP     NEXT OPERAND     " PROCEED WITH NEXT OPERANDS
 32 29    +  31 49 00 00 47      533  DIV X:            JMP     READ AT EXPRESSION
*32 03    +  32 29 00 01 50
 32 30    +  00 00 00 02 60      534                    ENTA    0                " RA = 0
 32 31    +  12 11 00 05 17      535                    LDX     EXPR             " RX = EXPR
 32 32    +  12 10 00 05 04      536                    DIV     AT EXPR          " EXECUTE DIVISION OPERATION
 32 33    +  31 93 00 00 47      537                    JMP     NEXT OPERAND     " PROCEED WITH NEXT OPERANDS
 32 34    +  31 49 00 00 47      538  DIV A:            JMP     READ AT EXPRESSION
*32 05    +  32 34 00 01 50
 32 35    +  12 11 00 05 10      539                    LDA     EXPR             " RA = EXPR
 32 36    +  00 00 00 02 67      540                    ENTX    0                " RX = 0
 32 37    +  12 10 00 05 04      541                    DIV     AT EXPR          " EXECUTE DIVISION OPERATION
 32 38    +  31 93 00 00 47      542                    JMP     NEXT OPERAND     " PROCEED WITH NEXT OPERANDS
                                 543
                                 544
                                 545  READ W VALUE:    " RA IS AN OUTPUT PARAMETER FOR THE VALUE OF THE WORD VALUE
 32 39                00 02 40  546                    STJ     9F               " SAVE RETURN ADDRESS
 32 40    +  12 13 00 05 41      547                    STZ     CONSTANT         " INITIALIZE CONSTANT
 32 41    +  32 43 00 00 47      548                    JMP     * + 2
 32 42    +  29 44 00 00 47      549  NEXT SPEC:       JMP     READ SYNT UNIT   " NEXT SYNTACTICAL UNIT
 32 43    +  31 85 00 00 47      550                    JMP     READ EXPRESSION  " READ AN EXPRESSION
 32 44    +  12 12 00 05 30      551                    STA     VALUE            " VALUE OF EXPRESSION
 32 45    +  12 05 00 05 10      552                    LDA     SYNT UNIT        " NOT BELONGING TO EXPRESSION
 32 46    +  00 71 00 01 60      553                    DECA    PAREN            " IS IT A PAREN ?
 32 47                00 04 50  554                    JANZ    STANDARD         " ELSE STANDARD FIELD SPECIFICATION
 32 48    +  29 44 00 00 47      555                    JMP     READ SYNT UNIT   " NEXT SYNTACTICAL UNIT
 32 49    +  31 85 00 00 47      556                    JMP     READ EXPRESSION  " READ FIELD SPECIFICATION
 32 50    +  00 05 00 03 06      557                    SRAX    5                " RX = FIELD SPECIFICATION, RA = 0
 32 51    +  28 35 00 05 04      558                    DIV     BASE             " RA = LEFT BYTE, RX = RIGHT BYTE
 32 52                00 00 50  559                    JAN     WRONG FIELD      " ERROR IF LEFT BYTE < 0
 32 53                00 05 37  560                    STX     AUX              " SAVE THE RIGHT BYTE
 32 54                00 05 70  561                    CMPA    AUX (1:5)        " LEFT BYTE > RIGHT BYTE ?
 32 55                00 06 47  562                    JG      WRONG FIELD      " THEN ERROR
 32 56    +  00 05 00 01 67      563                    DECX    5                " RIGHT BYTE > 5 ?
 32 57                00 02 57  564                    JXP     WRONG FIELD      " THEN ERROR
 32 58    +  12 12 00 05 10      565                    LDA     VALUE            " VALUE TO BE STORED
 32 59    +  12 11 00 05 11      566                    LD1     EXPR             " FIELD SPECIFICATION
 32 60    +  32 61 00 44 31      567                    ST1     * + 1 (4 : 4)    " MODIFY NEXT INSTRUCTION
 32 61    +  12 13 00 00 30      568                    STA     CONSTANT (MOD)   " STORE VALUE INTO BYTES SPECIFIED
 32 62    +  12 05 00 05 10      569  1H:              LDA     SYNT UNIT        " NOT BELONGING TO EXPRESSION
 32 63    +  00 72 00 01 60      570                    DECA    THESIS           " IS SYNTACTICAL UNIT A THESIS ?
 32 64                00 01 50  571                    JAZ     2F               " THEN O.K.
 32 65    +  00 42 00 02 66      572                    ENT6    42               " REPORT ERROR:
 32 66    +  29 26 00 00 47      573                    JMP     ERRORM           " ) REQUIRED
 32 67                00 00 47  574                    JMP     3F               " PROCEED WITH NEXT BYTES
 32 68    +  00 41 00 02 66      575  WRONG FIELD:     ENT6    41               " REPORT ERROR:
*32 57    +  32 68 00 02 57
*32 99    +  32 68 00 06 47
*32 92    +  32 68 00 00 50
```

```
 32 69    +  29 26 00 00 47      576                  JMP     ERRORM          " ILLEGAL FIELD
 32 70    +  32 62 00 00 47      577                  JMP     1B              " CONTINUE
 32 71    +  12 12 00 05 17      578  STANDARD:       LDX     VALUE           " VALUE TO BE STORED
*32 47    +  32 71 00 04 50
 32 72    +  12 13 00 05 37      579                  STX     CONSTANT        " STORE THE VALUE IN ALL BYTES
 32 73               00 00 47    580                  JMP     3F              " PROCEED WITH NEXT BYTES
 32 74    +  29 44 00 00 47      581  2H:             JMP     READ SYNT UNIT  " FOLLOWING THE THESIS
*32 64    +  32 74 00 01 50
 32 75    +  12 05 00 05 10      582  3H:             LDA     SYNT UNIT       " SYNTACTICAL UNIT
*32 73    +  32 75 00 00 47
*32 67    +  32 75 00 00 47
 32 76    +  00 60 00 01 60      583                  DECA    COMMA           " IS IT A COMMA ?
 32 77    +  32 42 00 01 50      584                  JAZ     NEXT SPEC       " THEN TREAT NEXT SPECIFICATION
 32 78    +  12 13 00 05 10      585                  LDA     CONSTANT        " RESULTING VALUE
 32 79    +  00 00 00 00 47      586  9H:             JMP     MOD             " RETURN
*32 39    +  32 79 00 02 40
                                 587
                                 588
                                 589  READ PROGRAM:   " NO PARAMETERS
 32 80               00 02 40    590                  STJ     9F              " SAVE RETURN ADDRESS
 32 81    +  28 37 00 05 10      591  NEXT INST:      LDA     END MARKER
 32 82    +  12 22 00 05 30      592                  STA     LAST LABEL      " INITIALIZE CHAIN OF LABELS
 32 83    +  29 44 00 00 47      593  NEXT SYMBOL:    JMP     READ SYNT UNIT  " READ POSSIBLE LABEL
 32 84    +  01 03 00 01 60      594                  DECA    SYMBOL          " IS IT A SYMBOL ?
 32 85               00 01 50    595                  JAZ     2F              " THEN JUMP
 32 86    -  00 02 00 01 60      596                  DECA    SEP - SYMBOL    " OR IS IT A SEPARATOR ?
 32 87    +  32 83 00 01 50      597                  JAZ     NEXT SYMBOL     " THEN SKIP IT
 32 88    +  00 51 00 02 66      598                  ENT6    51              " REPORT ERROR:
 32 89    +  29 26 00 00 47      599  1H:             JMP     ERRORM          " SYMBOL REQUIRED
 32 90    +  32 83 00 00 47      600                  JMP     NEXT SYMBOL     " SKIP SYNTACTICAL UNIT
 32 91    +  12 24 00 05 31      601  2H:             ST1     INST            " SAVE POINTER TO THE SYMBOL
*32 85    +  32 91 00 01 50
 32 92    +  29 44 00 00 47      602                  JMP     READ SYNT UNIT  " NEXT SYNTACTICAL UNIT
 32 93    +  00 63 00 01 60      603                  DECA    COLON           " IS IT A COLON ?
 32 94               00 04 50    604                  JANZ    TREAT INSTRUCTION       " ELSE JUMP
 32 95    +  12 07 00 05 10      605                  LDA     TYPE            " TYPE OF LABEL
 32 96               00 01 50    606                  JAZ     UNDEF SYMB      " JUMP IF IT IS AN UNDEF. SYMBOL
 32 97    +  00 02 00 01 60      607                  DECA    LOCAL LABEL     " IS IT A LOCAL LABEL ?
 32 98               00 11 50    608                  JAZ     LOC LAB         " THEN JUMP
 32 99    +  00 52 00 02 66      609                  ENT6    52              " REPORT ERROR:
 33 00    +  32 89 00 00 47      610                  JMP     1B              " ILLEGAL LABEL
 33 01    +  13 26 01 02 66      611  UNDEF SYMB:     ENT6    TABLE - 2 ,1    " POINTER TO FIELD VALUE1
*32 96    +  33 01 00 01 50
 33 02    +  12 02 00 05 10      612                  LDA     LINE NUMBER             ") NUMBER OF THIS LINE
 33 03    +  13 27 01 23 30      613                  STA     TABLE - 1 ,1 (2 : 3)    ") IN INFO CELL
 33 04    +  33 06 00 00 47      614                  JMP     * + 2
 33 05    +  12 48 01 02 66      615  LOC LAB:        ENT6    HERE, 1         " POINTER TO FIELD VALUE1
*32 98    +  33 05 00 01 50
 33 06    +  00 00 06 02 10      616                  LDA     0 ,6 (VALUE1)   " CONTENTS OF FIELD VALUE1
 33 07               00 01 50    617                  JAZ     3F              " JUMP IF INITIAL CONTENTS
 33 08    +  00 53 00 02 66      618                  ENT6    53              " REPORT ERROR:
 33 09    +  32 89 00 00 47      619                  JMP     1B              " (LOCAL) LABEL USED TWICE
 33 10    +  12 22 00 05 10      620  3H:             LDA     LAST LABEL      " CHAIN OF LABELS
*33 07    +  33 10 00 01 50
 33 11    +  00 00 06 02 30      621                  STA     0 ,6 (VALUE1)   ") ADD NEW LABEL TO
 33 12    +  12 22 00 05 31      622                  ST1     LAST LABEL      ") CHAIN OF LABELS
 33 13    +  32 83 00 00 47      623                  JMP     NEXT SYMBOL     " TRY NEXT LABELS
                                 624  TREAT INSTRUCTION:
 33 14    +  12 19 00 05 10      625                  LDA     LC              " LOCATION COUNTER
*32 94    +  33 14 00 04 50
```

```
 33 15     +  12 23 00 05 30      626          STA   EQUIV          " INITIALIZE EQUIV
 33 16     +  12 24 00 05 10      627          LDA   INST           " INSTRUCTION SYMBOL
 33 17     +  28 29 00 05 02      628          SUB   LAST INST      " IS IT A MIX MACHINE INSTRUCTION ?
 33 18                00 00 50   629          JAN   6F             " ELSE JUMP
 33 19     +  12 24 00 05 14      630          LD4   INST           " POINTER TO INSTRUCTION SYMBOL
 33 20                00 00 47   631          JMP   READ MIX MACHINE INST
 33 21     +  12 19 00 05 17      632  4H:     LDX   LC             " LOCATION COUNTER
 33 22                00 00 47   633          JMP   PRODUCE        " PRODUCE CODE
 33 23     +  12 05 00 05 10      634  5H:     LDA   SYNT UNIT      " SYNTACTICAL UNIT
 33 24     +  01 01 00 01 60      635          DECA  SEP            " IS IT A SEPARATOR ?
 33 25                00 01 50   636          JAZ   2F             " THEN O.K.
 33 26     +  00 55 00 02 66      637          ENT6  55             " REPORT ERROR:
 33 27     +  29 26 00 00 47      638          JMP   ERRORM         " SEPARATOR REQUIRED
 33 28     +  29 44 00 00 47      639  1H:     JMP   READ SYNT UNIT " ) SKIP
 33 29     +  01 01 00 01 60      640          DECA  SEP            " ) UNTIL
 33 30     +  33 28 00 04 50      641          JANZ  1B             " ) SEPARATOR
 33 31     +  12 22 00 05 15      642  2H:     LD5   LAST LABEL     " CHAIN OF LABELS
*33 25     +  33 31 00 01 50
 33 32                00 00 47   643          JMP   UPDATE         " UPDATE THIS CHAIN
 33 33     +  32 81 00 00 47      644          JMP   NEXT INST      " PROCEED WITH NEXT INSTRUCTION
 33 34     +  12 24 00 05 10      645  6H:     LDA   INST           " INSTRUCTION SYMBOL
*33 18     +  33 34 00 00 50
 33 35     +  28 34 00 05 02      646          SUB   ORIG           " IS IT ORIG ?
 33 36                00 04 50   647          JANZ  6F             " ELSE TRY OTHER
 33 37     +  32 39 00 00 47      648          JMP   READ W VALUE   " READ THE WORD VALUE
 33 38     +  12 19 00 05 30      649          STA   LC             " NEW VALUE FOR LOCATION COUNTER
 33 39     +  33 23 00 00 47      650          JMP   5B             " GENERAL TREATMENT
 33 40     +  12 24 00 05 10      651  6H:     LDA   INST           " INSTRUCTION SYMBOL
*33 36     +  33 40 00 04 50
 33 41     +  28 33 00 05 02      652          SUB   EQU            " IS IT EQU ?
 33 42                00 04 50   653          JANZ  6F             " ELSE TRY OTHER
 33 43     +  32 39 00 00 47      654          JMP   READ W VALUE   " READ THE WORD VALUE
 33 44     +  12 23 00 05 30      655          STA   EQUIV          " NEW VALUE FOR EQUIV
 33 45     +  33 23 00 00 47      656          JMP   5B             " GENERAL TREATMENT
 33 46     +  12 24 00 05 10      657  6H:     LDA   INST           " INSTRUCTION SYMBOL
*33 42     +  33 46 00 04 50
 33 47     +  28 31 00 05 02      658          SUB   CON            " IS IT CON ?
 33 48                00 04 50   659          JANZ  6F             " ELSE TRY OTHER
 33 49     +  32 39 00 00 47      660          JMP   READ W VALUE   " READ THE WORD VALUE
 33 50     +  33 21 00 00 47      661          JMP   4B             " GENERAL TREATMENT
 33 51     +  12 24 00 05 10      662  6H:     LDA   INST           " INSTRUCTION SYMBOL
*33 48     +  33 51 00 04 50
 33 52     +  28 30 00 05 02      663          SUB   ALF            " IS IT ALF ?
 33 53                00 04 50   664          JANZ  6F             " ELSE TRY OTHER
 33 54     +  28 94 00 00 47      665          JMP   READ 5 CHAR    " READ THE FIVE CHARACTERS
 33 55     +  33 21 00 00 47      666          JMP   4B             " GENERAL TREATMENT
 33 56     +  12 24 00 05 10      667  6H:     LDA   INST           " INSTRUCTION SYMBOL
*33 53     +  33 56 00 04 50
 33 57     +  28 32 00 05 02      668          SUB   END            " IS IT END ?
 33 58                00 04 50   669          JANZ  6F             " ELSE TRY OTHER
 33 59     +  32 39 00 00 47      670          JMP   READ W VALUE   " READ THE WORD VALUE
 33 60     +  12 27 00 05 30      671          STA   START ADDRESS  " VALUE FOR START ADDRESS
 33 61                00 00 47   672          JMP   INSERT CONSTANTS " INSERTION OF CONSTANTS
 33 62     +  12 19 00 05 10      673          LDA   LC             " LOCATION COUNTER (AFTER INSERT)
 33 63     +  12 23 00 05 30      674          STA   EQUIV          " NEW VALUE FOR EQUIV.
 33 64     +  12 22 00 05 15      675          LD5   LAST LABEL     " CHAIN OF LABELS PRECEDING END
 33 65                00 00 47   676          JMP   UPDATE         " UPDATE THIS CHAIN
 33 66     +  00 00 00 00 47      677  9H:     JMP   MOD            " RETURN
*32 80     +  33 66 00 02 40
 33 67     +  00 54 00 02 66      678  6H:     ENT6  54             " REPORT ERROR:
```

```
*33 96    +  33 67 00 04 50
 33 96    +  29 26 00 00 47      679                  JMP     ERRORM          " UNKNOWN INSTRUCTION
 33 97    +  33 23 00 00 47      680                  JMP     5B              " GENERAL TREATMENT
                                 681
                                 682
                                 683  POSSIBLE:       " RX CONTAINS THE ADDRESS,
                                 684                  " RI1 IS A BOOLEAN OUTPUT PARAMETER: RI1 ≠ 0 ≡ POSSIBLE
 33 98                 00 02 40 685                  STJ     9F              " SAVE RETURN ADDRESS
 33 99    +  12 14 00 05 37      686                  STX     ADDRESS         " SAVE THE ADDRESS
 33 9A    +  28 39 00 05 11      687                  LD1     LENGTH OF MEMORY ") LAST ADDRESS OF MEMORY
 33 9B    +  00 00 01 01 67      688                  DECX    0, 1             ") < ADDRESS ?
 33 9C                 00 02 57 689                  JXP     1F              " THEN NOT POSSIBLE
 33 9D    +  00 00 01 00 67      690                  INCX    0, 1            ") ADDRESS
 33 9E                 00 01 67 691                  DECX    END OF ASS      ") ≥ END OF ASSEMBLER ?
 33 9F                 00 03 57 692                  JXNN    8F              " THEN POSSIBLE ADDRESS
 33 A0                 00 00 67 693                  INCX    END OF ASS      ") ADDRESS
 33 A1    +  12 00 00 01 67      694                  DECX    BEGIN OF ASS    ") ≥ BEGIN OF ASSEMBLER ?
 33 A2                 00 03 57 695                  JXNN    1F              " THEN NOT POSSIBLE
 33 A3    +  12 00 00 00 67      696                  INCX    BEGIN OF ASS    " ADDRESS
 33 A4                 00 00 57 697                  JXN     1F              " < 0 ? THEN NOT POSSIBLE
 33 A5    +  12 14 00 05 17      698  8H:             LDX     ADDRESS         " RESTORE ADDRESS INTO RX
*33 A7    +  33 83 00 03 57
 33 A8    +  00 00 00 00 47      699  9H:             JMP     MOD             " RETURN
*33 AC    +  33 84 00 02 40
 33 AD    +  00 00 00 02 61      700  1H:             ENT1    0               " NOT POSSIBLE ADDRESS INDICATION
*33 B2    +  33 85 00 00 57
*33 B5    +  33 85 00 03 57
*33 B8    +  33 85 00 02 57
 33 BB    +  33 83 00 00 47      701                  JMP     8B              " READY
                                 702
                                 703
                                 704  READ MIX MACHINE INST: " R14 CONTAINS A POINTER TO THE INSTRUCTION SYMBOL,
                                 705                  " RA IS AN OUTPUT PARAMETER CONTAINING THE
                                 706                  " VALUE WHICH HAS BEEN ASSEMBLED
 33 BC                 00 02 40 707                  STJ     9F              " SAVE RETURN ADDRESS
*33 C0    +  33 87 00 00 47
 33 C1    +  12 15 00 05 41      708                  STZ     A PART          " INITIALIZE A PART
 33 C2    +  12 16 00 05 41      709                  STZ     I PART          " INITIALIZE I PART
 33 C3    +  13 27 04 44 10      710                  LDA     TABLE - 1 ,4 (4 : 4)
 33 C4    +  12 17 00 05 30      711                  STA     F PART          " STANDARD FIELD INTO F PART
 33 C5    +  13 27 04 55 10      712                  LDA     TABLE - 1, 4 (5 : 5)
 33 C6    +  12 18 00 05 30      713                  STA     INSTRUCTION     " SAVE INSTRUCTION CODE
 33 C7    +  12 05 00 05 10      714                  LDA     SYNT UNIT       " LAST SYNTACTICAL UNIT
 33 C8    +  00 43 00 01 60      715                  DECA    EQUAL SIGN      " IS IT AN EQUAL SIGN ?
 33 C9                 00 01 50 716                  JAZ     LIT CONS        " THEN TREAT LITERAL CONSTANT
 33 CA    +  00 60 00 01 60      717                  DECA    SYMBOL - EQUAL SIGN     " IS IT A SYMBOL ?
 33 CB                 00 01 50 718                  JAZ     FUT SYMB        " THEN TRY FUTURE REFERENCE
 33 CC    -  00 43 00 01 60      719                  DECA    COMMA - SYMBOL  " IS IT A COMMA ?
 34 00                 00 01 50 720                  JAZ     COMMA CHECKED   " THEN TREAT I PART
 34 01    +  00 11 00 01 60      721                  DECA    PAREN - COMMA   " IS IT A PAREN ?
 34 02                 00 01 50 722                  JAZ     PAREN CHECKED   " THEN TREAT F PART
 34 03    +  00 30 00 01 60      723                  DECA    SEP - PAREN     " IS IT A SEPARATOR ?
 34 04                 00 01 50 724                  JAZ     8F              " THEN READY
 34 05    +  31 85 00 00 47      725  COMMON EXPR:    JMP     READ EXPRESSION " READ A PART
 34 06    +  12 15 00 05 30      726                  STA     A PART          " SAVE THE VALUE
 34 07                 00 00 47 727                  JMP     CHECK COMMA.    " PROCEED WITH OTHER PARTS
 34 08    +  29 44 00 00 47      728  LIT CONS:       JMP     READ SYNT UNIT  " NEXT SYNTACTICAL UNIT
*33 96    +  34 08 00 01 50
 34 09    +  32 39 00 00 47      729                  JMP     READ W VALUE    " READ THE WORD VALUE
 34 10    +  12 26 00 05 16      730                  LD6     LIT CONS PTR    " STACK POINTER FOR LITERALS
```

```
 34 11    +  13 28 06 05 30        731                 STA   TABLE ,6          ") STACK
 34 12    +  00 01 00 00 66        732                 INC6  1                 ") THE
 34 13    +  12 26 00 05 36        733                 ST6   LIT CONS PTR      ") WORD VALUE
 34 14    +  28 28 00 05 14        734                 LD4   TABLE PTR1        " TABLE POINTER
 34 15    +  00 00 06 01 64        735                 DEC4  0 ,6              " SPACE AVAILABLE ?
 34 16             00 00 54        736                 J4N   EXIT              " ELSE EXIT
 34 17    +  12 05 00 05 10        737                 LDA   SYNT UNIT         " SYNTACTICAL UNIT
 34 18    +  00 43 00 01 60        738                 DECA  EQUAL SIGN        " IS IT AN EQUAL SIGN ?
 34 19    +  12 25 00 02 66        739                 ENT6  LAST CONSTANT     " CHAIN OF LITERAL CONSTANTS
 34 20             00 02 36        740                 ST6   FUT REF (0 : 2)  " MODIFY INSTRUCTION
 34 21             00 01 50        741                 JAZ   2F                " READ, AND EXTEND CHAIN
 34 22    +  00 61 00 02 66        742                 ENT6  61                " REPORT ERROR:
 34 23    +  29 26 00 00 47        743                 JMP   ERRORM            " = REQUIRED
 34 24             00 00 47        744                 JMP   FUT REF           " EXTEND CHAIN OF FUTURE REFERENCES
 34 25    +  12 07 00 05 10        745  FUT SYMB:      LDA   TYPE              " UNDEFINED SYMBOL ?
*34 26    +  34 25 00 01 50
 34 27             00 04 50        746                 JANZ  LOC               " ELSE TRY LOCAL FORWARD
 34 27    +  13 26 01 02 66        747                 ENT6  TABLE - 2 ,1      " POINTER TO FIELD VALUE2
 34 28             00 00 47        748                 JMP   1F                " TREAT FUTURE REFERENCE
 34 29    +  00 04 00 01 60        749  LOC:           DECA  LOCAL FORWARD     " LOCAL FORWARD OCCURRING ?
*34 26    +  34 29 00 04 50
 34 30    +  34 05 00 04 50        750                 JANZ  COMMON EXPR       " ELSE TRY COMMON EXPRESSION
 34 31    +  12 48 01 02 66        751                 ENT6  HERE ,1           " POINTER TO FIELD VALUE2
 34 32             00 02 36        752  1H:            ST6   FUT REF (0 : 2)  " MODIFY INSTRUCTION
*34 28    +  34 32 00 00 47
 34 33    +  29 44 00 00 47        753  2H:            JMP   READ SYNT UNIT    " NEXT SYNTACTICAL UNIT
*34 21    +  34 33 00 01 50
 34 34    +  00 00 00 02 66        754  FUT REF:       ENT6  MOD               " RESTORE R16
*34 32    +  34 34 00 02 36
*34 24    +  34 34 00 00 47
*34 29    +  34 34 00 02 36
 34 35    +  00 00 06 45 10        755                 LDA   0 ,6 (VALUE2)     ") CHAIN OF FUTURE REFERENCES
 34 36    +  12 15 00 05 30        756                 STA   APART             ") INTO A PART
 34 37    +  00 01 00 02 64        757                 ENT4  1
 34 38    +  12 20 00 05 34        758                 ST4   FUTURE            " NOTE OCCURRENCE OF FUTURE REFERENCE
 34 39    +  12 19 00 05 17        759                 LDX   LC                " LOCATION COUNTER
 34 40    +  33 70 00 00 47        760                 JMP   POSSIBLE          " IS IT A POSSIBLE ADDRESS ?
 34 41             00 05 51        761                 J2NP  CHECK COMMA       " ELSE SKIP ADMINISTRATION
 34 42    +  12 19 00 05 14        762                 LD4   LC                " LOCATION COUNTER
 34 43    +  00 00 04 05 10        763                 LDA   0 ,4              " CONTENTS OF THIS ADDRESS
 34 44             00 04 50        764                 JANZ  CHECK COMMA       " JUMP IF USED ALREADY
 34 45    +  12 21 00 05 10        765                 LDA   DANGEROUS         " DANGEROUS ADDRESS
 34 46    +  00 00 04 01 60        766                 DECA  0 ,4              " EQUAL TO THIS ADDRESS ?
 34 47             00 01 50        767                 JAZ   CHECK COMMA       " THEN SKIP ADMINISTRATION
 34 48    +  00 00 06 45 34        768                 ST4   0 ,6 (VALUE2)     " JOIN THE ADDRESS TO THE CHAIN
 34 49    +  12 05 00 05 10        769  CHECK COMMA:   LDA   SYNT UNIT         " SYNTACTICAL UNIT
*34 47    +  34 49 00 01 50
*34 44    +  34 49 00 04 50
*34 41    +  34 49 00 05 51
*34 07    +  34 49 00 00 47
 34 50    +  00 60 00 01 60        770                 DECA  COMMA             " IS IT A COMMA ?
 34 51             00 04 50        771                 JANZ  CHECK PAREN       " ELSE TRY FIELD SPECIFICATION
 34 52    +  29 44 00 00 47        772  COMMA CHECKED: JMP   READ SYNT UNIT    " NEXT SYNTACTICAL UNIT
*34 08    +  34 52 00 01 50
 34 53    +  31 85 00 00 47        773                 JMP   READ EXPRESSION   " READ I PART
 34 54    +  12 16 00 05 30        774                 STA   I PART            " SAVE THE VALUE
 34 55    +  12 05 00 05 10        775  CHECK PAREN:   LDA   SYNT UNIT         " SYNTACTICAL UNIT
*34 51    +  34 55 00 04 50
 34 56    +  00 71 00 01 60        776                 DECA  PAREN             " IS IT A PAREN ?
 34 57             00 04 50        777                 JANZ  CHECK VALUES      " ELSE JUMP
```

```
 34 58     +  29 44 00 00 47      778  PAREN CHECKED:  JMP     READ SYNT UNIT   " NEXT SYNTACTICAL UNIT
*34 02     +  34 58 00 01 50
 34 59     +  31 85 00 00 47      779                  JMP     READ EXPRESSION " READ F PART
 34 60     +  12 17 00 05 30      780                  STA     F PART          " SAVE THE VALUE
 34 61     +  12 05 00 05 10      781                  LDA     SYNT UNIT       " SYNTACTICAL UNIT
 34 62     +  00 72 00 01 60      782                  DECA    THESIS          " IS IT A THESIS ?
 34 63        00 04 50            783                  JANZ    7F              " ELSE ERROR
 34 64     +  29 44 00 00 47      784                  JMP     READ SYNT UNIT  " NEXT SYNTACTICAL UNIT
 34 65     +  28 37 00 05 10      785  CHECK VALUES:   LDA     MAX ADDRESS     " MAX. VALUE OF A PART
*34 67     +  34 65 00 04 50
 34 66     +  12 15 00 15 70      786                  CMPA    A PART (1 : 5)  " ABS(A PART) O.K. ?
 34 67        00 07 47            787                  JGE     1F              " THEN CHECK I PART
 34 68     +  00 63 00 02 66      788                  ENT6    63              " REPORT ERROR:
 34 69     +  29 26 00 00 47      789                  JMP     ERRORM          " VALUE OF A PART NOT O.K.
 34 70     +  12 15 00 05 41      790                  STZ     A PART          " REPLACE ERRONEOUS A PART
 34 71     +  28 36 00 05 10      791  1H:             LDA     MAX BYTE        " MAX. VALUE OF I PART
*34 67     +  34 71 00 07 47
 34 72     +  12 16 00 05 70      792                  CMPA    I PART          " I PART TOO LARGE ?
 34 73        00 04 47            793                  JL      3F              " THEN ERROR
 34 74     +  12 16 00 05 17      794                  LDX     I PART          " I PART ≥ 0 ?
 34 75        00 03 57            795                  JXNN    1F              " THEN CHECK F PART
 34 76     +  00 64 00 02 66      796  3H:             ENT6    64              " REPORT ERROR:
*34 73     +  34 76 00 04 47
 34 77     +  29 26 00 00 47      797                  JMP     ERRORM          " VALUE OF I PART NOT O.K.
 34 78     +  12 16 00 05 41      798                  STZ     I PART          " REPLACE ERRONEOUS I PART
 34 79     +  28 36 00 05 10      799                  LDA     MAX BYTE        " MAX VALUE OF F PART
 34 80     +  12 17 00 05 70      800  1H:             CMPA    F PART          " F PART TOO LARGE ?
*34 79     +  34 80 00 03 57
 34 81        00 04 47            801                  JL      3F              " THEN ERROR
 34 82     +  12 17 00 05 17      802                  LDX     F PART          " F PART ≥ 0 ?
 34 83        00 03 57            803                  JXNN    8F              " THEN CHECKS O.K.
 34 84     +  00 65 00 02 66      804  3H:             ENT6    65              " REPORT ERROR:
*34 81     +  34 84 00 04 47
 34 85     +  29 26 00 00 47      805                  JMP     ERRORM          " VALUE OF F PART NOT O.K.
 34 86     +  12 17 00 05 41      806                  STZ     F PART          " REPLACE ERRONEOUS F PART
 34 87     -  12 15 00 05 10      807  8H:             LDA     A PART                  ")
*34 83     +  34 87 00 03 57
*34 04     +  34 87 00 01 50
 34 88     +  12 18 00 02 30      808                  STA     INSTRUCTION (0 : 2)     ") ASSEMBLE
 34 89     +  12 16 00 05 10      809                  LDA     I PART                  ") THE
 34 90     +  12 18 00 03 30      810                  STA     INSTRUCTION (3 : 3)     ") WHOLE
 34 91     +  12 17 00 05 10      811                  LDA     F PART                  ") INSTRUCTION
 34 92     +  12 18 00 44 30      812                  STA     INSTRUCTION (4 : 4)     ")
 34 93     +  12 18 00 05 10      813                  LDA     INSTRUCTION     " RESULTING VALUE
 34 94     +  00 00 00 00 47      814  9H:             JMP     MOD             " RETURN
*33 87     +  34 94 00 02 40
 34 95     +  00 62 00 02 66      815  7H:             ENT6    62              " REPORT ERROR:
*34 63     +  34 95 00 04 50
 34 96     +  29 26 00 00 47      816                  JMP     ERRORM          " ) REQUIRED
 34 97     +  34 65 00 00 47      817                  JMP     CHECK VALUES    " CONTINUE
                                  818
                                  819
                                  820  PRODUCE:        " RA CONTAINS THE VALUE TO BE PRODUCED,
                                  821                  " RX CONTAINS THE ADDRESS
 34 98        00 02 40            822                  STJ     9F              " SAVE RETURN ADDRESS
*33 22     +  34 98 00 00 47
 34 99     +  33 70 00 00 47      823                  JMP     POSSIBLE        " POSSIBLE ADDRESS TO BE FILLED IN ?
 35 00        00 02 51            824                  J1P     3F              " JUMP IF POSSIBLE ADDRESS
 35 01     +  00 71 00 02 66      825                  ENT6    71              " REPORT ERROR:
 35 02     +  29 26 00 00 47      826                  JMP     ERRORM          " WRONG ADDRESS
```

```
35 03   +  12 14 00 05 17          LDX    ADDRESS             " IMPOSSIBLE ADDRESS
35 04   +  35 08 00 03 57          JXNN   * + 4               " JUMP IF NOT NEGATIVE
35 05   +  00 38 00 02 64          ENT4   MINUS               ") 'PRINT'
35 06   +  13 04 00 44 34          ST4    OUTBUF (4 : 4)      ") MINUS SIGN
35 07   +  35 10 00 00 47          JMP    * + 3
35 08   +  00 37 00 02 64          ENT4   PLUS                ") 'PRINT'
35 09   +  13 04 00 44 34          ST4    OUTBUF (4 : 4)      ") PLUS SIGN
35 10   +  00 05 00 02 06          SLAX   5                   " RA = IMPOSSIBLE ADDRESS, RX = 0
35 11   +  00 00 00 01 05          CHAR                       " CONVERT TO CHARACTERS
35 12   +  13 05 00 25 30          STA    OUTBUF + 1          ") 'PRINT'
35 13   +  13 06 00 05 37          STX    OUTBUF + 2          ") THEM
35 14            00 00 47          JMP    6F                  " GIVE UP
35 15   +  12 03 00 05 14     3H:  LD4    CODE                ") IF CODE PRODUCED ALREADY
*35 00  +  35 15 00 02 51
35 16   +  29 16 00 02 54          J4P    PRINT LINE          ") THEN PRINT THE LINE
35 17   +  12 14 00 05 11          LD1    ADDRESS             " POSSIBLE ADDRESS
35 18   +  12 19 00 05 17          LDX    LC                  " LOCATION COUNTER
35 19   +  00 00 01 01 67          DECX   0 ,1                " ARE THEY UNEQUAL ?
35 20            00 04 57          JXNZ   4F                  " THEN SKIP TEST
35 21   +  00 00 01 05 17          LDX    0 ,1                " CONTENTS OF ADDRESS, ZERO ?
35 22            00 04 57          JXNZ   1F                  " ELSE ERROR
35 23   +  12 21 00 05 17          LDX    DANGEROUS           ") ADDRESS UNEQUAL
35 24   +  00 00 01 01 67          DECX   0 ,1                ") TO DANGEROUS ONE ?
35 25            00 04 57          JXNZ   4F                  " THEN O.K.
35 26   +  00 72 00 02 66     1H:  ENT6   72                  " REPORT ERROR:
*35 22  +  35 26 00 04 57
35 27   +  29 26 00 00 47          JMP    ERRORM              " ADDRESS ALREADY USED
35 28   +  12 14 00 05 11          LD1    ADDRESS
35 29            00 00 47          JMP    6F                  " PRODUCE OUTPUT ONLY
35 30   +  12 12 00 05 30     4H:  STA    VALUE               " SAVE THE VALUE TO BE PRODUCED
*35 25  +  35 30 00 04 57
*35 20  +  35 30 00 04 57
35 31            00 04 50          JANZ   2F                  " JUMP IF IT IS NOT ZERO
35 32   +  12 20 00 05 17          LDX    FUTURE              " FUTURE REFERENCE TO BE PRODUCED ?
35 33            00 05 57          JXNP   2F                  " ELSE JUMP
35 34   +  12 21 00 05 31          ST1    DANGEROUS           " DANGEROUS FUTURE REFERENCE PRODUCED
35 35   +  00 00 00 01 05     2H:  CHAR                       " CONVERT THE VALUE TO CHARACTERS
*35 33  +  35 35 00 05 57
*35 31  +  35 35 00 04 50
35 36   +  13 07 00 55 30          STA    OUTBUF + 3 (5 : 5)
35 37   +  13 08 00 05 37          STX    OUTBUF + 4          " 'PRINT' SIX DECIMAL POSITIONS
35 38   +  12 20 00 05 14          LD4    FUTURE              " FUTURE REFERENCE ?
35 39            00 02 54          J4P    5F                  " THEN OMIT A PART
35 40   +  13 07 00 05 30          STA    OUTBUF + 3          " 'PRINT' A PART
35 41   +  35 45 00 03 50          JANN   * + 4               " JUMP IF VALUE ≥ 0
35 42   +  00 38 00 02 64          ENT4   MINUS
35 43   +  13 06 00 33 34          ST4    OUTBUF + 2 (3 : 3)      ") 'PRINT' MINUS SIGN
35 44            00 00 47          JMP    5F
35 45   +  00 37 00 02 64          ENT4   PLUS
35 46   +  13 06 00 33 34          ST4    OUTBUF + 2 (3 : 3)      ") 'PRINT' PLUS SIGN
35 47   +  12 12 00 05 10     5H:  LDA    VALUE               " VALUE TO BE PRODUCED
*35 44  +  35 47 00 00 47
*35 39  +  35 47 00 02 54
35 48   +  00 00 01 05 30          STA    0 ,1                " STORE IT INTO MEMORY
35 49   +  00 00 01 02 60     6H:  ENTA   0 ,1                " ADDRESS
*35 29  +  35 49 00 00 47
35 50   +  00 00 00 01 05          CHAR                       " CONVERT TO CHARACTERS
35 51   +  13 04 00 25 37          STX    OUTBUF (2 : 5)      " 'PRINT' IT
35 52   +  12 19 00 05 17          LDX    LC                  " LOCATION COUNTER
35 53   +  00 00 01 01 67          DECX   0 ,1                " UNEQUAL TO ADDRESS ?
```

```
 35 54     +  35 58 00 04 57        878                 JXNZ    * + 4           " THEN JUMP
 35 55     +  00 01 00 00 61        879                 INC1    1               ") INCREASE LOCATION COUNTER
 35 56     +  12 29 00 05 31        880                 ST1     LC              ") BY ONE
 35 57              00 00 47        881                 JMP     8F              " READY
 35 58     +  00 39 00 02 64        882                 ENT4    ASTERISK        " CHARACTER CODE OF ASTERISK
 35 59     +  13 04 00 01 34        883                 ST4     OUTBUF (1 : 1)  " 'PRINT' IT
 35 60     +  12 20 00 05 41        884  8H:            STZ     FUTURE          " RESET FUTURE ON 'FALSE
*35 61     +  35 60 00 00 47
*35 14     +  35 60 00 00 47
 35 61     +  00 01 00 02 61        885                 ENT1    1               ") NOW, CODE
 35 62     +  12 03 00 05 34        886                 ST4     CODE            ") HAS BEEN PRODUCED
 35 63     +  00 00 00 00 47        887  9H:            JMP     *+0             " RETURN
*35 98     +  35 63 00 02 40
                                    888
                                    889
                                    890  UPDATE:         " R5 CONTAINS A POINTER TO THE CHAIN TO BE UPDATED
 35 64              00 02 40        891                 STJ     9F              " SAVE REURN ADDRESS
*35 65     +  35 64 00 00 47
*35 32     +  35 64 00 00 47
 35 65     +  28 37 00 05 75        892  LOOP1:         CMP5    END MARKER      " REMAINING CHAIN EMPTY ?
 35 66     +  00 00 00 05 47        893  9H:            JE      *+0             " THEN RETURN
*35 64     +  35 66 00 02 40
 35 67     +  00 00 05 02 64        894                 ENT4    0 ,5            " REMAINING CHAIN
 35 68              00 05 55        895                 J5NP    1F              " JUMP IF REFERENCE TO LOCAL LABEL
 35 69     +  13 26 04 02 15        896                 LD5     TABLE - 2 ,4 (VALUE1)   " DELETE LAST LABEL
 35 70     +  13 26 04 45 16        897                 LD6     TABLE - 2 ,4 (VALUE2)   " CHAIN OF FUTURE REFERENCES
 35 71     +  00 01 00 02 60        898                 ENTA    DEFINED SYMB            ") SYMBOL
 35 72     +  13 27 04 01 30        899                 STA     TABLE - 1 ,4 (0 : 1)    ") DEFINED NOW
 35 73     +  12 23 00 05 10        900                 LDA     EQUIV           ") VALUE OF
 35 74     +  13 26 04 45 30        901                 STA     TABLE - 2 ,4    ") DEFINED SYMBOL
 35 75              00 00 47        902                 JMP     LOOP2           " UPDATE CHAIN OF FUTURE REFERENCES
 35 76     +  12 48 04 02 15        903  1H:            LD5     HERE ,4 (VALUE1) " DELETE LAST LABEL FROM CHAIN
*35 68     +  35 76 00 05 55
 35 77     +  12 48 04 02 41        904                 STZ     HERE ,4 (VALUE1) " INITIALIZE CHAIN AGAIN
 35 78     +  12 48 04 45 16        905                 LD6     HERE ,4 (VALUE2) " CHAIN OF FUTURE REFERENCES
 35 79     +  28 37 00 05 10        906                 LDA     END MARKER
 35 80     +  12 48 04 45 30        907                 STA     HERE ,4 (VALUE2) " EMPTY AGAIN
 35 81     +  12 23 00 05 10        908                 LDA     EQUIV           ") VALUE OF
 35 82     +  12 38 04 05 30        909                 STA     BACKWARD ,4     ") LOCAL BACKWARD
 35 83     +  00 01 00 02 61        910                 ENT1    DEFINED SYMB    " NOTE OCCURRENCE OF
 35 84     +  12 48 04 33 31        911                 ST1     HERE ,4 (3 : 3) " LOCAL BACKWARD
 35 85     +  28 37 00 05 76        912  LOOP2:         CMP6    END MARKER      " REMAINING CHAIN OF FUTURE REFERENCES
*35 73     +  35 85 00 00 47
 35 86     +  35 65 00 05 47        913                 JE      LOOP1           " IF EMPTY, PROCEED WITH LABEL CHAIN
 35 87     +  12 23 00 15 10        914                 LDA     EQUIV (1 : 5)   ") ABS(EQUIV)
 35 88     +  28 37 00 05 70        915                 CMPA    MAX ADDRESS     ") NOT TOO LARGE ?
 35 89              00 09 47        916                 JLE     1F              " THEN O.K.
 35 90     +  00 81 00 02 66        917                 ENT6    81              " REPORT ERROR:
 35 91     +  29 26 00 00 47        918                 JMP     ERRORM          " VALUE FOR FUTURE REFERENCE TOO LARGE
 35 92     +  35 65 00 00 47        919                 JMP     LOOP1           " CONTINUE WITH LABEL CHAIN
 35 93     +  00 00 00 02 61        920  1H:            ENT1    0 ,6            " ADDRESS TO BE UPDATED
*35 89     +  35 93 00 09 47
 35 94     +  00 00 01 12 16        921                 LD6     0 ,1 (0 : 2)    " NEXT ADDRESS IN CHAIN
 35 95     +  12 23 00 05 10        922                 LDA     EQUIV           ") UPDATE A PART
 35 96     +  00 00 01 02 30        923                 STA     0 ,1 (0 : 2)    ") OF THE ADDRESS
 35 97     +  00 00 01 02 67        924                 ENTX    0 ,1            " ADDRESS IN RX
 35 98     +  00 00 01 05 10        925                 LDA     0 ,1            " NEW CONTENTS IN RA
 35 99     +  34 98 00 00 47        926                 JMP     PRODUCE         " PRODUCE OUTPUT
 36 00     +  35 85 00 00 47        927                 JMP     LOOP2           " CONTINUE WITH FUTURE REFERENCES
                                    928
```

```
                                                  929
                                                  930  TRAVERSE TREE:  " R14 CONTAINS A SUBROUTINE FOR INVESTIGATE
 36 01                    00 02 40                 931                  STJ     9F                 " SAVE RETURN ADDRESS
 36 02                    00 02 34                 932                  ST4     INVESTIGATE (0 : 2)     " ACTUAL SUBROUTINE INVESTIGATE
 36 03    +  14 99 00 02 62                        933                  ENT2    LENGTH OF TABLE - 1     " POINTER TO ROOT OF TREE
 36 04    +  13 28 02 23 14                        934  1H:             LD4     TABLE ,2 (2 : 3) " LEFT BRANCH, EMPTY ?
 36 05                    00 01 54                 935                  J4Z     INVESTIGATE        " THEN JUMP
 36 06    +  00 00 04 02 62                        936                  ENT2    0 ,4               " TRAVERSE LEFT BRANCH
 36 07    +  36 04 00 00 47                        937                  JMP     1B                 " REPEAT
 36 08    +  00 00 00 00 47                        938  INVESTIGATE:    JMP     MOD                " INVESTIGATE THE NODE
*36 05    +  36 08 00 01 54
*36 02    +  36 08 00 02 34
 36 09    +  13 28 02 05 10                        939                  LDA     TABLE ,2           " THREAD ?
 36 10    +  13 28 02 45 12                        940                  LD2     TABLE ,2 (4 : 5) " RIGHT BRANCH
 36 11    +  36 04 00 02 50                        941                  JAP     1B                 " TRAVERSE TO LEFT IF NOT A THREAD
 36 12    +  28 37 00 05 72                        942                  CMP2    END MARKER         " WHOLE TREE TRAVERSED ?
 36 13    +  36 08 00 08 47                        943                  JNE     INVESTIGATE        " ELSE CONTINUE
 36 14    +  00 00 00 00 47                        944  9H:             JMP     MOD                " RETURN
*36 01    +  36 14 00 02 40
                                                  945
                                                  946
                                                  947  BUFFER CHAR:    " RA CONTAINS THE CHARACTER TO BE BUFFERED,
                                                  948                  " R14 AND R15 ARE PERMANENT FILL POINTERS
 36 15                    00 02 40                 949                  STJ     9F                 " SAVE RETURN ADDRESS
 36 16    +  36 17 00 44 35                        950                  ST5     * + 1 (4 : 4)      " MODIFY NEXT INSTRUCTION
 36 17    +  13 04 04 00 30                        951                  ST4     OUTBUF ,4 (MOD) " PRINT THE CHARACTER
 36 18    -  00 55 05 03 65                        952                  ENN5    5 : 5 /N ,5        " BYTE FIVE OF A WORD FILLED ?
 36 19                    00 02 55                 953                  J5P     8F                 " ELSE JUMP
 36 20    +  00 55 00 02 65                        954                  ENT5    5 : 5              ") BYTE 0 OF
 36 21    +  00 01 00 00 64                        955                  INC4    1                  ") NEXT WORD
 36 22    -  00 66 05 03 65                        956  8H:             ENN5    6 : 6 /N ,5        " POINTER TO NEXT BYTE
*36 19    +  36 22 00 02 55
 36 23    +  00 00 00 00 47                        957  9H:             JMP     MOD                " RETURN
*36 15    +  36 23 00 02 40
                                                  958
                                                  959
                                                  960  BUFFER SYMB:    " R12 CONTAINS A POINTER TO THE SYMBOL TO BE BUFFERED,
                                                  961                  " R14 AND R15 CONTAIN THE POSITION FOR THE FIRST LETTER
 36 24                    00 02 40                 962                  STJ     9F                 " SAVE RETURN ADDRESS
 36 25                    00 02 52                 963                  J2P     4F                 " JUMP IF ORDINARY SYMBOL
 36 26    +  00 01 02 03 60                        964                  ENNA    1 ,2               " DIGIT OF LOCAL SYMBOL
 36 27    +  36 15 00 00 47                        965                  JMP     BUFFER CHAR        " 'PRINT' IT
 36 28    +  00 17 00 02 60                        966                  ENTA    LETTER H           " LETTER H
 36 29    +  36 15 00 00 47                        967                  JMP     BUFFER CHAR        " 'PRINT' IT
 36 30    +  00 00 00 00 47                        968  9H:             JMP     MOD                " RETURN
*36 24    +  36 30 00 02 40
 36 31    -  00 03 02 02 61                        969  4H:             ENT1    - 3 ,2             " POINTER TO CHARACTERS
*36 25    +  36 31 00 02 52
 36 32    +  13 28 01 05 17                        970  1H:             LDX     TABLE ,1           " NEXT FIVE CHARACTERS
 36 33    +  00 55 00 02 66                        971                  ENT6    5 : 5              " INITIALIZE BYTE POINTER
 36 34    -  00 66 06 03 66                        972  2H:             ENN6    6 : 6 /N ,6        " POINTER TO NEXT BYTE
 36 35    +  36 36 00 44 36                        973                  ST6     * + 1 (4 : 4)      " MODIFY NEXT INSTRUCTION
 36 36    +  13 28 01 00 10                        974                  LDA     TABLE ,1 (MOD)     " 'READ' NEXT CHARACTER
 36 37    +  36 30 00 01 50                        975                  JAZ     9B                 " IF EMPTY THEN READY
 36 38    +  00 01 00 01 60                        976                  DECA    1                  " CANCEL MODIFICATION OF CHARACTER
 36 39    +  36 15 00 00 47                        977                  JMP     BUFFER CHAR        " 'PRINT' THE CHARACTER
 36 40    -  00 55 06 03 66                        978                  ENN6    5 : 5 /N ,6        " BYTE FIVE 'READ' ?
 36 41    +  36 34 00 02 56                        979                  J6P     2B                 " ELSE LOOP
 36 42    +  36 30 00 00 57                        980                  JXN     9B                 " READY IF LAST WORD
 36 43    +  00 01 00 01 61                        981                  DEC1    1                  " NEXT WORD WITH CHARACTERS
```

```
 0044   +  36 32 00 00 47      982                 JMP     1B              " CONTINUE
                              983
                              984
                              985  BUFFER CONSTANT: " R1 CONTAINS A POINTER TO THE SYMBOL
 0045            00 02 40     986                 STJ     9F              " SAVE RETURN ADDRESS
 0046   +  12 03 00 05 10     987                 LDA     CODE            " CODE PRODUCED ?
 0047   +  12 04 00 05 01     988                 ADD     TEXT            " OR TEXT AVAILABLE ?
 0048   +  29 16 00 02 50     989                 JAP     PRINT LINE      " THEN PRINT THE LINE
 0049   +  00 08 00 02 64     990                 ENT4    8               ") POSITION IN BUFFER
 0050   +  00 11 00 02 65     991                 ENT5    1 : 1           ") FOR SYMBOL
 0051   +  36 24 00 00 47     992                 JMP     BUFFER SYMB     " 'PRINT' THE SYMBOL
 0052   +  00 63 00 02 60     993                 ENTA    COLON
 0053   +  36 15 00 00 47     994                 JMP     BUFFER CHAR     " 'PRINT' A COLON
 0054   +  13 16 00 11 10     995                 LDA     OUTBUF+8 + 4 (1 : 1)    " POSITION FOR FIRST LETTER
 0055   +  00 66 00 01 60     996                 DECA    SPACE           " USED ALREADY ?
 0056   +  29 16 00 04 50     997                 JANZ    PRINT LINE      " THEN PRINT THE LINE
 0057   +  00 12 00 02 64     998                 ENT4    8 + 4           ") POSITION IN BUFFER
 0058   +  00 11 00 02 65     999                 ENT5    1 : 1           ") FOR THIS SYMBOL
 0059            00 02 32    1000                 ST2     1F (0 : 2)      " SAVE POINTER TO SYMBOL
 0060   +  28 31 00 05 12    1001                 LD2     CON             " POINTER TO CONSTANT INSTRUCTION SYMBOL
 0061   +  36 24 00 00 47    1002                 JMP     BUFFER SYMB     " 'PRINT' CONSTANT INSTRUCTION SYMBOL
 0062   +  00 00 00 02 62    1003  1H:            ENT2    MOD             " RESTORE POINTER TO SYMBOL
 0063   +  36 62 00 02 32
 0064   +  00 01 00 00 64    1004                 INC4    1
 0065   +  13 04 04 22 41    1005                 STZ     OUTBUF ,4 (2 : 2)" 'PRINT' WORD VALUE ZERO
 0066   +  00 00 00 00 47    1006  9H:            JMP     MOD             " RETURN
 0067   +  36 65 00 02 40
                              1007
                              1008
 0068            00 02 40    1009  DEFINE:       " R2 CONTAINS A POINTER TO THE SYMBOL TO BE DEFINED
 0069   +  13 27 02 01 10    1010                 STJ     9F              " SAVE RETURN ADDRESS
 0070            00 02 50    1011                 LDA     TABLE - 1 ,2 (0 : 1)    " DEFINED SYMBOL ?
 0071   +  13 26 02 02 10    1012                 JAP     9F              " THEN READY
 0072            00 04 50    1013                 LDA     TABLE - 2 ,2 (VALUE1)   " CONTENTS OF FIELD VALUE1
 0073   +  13 26 02 02 63    1014                 JANZ    9F              " READY IF NOT INITIAL VALUE
                             1015                 ENT3    TABLE - 2 ,2    " POINTER TO FIELD VALUE2
 0074            00 02 40    1016  DEFINE1:      " R3 CONTAINS A POINTER TO FIELD VALUE2
 0075   +  00 00 03 45 10    1017                 STJ     9F              " SAVE RETURN ADDRESS (AGAIN)
 0076   +  28 37 00 05 70    1018                 LDA     0 ,3 (VALUE2)   " CHAIN OF FUTURE REFERENCES
 0077            00 05 47    1019                 CMPA    END MARKER      " EMPTY CHAIN ?
 0078   +  36 45 00 00 47    1020                 JE      9F              " THEN READY
 0079   +  12 19 00 05 17    1021                 JMP     BUFFER CONSTANT " 'PRINT' A CONSTANT INSTRUCTION
 0080   +  12 23 00 05 37    1022                 LDX     LC              " LOCATION COUNTER
 0081   +  00 00 00 02 60    1023                 STX     EQUIV           " NEW VALUE FOR EQUIV
 0082   +  34 98 00 00 47    1024                 ENTA    0               " WORD VALUE OF CONSTANT INSTRUCTION
 0083   +  28 37 00 05 10    1025                 JMP     PRODUCE         " PRODUCE IT
 0084   +  00 00 03 02 30    1026                 LDA     END MARKER      ") SIMULATE OCCURRENCE
 0085   +  00 00 02 02 65    1027                 STA     0 ,3 (VALUE1)   ") AS LABEL (FOR UPDATE)
 0086   +  35 64 00 00 47    1028                 ENT5    0 ,2            " PARAMETER FOR UPDATE
 0087   +  00 00 00 00 47    1029                 JMP     UPDATE          " UPDATE THE CHAIN
*0088   +  36 85 00 05 47    1030  9H:            JMP     MOD             " RETURN
*0089   +  36 85 00 02 40
*0090   +  36 85 00 04 50
*0091   +  36 85 00 02 50
*0092   +  36 85 00 02 40
                             1031
                             1032
                             1033  INSERT CONSTANTS:      " NO PARAMETERS
 0093            00 02 40    1034                 STJ     9F              " SAVE RETURN ADDRESS
```

```
*36 61    +  36 86 00 00 47
 36 87    +  00 01 00 02 60      1035
 36 88    +  12 03 00 05 30      1036
 36 89    -  00 10 00 02 62      1037
 36 90    +  12 48 02 02 63      1038   1H:     ENT2   -10                 " INITIALIZE LOCAL SYMBOL POINTER
 36 91    +  36 72 00 00 47      1039           ENT3   HERE ,2              " POINTER TO FIELD VALUE2
 36 92    +  00 01 00 00 62      1040           JMP    DEFINE1             " DEFINE THE SYMBOL IF NECESSARY
 36 93    +  36 90 00 00 52      1041           INC2   1                   " INCREASE POINTER, READY ?
 36 94    +  36 66 00 02 64      1042           J2N    1B                  " ELSE CONTINUE
 36 95    +  36 01 00 00 47      1043           ENT4   DEFINE              " PARAMETER FOR TRAVERSE TREE
 36 96    +  12 25 00 05 15      1044   2H:     JMP    TRAVERSE TREE       " DEFINE SYMBOLS IF NECESSARY
 36 97    +  28 37 00 05 75      1045           LD5    LAST CONSTANT       " CHAIN OF LITERAL CONSTANTS
 36 98    +  00 00 00 05 47      1046   9H:     CMP5   END MARKER          " IS IT EMPTY ?
*36 99    +  36 98 00 02 40                     JE     MOD                 " THEN RETURN
 36 99    +  00 00 05 02 10      1047
 37 00    +  12 25 00 05 30      1048           LDA    0 ,5 (0 : 2)        ") DELETE LAST CONSTANT
 37 01    +  12 19 00 05 17      1049           STA    LAST CONSTANT       ") FROM CHAIN
 37 02    +  00 00 05 02 37      1050           LDX    LC                  " LOCATION COUNTER
 37 03    +  12 26 00 05 16      1051           STX    0 ,5 (0 : 2)        " UPDATE A PART OF ADDRESS
 37 04    +  00 01 00 01 66      1052           LD6    LIT CONS PTR        ") UNSTACK
 37 05    +  12 26 00 05 36      1053           DEC6   1                   ") VALUE FOR
 37 06    +  13 28 06 05 10      1054           ST6    LIT CONS PTR        ") LITERAL CONSTANT
 37 07    +  34 98 00 00 47      1055           LDA    TABLE ,6            " VALUE OF IT
 37 08    +  00 00 05 02 67      1056           JMP    PRODUCE             " PRODUCE THE INSERTED WORD VALUE
 37 09    +  00 00 05 05 10      1057           ENTX   0 ,5                " UPDATED ADDRESS
 37 10    +  34 98 00 00 47      1058           LDA    0 ,5                " NEW CONTENTS OF IT
 37 11    +  36 96 00 00 47      1059           JMP    PRODUCE             " PRODUCE OUTPUT
                                 1060           JMP    2B                  " CONTINUE
                                 1061
                                 1062   BUFFER LN:      " R4 CONTAINS THE LINE NUMBER
 37 12                00 02 40  1063                   STJ    9F                  " SAVE RETURN ADDRESS
*28 00    +  37 12 00 00 47
 37 13    +  00 00 00 01 05      1064                   CHAR                       " CONVERT LINE NUMBER TO CHARACTERS
 37 14    +  13 10 00 25 37      1065                   STX    OUTBUF+8 - 2 (2 : 5)    " STORE THEM INTO BUFFER
 37 15    +  00 66 00 02 60      1066                   ENTA   SPACE               " INTERNAL CODE OF A SPACE
 37 16    +  13 10 00 22 17      1067                   LDX    OUTBUF+8 - 2 (2 : 2)    " FIRST DECIMAL
 37 17                00 04 57  1068                   JXNZ   9F                  " READY IF NOT A ZERO
 37 18    +  13 10 00 22 30      1069                   STA    OUTBUF+8 - 2 (2 : 2)    " REPLACE THE ZERO BY A SPACE
 37 19    +  13 10 00 03 17      1070                   LDX    OUTBUF+8 - 2 (3 : 3)    " SECOND DECIMAL
 37 20                00 04 57  1071                   JXNZ   9F                  " READY IF NOT A ZERO
 37 21    +  13 10 00 33 30      1072                   STA    OUTBUF+8 - 2 (3 : 3)    " REPLACE THE ZERO BY A SPACE
 37 22    +  13 10 00 44 17      1073                   LDX    OUTBUF+8 - 2 (4 : 4)    " THIRD DECIMAL
 37 23                00 04 57  1074                   JXNZ   9F                  " READY IF NOT A ZERO
 37 24    +  13 10 00 44 30      1075                   STA    OUTBUF+8 - 2 (4 : 4)    " REPLACE THE ZERO BY A SPACE
 37 25    +  00 00 00 00 47      1076   9H:             JMP    MOD                 " RETURN
*37 25    +  37 25 00 04 57
*37 26    +  37 25 00 04 57
*37 17    +  37 25 00 04 57
*37 12    +  37 25 00 02 40
                                 1077
                                 1078
                                 1079   DUMP:           " R12 CONTAINS A POINTER TO THE SYMBOL
 37 26                00 02 40  1080                   STJ    9F                  " SAVE RETURN ADDRESS
 37 27    +  13 27 02 01 10      1081                   LDA    TABLE - 1 ,2 (0 : 1)
 37 28                00 01 50  1082                   JAZ    9F                  " READY IF SYMBOL UNDEFINED YET
 37 29    +  13 26 02 05 10      1083                   LDA    TABLE - 2 ,2        " VALUE OF SYMBOL
 37 30    +  00 00 00 01 05      1084                   CHAR                       ") 'PRINT'
 37 31    +  13 07 00 05 30      1085                   STA    OUTBUF+8 - 5        ") THIS
 37 32    +  13 08 00 05 37      1086                   STX    OUTBUF+8 - 4        ") VALUE
 37 33                00 00 50  1087                   JAN    1F                  " JUMP IF THE VALUE IS NEGATIVE
```

```
 37 54    +  00 37 00 02 60    1088                 ENTA   PLUS              ") 'PRINT' PLUS SIGN
 37 55    +  13 06 00 44 30    1089                 STA    OUTBUF+8 - 6 (4 : 4)
 37 56                 00 00 47    1090                 JMP    2F
 37 57    +  00 38 00 02 60    1091    1H:          ENTA   MINUS             ") 'PRINT' MINUS SIGN
*37 60    +  37 37 00 00 50
 37 61    +  13 06 00 44 30    1092                 STA    OUTBUF+8 - 6 (4 : 4)
 37 62    +  13 27 02 23 10    1093    2H:          LDA    TABLE - 1 ,2 (2 : 3)    " LINE OF DEFINING OCCURRENCE
*37 63    +  37 39 00 00 47
 37 64                 00 04 50    1094                 JANZ   3F                " JUMP IF IT HAS BEEN OCCURRED
 37 65    +  00 37 00 02 60    1095                 ENTA   PLUS                     ") 'PRINT' A PLUS SIGN
 37 66    +  13 11 00 11 30    1096                 STA    OUTBUF+8 - 1 (1 : 1)     ") INSTEAD OF LINE NUMBER
 37 67                 00 00 47    1097                 JMP    4F
 37 70    +  37 12 00 00 47    1098    3H:          JMP    BUFFER LN         " 'PRINT' THE LINE NUMBER
-37 71    +  37 44 00 04 50
 37 72    +  00 08 00 02 64    1099    4H:          ENT4   8                 ") POSITION IN BUFFER
*37 73    +  37 45 00 00 47
 37 74    +  00 11 00 02 65    1100                 ENT5   1 : 1             ") FOR SYMBOL
 37 75    +  36 24 00 00 47    1101                 JMP    BUFFER SYMB       " 'PRINT' THE SYMBOL
 37 76    +  29 16 00 00 47    1102                 JMP    PRINT LINE        " PRINT THE LINE
 37 77    +  00 00 00 00 47    1103    9H:          JMP    MOD               " RETURN
*37 20    +  37 49 00 01 50
*37 21    +  37 49 00 02 40
                              1104
                              1105
                              1106    INITIALIZE:  " NO PARAMETERS
 37 22                 00 02 40    1107                 STJ    9F                " SAVE RETURN ADDRESS
 37 23    +  28 28 00 05 11    1108                 LD1    TABLE PTR1        ") FIRST ADDRESS PRECEDING RELEVANT
 37 24    +  13 28 01 02 61    1109                 ENT1   TABLE ,1          ") INFORMATION FOR ASSEMBLER
 37 25                 00 00 51    1110    1H:          J1N    2F                " JUMP IF ADDRESS ZERO TREATED
 37 26    +  00 00 01 05 41    1111                 STZ    0 ,1              " INITIALIZE ADDRESS ON ZERO
 37 27    +  00 01 00 01 61    1112                 DEC1   1                 " NEXT ADDRESS
 37 30    +  37 53 00 00 47    1113                 JMP    1B                " CONTINUE
 37 31    +  28 37 00 05 20    1114    2H:          LDAN   END MARKER
*37 32    +  37 57 00 00 51
 37 33    +  12 21 00 05 30    1115                 STA    DANGEROUS         " INITIALIZE DANGEROUS
 37 34    +  28 37 00 05 10    1116                 LDA    END MARKER
 37 35    +  12 25 00 05 30    1117                 STA    LAST CONSTANT     " INITIALIZE LAST CONSTANT
 37 36    +  12 36 00 05 30    1118                 STA    HERE - 10         ") INITIALIZE
 37 37    +  12 39 00 02 61    1119                 ENT1   HERE - 9          ") ARRAYS 'H' AND 'F'
 37 40    +  12 38 00 09 07    1120                 MOVE   HERE - 10 (9)
 37 41    +  12 02 00 05 41    1121    INIT READING: STZ   LINE NUMBER        " INITIALIZE LINE NUMBER
 37 42    +  13 04 00 02 61    1122                 ENT1   OUTBUF
 37 43    +  28 41 00 01 07    1123                 MOVE   SPACES (1)        ") INITIALIZE
 37 44    +  13 04 00 23 07    1124                 MOVE   OUTBUF (23)       ") OUTPUT BUFFER
 37 45    +  12 48 00 16 44    1125                 IN     INBUF X (READER)  " READ THE FIRST CARD
 37 46    +  00 01 00 02 62    1126                 ENT2   1                 ") INITIALIZE
 37 47    +  00 55 00 02 63    1127                 ENT3   5 : 5             ") READING
 37 50    +  00 66 00 02 60    1128                 ENTA   SPACE             ") POINTERS
 37 51    +  12 00 00 05 30    1129                 STA    CHAR              ")
 37 52    +  00 00 00 00 47    1130    9H:          JMP    MOD               " RETURN
*37 53    +  37 73 00 02 40
                              1131
                              1132
                              1133    INIT TABLE:  " NO PARAMETERS
 37 54                 00 02 40    1134                 STJ    9F                " SAVE RETURN ADDRESS
 37 55                 00 02 61    1135                 ENT1   1F                ") RETURN ADDRES AFTER INIT READING
 37 56    +  37 73 00 02 31    1136                 ST1    9B (0 : 2)        ") INTO RETURN OF INIT READING
 37 57    +  37 64 00 00 47    1137                 JMP    INIT READING      " INITIALIZE READING POINTERS
 37 60    +  14 99 00 02 61    1138    1H:          ENT1   LENGTH OF TABLE - 1     " POINTER TO ROOT OF TREE
*37 61    +  37 78 00 02 61
```

```
 37 79    +  28 28 00 05 31      1139                    ST1     TABLE PTR1        " INITIALIZE TABLE POINTER
 37 80    +  29 44 00 00 47      1140  OPERS:            JMP     READ SYNT UNIT    " NEXT SYNTACTICAL UNIT
 37 81    +  01 03 00 01 60      1141                    DECA    SYMBOL            " IS IT A SYMBOL ?
 37 82             00 04 50      1142                    JANZ    2F                " ELSE TRY SEPARATOR
 37 83    +  31 49 00 00 47      1143                    JMP     READ AT EXPRESSION      " READ THE NUMBER
 37 84    +  12 06 00 05 11      1144                    LD1     LAST SYMBOL       " POINTER TO INSTRUCTION SYMBOL
 37 85    +  13 27 01 45 30      1145                    STA     TABLE - 1 ,1 (4 : 5)    " STORE STANDARD FIELD AND CODE
 37 86    +  29 44 00 00 47      1146                    JMP     READ SYNT UNIT    " NEXT SYNTACTICAL UNIT
 37 87    +  01 03 00 01 60      1147                    DECA    SYMBOL            ")
 37 88    -  00 02 00 01 60      1148  2H:               DECA    SEP - SYMBOL      ") IS IT A SEPARATOR ?
*37 82    +  37 88 00 04 50
 37 89    +  37 91 00 02 52      1149                    J2P     * + 2             " JUMP IF NOT A SEMICOLON
 37 90    +  12 04 00 05 41      1150                    STZ     TEXT              " PREVENT NEW LINE
 37 91    +  37 80 00 01 50      1151                    JAZ     OPERS             " CONTINUE IF SEPARATOR READ
 37 92    -  00 30 00 01 60      1152                    DECA    PAREN - SEP       " IS SYNTACTICAL UNIT A PAREN ?
 37 93             00 01 50      1153                    JAZ     PSEUDO INSTS      " THEN PROCEED WITH PSEUDO INSTRUCTIONS
 37 94    -  00 01 00 02 66      1154                    ENT6    01                " REPORT ERROR:
 37 95    +  29 26 00 00 47      1155                    JMP     ERRORM            " INSTRUCTION DEFINITIONS NOT O.K.
 37 96    +  00 00 00 02 05      1156                    HLT                       " STOP THE MACHINE
 37 97    +  12 06 00 05 11      1157  PSEUDO INSTS:     LD1     LAST SYMBOL       " LAST INSTRUCTION SYMBOL
*37 93    +  37 97 00 01 50
 37 98    +  28 29 00 05 31      1158                    ST1     LAST INST         " INITIALIZE LAST INST
 37 99    +  29 44 00 00 47      1159                    JMP     READ SYNT UNIT
 38 00    +  28 30 00 05 31      1160                    ST1     ALF               " INITIALIZE ALF
 38 01    +  29 44 00 00 47      1161                    JMP     READ SYNT UNIT
 38 02    +  28 31 00 05 31      1162                    ST1     CON               " INITIALIZE CON
 38 03    +  29 44 00 00 47      1163                    JMP     READ SYNT UNIT
 38 04    +  28 32 00 05 31      1164                    ST1     END               " INITIALIZE END
 38 05    +  29 44 00 00 47      1165                    JMP     READ SYNT UNIT
 38 06    +  28 33 00 05 31      1166                    ST1     EQU               " INITIALIZE EQU
 38 07    +  29 44 00 00 47      1167                    JMP     READ SYNT UNIT
 38 08    +  28 34 00 05 31      1168                    ST1     ORIG              " INITIALIZE ORIG
 38 09    +  29 44 00 00 47      1169                    JMP     READ SYNT UNIT    " NEXT SYNTACTICAL UNIT
 38 10    +  00 72 00 01 60      1170                    DECA    THESIS            " IS IT A THESIS ?
 38 11    +  29 16 00 00 47      1171                    JMP     PRINT LINE
 38 12    +  00 00 00 01 50      1172  9H:               JAZ     MOD               " THEN RETURN
*37 74    +  38 12 00 02 40
 38 13    +  00 03 00 02 66      1173                    ENT6    03                " REPORT ERROR:
 38 14    +  29 26 00 00 47      1174                    JMP     ERRORM            " PSEUDO INSTR. DEFINITIONS NOT O.K.
 38 15    +  00 00 00 02 05      1175                    HLT                       " STOP THE MACHINE
                                 1176
                                 1177
                                 1178  RESET TABLE:      " NO PARAMETERS
 38 16             00 02 40      1179                    STJ     9F                " SAVE RETURN ADDRESS
*28 55    +  38 16 00 00 47
 38 17    +  28 34 00 05 12      1180                    LD2     ORIG              " LAST (PSEUDO) INSTRUCTION SYMBOL
 38 18    +  00 03 00 01 62      1181                    DEC2    3                 " POINTER TO CHARACTERS
 38 19    +  13 28 02 05 10      1182  3H:               LDA     TABLE ,2          " LAST WORD WITH CHARACTERS ?
 38 20    +  00 01 00 01 62      1183                    DEC2    1                 ") ELSE SEARCH FOR
 38 21    +  38 19 00 02 50      1184                    JAP     3B                ") THE LAST ONE
 38 22    +  28 28 00 05 32      1185                    ST2     TABLE PTR1        " NEW FREE SPACE
 38 23    +  14 99 00 02 62      1186                    ENT2    LENGTH OF TABLE - 1     " ROOT OF TREE
 38 24    +  13 28 02 23 14      1187  4H:               LD4     TABLE ,2 (2 : 3)  " LEFT BRANCH, EMPTY ?
 38 25             00 01 54      1188                    J4Z     VISIT             " THEN VISIT NODE TO BE SAVED
 38 26    +  28 28 00 05 74      1189                    CMP4    TABLE PTR1        " THIS NODE TO BE DELETED ?
 38 27             00 09 47      1190                    JLE     DELETE            " THEN DELETE IT
 38 28    +  00 00 04 02 62      1191                    ENT2    0 ,4              ") TRAVERSE LEFT BRANCH
 38 29    +  38 24 00 00 47      1192                    JMP     4B                ") AND CONTINUE
 38 30    +  13 28 02 23 41      1193  DELETE:           STZ     TABLE ,2 (2 : 3)  " EMPTY LEFT BRANCHE NOW
*38 27    +  38 30 00 09 47
```

```
 38 31     +  00 00 02 02 63      1194  VISIT:      ENT3   0 ,2              " POINTER TO SAVED NODE
*38 29     +  38 31 00 01 54
 38 32     +  13 28 03 05 10      1195                LDA    TABLE ,3          ") THREAD ? OR
 38 33     +  13 28 03 45 12      1196                LD2    TABLE ,3 (4 : 5)") RIGHT BRANCH ?
 38 34     +  13 27 03 03 41      1197                STZ    TABLE - 1 ,3 (0 : 3)    " RESET TYPE AND LINK NUMBER
 38 35     +  28 37 00 05 17      1198                LDX    END MARKER        ") RESET THE FIELDS
 38 36     +  13 26 03 05 37      1199                STX    TABLE - 2 ,3      ") VALUE1 AND VALUE2
 38 37              00 00 50      1200                JAN    CMP END MARKER    " JUMP IF THREAD
 38 38     +  28 28 00 05 72      1201                CMP2   TABLE PTR1        " RIGHT BRANCH
 38 39     +  38 24 00 06 47      1202                JG     4B                " JUMP IF RIGHT NODE TO BE SAVED
 38 40     +  13 28 02 05 10      1203  5H:           LDA    TABLE ,2          ") THREAD ? OR
 38 41     +  13 28 02 45 12      1204                LD2    TABLE ,2 (4 : 5)") RIGHT BRANCH ?
 38 42     +  38 40 00 02 50      1205                JAP    5B                " ELSE SEARCH FOR THE THREAD
 38 43     +  00 00 02 03 60      1206                ENNA   0 ,2              " TAKE THE THREAD (NEGATIVE)
 38 44     +  13 28 03 45 30      1207                STA    TABLE ,3 (4 : 5)  " STORE THE THREAD IN SAVED NODE
 38 45     +  13 28 03 00 30      1208                STA    TABLE ,3 (0 : 0)  " THREAD INDICATION
 38 46     +  28 37 00 05 72      1209  CMP END MARKER: CMP2   END MARKER        " IS THIS THREAD THE LAST ONE ?
*38 37     +  38 46 00 00 50
 38 47     +  38 31 00 08 47      1210                JNE    VISIT             " ELSE CONTINUE
 38 48     +  00 00 00 00 47      1211  9H:           JMP    MOD               " RETURN
*38 10     +  38 48 00 02 40
                                  1212
                                  1213
                                  1214  MIXAL ASSEMBLER:      " RA IS A BOOLEAN OUTPUT PARAMETER: RA = - ERRONEOUS,
                                  1215                        " R1 IS AN OUTPUT PARAMETER FOR THE START ADDRESS
 38 49              00 02 40      1216                STJ    9F                " SAVE RETURN ADDRESS
 38 50     +  37 50 00 00 47      1217                JMP    INITIALIZE        " INITIALIZE THE MEMORY
 38 51     +  00 00 00 18 43      1218                IOC    0 (PRINTER)       " NEW PAGE
 38 52     +  32 80 00 00 47      1219                JMP    READ PROGRAM      " ASSEMBLE THE PROGRAM
 38 53     +  29 16 00 00 47      1220                JMP    PRINT LINE        " PRINT THE LAST LINE
 38 54     +  12 27 00 05 17      1221                LDX    START ADDRESS     " START ADDRESS
 38 55     +  33 70 00 00 47      1222                JMP    POSSIBLE          " IS IT A POSSIBLE ADDRESS ?
 38 55              00 02 51      1223                J1P    1F                " THEN JUMP
 38 57     +  00 91 00 02 66      1224                ENT6   91                " REPORT ERROR:
 38 58     +  29 26 00 00 47      1225                JMP    ERRORM            " WRONG START ADDRESS
 38 59     +  12 09 00 05 10      1226  1H:           LDA    ERRONEOUS         " ERROR RECOVERED ?
*38 56     +  38 59 00 02 51
 38 60              00 01 50      1227                JAZ    2F                " ELSE JUMP
 38 61     +  00 92 00 02 66      1228                ENT6   92                " REPORT ERROR:
 38 62     +  29 26 00 00 47      1229                JMP    ERRORM            " PROGRAM INCORRECT
 38 63     +  00 00 00 18 43      1230  2H:           IOC    0 (PRINTER)       " NEW PAGE
*38 60     +  38 63 00 01 50
 38 64     +  37 26 00 02 64      1231                ENT4   DUMP              " PARAMETER FOR TRAVERSE TREE
 38 65     +  36 01 00 00 47      1232                JMP    TRAVERSE TREE     " DUMP THE SYMBOL TABLE
 38 66     +  29 16 00 00 47      1233                JMP    PRINT LINE
 38 67     +  38 16 00 00 47      1234                JMP    RESET TABLE       " INITIALIZE SYMBOL TABLE AGAIN
 38 68     +  12 09 00 05 20      1235                LDAN   ERRONEOUS         " COMPLEMENT OF ERRONEOUS IN RA
 38 69     +  12 27 00 45 11      1236                LD1    START ADDRESS (4 : 5)    " START ADDRESS IN R1
 38 70     +  00 00 00 00 47      1237  9H:           JMP    MOD               " RETURN
*38 49     +  38 70 00 02 40
 38 71     +  00 99 00 02 66      1238  EXIT:       ENT6   99                " REPORT ERROR:
*38 15     +  38 71 00 00 54
*29 81     +  38 71 00 02 56
*29 58     +  38 71 00 02 56
 38 72     +  29 26 00 00 47      1239                JMP    ERRORM            " OVERFLOW SYMBOL TABLE
 38 73     +  28 45 00 00 47      1240                JMP    READ CHAR         ") SKIP (BUT PRINT)
 38 74     +  38 73 00 00 47      1241                JMP    * - 1             ") REST OF INPUT
                                  1242
                                  1243
                                  1244
```

```
                                    1245  "   NEXT FOLLOWS THE PIECE OF PROGRAM, SERVING AS A VERY SIMPLE MONITORING
                                    1246  " PROGRAM, ACCEPTING SEVERAL MIXAL PROGRAMS, BEING SEPARATED FROM EACH OTHER
                                    1247  " BY A JOB SEPARATOR ("" IN THE COLUMNS 79 AND 80).
                                    1248  " THESE MIXAL PROGRAMS ARE TRANSLATED AND EXECUTED (IF CORRECT) CONSECUTIVELY.
                                    1249  " THE JOB SEPARATOR IS DETECTED BY THE ROUTINE "READ CHAR" ONLY; SO DURING
                                    1250  " EXECUTION OF A PROGRAM, THE JOB SEPARATOR WILL BE IGNORED.
                                    1251  "   AS THE ASSEMBLER ALWAYS READS ONE CARD IN ADVANCE, THE CARD SUCCEEDING
                                    1252  " THE END INSTRUCTION IS READ TOO BY THE ROUTINE "MIXAL ASSEMBLER".
                                    1253  " THIS FACT HAS TO BE TAKEN INTO ACCOUNT WHEN IN - INSTRUCTIONS ARE PERFORMED
                                    1254  " BY A PARTICULAR MIXAL PROGRAM.
                                    1255  "   THE MONITOR MAKES USE OF THE AUXILIARY ROUTINES "SKIP INPUT" AND
                                    1256  " "PUNCH TRAJECT".
                                    1257
                                    1258
38 75     +  28 28 00 05 12         1259  START0:         LD2     TABLE PTR1
38 76     +  13 29 02 02 62         1260                  ENT2    TABLE + 1 ,2     " START ADDRESS OF TRAJECT
38 77     +  00 00 02 02 63         1261                  ENT3    0 ,2             ") LENGTH OF
38 78              00 01 63         1262                  DEC3    END OF ASS       ") TRAJECT (NEGATIVE)
38 79              00 00 47         1263                  JMP     PUNCH TRAJECT    " PUNCH PERMANENT OF ASSEMBLER
38 80     +  37 74 00 00 47         1264  START1:         JMP     INIT TABLE       " INITIALIZE THE SYMBOL TABLE
38 81     +  38 49 00 00 47         1265  NEXT PROGRAM:   JMP     MIXAL ASSEMBLER  " TRANSLATE THE PROGRAM
*28 25    +  38 81 00 00 47
38 82     +  00 00 00 18 43         1266                  IOC     0 (PRINTER)      " NEW PAGE
38 83     +  38 83 00 16 42         1267                  JBUS    * (READER)       " SKIP THE FIRST INPUT CARD
38 84     +  00 00 01 03 50         1268                  JANN    0 ,1             " EXECUTE THE PROGRAM IF CORRECT
38 85              00 00 47         1269                  JMP     SKIP INPUT       " SKIP REST OF INPUT OF PROGRAM
38 86     +  38 81 00 00 47         1270                  JMP     NEXT PROGRAM     " TREAT THE NEXT PROGRAM
                                    1271
                                    1272
                                    1273  SKIP INPUT:     " NO PARAMETERS
38 87              00 02 40         1274                  STJ     9F               " SAVE RETURN ADDRESS
*38 87    +  38 87 00 00 47
38 88     +  12 64 00 16 44         1275  1H:             IN      INBUF (READER)   " READ NEXT CARD
38 89     +  38 89 00 16 42         1276                  JBUS    * (READER)       " WAIT FOR CARD READER
38 90     +  12 79 00 15 10         1277                  LDA     INBUF + 15       " POSSIBLE JOB SEPARATOR
38 91     +  28 42 00 45 70         1278                  CMPA    JOB SEPARATOR (4 : 5)   " IS IT A JOB SEPARATOR ?
38 92     +  38 88 00 08 47         1279                  JNE     1B               " ELSE CONTINUE
38 93     +  00 00 00 00 47         1280  9H:             JMP     MOD              " RETURN
*38 87    +  38 93 00 02 40
                                    1281
                                    1282
                                    1283  PUNCH TRAJECT:  " R2 CONTAINS THE START ADDRESS,
                                    1284                  " R3 CONTAINS THE LENGTH OF THE TRAJECT (NEGATIVE)
38 94              00 02 40         1285                  STJ     9F               " SAVE RETURN ADDRESS
*38 79    +  38 94 00 00 47
38 95     +  13 03 00 02 61         1286                  ENT1    OUTBUF - 1       ")
38 96     +  28 41 00 01 07         1287                  MOVE    SPACES (1)       ") INITIALIZE
38 97     +  13 03 00 16 07         1288                  MOVE    OUTBUF - 1 (16)  ") OUTPUT BUFFER
38 98     +  00 37 00 02 65         1289                  ENT5    PLUS             " CODE OF PLUS SIGN
38 99     +  00 38 00 02 66         1290                  ENT6    MINUS            " CODE OF MINUS SIGN
39 00     +  00 00 02 02 60         1291                  ENTA    0 ,2             " START ADDRESS
39 01     +  00 00 00 01 05         1292                  CHAR                     " CONVERT IT TO CHARACTERS
39 02     +  13 04 00 15 37         1293                  STX     OUTBUF           " STORE THEM INTO THE OUTPUT BUFFER
39 03     +  00 00 03 03 60         1294                  ENNA    0 ,3             " LENGTH OF TRAJECT (POSITIVE)
39 04     +  00 00 00 01 05         1295                  CHAR                     " CONVERT IT TO CHARACTERS
39 05     +  13 06 00 15 37         1296                  STX     OUTBUF + 2       " STORE THEM INTO THE OUTPUT BUFFER
39 06     +  00 00 03 01 62         1297                  DEC2    0 ,3             " START ADDRESS + LENGTH OF TRAJECT
39 07              00 02 32         1298                  ST2     LOAD (0 : 2)     " MODIFY INSTRUCTION
39 08     +  13 04 00 17 45         1299  OUT:            OUT     OUTBUF (CPUNCH)  " PUNCH A CARD
39 09     +  39 09 00 17 42         1300                  JBUS    * (CPUNCH)       " WAIT FOR CARD PUNCH
```

```
 39 10    +  00 00 00 01 53      1301   9H:        J3Z     MOD             " RETURN IF TRAJECT COMPLETED
*38 94    +  39 10 00 02 40
 39 11    +  13 04 00 02 61      1302               ENT1    OUTBUF          ") INITIALIZE
 39 12    +  13 03 00 16 07      1303               MOVE    OUTBUF - 1 (16) ") OUTPUT BUFFER AGAIN
 39 13    +  00 15 00 03 64      1304               ENN4    15              " INITIALIZE POSITION ON CARD
 39 14    +  00 00 03 05 10      1305   LOAD:       LDA     MOD ,3          " NEXT WORD
*39 07    +  39 14 00 02 32
 39 15               00 00 50   1306               JAN     1F              " JUMP IF NEGATIVE
 39 16    +  13 19 04 33 35      1307               ST5     OUTBUF + 15 ,4 (3 : 3)  " STORE PLUS SIGN INTO BUFFER
 39 17               00 00 47   1308               JMP     2F
 39 18    +  13 19 04 33 36      1309   1H:         ST6     OUTBUF + 15 ,4 (3 : 3)  " STORE MINUS SIGN INTO BUFFER
*39 15    +  39 18 00 00 50
 39 19    +  00 00 00 01 05      1310   2H:         CHAR                    " CONVERT WORD TO CHARACTERS
*39 17    +  39 19 00 00 47
 39 20    +  13 20 04 05 30      1311               STA     OUTBUF + 16 ,4  ") STORE THEM
 39 21    +  13 21 04 05 37      1312               STX     OUTBUF + 17 ,4  ") INTO OUTPUT BUFFER
 39 22    +  00 03 00 00 64      1313               INC4    3               " NEXT POSITION ON CARD
 39 23    +  00 01 00 00 63      1314               INC3    1               " NEXT WORD FROM MEMORY
 39 24    +  39 08 00 01 54      1315               J4Z     OUT             " JUMP IF CARD COMPLETED
 39 25    +  39 14 00 04 53      1316               J3NZ    LOAD            " JUMP IF TRAJECT NOT COMPLETED
 39 26    +  39 08 00 00 47      1317               JMP     OUT             " ELSE PUNCH LAST CARD
                                 1318
                                 1319
                                 1320   END OF ASS:  END     START0
 39 27    +  00 00 00 00 00             AUX:        CON     0
*32 54    +  39 27 00 25 70
*32 53    +  39 27 00 05 37
*30 25    +  39 27 00 05 03
*30 25    +  39 27 00 05 30
 39 28    +  00 00 00 00 11
*36 19    +  39 28 00 05 04
*35 78    +  39 29 00 01 63
*35 78    +  39 29 00 00 67
*35 76    +  39 29 00 01 67

START ADDRESS:       *3875

TRANSLATION TIME:  330.54 SEC.
```

VALUE AND LINE NUMBER OF DEFINED SYMBOLS:

```
+  00 00 00 12 01        57   5CHAR
+  00 00 00 32 14       518   ADD
+  00 00 00 12 14        70   ADDRESS
+  00 00 00 28 30       102   ALF
+  00 00 00 12 15        71   APART
+  00 00 00 32 25       529   ARITHMOV
+  00 00 00 00 39        19   ASTERISK
+  00 00 00 31 82       482   ASTSK
+  00 00 00 12 10        66   ATEXPR
+  00 00 00 31 52       452   ATEXPRESSION
+  00 00 00 39 27         +   AUX
+  00 00 00 30 66       362   B
+  00 00 00 12 38        88   BACKWARD
+  00 00 00 28 35       107   BASE
+  00 00 00 12 00        56   BEGINOPASS
+  00 00 00 36 15       949   BUFFERCHAR
+  00 00 00 36 45       986   BUFFERCONSTANT
+  00 00 00 37 12      1063   BUFFERLN
+  00 00 00 36 24       962   BUFFERSYMB
+  00 00 00 01 00       108   BYTESIZE
+  00 00 00 12 00        56   CHAR
+  00 00 00 34 49       769   CHECKCOMMA
+  00 00 00 34 55       775   CHECKPAREN
+  00 00 00 34 65       785   CHECKVALUES
+  00 00 00 38 46      1209   CMPENDMARKER
+  00 00 00 12 03        59   CODE
+  00 00 00 00 63        25   COLON
+  00 00 00 00 60        22   COMMA
+  00 00 00 34 52       772   COMMACHECKED
+  00 00 00 34 05       725   COMMONEXPR
+  00 00 00 29 88       283   COMPL
+  00 00 00 28 31       103   CON
+  00 00 00 12 13        69   CONSTANT
+  00 00 00 30 47       343   CONTAINSLETTER
+  00 00 00 28 44       119   CORRECTION
+  00 00 00 00 17         7   CPUNCH
+  00 00 00 12 21        77   DANGEROUS
+  00 00 00 00 63        25   DEC
+  00 00 00 36 66      1010   DEFINE
+  00 00 00 36 72      1017   DEFINE1
+  00 00 00 31 72       472   DEFINED
+  00 00 00 00 01        40   DEFINEDSYMB
+  00 00 00 38 30      1193   DELETE
+  00 00 00 31 34       431   DIV
+  00 00 00 00 40        20   DIV1
+  00 00 00 01 04        36   DIV2
+  00 00 00 32 34       538   DIVA
+  00 00 00 32 29       533   DIVX
+  00 00 00 37 26      1060   DUMP
+  00 00 00 28 32       104   END
+  00 00 00 28 37       111   ENDMARKER
+  00 00 00 39 29      1320   ENDOFASS
+  00 00 00 28 33       105   EQU
+  00 00 00 00 43        21   EQUALSIGN
+  00 00 00 12 23        79   EQUIV
+  00 00 00 12 09        65   ERRONEOUS
+  00 00 00 28 43       118   ERROR
+  00 00 00 29 26       217   ERRORM
```

```
+  00 00 00 38 71      1238  EXIT
+  00 00 00 12 11        67  EXPR
+  00 00 00 00 64       360  F
+  00 00 00 31 09       406  FOUND
+  00 00 00 12 17        73  FPART
+  00 00 00 34 34       754  FUTREF
+  00 00 00 34 25       745  FUTSYMB
+  00 00 00 12 20        76  FUTURE
+  00 00 00 12 48        90  HERE
+  00 00 00 28 40       115  HUNDREDTHOUSAND
+  00 00 00 12 64        91  INBUF
+  00 00 00 12 48        90  INBUFX
+  00 00 00 37 50      1107  INITIALIZE
+  00 00 00 37 64      1121  INITREADING
+  00 00 00 37 74      1134  INITTABLE
+  00 00 00 36 86      1034  INSERTCONSTANTS
+  00 00 00 12 24        80  INST
+  00 00 00 12 18        74  INSTRUCTION
+  00 00 00 36 08       938  INVESTIGATE
+  00 00 00 12 16        72  IPART
+  00 00 00 28 42       117  JOBSEPARATOR
+  00 00 00 12 25        81  LASTCONSTANT
+  00 00 00 28 29       101  LASTINST
+  00 00 00 12 22        78  LASTLABEL
+  00 00 00 12 06        62  LASTSYMBOL
+  00 00 00 12 19        75  LC
+  00 00 00 30 86       383  LEFT
+  00 00 00 28 39       114  LENGTHOFMEMORY
+  00 00 00 15 00        48  LENGTHOFTABLE
+  00 00 00 29 50       245  LETGIT
+  00 00 00 00 11        12  LETTERB
+  00 00 00 00 15        13  LETTERF
+  00 00 00 00 17        14  LETTERH
+  00 00 00 00 35        15  LETTERZ
+  00 00 00 12 02        59  LINENUMBER
+  00 00 00 34 08       728  LITCONS
+  00 00 00 12 26        82  LITCONSPTR
+  00 00 00 39 14      1305  LOAD
+  00 00 00 34 29       749  LOC
+  00 00 00 00 03        42  LOCALBACKWARD
+  00 00 00 00 04        43  LOCALFORWARD
+  00 00 00 00 02        41  LOCALLABEL
+  00 00 00 31 74       474  LOCDEFINED
+  00 00 00 33 05       615  LOCLAB
+  00 00 00 30 70       367  LOOKFORSYMBOL
+  00 00 00 35 65       892  LOOP1
+  00 00 00 35 85       912  LOOP2
+  00 00 00 28 37       111  MAXADDRESS
+  00 00 00 28 36       109  MAXBYTE
+  00 00 00 28 38       112  MAXWORD
+  00 00 00 00 38        17  MINUS
+  00 00 00 38 49      1216  MIXALASSEMBLER
+  00 00 00 00 00        46  MOD
+  00 00 00 32 21       525  MUL
-  00 00 00 00 01        47  N
+  00 00 00 31 01       398  NEW
+  00 00 00 30 71       388  NEXT
+  00 00 00 32 81       592  NEXTINST
+  00 00 00 31 93       496  NEXTOPERAND
+  00 00 00 38 81      1265  NEXTPROGRAM
```

```
+  00 00 00 32 42      549   NEXTSPEC
+  00 00 00 32 83      593   NEXTSYMBOL
+  00 00 00 00 92       30   NLCR
+  00 00 00 31 80      480   NUMB
+  00 00 00 01 02       36   NUMBER
+  00 00 00 37 80     1140   OPERS
+  00 00 00 28 34      106   ORIG
+  00 00 00 31 44      441   OTHER
+  00 00 00 39 08     1299   OUT
+  00 00 00 13 04       93   OUTBUF
+  00 00 00 12 80       92   OUTBUFX
+  00 00 00 30 43      388   OVERFLOW
+  00 00 00 00 71       28   PAREN
+  00 00 00 34 58      778   PARENCHECKED
+  00 00 00 00 37       16   PLUS
+  00 00 00 00 61       23   POINT
+  00 00 00 33 70      685   POSSIBLE
+  00 00 00 00 18        8   PRINTER
+  00 00 00 29 16      204   PRINTLINE
+  00 00 00 34 98      822   PRODUCE
+  00 00 00 37 97     1157   PSEUDOINSTS
+  00 00 00 38 94     1265   PUNCHTRAJECT
+  00 00 00 00 94       31   QUOTE
+  00 00 00 29 63      258   READ
+  00 00 00 28 94      179   READ5CHAR
+  00 00 00 31 49      449   READATEXPRESSION
+  00 00 00 28 76      158   READC
+  00 00 00 28 45      127   READCHAR
+  00 00 00 00 16        6   READER
+  00 00 00 31 85      488   READEXPRESSION
+  00 00 00 33 87      707   READMIXMACHINEINST
+  00 00 00 32 80      590   READPROGRAM
+  00 00 00 29 44      239   READSYNTUNIT
+  00 00 00 32 39      546   READWVALUE
+  00 00 00 38 16     1179   RESETTABLE
+  00 00 00 30 95      392   RIGHT
+  00 00 00 01 00      108   S
+  00 00 00 00 64       26   SEMICOLON
+  00 00 00 01 01       35   SEP
+  00 00 00 31 19      416   SEPARATOR
+  00 00 00 38 87     1274   SKIPINPUT
+  00 00 00 00 66       27   SPACE
+  00 00 00 28 41      116   SPACES
+  00 00 00 32 71      578   STANDARD
+  00 00 00 38 75     1259   START0
+  00 00 00 38 80     1264   START1
+  00 00 00 12 27       83   STARTADDRESS
+  00 00 00 32 17      527   SUB
+  00 00 00 31 64      464   SYMB
+  00 00 00 01 03       37   SYMBOL
+  00 00 00 12 05       61   SYNTUNIT
+  00 00 00 13 28       94   TABLE
+  00 00 00 28 28      100   TABLEPTR1
+  00 00 00 12 04       60   TEXT
+  00 00 00 00 72       29   THESIS
+  00 00 00 00 39       19   TIMES
+  00 00 00 32 10      514   TIMESBASEPLUS
+  00 00 00 36 01      931   TRAVERSETREE
+  00 00 00 33 14      525   TREATINSTRUCTION
+  00 00 00 12 07       63   TYPE
```

```
+  00 00 00 00 00        39   UNDEFINEDSYMB
+  00 00 00 33 01       611   UNDEFSYMB
+  00 00 00 35 64       891   UPDATE
+  00 00 00 12 12        68   VALUE
+  00 00 00 00 02        44   VALUE1
+  00 00 00 00 45        45   VALUE2
+  00 00 00 12 08        64   VALUEOFNUMBER
+  00 00 00 38 31      1194   VISIT
+  00 00 00 32 68       575   WRONGFIELD
```



```
1     J3NP  553;  ENT5  265;  LD4   514;  DEC6  166;  INC6  066
2     JL    447;  SLAX  206;  CMP6  576;  ENN5  365;  FSUB  602
3     J1P   251;  J6N   056;  JXP   257;  LDX   517;  ST4   534
4     CMP2  572;  DEC2  162;  ENN1  361;  ENT1  261;  FADD  601
5     INC2  062;  J1N   051;  J2NP  552;  J4P   254;  JANP  550
6
7     JXN   057;  LD2   512;  LD6   516;  NOP   000;  SRC   506
8     STJ   240;  CHAR  105;  CMP4  574;  CMPX  577;  DEC4  164
9     DECX  167;  ENN3  363;  ENNA  360;  ENT3  263;  ENTA  260
10    FDIV  604;  IN    044;  INC4  064;  INCX  067;  J1NP  551
11    J2N   052;  J2P   252;  J4N   054;  J5NP  555;  J6P   256
12
13    JBUS  042;  JNOV  347;  JXNP  557;  LD1   511;  LD3   513
14    LD5   515;  LDA   510;  MOVE  107;  OUT   045;  SRA   106
15    ST2   532;  ST6   536;  STZ   541;  ADD   501;  CMP1  571
16    CMP3  573;  CMP5  575;  CMPA  570;  DEC1  161;  DEC3  163
17    DEC5  165;  DECA  160;  DIV   504;  ENN2  362;  ENN4  364
18
19    ENN6  366;  ENNX  367;  ENT2  262;  ENT4  264;  ENT6  266
20    ENTX  267;  FCMP  670;  FMUL  603;  HLT   205;  INC1  061
21    INC3  063;  INC5  065;  INCA  060;  IOC   043;  J1NN  351
22    J1NZ  451;  J1Z   151;  J2NN  352;  J2NZ  452;  J3N   053
23    J3P   253;  J4NP  554;  J5N   055;  J5P   255;  J6NP  556
24
25    JAN   050;  JAP   250;  JG    647;  JMP   047;  JRED  046
26    JXNN  357;  JXNZ  457;  JXZ   157;  LD1N  521;  LD2N  522
27    LD3N  523;  LD4N  524;  LD5N  525;  LD6N  526;  LDAN  520
28    LDXN  527;  MUL   503;  NUM   005;  SLA   006;  SLC   406
29    SRAX  306;  ST1   531;  ST3   533;  ST5   535;  STA   530
30
31    STX   537;  SUB   502;  J2Z   152;  J3NN  353;  J3NZ  453
32    J3Z   153;  J4NN  354;  J4NZ  454;  J4Z   154;  J5NN  355
33    J5NZ  455;  J5Z   155;  J6NN  356;  J6NZ  456;  J6Z   156
34    JANN  350;  JANZ  450;  JAZ   150;  JE    547;  JGE   747
35    JLE   947;  JNE   847;  JOV   247;  JSJ   147
36    (ALF      CON       END       EQU       ORIG)
```

8.1 Table of errormessages.

error number:	description:
ER 01	instruction definitions not o.k.
ER 03	pseudo instruction definitions not o.k.
ER 10	. required.
ER 11	value of number too large.
ER 21	atomic expression required.
ER 22	undefined local backward in expression.
ER 23	undefined symbol in expression.
ER 31	overflow by arithmetic operation.
ER 41	illegal field.
ER 42	) required.
ER 51	symbol required.
ER 52	illegal label.
ER 53	(local) label used twice.
ER 54	unknown instruction.
ER 55	separator required.
ER 61	= required.
ER 62	) required.
ER 63	value of a part not o.k.
ER 64	value of i part not o.k.
ER 65	value of f part not o.k.
ER 71	wrong address.
ER 72	address already used.
ER 81	value for future reference too large.
ER 91	wrong start address.
ER 92	program incorrect.
ER 98	end of input.
ER 99	overflow symbol table.

9. A loading program.

Before discussing the loading program, we have to treat a feature of the MIX simulator, concerning the start of the computer. With respect to the MIXAL system, consisting of simulator and assembler (see introduction), there is no question of a start of the simulator from the point that no relevant information is in the memory; the proper MIX simulator, without assembler, has the problem of how to load the first instructions into memory. For this purpose, the proper simulator has been provided with a simulated "start button", being "pressed" at the beginning of the execution of the MIX simulator. Pressing this start button, the MIX instruction "IN 0(16)" is executed, which causes the activation of the card reader; after one card has been read, and the information of it has been stored into the locations zero upwards, the normal cycle of the computer is started at location zero.

Preparing this first card by hand, we thus can get the first simple loading program, consisting of at most 16 instructions, into the machine. Each position on this card corresponds to one byte; this byte is filled with the internal code of the character on that position (this is the usual effect of an IN instruction).
Unfortunately, this function on the collection of characters into the numbers $\{n \mid 0 \leq n < 100\}$ is not a surjective one; e.g. the instruction codes of the IN and JBUS instructions (44 and 42, resp.) do not correspond to any MIX character. As these two instructions are indispensable in a loading program, we have to find another way to get these instructions into the memory. An easy solution is: constructing these instructions programmatically (e.g. by means of the instructions: "ENT2 40; INC2 4; ST2 X(5:5)", when an IN instruction has to be loaded at location X). Till so far, we have discussed the start of this special MIX simulator, and noticed some difficulties and possibilities of it.

The loading program of this section consists of three cards, the proper loader being contained in the second and third card. The first card serves to construct the instructions for reading the next two cards, and, after these cards have been read, to construct the IN and JBUS instructions of the proper loader. Next, control is passed on to the proper loader, and the

locations 0 - 15 are used as an input buffer by the loader. Much attention had to be paid to arranging the instructions of the proper loader in such a way, that the value of the bytes of all instructions did correspond to an existing MIX character.

Eventually, the code of the information to be loaded is given. The input of the loader consists of a number of blocks of information, each block consisting of a number of information cards, preceded by one traject card. On the traject card, two numbers have been punched: a startaddress (columns 1 - 5), and a quantity indicating the number of consecutive locations (beginning at the startaddress) to be loaded (columns 11 - 15). Each information card contains the information to be loaded into (the next) five consecutive locations. Column 3 contains the sign (+ or -) of the first location, columns 6 - 15 contain the numerical value for this location in decimal notation (two columns for each byte); columns 18 and 21 - 30 contain the information for the second location, etc..
The last of the information cards contains as much information as is necessary to complete this particular block.
A traject card, which contains a zero as second number, indicates the final block. In this case, the startaddress on the traject card is interpreted as the location at which the loaded program has to be started.
This kind of code is produced by the routine called "punch traject" of the MIXAL version of the assembler. This routine punches one block of information, including the traject card. The startaddress is given by the value of index register 2, and the number of consecutive locations to be punched out, is given by the negative value of index register 3.

Next follows the MIXAL text of the entire loading program (some instructions have been added, from location 100 upwards, to punch the three initial cards, mentioned before, which contain the loading system), in the way it has been printed by the MIXAL version of the MIXAL assembler.

```
0000  +  1640000261   1   INBUF:    ENT1  1640
0001  +  0002010262   2             ENT2  2 ,1
0002  +  0004000061   3             INC1  4
0003       004531     4             ST1   INCARD1 (4:5)
0004       004531     5             ST1   INCARD2 (4:5)
0005       004532     6             ST2   JBUS (4:5)
0006  +  0016000000   7   INCARD1:  NOP   16
*0006  +  0006004531
0007  +  0032000000   8   INCARD2:  NOP   32
*0008  +  0007004531
0008  +  0008000000   9   JBUS:     NOP   *
*0009  +  0008004532
0009       004531    10             ST1   READINFCARDS (4:5)
0010       004532    11             ST2   WAITINFCARDS (4:5)
0011       004531    12             ST1   READTRAJECT (4:5)
0012       004532    13             ST2   WAITTRAJECT (4:5)
0013       000047    14             JMP   READTRAJECT
                     15
                     16             ORIG  15
0015  +  0000000000  17   AUX:      CON   0
0016  +  0000000038  18   MINUS:    ALF   .0000-
0017  +  0003000063  19   1H:       INC3  3
0018       000053    20             J3NZ  2F
0019  +  0015000063  21             ENN3  15
0020  +  0000000000  22   READINFCARDS:  NOP  INBUF
*0009  +  0020004531
0021  +  0021000000  23   WAITINFCARDS:  NOP  *
*0010  +  0021004532
0022  +  0015033311  24   2H:       LD1   INBUF + 15 ,3 (3:3)
*0018  +  0022000453
0023  +  0016039510  25             LDA   INBUF + 16 ,3
0024  +  0017030517  26             LDX   INBUF + 17 ,3
0025  +  0000000005  27             NUM
0026  +  0016000571  28             CMP1  MINUS
0027       000647    29             JNE   STORE
0028  +  0015000530  30             STA   AUX
0029  +  0015000520  31             LDAN  AUX
0030  +  0000020530  32   STORE:    STA   0 ,2
*0027  +  0030000847
0031  +  0001000062  33             INC2  1
0032  +  0017000452  34             J2NZ  1B
0033  +  0000000000  35   READTRAJECT:   NOP  INBUF
*0013  +  0033000047
*0011  +  0033004531
0034  +  0034000000  36   WAITTRAJECT:   NOP  *
*0012  +  0034004532
0035  +  0000000260  37             ENTA  0
0036  +  0002000517  38             LDX   INBUF + 2
0037  +  0000000005  39             NUM
0038  +  0039000230  40             STA   * + 1 (0:2)
0039  +  0000000362  41             ENN2  0
0040  +  0000000260  42             ENTA  0
0041  +  0000000517  43             LDX   INBUF
0042  +  0000000005  44             NUM
0043  +  0045000230  45             STA   * + 2 (0:2)
0044  +  0000020160  46             DECA  0 ,2
0045  +  0000000152  47             J2Z   0
0046  +  0030000230  48             STA   STORE (0:2)
0047  +  0019000047  49             JMP   READINFCARDS - 1
                     50
```

```
                                    51                  ORIG    100
 0100                  000240       52                  STJ     9F
 0101          +  0000001745        53                  OUT     0 (17)
 0102          +  0016001745        54                  OUT     16 (17)
 0103          +  0032001745        55                  OUT     32 (17)
 0104          +  0104001742        56                  JBUS    * (17)
 0105          +  0102000147        57        9H;       JMP     *
*0100          +  0105000240
                                    58                  END     100
```



```
          + 0000000015        17      AUX
          + 0000000000         1      INBUF
          + 0000000006         7      INCARD1
          + 0000000007         8      INCARD2
          + 0000000008         9      JBUS
          + 0000000016        10      MINUS
          + 0000000020        22      READINFCARDS
          + 0000000033        35      READTRAJECT
          + 0000000030        32      STORE
          + 0000000021        23      WAITINFCARDS
          + 0000000034        36      WAITTRAJECT
```

References.

[1]. Knuth, D.E., The Art of Computer Programming, Vol. 1
Fundamental Algorithms.
Reading (Mass.), Addison-Wesley, 1968.

[2]. van de Riet, R.P.,
Inleiding in de Informatica,
Syllabus bij het Oriënterend Colloquium Informatica,
CR 21/70 december, Mathematisch Centrum.

[3]. Grune, D., Handleiding Milli-systeem voor de EL X8,
LR. 1.1, Mathematisch Centrum.