DOS 3.3 ProDOS

AMPERGO

Improve both your program's readability and its execution speed by using labels in your GOTO and GOSUB statements. This handy ampersand utility is all you need for the task.

by Cornelis Bongers
Erasmus University
Postbox 1738
3000 DR Rotterdam
The Netherlands

One of the most severe limitations of Applesoft is its lack of label support. In Pascal and most other programming languages, subroutines can be called by name, for example, CALL PRINTPAGE. In most versions of BASIC, including Applesoft, the corresponding statement would be something like GOSUB 63241.

The obvious advantage of using names rather than line numbers in subroutine calls is that they make programs considerably easier to read. This is especially pertinent to longer programs, since short programs usually do not need many GOSUB statements. When you are working on a long program, say five to ten pages, the only way to maintain control is to modularize the program. Such programs will, therefore, contain many subroutines that handle user input, errors, disk access, etc. A randomly chosen line from a large program may then very well read:

```
3540 GOSUB 230 : GOSUB 300
   : GOSUB 41530
```

which stands for get user input, format it and write it to disk.

Lines like this are not a problem when the program is being developed, but if it is updated a month after completion, there is a good chance that the programmer will have forgotten what GOSUB 41530 does and will have to search through the listing. Of course, this trap can be avoided by inserting REM statements behind every GOTO/GOSUB statement. Unfortunately, although this method is theoretically sound, few people apply it in practice, as it is tedious and time-consuming.

A more logical solution is to work with labels rather than line numbers. The function of the subroutine can then be summarized in the label name. For example, the statement GOSUB USER INPUT clearly indicates the subroutine's function.

Strangely enough, Applesoft allows names in the CALL statement (you can specify CALL DISKIO if the variable DISKIO is initialized to the proper value), but not in the far more common GOSUB and GOTO statements. It's hard to guess why labels are not supported by BASIC, for the implementation is rather simple and requires only moderate overhead.

Now you *can* work with labels in Applesoft; the program AMPERGO (**Listing 1**) can handle GOSUB, GOTO, ON-GOTO/GOSUB and LIST statements with label references. To use AMPERGO in your program, simply BRUN AMPERGO and follow the syntax rules described below.

Syntax of the Statements Supported by AMPERGO

& GOSUB and & GOTO

The GOSUB statement may be specified in three different forms:

& GOSUB line number
& GOSUB label
& GOSUB (expression)

The & statement transfers control to the AMPERGO routine. All keywords and text behind the ampersand (&) up to the next colon (:) or end of line symbol are analyzed and subsequently processed by AMPERGO.

The first statement above, & GOSUB line number, is identical to the normal Applesoft GOSUB statement and is only included in AMPERGO for completeness.

The second statement, & GOSUB label, is the most important; execution of this statement transfers control to the line that contains the label statement that matches the label specified behind the GOSUB keyword. The label statement has the following format:

& > label

This statement must be the *first* statement of a program line — in other words, the statement must immediately follow the line number.

An example of an & GOSUB application is presented in AMPERGO.DEMO1 (**Listing 2**). The & GOSUB statement and the & > label statement may be followed by other statements on the same line, provided a colon is used as separator (see **lines 270 and 310**). Note also that the & > label statement has no effect on program execution; Applesoft simply skips the statement just as it skips a DATA statement (see **line 230**).

A label must satisfy two conditions. The first and most important condition is that the first character of a label must be a letter. If the label consists of more than one character, the second character must be either a letter, a digit, or a blank. Hence, the first two characters of a label must satisfy the same conditions as the name of a real variable in Applesoft. There are no restrictions on the remaining characters. For instance, labels such as PP?, PRODUCT# and GOLA(10) are correct, but the labels P%, #PRODUCTS and A(10) are incorrect.

Unfortunately, Applesoft sometimes converts parts of labels that satisfy the above conditions to tokens. This happens if Applesoft recognizes a keyword in the label. This would be the case, for instance, if you used the label FRE. Although FRE is syntactically correct, AMPERGO will not accept it as a label since FRE is tokenized to a one-byte value (upon input of the line containing FRE), and this value does not correspond to a letter. A similar problem arises if you use the label GATE, in which Applesoft recognizes the keyword AT. In this case, the second and third characters of the label will be converted to a token.

You can avoid problems of this kind by enclosing in quotation marks any labels that contain a keyword that starts in the first or second position. Thus the line,

```
50 & GOSUB "FREE SPACE"
   : GOSUB "GATE"
```

and the corresponding label instructions,

```
200 & > "FREE SPACE" : . . .
300 & > "GATE" : . . .
```

are correct. Note that "quoting" a label is also necessary if you want to use blanks in the label. For example, GOSUB P A R I T Y will be parsed by Applesoft as GOSUB PARITY. If you quote the label (i.e., & GOSUB "P A R I T Y"), all blanks will be kept in position and none will be removed.

The second condition a label must satisfy is that it may not contain commas. Although commas are not allowed, no explicit check is performed because the comma is used as a label separator in the & ON-GOTO/GOSUB statement. If a comma is included in a label, the part of the label behind the comma will be ignored. So if you specify:

& GOSUB MAR,JO

AMPERGO will search for the label statement & > MAR rather than for & > MAR,JO.

By the way, the name you use for a label does not influence any Applesoft variables with the same name. For example, the values of the variables EVA, EVA% and EVA$ will not be affected by execution of the statement & GOSUB EVA.

The third form of the & GOSUB statement is & GOSUB (expression). The expression may be either a numerical or a string expression and must be enclosed in parentheses. If the result of the evaluation is numerical, AMPERGO will interpret it as a line number and execute a GOSUB to this line number. Hence,

```
10 A=100 : & GOSUB (A)
```

will execute the subroutine at line 100. Thus, the statements at **line 10** above have the same effect as the Applesoft GOSUB 100 statement.

On the other hand, if the expression evaluates to a string, this string will be interpreted as a label. So, if you specify:

```
100 A$ = "KEES": & GOSUB (A$)
```

a GOSUB will be done to the line starting with:

```
& > KEES
```

Note that the quotation marks in A$ = "KEES" serve only as terminators in the assignment and are not a part of the label.

The label that results from evaluation of a string expression must also satisfy the two conditions mentioned above: the first two characters must form a syntactically correct name of a real variable and the label may contain no commas.

One application of the indirect GOSUB statement is to use it instead of the Applesoft ON-GOSUB statement. A common situation is, for example,

1. Display menu
2. Get user choice in the variable A
3. ON A GOSUB 1000,2000,3000,4000, 5000

The ON-GOSUB statement can be replaced with & GOSUB (1000*A), which will save some memory and speed up execution. The indirect GOSUB statement is extremely handy in some applications. However, I am not particularly enthusiastic about this construction, since things can go terribly wrong if you renumber a program containing such statements.

Since the syntax of & GOTO is identical to that of & GOSUB, it need not be discussed separately here.

& ON-GOTO/GOSUB

AMPERGO also supports the Applesoft equivalent of the ON-GOTO/GOSUB statement. In the AMPERGO statement & ON-GOTO/GOSUB, you may use line numbers, labels or expressions. **Line 110** below is a correct & ON-GOSUB statement.

```
100 A$="JANJAAP" : A=200
    : A%=300
110 & ON K GOSUB (A$) , JANJAAP ,
    (A) , 400 , (A%)
```

If K=1 or 2, the subroutine starting with the label statement & > JANJAAP will be executed; if K=3, the subroutine at **line 200** will be executed; if K=4, the subroutine at **line 400** will be executed; and if K=5, the subroutine at **line 300** will be executed. If K>5, control will pass to the statement following the & ON-GOSUB statement (similar to the Applesoft ON-GOSUB).

When used with labels, AMPERGO's & ON-GOTO/GOSUB statement offers an attractive alternative to the Applesoft ON-GOTO/GOSUB statement, since just a quick glance is needed to determine which subroutines are invoked by its execution. This is illustrated by the following statement:

```
100 & ON CHOICE GOSUB DISPLAY ,
    EDIT , ADD , "TRANSFORM" ,
    "SAVE"
```

& LIST

The final option of AMPERGO is the & LIST statement. Let me first explain how LIST fits into the GOSUB, GOTO, ON-GOTO/GOSUB pattern. When I had substituted & GOTO/GOSUB and & ON-GOTO/GOSUB statements in some large programs, everything worked as expected and the listings were much easier to read. However, on editing these programs, I soon discovered that something was still wrong. Though the labeled subroutines made it easier to follow the flow of program execution, the lack of line numbers made it difficult to actually find the labeled routines for editing.

The search for subroutines took so much time that I decided to add the & LIST option to AMPERGO. The & LIST option lists lines containing label statements, so that you can easily find the location of particular subroutines. As with & GOSUB and & GOTO, you must specify either a line number, a label or a quoted expression behind & LIST.

For example, if you specify **& LIST MARJO** the line containing the label statement & > MARJO will be displayed (if present). If you want to list more than one line, the final character of the & LIST statement must be a comma, for example **& LIST MARJO,**.

Note that & LIST does not support multiple arguments, so the statement:

& LIST label1,label2

will list only the line containing the label statement & > label1.

Benefits and Drawbacks of AMPERGO

Let us consider some of the advantages and disadvantages of using the AMPERGO routines instead of their Applesoft equivalents. An obvious disadvantage is that — apart from the overhead of AMPERGO itself (451 bytes) — a program containing & GOSUB/GOTO statements will usually occupy more memory. Assuming the average length of a label is 8 characters, there is a fixed overhead of 11 characters for the label statement:

(8 + 1 (&) + 1 (>) + 1 (:))

Seven additional bytes may be needed (but not always) for the label-pointer, for a total of 18 bytes of overhead per subroutine. If the average length of a line number is 3, there is a net overhead of 8 + 1 (&) - 3 = 6 bytes per & GOTO/GOSUB statement. Overall, this implies that working with & GOTO/GOSUB statements increases program length by a maximum of:

18 * number of subroutines + 6 * number of & GOTO/GOSUB statements

This demonstrates that, especially in large programs, it is not advisable to use & GOTO/GOSUB statements indiscriminately at every GOTO/GOSUB. My own strategy is to avoid long jumps (i.e., 10 GOTO 63200) and use the normal Applesoft GOTO for short jumps. For all subroutine calls, I use the & GOSUB label or the & ON-GOSUB label1,label2,... construction. In my experience, the time gain realized when editing or updating these more readable programs amply compensates for the reduction in "free memory" caused by the overhead of AMPERGO and the & GOSUB/GOTO statements.

As to the benefits of AMPERGO, apart from improved readability, AMPERGO provides faster program execution. The reason for this phenomenon is that, for each label, a pointer to the line containing the corresponding label statement is stored in the program's variable space.

The first time an & GOTO/GOSUB statement is encountered, AMPERGO searches the program for the line with the matching label. If the line is found, the program sets a pointer in the variable space to the start of the line. The next time the & GOSUB label statement is executed, AMPERGO will detect that the variable space contains a pointer associated with the label following the & GOSUB statement. AMPERGO will then check whether this label is identical to the label specified in the label statement on the line referenced by the pointer. If so, the GOSUB statement is executed immediately; if not, the program is again searched for the matching label.

The pointers generated by AMPERGO are stored in a way that is completely transparent to the user; that is, the user cannot access the pointers and they will not be affected by any Applesoft actions. Still, the pointers are stored in the normal BASIC variable space, which you can also use simultaneously for program variables. This is possible because Applesoft uses only three of the five bytes allocated for storage of the string descriptor. The remaining two bytes are not used by Applesoft and provide us with the room needed to store a two-byte pointer.

This is how it works. If AMPERGO encounters a label (or an expression that evaluates to a label) following an & GOSUB statement, the label is interpreted as the name of a string variable. AMPERGO then orders Applesoft to search the variable space for a string variable with this name. (If it is not present, Applesoft will create one.) Next, bytes 4 and 5 of the string descriptor storage area are checked to see if these are zero; if so, AMPERGO concludes that the specified label has not yet been referenced. AMPERGO then searches the program for the corresponding label statement and stores the pointer to the line containing this label statement in bytes 4 and 5 of the string variable.

On the other hand, if bytes 4 and 5 of the string variable are not zero, & GOLA retrieves the label-pointer from these bytes and checks to see if the line referenced by the label-pointer contains a label (statement) that matches the label specified in the & GOSUB statement. If the labels are identical, control is transferred to the statement following the label statement; if not, the program searches for the label once again.

Note that the fact that label-pointers are stored in normal string variables implies that not every label requires seven bytes in the variable space for the label-pointer. For example, in AMPERGO.DEMO1 the label DIGIT and the string variable DIG$ are used. Since the first two characters of DIGIT and DIG$ are the same, the label-pointer can be stored in the storage area of DIG$ and thus requires no extra room.

It is worth stressing that unnecessary label searches may occur if you use two (or more) labels in which the first two characters are identical. The reason for this is that in Applesoft only the first two characters of a variable name are significant. So, if you use the labels TEST1 and TEST2, the string variable TE$ will alternately be used to store the pointers to the corresponding label statements.

See, for example, AMPERGO.DEMO2 (**Listing 3**). If **line 110** is executed, AMPERGO will search for the label TEST2 and subsequently store the pointer to **line 120** (where TEST2 is located) in the string variable TE$. Upon execution of **line 120**, AMPERGO will detect that the string variable that corresponds to the label TEST1 (i.e., TE$) contains a pointer. However, this pointer does not point to the right label statement (i.e., the pointer points to **line 120**), so the program is searched for the label TEST1 and the pointer to **line 110** (in which TEST1 occurs) is stored in TE$. When **line 110** is executed, the same thing happens again, and so on.

Although execution will not give rise to any errors (since AMPERGO checks labels before control is transferred), you will get the most out of AMPERGO if you make sure that the first two label characters differ. When in doubt, check whether execution of a particular & GOSUB/GOTO label statement forces a search through the program (rather than an immediate transfer of control) by PEEKing location 255. If AMPERGO searches the program for a particular label, location 255 is set to 255 ($FF); if not, location 255 is set to zero.

As the program output of AMPERGO .DEMO 2 will show, each time an & GOTO statement is executed, the program searches for one of the labels, TEST1 or TEST2. This problem has been eliminated in AMPERGO .DEMO3 (**Listing 4**). Note that the only difference between the two programs is that the label TEST2 has been replaced by the label TTEST2 (**line 120**). When you run the program, the value 255 will be output twice, indicating that AMPERGO performed two searches through the program. This is correct, for the program contains two different labels. The rest of the output values will be zero, which signals that AMPERGO has jumped directly to the correct line(s).

Now for the happy conclusion! It is no longer necessary to position the most frequently used subroutines near the start of your program (or just below the caller). This has been the rule in Applesoft to get a reasonable execution time for large programs, since Applesoft starts its search for line numbers specified in GOTO/GOSUB statements at the beginning of your program. Another reason to position frequently called subroutines near the beginning of the program was that, for instance, evaluation of GOSUB 100 takes less time than GOSUB 50000, as more digits must be processed in the latter case.

With AMPERGO, it doesn't matter where subroutines are positioned in the program. As long as the first two characters of your labels are unique, the program will be searched one time only for each label, after which the label-pointer will be used.

However, execution speed of AMPERGO statements will depend on the number of active variables. If there are many variables, the search for a particular label-variable will, on the average, last longer. Fortunately, the number of variables in a large program is usually much smaller than the number of program lines. A program consisting of 500 program lines will usually have considerably less variables, say 100. (The number of arrays is not relevant.)

I tested AMPERGO on a program of this size. The time required to execute an 8-character GOSUB label statement varied — depending on the position of the label-variable in the variable space — between 1.7 milliseconds (if the label-variable was the first variable referenced in the program) and 5 milliseconds (if the label-variable was the last (101st) variable referenced).

The execution time of the original Applesoft GOSUB statement depends on the number of lines Applesoft encounters before it finds the line for which it is searching. In this case, execution time does not depend on the number of variables. If the number of lines between the GOSUB statement and the target line equals 30, execution time of a GOSUB statement is about 3 milliseconds. If the number of lines equals 100, execution time increases to 7 milliseconds, and if the number of lines equals 250 or 500, the execution times are 15 milliseconds and 29 milliseconds, respectively.

Roughly speaking, these figures indicate that for execution speed there is a break-even point between & GOSUB and the Applesoft GOSUB if the distance between the source and destination is about 20-30 lines. If the distance increases, AMPERGO's execution time remains constant, but Applesoft's execution time increases linearly with the distance. If you want to see the difference between Applesoft and AMPERGO execution speed, load your largest Applesoft program and note the number of the last line of the program. Next, enter the following lines (here we assume that the largest line number is 29999):

```
0 FOR I=1 TO 500:GOSUB 30000
  :NEXT:STOP
1 FOR I=1 TO 500: & GOSUB LABEL
  :NEXT:STOP
30000 PRINT "A"; :RETURN
30001 & > LABEL: PRINT "A";
  :RETURN
```

Type **RUN 0** and **RUN 1**. You can evaluate the respective performances of Applesoft and AMPERGO by watching the speed at which the characters are printed on the screen.

Use of the AMPERGO statements provides still another advantage that may not be readily apparent. AMPERGO lets you write position independent BASIC subroutines. For example, when you overlay the last part of a main program with a new module, this module will usually use routines from the part of the main program that is not overlayed. You will get into trouble if you change the line numbers in the first part (the main program), for this implies that you have to change all corresponding line number references in all your modules.

A similar problem arises if you work with a library of BASIC subroutines. When you change line numbers in one or more of the library routines, all the other routines — and possibly some main programs too — have to be checked to see if the line number references are still correct. Depending on the number of library routines and their internal structure, this may be a lot of work.

Using label references eliminates all of these problems, for no matter where your routines are positioned in the program, AMPERGO will find them.

Error Messages

AMPERGO can return three error messages. These and their most probable causes are outlined in **Table 1**.

The Machine Language Program

Entering the Program

To use AMPERGO, simply enter the program shown in **Listing 1**. If you have an assembler, use it to enter the source code and assemble. If you do not have an assembler, the hex code may be entered directly in the Monitor as described in "A Welcome to New *Nibble* Readers" in the beginning of this issue. You may then save the program on disk with the command:

```
BSAVE AMPERGO,A$9400,L$1C3
```

The program can be installed with the BRUN command. This command loads AMPERGO and executes an initialization routine that sets HIMEM to $9000 and installs the & vector. Note that the string pointers are also reset to the value of HIMEM, so if you want to install AMPERGO from within an Applesoft program, insert the BRUN command as the first line. AMPERGO makes extensive use of Applesoft routines; it assumes that these are in ROM or in the Language Card.

Acknowledgement: *My thanks to Hans Geilenkirchen for proofreading the first draft of this article.*

Table 1: Errors When Using AMPERGO

Syntax Error

- No ampersand (&) or "greater than" (>) symbols were specified in an AMPERGO statement.
- The first two characters of a label do not form a legal Applesoft name of a real variable.
- The label contains a keyword that starts in the first or second position (insert quotation marks around the label).

Illegal Quantity Error

- The length of the label equals zero (i.e., & GOTO (A$), where A$ is empty).
- The expression evaluates to a negative number (i.e., & GOTO (−1)).

Undefined Statement Error

- No "matching" label was found. Check your program for the presence of the corresponding label statement and see if the labels are identical.
- No > symbol was specified in the label statement.

LISTING 1: AMPERGO

```
                *
                * AMPERGO
                * BY CORNELIS BONGERS
                * COPYRIGHT (C) 1984
                * BY MICROSPARC, INC.
                * CONCORD, MA Ø1742
                *
                * MERLIN ASSEMBLER
                *
                * SYNTAX : & GOSUB LABEL/(EXPRESSION)
                *        : & ON A GOSUB LABEL/(EXPRESSION)
                *        : & LIST LABEL/(EXPRESSION)
                *
                         ORG  $94ØØ
                *
                * ZERO PAGE ADDRESSES
                *
                TEMP1    =    $Ø6         ;POINTER TO LABEL
                VALTYP   =    $11         ;TYPE EXPRESSION
                LINNUM   =    $5Ø         ;LINE NO FROM LINGET
                TXTTAB   -    $67         ;POINTER TO START OF PROGRAM
                FRETOP   =    $6F         ;BOTTOM OF STRINGPOOL
                MEMSIZ   =    $73         ;HIMEM
                CURLIN   =    $75         ;CURRENT LINE NUMBER
                DESCPTR  =    $AØ         ;POINTER TO DESCRIPTOR
                SIGN     =    $A2         ;SIGN MFP ACCU
                LOWTR    =    $9B         ;SEARCH POINTER
                TXTPTR   =    $B8         ;TEXTPOINTER
                TEMP5    =    $FA         ;TEMPORARY
                TEMP7    =    $FE         ;LIST FLAG
                *
                * TOKENS
                *
                GOTOT    =    171
                GOSUBT   =    176
                ONT      =    18Ø
                LISTT    =    188
                GRTT     =    2Ø7         ;> TOKEN
                AMPT     =    175         ;& TOKEN
                *
                * APPLESOFT ROUTINES
                *
                CHARGET  =    $B1         ;GET NEXT CHARACTER
                CHARGOT  =    $B7         ;GET CURRENT CHAR
                DATA     =    $D995       ;DATA HANDLER
                NEWSTT   =    $D.'D2      ;RESTART APPLESOFT
                GOTO     =    $D93E       ;GOTO HANDLER
                PARCHK   =    $DEB2       ;EVALUATE EXPR AND CHECK ()
                CHKSTACK =    $D3D6       ;CHECK STACK
                GETADR   =    $E752       ;CONVERT FAC TO INTEGER
                PTRGET   =    $DFE3       ;EVALUATE NAME
                FREFAC   =    $E6ØØ       ;FREE TEMP DESCRIPTOR
                GETBYT   =    $E6F8       ;CONVERT ASCII'S TO INTEGER
                ADDON    =    $D998       ;TXTPTR=TXTPTR+Y
                LISTP1   =    $D6D4       ;1 TH PART LIST HANDLER
                LISTP2   =    $D6DA       ;2 TH PART LIST HANDLER
                SYNT     =    $DEC9       ;SYNTAX ERROR
                UNDEFS   =    $D97C       ;UNDEFINED STATEMENT
                ILLQ     =    $E199       ;ILLEGAL QUANTITY ERROR
                *
                * & VECTOR, INPUT BUFFER
                *
                BJP      =    $3F5        ;& VECTOR
                IBUFP    =    $2FC        ;INPUT BUFFER (LAST PART)
                *
                * INITIALIZATION
                *
94ØØ: A9 ØØ     BEGIN    LDA  #BEGIN
94Ø2: 85 73              STA  MEMSIZ      ;SET HIMEM
94Ø4: 85 6F              STA  FRETOP      ;AND BOTTOM OF STRINGPOOL
94Ø6: A9 9Ø              LDA  #>BEGIN-$4ØØ ;LEAVE SPACE FOR PRODOS BUFFERS
94Ø8: 85 74              STA  MEMSIZ+1
94ØA: 85 7Ø              STA  FRETOP+1
94ØC: A9 17              LDA  #START      ;SET & VECTOR
94ØE: 8D F6 Ø3           STA  BJP+1
9411: A9 94              LDA  #>START
9413: 8D F7 Ø3           STA  BJP+2
9416: 6Ø                 RTS
                *
                * MAIN PROGRAM
                *
9417: A2 ØØ     START    LDX  #$ØØ
```

```
9419: 86 FE             STX  TEMP7          ;CLEAR LIST FLAG
941B: C9 B4             CMP  #ONT           ;ON TOKEN ?
941D: DØ 17             BNE  LISS           ;BRANCH IF NOT
941F: 2Ø B1 ØØ          JSR  CHARGET        ;ADVANCE TEXTPOINTER
9422: 2Ø F8 E6          JSR  GETBYT         ;CONVERT ON VAR TO INTEGER
9425: A8                TAY                 ;SAVE TOKEN BEHIND VAR IN Y
9426: C6 A1    TRNXTL   DEC  DESCPTR+1      ;COUNT !
9428: FØ 17             BEQ  EXGOTSUB       ;EXECUTE GOTO/GOSUB IF CNTR=Ø
942A: 2Ø B1 ØØ CCSEARCH JSR  CHARGET        ;SEARCH FOR NEXT COMMA
942D: FØ Ø6             BEQ  NOFIND         ;RTS IF NO CORRESP. LABEL
942F: C9 2C             CMP  #',            ;COMMA ?
9431: DØ F7             BNE  CCSEARCH       ;CONTINUE SEARCH IF NOT
9433: FØ F1             BEQ  TRNXTL         ;TRY NEXT ONE
9435: 6Ø       NOFIND   RTS                 ;BACK TO BASIC
9436: C9 BC    LISS     CMP  #LISTT         ;LIST TOKEN ?
9438: FØ 16             BEQ  LISTH          ;BRANCH IF SO
943A: C9 CF             CMP  #GRTT          ;'GREATER THAN' TOKEN ?
943C: DØ Ø4             BNE  GOT            ;TRY GOTO IF NOT
943E: 4C 95 D9          JMP  DATA           ;PROCESS AS DATA STATEMENT
9441: 98       EXGOTSUB TYA                 ;GET TOKEN FROM Y-REG
9442: C9 AB    GOT      CMP  #GOTOT         ;GOTO TOKEN ?
9444: FØ 5F             BEQ  GOTOH          ;BRANCH IF SO
9446: C9 BØ             CMP  #GOSUBT        ;GOSUB TOKEN ?
9448: FØ 12             BEQ  GOSUBH         ;BRANCH IF SO
944A: 2Ø B7 ØØ GETA     JSR  CHARGOT        ;FOR FUTURE AMPERSOFT EXT
944D: 4C C9 DE          JMP  SYNT           ;SYNTAX ERROR (CONNECT)
945Ø: AØ Ø1    LISTH    LDY  #$Ø1
9452: B1 B8             LDA  (TXTPTR),Y     ;CHECK ON EOL/EOS
9454: FØ F4             BEQ  GETA           ;SYNTAX ERROR IF EOL
9456: C9 3A             CMP  #':
9458: FØ FØ             BEQ  GETA           ;OR EOS
945A: C6 FE             DEC  TEMP7          ;SET LIST FLAG
945C: A9 Ø3    GOSUBH   LDA  #$Ø3
945E: 2Ø D6 D3          JSR  CHKSTACK       ;CHECK ON STACK OVERFLOW
9461: A5 B9             LDA  TXTPTR+1
9463: 48                PHA
9464: A5 B8             LDA  TXTPTR         ;PUSH STACKPOINTER
9466: 48                PHA
9467: A5 76             LDA  CURLIN+1       ;PUSH CURRENT LINE NUMBER
9469: 48                PHA
946A: A5 75             LDA  CURLIN
946C: 48                PHA
946D: A9 BØ             LDA  #GOSUBT
946F: 48                PHA                 ;PUSH GOSUB TOKEN
947Ø: 2Ø A5 94          JSR  GOTOH          ;EXECUTE A GOTO
9473: 24 FE             BIT  TEMP7          ;LIST FLAG ON ?
9475: 3Ø Ø3             BMI  LISTH2         ;BRANCH IF SO
9477: 4C D2 D7          JMP  NEWSTT         ;JMP TO BASIC (NO RTS)
947A: 68       LISTH2   PLA                 ;PULL GOSUB TOKEN
947B: 68                PLA
947C: 85 75             STA  CURLIN         ;RESTORE CURRENT LINE NUMBER
947E: 68                PLA
947F: 85 76             STA  CURLIN+1
9481: 68                PLA
9482: 85 B8             STA  TXTPTR         ;RESTORE TEXTPOINTER
9484: 68                PLA
9485: 85 B9             STA  TXTPTR+1
9487: 68                PLA                 ;PULL EXSTMNT'S RETURN ADDRESS
9488: 68                PLA
9489: 2Ø 95 D9          JSR  DATA           ;SET TXTPTR TO END INSTRUCTION
948C: A5 B8             LDA  TXTPTR
948E: DØ Ø2             BNE  NODDEC         ;TXTPTR=TXTPTR-1
949Ø: C6 B9             DEC  TXTPTR+1
9492: C6 B8    NODDEC   DEC  TXTPTR
9494: 2Ø B7 ØØ          JSR  CHARGOT        ;GET LAST CHAR
9497: A8                TAY                 ;SAVE IT
9498: 2Ø B1 ØØ          JSR  CHARGET        ;RESTORE TEXTPOINTER
949B: CØ 2C             CPY  #',            ;COMMA ?
949D: FØ Ø3             BEQ  LISTALL        ;BRANCH IF SO
949F: 4C DA D6          JMP  LISTP2         ;LIST LINE
94A2: 4C D4 D6 LISTALL  JMP  LISTP1         ;LIST FROM CURRENT LINE
94A5: 2Ø B1 ØØ GOTOH    JSR  CHARGET        ;GET CHAR BEHIND GOTO/GOSUB
94A8: A2 ØØ             LDX  #$ØØ
94AA: 86 FF             STX  TEMP7+1        ;SET SEARCH INDICATOR
94AC: BØ Ø3             BCS  NODIG          ;BRANCH IF NO DIGIT
94AE: 4C 3E D9          JMP  GOTO           ;EXECUTE APPLESOFT'S GOTO
94B1: C9 28    NODIG    CMP  #'(            ;PARENTHESIS ?
94B3: DØ 2D             BNE  NOIND          ;BRANCH IF NOT
94B5: 2Ø B2 DE          JSR  PARCHK         ;EVALUATE EXPRESSION
94B8: 24 11             BIT  VALTYP         ;STRING EXPRESSION ?
94BA: 3Ø 13             BMI  STRING         ;BRANCH IF SO
94BC: 24 A2             BIT  SIGN           ;RESULT MUST BE POSITIVE
94BE: 3Ø Ø9             BMI  ILL            ;ERROR IF NOT
94CØ: 2Ø 52 E7          JSR  GETADR         ;CONVERT FAC TO INTEGER
94C3: 4C 41 D9          JMP  GOTO+3         ;EXECUTE APPLESOFT'S GOTO
94C6: 4C C9 DE SYN      JMP  SYNT           ;SYNTAX ERROR
94C9: 4C 99 E1 ILL      JMP  ILLQ           ;ILLEGAL QUANTITY ERROR
94CC: 4C 7C D9 UNDEF    JMP  UNDEFS         ;UNDEFINED STATEMENT
94CF: 2Ø ØØ E6 STRING   JSR  FREFAC         ;FRE TEMP DESCRIPTOR
94D2: AØ Ø2             LDY  #$Ø2
94D4: B1 AØ    STXTP    LDA  (DESCPTR),Y    ;SET TEXTPOINTER TO START
94D6: 99 B7 ØØ          STA  TXTPTR-1,Y     ;OF LABEL
94D9: 88                DEY
94DA: DØ F8             BNE  STXTP
94DC: B1 AØ             LDA  (DESCPTR),Y    ;GET LENGTH
94DE: FØ E9             BEQ  ILL            ;MUST BE <> Ø
94EØ: AA                TAX
94E1: E8                INX
94E2: 86 FA    NOIND    STX  TEMP5          ;SAVE LENGTH+1
94E4: A5 B8             LDA  TXTPTR         ;SAVE TEXTPOINTER IN TEMP1
94E6: 85 Ø6             STA  TEMP1
94E8: A5 B9             LDA  TXTPTR+1
94EA: 85 Ø7             STA  TEMP1+1
94EC: AØ Ø4             LDY  #$Ø4
94EE: B9 BE 95 DATAMO   LDA  DATAS-1,Y      ;INIT LABEL VARIABLE
94F1: 99 FB Ø2          STA  IBUFP-1,Y
94F4: 88                DEY
94F5: DØ F7             BNE  DATAMO
94F7: CA       DECXX    DEX                 ;COUNT # OF CHARS
94F8: FØ 2Ø             BEQ  ADDDOL         ;BRANCH IF END OF LABEL
94FA: B1 B8             LDA  (TXTPTR),Y     ;GET NEXT CHAR
94FC: FØ 1C             BEQ  ADDDOL         ;EOL/EOS SYMB IS TERMINATOR
94FE: C9 3A             CMP  #':
95ØØ: FØ 18             BEQ  ADDDOL
95Ø2: C9 22             CMP  #'"            ;QUOTE ?
95Ø4: DØ Ø8             BNE  NOSKIP         ;BRANCH IF NOT
95Ø6: E6 B8             INC  TXTPTR         ;ADVANCE TEXTPOINTER
95Ø8: DØ ED             BNE  DECXX
95ØA: E6 B9             INC  TXTPTR+1
95ØC: DØ E9             BNE  DECXX          ;ALWAYS
95ØE: C9 2C    NOSKIP   CMP  #',            ;COMMA IS SEPARATOR
951Ø: FØ Ø8             BEQ  ADDDOL
9512: 99 FC Ø2          STA  IBUFP,Y        ;SAVE FIRST 2 CHARACTERS
9515: C8                INY                 ;OF LABEL IN INPUT BUFFER
9516: CØ Ø2             CPY  #$Ø2
9518: DØ DD             BNE  DECXX
951A: A9 FC    ADDDOL   LDA  #<IBUFP
951C: 85 B8             STA  TXTPTR         ;SET TXTPTR TO INPUT BUFFER
951E: A9 Ø2             LDA  #>IBUFP
952Ø: 85 B9             STA  TXTPTR+1
9522: 2Ø E3 DF          JSR  PTRGET         ;SEARCH LABEL VARIABLE
9525: 85 AØ             STA  DESCPTR        ;SAVE POINTER TO DESC
9527: 84 A1             STY  DESCPTR+1
9529: A5 B8             LDA  TXTPTR         ;CHECK TEXTPOINTER
952B: C9 FF             CMP  #<IBUFP+3
952D: DØ 97             BNE  SYN            ;SYNTAX ERROR IF ILLEGAL LABEL
952F: AØ Ø4             LDY  #$Ø4
9531: B1 AØ             LDA  (DESCPTR),Y    ;PNTR TO GOTO/GOSUB PRESENT ?
9533: FØ ØE             BEQ  LABELINI       ;BRANCH IF NOT
9535: 85 9C             STA  LOWTR+1        ;COPY POINTER IN LOWTR
9537: 88                DEY
9538: B1 AØ             LDA  (DESCPTR),Y
953A: 85 9B             STA  LOWTR
953C: 2Ø 81 95          JSR  CIMPARE        ;COMPARE LABELS
953F: AØ Ø2             LDY  #$Ø2           ;PREPARE FOR UPDATE
9541: BØ 13             BCS  UPDATE         ;BRANCH IF MATCH
9543: 2Ø 64 95 LABELINI JSR  SEARCH         ;SEARCH PROGRAM FOR LABEL
9546: 9Ø 84             BCC  UNDEF          ;UNDEFINED STMNT IF NOT FOUND
9548: C6 FF             DEC  TEMP7+1        ;SET SEARCH INDICATOR TO $FF
954A: AØ Ø4             LDY  #$Ø4
954C: A5 9C             LDA  LOWTR+1
954E: 91 AØ             STA  (DESCPTR),Y    ;SAVE POINTER TO START LINE
955Ø: 88                DEY                 ;IN VARIABLE SPACE
9551: A5 9B             LDA  LOWTR
9553: 91 AØ             STA  (DESCPTR),Y
9555: 88                DEY
9556: B1 9B    UPDATE   LDA  (LOWTR),Y      ;INSTALL NEW LINE NO
9558: 85 75             STA  CURLIN
955A: 85 5Ø             STA  LINNUM         ;SAVE IN LINNUM TOO FOR LIST
955C: C8                INY
955D: B1 9B             LDA  (LOWTR),Y
955F: 85 76             STA  CURLIN+1
9561: 85 51             STA  LINNUM+1
9563: 6Ø                RTS                 ;BACK TO CALLER
                *
                * SEARCH : FIND MATCHING & > LABEL LINE
                *   EXIT WITH CC IF NO MATCH, ELSE CS
                *
9564: A6 67    SEARCH   LDX  TXTTAB         ;GET PNTR TO START PROGRAM
9566: A5 68             LDA  TXTTAB+1
9568: AØ Ø1             LDY  #$Ø1
956A: 86 9B    CNTSR1   STX  LOWTR          ;UPDATE SEARCH POINTER
956C: 85 9C             STA  LOWTR+1
956E: B1 9B             LDA  (LOWTR),Y      ;END OF PROGRAM ?
957Ø: FØ 42             BEQ  MISMATCH       ;EXIT WITH CC IF SO
9572: 2Ø 81 95          JSR  CIMPARE        ;COMPARE LABELS
9575: BØ 47             BCS  MATCH          ;BRANCH IF EQUAL
9577: AØ ØØ             LDY  #$ØØ
9579: B1 9B             LDA  (LOWTR),Y
957B: AA                TAX
957C: C8                INY                 ;GET POINTER TO NEXT LINE
957D: B1 9B             LDA  (LOWTR),Y
957F: DØ E9             BNE  CNTSR1         ;ALWAYS
                *
                * COMPARE : CHECK ON & > AND COMPARE LABEL BEHIND
                *   & GOTO/GOSUB WITH LABEL BEHIND & >. UPDATE
                *   TXTPTR TO END OF & > STATEMENT IF MATCH.
                *   EXIT WITH CC IF NO MATCH, ELSE CS
                *
9581: AØ Ø4    CIMPARE  LDY  #$Ø4
9583: B1 9B             LDA  (LOWTR),Y      ;LOWTR POINTS TO START LINE
9585: C9 AF             CMP  #AMPT          ;& TOKEN ?
9587: DØ 2B             BNE  MISMATCH       ;BRANCH IF NOT
9589: C8                INY
958A: B1 9B             LDA  (LOWTR),Y
958C: C9 CF             CMP  #GRTT          ;'GREATER THAN' TOKEN ?
958E: DØ 24             BNE  MISMATCH       ;BRANCH IF NOT
959Ø: A5 9B             LDA  LOWTR          ;NOW COMPARE LABELS
9592: 69 Ø5             ADC  #$Ø5           ;CARRY IS SET
9594: 85 B8             STA  TXTPTR         ;INIT TXTPTR TO LOWTR+6
9596: A5 9C             LDA  LOWTR+1
9598: 69 ØØ             ADC  #$ØØ
959A: 85 B9             STA  TXTPTR+1
959C: A6 FA             LDX  TEMP5          ;GET LENGTH OF LABEL
959E: AØ FF             LDY  #$FF
95AØ: C8       CMPNXT   INY
95A1: CA                DEX
95A2: FØ 12             BEQ  ADD            ;READY IF AT END OF LABEL
95A4: B1 Ø6             LDA  (TEMP1),Y      ;GET CHAR OF LABEL
95A6: FØ ØE             BEQ  ADD            ;BRANCH IF EOL
95A8: C9 3A             CMP  #':            ;COLON ?
95AA: FØ ØA             BEQ  ADD            ;BRANCH IF EOS
95AC: C9 2C             CMP  #',            ;COMMA ?
95AE: FØ Ø6             BEQ  ADD            ;END REACHED IF SO
95BØ: D1 B8             CMP  (TXTPTR),Y     ;COMPARE WITH CURRENT LABEL
95B2: FØ EC             BEQ  CMPNXT         ;CONT COMPARISON IF MATCH
95B4: 18       MISMATCH CLC                 ;INDICATE FAILED SEARCH
95B5: 6Ø                RTS
95B6: 2Ø 98 D9 ADD      JSR  ADDON          ;UPDATE TEXTPOINTER
95B9: 2Ø B7 ØØ          JSR  CHARGOT        ;MUST POINT TO : OR Ø
95BC: DØ F6             BNE  MISMATCH       ;ELSE NO MATCH
95BE: 6Ø       MATCH    RTS                 ;RTS WITH CARRY SET
                *
95BF: 2Ø 2Ø 24 DATAS    ASC  '  $'
95C2: ØØ                DFB  $ØØ
                *
                * END OF PROGRAM
                *

--End assembly--

451 bytes

Errors: Ø
```

```
            KEY PERFECT 4.Ø
                RUN ON
               AMPERGO
     ===========================
       CODE       ADDR# - ADDR#
      ------      -------------
       242B        94ØØ - 944F
       26A8        945Ø - 949F
       26E1        94AØ - 94EF
       2853        94FØ - 953F
       235F        954Ø - 958F
       17C6        959Ø - 95C2
    PROGRAM CHECK IS : Ø1C3
```

```
            CHECK CODE 3.Ø

            ON: AMPERGO
            TYPE: B

            LENGTH: Ø1C3
            CHECKSUM: 3F
```

LISTING 2: AMPERGO.DEMO1

```
10Ø  REM ***********************
20Ø  REM *    AMPERGO.DEMO1    *
30Ø  REM * BY CORNELIS BONGERS *
40Ø  REM * COPYRIGHT (C)  1984 *
50Ø  REM * BY MICROSPARC, INC. *
60Ø  REM * CONCORD, MA   Ø1742 *
70Ø  REM ***********************
80Ø  PRINT  CHR$ (4)"BRUN AMPERGO"
90Ø  REM
10ØØ  REM MAIN PROGRAM (ADD TWO DIGITS)
11ØØ  REM
12ØØ  &  GOSUB INIT
13ØØ  &  GOSUB USER INPUT :FDIG = DIG
14ØØ  &  GOSUB USER INPUT
15ØØ  PRINT : PRINT "THE SUM OF ";
16ØØ  PRINT FDIG;" AND ";DIG;" IS ";FDIG + DIG
17ØØ  END
18ØØ  REM
19ØØ  REM USER INPUT
20ØØ  REM
21ØØ  &  > USER INPUT
22ØØ  PRINT : PRINT "ENTER A DIGIT ";
23ØØ  &  > DIGIT: GET DIG$
24ØØ  CALL CLREOS
25ØØ  &  GOSUB CHECK
26ØØ  IF  NOT ER THEN  PRINT DIG$: RETURN
27ØØ  &  GOSUB "ERROR": &  GOTO DIGIT
28ØØ  REM
29ØØ  REM CHECK INPUT
30ØØ  REM
31ØØ  &  > CHECK:ER = Ø
32ØØ  LET DIG =  VAL (DIG$): IF (DIG) THEN  RETURN
33ØØ  IF DIG$ <  > "Ø" THEN ER = 1
34ØØ  IF DIG$ =  CHR$ (3) THEN  STOP
35ØØ  RETURN
36ØØ  REM
37ØØ  REM INITIALIZE
38ØØ  REM
39ØØ  &  > INIT: TEXT : HOME
40ØØ  LET CLREOS =  - 958: RETURN
41ØØ  REM
42ØØ  REM ERROR HANDLER
43ØØ  REM
44ØØ  &  > "ERROR"
45ØØ  LET HPO =  POS (Ø) + 1:VPO =  PEEK (37) +
      1
46ØØ  HTAB 1Ø: VTAB 24: INVERSE
47ØØ  PRINT "NOT A DIGIT, REENTER";
48ØØ  HTAB HPO: VTAB VPO: NORMAL
49ØØ  RETURN
```

LISTING 3: AMPERGO.DEMO2

```
10Ø  REM ***********************
20Ø  REM *    AMPERGO.DEMO2    *
30Ø  REM * BY CORNELIS BONGERS *
40Ø  REM * COPYRIGHT (C) 1984  *
50Ø  REM * BY MICROSPARC, INC  *
60Ø  REM * CONCORD, MA. Ø1742  *
70Ø  REM ***********************
80Ø  PRINT  CHR$ (4)"BRUN AMPERGO"
90Ø  REM THIS PROGRAM FORCES CONTINUOUS LABEL
       SEARCHING
10ØØ  &  GOTO TEST1: REM  INIT LOC 255 TO 255
11ØØ  &  > TEST1: PRINT  PEEK (255): &  GOTO
       TEST2
12ØØ  &  > TEST2: PRINT  PEEK (255): &  GOTO
       TEST1
```

LISTING 4: AMPERGO.DEMO3

```
10Ø  REM ***********************
20Ø  REM *    AMPERGO.DEMO3    *
30Ø  REM * BY CORNELIS BONGERS *
40Ø  REM * COPYRIGHT (C) 1984  *
50Ø  REM * BY MICROSPARC, INC  *
60Ø  REM * CONCORD, MA. Ø1742  *
70Ø  REM ***********************
80Ø  PRINT  CHR$ (4)"BRUN AMPERGO"
90Ø  REM THIS PROGRAM USES THE LABEL-POINTERS
10ØØ  &  GOTO TEST1: REM  INIT LOC 255 TO 255
11ØØ  &  > TEST1: PRINT  PEEK (255): &  GOTO
       TTEST2
12ØØ  &  > TTEST2: PRINT  PEEK (255): &  GOTO
       TEST1
```