The SAM76 Language

by Ancelme Roichel

This text was formatted from a raw text file employing procedures which make use of the

<ncra> 21

SAM76 language processor

Typographical and code assignment table

	0 1 2 3 4 5 6 7	8 9 A B C D E F
	0 0 0 0 1 1 1 1	2 2 2 2 3 3 3 3
	0 2 4 6 0 2 4 6	0 2 4 6 0 2 4 6
0 00	0 @ P ` p	0 @ P ` p
1 01	! 1 A Q a q	! 1 A Q a q
2 02	" 2 B R b r	" 2 B R b r
3 03	# 3 C S c s	# 3 C S c s
4 04	$ 4 D T d t	$ 4 D T d t
5 05	% 5 E U e u	% 5 E U e u
6 06	& 6 F V f v	& 6 F V f v
7 07	' 7 G W g w	' 7 G W g w
8 10	(8 H X h x	(8 H X h x
9 11	) 9 I Y i y	) 9 I Y i y
A 12	* : J Z j z	* : J Z j z
B 13	+ ; K [k {	+ ; K [k {
C 14	, < L \ l |	, < L \ l |
D 15	- = M] m }	- = M] m }
E 16	. > N ^ n ~	. > N ^ n ~
F 17	/ ? O _ o	/ ? O _ o

The SAM 76 Language

The SAM76 language combines into a single interpretive processor characteristics of two different string and general purpose macro generators and one (or more) infix operator mathematical systems.

SAM76 was primarily inspired by the "M6 MACRO PROCESSOR" designed by M. D. Mc Ilroy and R. Morris of the Bell Telephone Laboratories. A description authored by Andrew D. Hall is given in the Bell Lab. Computing Science Technical Report #2, and other places.

The second source of inspiration came from the syntax of "GPM - a GENERAL PURPOSE MACRO GENERATOR" described by its author - C. Strachey.

GPM syntax very significantly improves the interface between the user and the language by eliminating an awkwardness inherent in the design of M6. This awkwardness arises out of the fact that in the M6 language the specification for the action to be performed on the result of an expression evaluation is at the right hand end of the expression to be evaluated; this requirement causes the need for a unique pair of characters to be designated for the purposes of "quotation"; consequently when the writer of procedures in M6 is terminating the writing of an expression he must mentally reconstruct the sequence of alternations of "quoted", "active" and "neutral" expressions which may be nested one within the other.

In GPM, where the nature of the expression is specified at the left end, only one character need be identified to serve to provide the right hand boundary for all three types.

This means that the user, when closing out his writings, need only supply enough such identical characters (plus a few for good measure) to enable correct action to subsequently take place.

An universal lament of users of both M6 and GPM relates to the extremely unnatural method in which arithmetic expressions must be organized. The simplest arithmetic expression in normal "infix" notation becomes a tortuous involved nest of strange symbols when expressed in either M6 or GPM. A simple solution exists, is incorporated in SAM76 and forms the third element of the language.

In addition inspiration for the selection and design of a number of the resident functions came from a variety of sources (see reference list).

Acknowledgement is made of the contribution made by L. Peter Deutsch [U.C. Berkeley] who conceived of nesting an expression calling for input inside an expression which causes output [Ref. 3] thus:

{OUTPUT,{INPUT}}

Finally acknowlegement and great appreciation is expressed for:
The help received from Neil Colvin [M.I.T.] who provided me with an 8080 simulator for the PDP-10 and who coded a preliminary version of the language to run in an 8080 based microprocessor system contributed also materially to the development of this language.

Barry Lubowsky [Rider College] who allowed me to make use of a PDP-10 and who was most helpful with some system software problems.

Roger Amidon [Delta Data Systems], whose actual 8080 based "ALTAIR" and "IMSAI" microcomputers were used to test and debug the software, was also most helpful in developing a family of "people" oriented mnemonics and functions particularly well suited for the "amateur" user.

Other people who contributed to various details and who "people" tested the language are Carl Galletti [T.D.L.], Marty Nichols [Newsweek], and Tom Kirk [N.J. Bell].

References

1. Strachey, C. - "A general purpose macrogenerator", Computer Journal, Vol. 8, No. 3, Oct. 1965 p. 225; |19651000|.

2. Hall, Andrew D. - "The M6 macroprocessor", Bell Telephone Labs; Computer Science Report No. 2; 1971;

3. Kagan, Claude A. R. - "A string language Processor for small machines", Proceedings of the ACM SIGPLAN Symposium on the Pedagogical Applications of small Computers, University of Kansas, Lawrence, Kansas |19711118|.

Notice This description of the SAM76 language is substantially the same that is available from the IEEE Computer Society Repository, R 76-301 - August 1976.

The three elements of SAM76

The first element is designed around the characteristics described by STRACHEY in his GPM (General Purpose Macro Generator) language. **GPM - S76**

Strachey defined its general forms as follows:

Active %function,arguments, ... :

Neutral %VAL,function,arguments, . . . :

In order to avoid ambiguity and to provide consistency the above is changed for the purposes of SAM76 as follows:

Active %function,arguments, ... /

Neutral &function,arguments, ... /

Protected ! . . . protected string . . . /

In addition changes and extensions are made to the language originally described by Strachey. The modified form incorporated in SAM76 is to be known as S76; this is a fully operational proper subset of SAM76.

The syntax used in connection with the S76 system is to be referred to as the "S" syntax of SAM 76.

The second element is substantially as described by Andy HALL for the M6 language attributed to Mc ILROY and MORRIS: **M6 - M76**

Active #function,arguments, ... :

Neutral #function,arguments, ... ;

Except for different function mnemonics no change was made in the syntax and notation of M6 for the purposes of SAM76.

In consequence routines and texts written for M6 processors distributed by the Bell Labs outside the Bell System should be compatible with SAM76.

Compatibility with M6 is insured by enabling user to stipulate whether "resident" or "user defined" functions are to be first searched whenever an expression is to be evaluated.

Expressions written using the M76 syntax will first cause a search of the user defined functions; if no user defined function of that name exists then a search is made of the resident functions. Expressions written using the S76 syntax will first search the resident functions.

Although there is syntactic compatibility with M6, the implementation in SAM76 is designated M76 because of the generally different complement of resident functions, and the significant enhancement of the scan algorithm.

The syntactic form of expressions based on the M76 system is to be referred to as the "M" syntax.

An infix operator system characterized in general by the following example for a simple arithmetic expression: **A76**

Example [5 + 5 * (144/12)]

The syntax of the foregoing system is to be referred to as the "A" syntax. The evaluation rules and internal syntax may vary between systems for this sub system; a method of identification is provided within the framework of SAM76.

Input - Activating - Scanning and Output

Input and Output The flow of characters, be they procedures (programs) or just plain text between the SAM76 processor and either the user or other sources or destinations is under the control of two resident SAM76 functions:

Input from user (or source) to SAM76 is under control of the "input string" function whose mnemonics are "IS"; this may be used in either the "S" or "M" syntax either as an active or neutral function thus:

type	active		neutral
S76	%IS/	or	&IS/
M76	#IS:	or	#IS;

Output from the SAM76 processor to the user (or other destination) is controlled by the resident "output string" function, mnemonic of which is "OS"; active or neutral use in both the "S" and "M" syntax is:

type	active		neutral
S76	%OS/	or	&OS/
M76	#OS:	or	#OS;

Initially and whenever its task has been completed, the SAM76 processor automatically loads a "restart" expression which is scanned and provides the means for accepting input and delivering output; this expression is initially defined as the S76 expression: **The "restart" expression**

&os,%is//=

The initial or standardized condition makes use of the equal sign "=" to ACTIVATE the processor; this "activating" character may be changed by the user through execution of the appropriate SAM76 resident function.

Activating the processor

Scanning

When expressions are nested one within the other, the SAM76 processor first executes the innermost, proceeding from left to right.

The innermost in the "restart" expression is:

```
%is/
```

and therefore input is sought; the user input is terminated with an activating character, the (=) will be shown to remind the user how to terminate his input.

Once the input is terminated, it is in turn examined by the scanner for possible further execution until finally the outermost expression of the "restart" expression is encountered:

```
&os, ... /
```

where the series of periods represents the result of processing the user input and in fact will be the output that is to be given the user.

Quoting or Protection

Frequently the user may wish to incorporate in his text certain characters which have special meaning to the scanning system of SAM76. Typical of these are the characters used to indicate the start and end of expressions. These are known as "warning characters".

For standardization purposes a processor for the SAM76 language is initialized with the following assignment of "warning characters":

Abbv.	Char.	Octal	Purpose
SA	%	45	Begin S76 active expression
SN	&	43	Begin S76 neutral expression
SP	!	41	Begin S76 protected string
SE	/	52	End S76 expr. or protected string
MB	#	43	Begin M76 expression
MA	:	72	End M76 active expression
MN	;	73	End M76 neutral expression
AS	,	77	Argument separator
QC	@	100	Quote single character
BQ	<	74	Begin quoted string - Pair 1
EQ	>	76	End quoted string - Pair 1
BP	(	50	Begin protected string - Pair 2
EP	)	51	End protected string - Pair 2
BA	[	133	Begin "A - syntax" expression
EA	]	135	End "A - syntax" expression

Quoting or protecting a string

In both the "S" and "M" forms of SAM76, the equivalent of string quotes may be provided by any pair of characters designated for that purpose. Initially two such pairs are defined and consist of the following balanced pairs of enclosures:

```
< . . . . > or ( . . . . )
```

Strings of symbols found between the above illustrated enclosures are said to be "quoted" or "protected" and are completely ignored during the scanning process.

The two rules that follow are applied to the scanning process with respect to the use of "quoting characters".

2

When an unprotected character designated as a "quoting character" is encountered in the left to right scan of an expression, the other character of the pair is then defined as the character which will end the quoted string; this means that for the two initially designated pairs shown above, quoting or protection may also be achieved as follows:

```
> . . . . < or ) . . . . (
```

1

In the process of finding the character which is to terminate a particular quoted string any occurrence of the same character as that which identified the beginning of that quoted string must be balanced by a corresponding "end of quoted string" character.

This is to be differentiated from the action which takes place in connection with the "protected string" of the "S" syntactic form:

```
! . . . protected string . . . /
```

In the foregoing a tally is made of "unquoted" warning characters in order to find the slant sign "/" which matches the exclamation mark:

The tally count is increased by a count of one - % & ! #

The tally count is decreased by a count of one - / : ;

It should, however, be noted that the "warning characters" (symbols used to delineate bounds of expressions) shown in the foregoing illustrations and examples are changeable by the user and are only representative of initial or standardized conditions.

Any of the foregoing "warning characters" may be changed using the appropriate SAM76 resident function; the abbreviations listed earlier are used to identify the particular warning character it is desired to change.

Quoting or protecting a single character

When an unprotected "character quoting symbol" is encountered, then the immediately following character is considered to be quoted. This is particularly useful when the user wishes to quote single characters, such a warning character, without having to maintain a balance of quoting character pairs.

Initially the "character quoting symbol" is set to be the "commercial at" (@).

Using the SAM76 syntax, functions, and trace

In the examples that follow the | ... | arrangement is used only to delineate output at the user terminal; if for instance the user wished to add two numbers making use of the "A" syntactic form, the following might be observed:

```
[ 5 + 5 ]=|10|
```

M76 example In the following example the M76 syntax is used to create a "macro" which when invoked will square any desired number:

```
#DT,SQ,<#MU,q1,q1:>:=          DT means Define Text
                               MU means Multiply
                               SQ is name given to text

#PT,SQ,q1:=                    PT means Partition Text
```

There are four possible actions which may ensue when the macro, or user defined function named SQ is to be invoked:

```
1.      #SQ,5:=|25|            value returned is 25

2.      #FT,SQ,5:=|25|         FT - Fetch Text, value 25

3.      #SQ,5;=|25|            value returned is 25

4.      #FT,SQ,5;|#MU,5,5:|    value is Text of SQ
```

S76 example The foregoing example is repeated below making use of the S76 syntax; however to illustrate its use, the infix system is used for the actual multiply function:

```
%DT,SQ,![q1 * q1]//=           q1 is dummy variable

%PT,SQ,q1/=                    q1 becomes partition valu
e
1
```

The four invocations of SQ are shown below:

```
1.      %SQ,5/=|25|            value is 25

2.      %FT,SQ,5/=|25|         value is again 25

3.      &SQ,5/=|25|            value is still 25

4.      &FT,SQ,5/=|[ 5 * 5 ]|  value is now "text" of SQ
```

Resident Functions In general the mnemonics used in SAM76 to identify the resident functions are composed of two alphabetic characters. This is not a restriction imposed on the language, and three or more alphabetic characters may be used to identify resident functions. In fact the user may wish to employ the "@F" ("wh@" (what) Functions) mnemonic to ascertain the mnemonics in actual being in his system.

It should be remembered that when the "@" sign serves the purpose of "quoting single character" then the expression to list the resident functions would be:

```
M76     #@@F, ... :

S76     %@@F, ... /
```

where " ... " is the string of characters which will precede each resident function name in the result of executing this expression.

Descriptions of the functions and examples in which either S76 or M76 syntax is applicable employ the | (Long Vertical Mark) character to bound the expression. In actual use the | is replaced by the appropriate pair of warning characters.

In the case of "null" valued functions where the neutral form of the expression is to be illustrated, the \ "reverse slash" is used in lieu of the |.

The formal description of the "DT" - "Define Text" function is given below to illustrate the manner of presenting SAM76 resident functions.

|DT,t,s,dl| Define Text with id number

This function places in the "text area" the string denoted by "s" in the expression shown above; this text is given the name denoted by "t". An arbitrary identification decimal number denoted by "dl" may also be included as part of the "text" record. Previously defined text with same name is erased and replaced with this definition.

Examples:

```
M76     #DT,NAME,THE SKY IS BLUE:

S76     %DT,NAME,THE SKY IS BLUE/
```

In the foregoing examples no id. number was desired, and consequently the default value of zero is stored as part of the record created by this function.

The "VT" View Text function Texts in the "text area" may be examined without evaluation using the "vt" function. This examination will display location and value of any "partitions", "multi-partitions", and the text divider(s) when at a location other than the head (or extreme left) end of the text.

Initial conventions for the display within a "text" of partitions, multi-partitions and text divider(s) are:

```
[1], [2], ... [d]              Partitions

[#1], [#2], ... [#d]           Multi-partitions

[^]                            Text Divider
```

Trace When in the "Trace" condition, active expression about to be evaluated will be shown bounded by the |, neutral expression by \, regardless of the syntax (M or S) of the expression.

"id" numbers The M6 language provides a unique identification number for each of the resident functions of the language, and means for the user to assign a similar type of number to defined functions. This numbering concept is preserved in SAM76; numbers between 1 and 99 inclusive are not assigned to any resident SAM76 function unless its name and operation is absolutely identical to its M6 counterpart. This permits user defined functions to be written to mimic M6 functions if desired. A list of M6 functions and their assigned "id" numbers is given in an appendix.

User defined function To illustrate, the M6 function "DEF" might be mimicked in the SAM76 system as follows:

```
1a.     %DT,DEF,!%NI,<&>,<%>/DTmpl</>//

2.      %MP,DEF,mpl/
```

examination of the defined text (or function) DEF can be made using the SAM76 "VT - View Text" resident function thus:

```
3.      %VT,DEF/=|%NI,<&>,<%>/DT[#1]</>|
```

in the foregoing "mpl" is a dummy for a "multi-partition" which will be created by the second expression and visualized by the VT function as [#1].

The function "NI - Neutral Implied" is used to provide the expression using DT with either the active or neutral warning character as used with an expression containing "DEF" as its operative argument.

Alternatively expression (1a) could have been written in one of the following ways,with exactly the same end effect:

```
1b.     %DT,DEF,<%NI,@&,@%/DTmp1/>/

1c.     #DT,DEF,<#NI,@&,@%:DTmp1/>:

1d.     #DT,DEF,<#DTmp1%NI,@:,@:/>:

1e.     #DT,DEF,<#DTmp1#NI,<;>,<:>:>:
```

Algorithms in the SAM 76 Language

introduction In the algorithms described and illustrated in this section, use is made of the procedure named "print" to display the script of the functions, expressions, and other text strings.

The "print" function is useful in that it augments the capability of the "vt - view text" function by prefixing each individual "text" displayed with its name bounded by angle brackets; assuming that texts have been created as shown:

```
        ~~~~~~~~~~~~~~~~~~~~~~~~
        {}  %dt,a,apple/=
        {}  %dt,b,bottle/=
        {}  %pt,b,t/=
        {}
        ~~~~~~~~~~~~~~~~~~~~~~~~~~
```

The use of "print" would cause the following display:

```
< a >
apple

< b >
bo[1][1]le
```

print The need and use of the "print" expression was described earlier; the expression to accomplish this task uses a multi partition value #2.

Typical use to print out name and contents of the text area which in this example only contains the "print" expression would be:

```
        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        {}  %print%lt,(,)//=
        {}  < print >
        {}  %ii,[1],,,!os,
        {}  (< [1] >)/%vt,[1]/%os,
        {}  /%print[#2]///
        {}
        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

do "n" Times A simple way of doing some procedure "proc" some desired "n" number of times is exemplified by the use of "ds - duplicate string" in the expression named "dnt":

```
< dnt >
%ds,[1],!%proc///
```

factorial The simple recursive function named "fac" generates the factorial of any number which replaces partition value 1 in its use.

```
< fac >
%ig,1,[1],1,!%mu,[1],%fac,%su,[1],1/////
```

```
        ~~~~~~~~~~~~~~~~~~
        {}  %fac,5/=120
        {}
        ~~~~~~~~~~~~~~~~~~~~~~
```

exponential This recursive function named "exp" is used to generate the exponential of "m" to the "n"th. power; in use "m" replaces partition value 1, and "n" partition value 2.

```
< exp >
%ig,[2],,!%mu,[1],%exp,[1],%su,[2],1////,1/
```

```
        ~~~~~~~~~~~~~~~~~~~~
        {}  %exp,3,7/=2187
        {}
        ~~~~~~~~~~~~~~~~~~~~~~~~~~
```

"m"th. root of "n" The recursive expression named "mrn" extracts the "m"th. root of any number "n"; it should be noted that this function makes use of the expression named "exp", which generates an exponential.

```
< mrn >
%ig,%exp,[3],[2]/,[1],
!%su,[3],1//,
!%mrn,[1],[2],%ad,[3],1////
```

```
        ~~~~~~~~~~~~~~~~~~~~
        {}  %mrn,2187,7/=3
        {}
        ~~~~~~~~~~~~~~~~~~~~~~
```

tensor Expression named "tensor" is used to generate a list of successive numbers separated by "."; the size of the list is specified by the integer number which replaces partition value 1 in the expression.

```
< tensor >
%ig,[1],1,!%tensor,%su,[1],1//.[1]/,1/
```

Decimal to Roman A frequently needed capability is that of generating "roman" numbers, as for text section and chapter headings; when function "dtr" is invoked with a decimal number as its argument, the value returned is the same in roman notation.

Thanks are extended to Jim Gimpel, of the Bell Telephone Laboratories for suggesting this as a most useful adjunct to the SAM76 language, as it is to the SNOBOL language.

```
< dtr >
%dt,x,[1]/%roml/%y/%et,x,y,z/

< roml >
%ig,%crd,x/,,!%rom2,%fc,x//%roml///

< rom2 >
%dt,z,[1]/%pt,z%d//%dt,z,%z%r///%pt,y%u//%dt,y,%y%t//%z//

< d >
,0,1,2,3,4,5,6,7,8,9

< r >
,,I,II,III,IV,V,VI,VII,VIII,IX

< u >
,I,V,X,L,C,D,M

< t >
,X,L,C,D,M,*,$
```

The "hex" freak can fulfill his desires by defining the "htr" expression to achieve "hex to roman" conversion:

hex to roman

```
< htr >
%dtr,%xd,[1]//
```

The user often finds it necessary to derive a series of answers resulting from the execution of some script for a range of numbers "n1" to "n2"; the script loop provides this capability by enabling the user to specify for partition 1 the name of the function, the lower of the two numbers "n1" as partition 2 and the higher number "n2" as partition 3.

loop

```
< loop >
%zs,6,[2]/%zs,7,%su,[3],[2]//%zi,7/%lpl,[1]/

< lpl >
%lp2,[1],%zq,6//%zi,6/%zd,7,,,!%lpl,[1]///

< lp2 >
!
/%ps,-4, ,[2]/    %[1],[2]/
```

For example the user wishes to ascertain the "roman" number equivalent of the decimal numbers from 105 to 116:

```
%loop,dtr,105,116/=
 105    CV
 106    CVI
 107    CVII
 108    CVIII
 109    CIX
 110    CX
 111    CXI
 112    CXII
 113    CXIII
 114    CXIV
 115    CXV
 116    CXVI
```

The expression named "hanoi" is a classical example of recursive capability; although the original is said to be based on monks moving rings between the three towers of Bhrama, the name "hanoi" seems to be better known for this puzzle.

hanoi

The expression shown below was conceived by Dick Stone; it solves the problem, with a minimum number of moves:

```
< hanoi >
%ii,[1],0,,!%hanoi,%su,[1],1/,[2],[4],[3]/!
Move Ring [1] from [2] to [3]/
%hanoi,%su,[1],1/,[4],[3],[2]///
```

Given the need to move four rings from "hither" to "yon", using "thither" as a way station, the monks needed only to follow the following sequence to do the task in a minimum length of time:

```
%hanoi,4,hither,thither,yon/=
 Move Ring 1 from hither to yon
 Move Ring 2 from hither to thither
 Move Ring 1 from yon to thither
 Move Ring 3 from hither to yon
 Move Ring 1 from thither to hither
 Move Ring 2 from thither to yon
 Move Ring 1 from hither to yon
 Move Ring 4 from hither to thither
 Move Ring 1 from yon to thither
 Move Ring 2 from yon to hither
 Move Ring 1 from thither to hither
 Move Ring 3 from yon to thither
 Move Ring 1 from hither to yon
 Move Ring 2 from hither to thither
 Move Ring 1 from yon to thither
```

Often times it is desired to execute a procedure on each and every one of some arbitrary number of "texts" in the storage area; as an example a script designed to provide a list of text names and the count of characters in each text will be illustrated; the key to this script was the use of "mt - multi part text"; it was used to create the multi partition value "#2" seen in the expression named "repeat".

repeat

```
< repeat >
%ii,[1],,,!%procl,[1]/%repeat[#2]///

< procl >
%os,
%ps,10, ,[1]/%ps,-5, ,%crd,[1]///
```

If, for instance, the only texts in the text area were "repeat" and "procl", then the use of repeat with an "lt - list text function" would appear as follows:

```
%repeat%lt,(,)//=
 repeat       26
 procl        32
```

The "skim" expression "skims off" the first appearance of each different character of the string which is placed as an argument to the expression; the result is returned as the value.

skim

```
< skim >
%dt,,![1]//%rs,%i//%et,/

< i >
%ig,%crd,/,,!%k,&fc,////

< k >
%pt,,[1]/%i,[1]/![1]/
```

A simple example of the use of "skim" follows:

```
%skim,MISSISSIPPI/=MISP
```

In his book "Algorithms in SNOBOL4" Jim Gimpel, the author, offers SKIM as an example of string manipulation; it is interesting to note the typing effort required for the same function in each language; it is assumed that a speed of ten words per minute (or one character per second) is reasonable for intellectual typing. At that rate the SAM76 script would take approximately 100 seconds, while the SNOBOL4 example would require approximately 143 seconds to type up.

SAM 76 Language Functions

The SAM76 language functions described in the following pages represent a first draft for test and evaluation purposes, and include all of the functions available in a processor for that language initially coded for the INTEL 8080 and Zilog Z-80 microprocessors.

In particular these functions properly represent the action expected for processors which have the <ncra> identification; see function [@n].

The references listed below should be used to provide information as to language syntax, method of use, and applications:

References

1. "SAM76 - A language based on "Strachey's GPM and Mc Ilroy's M6", by Claude A. R. Kagan, dated august 1976, and available as report R 76-301 from the IEEE Computer Society, 5555 Naples Plaza, Long Beach, CA, 90803.

2. "SAM76 Language Primer" to be published.

3. Notes on loading and using SAM76, from the Amateur Computer Group of New Jersey.

The author would greatly appreciate feedback, particularly if errors are found in this document, or if the definitions do not seem to correspond to the action encountered in a given processor.

Ancelme Roichel
Box 257 - RR1
Pennington - N. J., 08534

Notation used in list of functions

```
 #      Preliminary definition - no ID number assigned

|function,arg1,arg2,...,argn|   Active form of execution
\function,arg1,arg2,...,argn\   Neutral form of execution

       @                       "at" (wh@ means "what")

       c2,c1, ... ,c           Single characters
       s2,s1, ... ,s           Character strings
       s0                      Prefixing character string
       t2,t1, ... ,t           Names of Texts
       f2,f1, ... ,f           Names of Files
       d2,d1, ... ,d           Decimal Numbers
       n2,n1, ... ,n           Numbers in current radix
       x2,x1, ... ,x           Binary numbers (octal/hex)
       vz                      Value if function fails
       vt                      Value if function True
       vf                      Value if function False
       v0                      Value if function zero
       v+                      Value if function positive
       v-                      Value if function negative
       dev                     Device
       a2,a1, ... ,a           Abbreviations
       xn1,yn1, ... xn,yn      Increments in X and Y

       con                     Condition
       EQ                      Equal
       GT                      Greater Than
       LT                      Less Than
       ZE                      Zero
       GE                      Greater than or Equal
       LE                      Less than or Equal
       UC                      Upper Case
       LC                      Lower Case
```

When a radix greater than 10 is used upper case alphabetic symbols represent the ensuing integers starting with "A" being equivalent to 10 in base 10, "B" to 11, and so on. Lower case characters are used when the radix exceeds 36.

Function Definitions

238 - |@f,s0| wh@ are Functions

The value of this function is a list of built-in functions available within the system. Each function mnemonic is preceded by the string of characters symbolized by "s0". It is advisable to use the "neutral" form of expression as shown below:

```
&@@f, /={value will be list of functions}
```

Note use of two "@"; since that symbol is a warning character it must be protected, and in the above example it is used as the single character protector to protect itself.

239 - |@n| wh@ is processor ser. Number

The value returned through execution of this function is the version number of the processor being used. This number is tested by some of the functions - and may also be so tested by the user - to ascertain compatibility between scripts and run time load modules that may be moved between systems and users.

237 - |@t| wh@ is processor Title

The value of this function provides information as to the authorship of the processor being used; a typical example is:

```
&@@t/=<ncra>   indicating that the authors are:

"Neil Colvin Claude Roichel Roger Amidon.
```

159 - |ab,s1,s2,vt,vf| Alphabetic Branch

This function compares the character strings denoted by "s1" and "s2" in terms of their "ASCII" binary value; if the string "s1" has a lesser value than the string "s2" then the value of the expression is the string denoted by "vt", else the value is the string denoted by "vf"; if the value strings are expected to contain executable or syntactically meaningful symbols they should be protected within the expression.

128 - |ad,n1,n2,n3,...,n| Add

The value of this expression is the arithmetic sum of the integer numbers denoted by "n1", "n2", ... "n". Each of the arguments is examined from right to left until a symbol not representing a valid number in the current number base is encountered; non numeric symbols prefixing the arguments are ignored with the exception of those in argument "n1" which is carried into the value string of the expression.

For example:

```
%ad,dog5,cat6/={dog11}
```

Negative numbers are indicated in the usual manner with a "-" sign which must immediately precede the string of integers it applies to thus:

```
%ad,dog-5,cat6/={dog1}
```

Ommitted arguments (such as adjacent commas) have the value of the "null string" and a numeric value of zero (0) as illustrated in the following example:

```
%ad,dog,/={dog0}
```

160 - |ai,s0,s1,s2,...,s| Alphabetic Insertion

The value of this function is a character string in which the string denoted by "s1" has been inserted ahead of the first of the subsequent argument string which has a greater alphabetic value; each of the strings which form the arguments of the expression are returned with a copy of the string symbolized by "s0" preceding it.

187 - |and,x1,x2| And the bits

The value of this function is the result of performing an anding of the two binary numbers represented in the current "X" base by "x1" and "x2".

161 - |as,s0,s1,s2,...,s| Alphabetic Sort

The value of this function is a character string in which the sequence of the strings denoted by "s1", "s2", ... "s" is rearranged to be in proper alphabetic sequence; each of these strings is preceded by the string denoted by "s0" in the result:

```
%as,(,),horse,cat,dog/={,dog,cat,horse}
```

Note the use of "(,)" for "s0"; this is typical of lists that the user may wish to use in a variety of functions which have arbitrary number of arguments.

220 - |bf,f,vz| Bring File

Execution of this function serves to bring from auxiliary storage the file whose name is denoted by "f"; if this file does not exist, then the value of the function becomes the string symbolized by "vz".

113 - |ca,s| Change Activator (current)
\ca,s\ Change Activator (initial)

The purpose of this function is to change the symbol which terminates input from its current value to that denoted by the first character of the string denoted by "s"; active form is used to change the operational activating character whereas the neutral form is used to change the initial table.

195 - |cfc,d1,s| Change Fill Character schema
\cfc,d1,s\ Change Fill Char. (initial)

The purpose of this function is to enable the user to specify the number of that character which is the first one of the string denoted by "s" that will be automatically output at the beginning of each "new line"; usually these will be "nulls" but they may be any other character the user wishes to use. Active form of the expression changes the table in active use, while the neutral form changes the initial table; see [cnb].

193 - |cin,t1,d1,...,t,d| Change Id Number

To be defined and coded.

148 - |cld,t| Characters Left of Divider

The value of this function is the (decimal base) number of characters located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

191 - |cll,d| Change Line Length (active)
\cll,d\ Change Line Length (initial)

The purpose of this function is to change the number of characters that the processor will output before automatically inserting a "new line" code; the desired value is the decimal number denoted by "d" in the expression; the active form changes the actively used constant, while the neutral form changes the initial condition table; see [cnb].

133 - |cnb,d| Change Number Base (active)
\cnb,d\ Change Number Base (initial)

The purpose of this function is to change the radix of the processor's arithmetic functions; this is always expressed as the decimal number symbolized by "d" in the expression; see [qnb]; active form of the expression changes the value of the radix used during the actual operations while the neutral form changes the initial table. Note that the neutral change can only be effective while the processor is in unprotected memory, and will have no effect if the processor is executed from read only or protected memory.

266 - |cpc,t1,d1,...,t,d| Change Protection Class

To be defined and coded.

147 - |crd,t| Characters Right of Divider

The value of this function is the (decimal base) number of characters located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

203 - |cro,s1| Change Rub Out char. schema
\cro,s1\ Change Rub Out (initial)

The purpose of this function is to permit the user to specify as the first character of the string denoted by "s1" that symbol desired to serve the character delete function; in addition the user may specify the next two characters of that string to be output before and after, respectively, the deleted string of characters. The active form changes the user table, while the neutral form of this function changes the initial table; see [cnb].

132 - |ct,t1,t2,t3,...,t| Combine Texts (superseding)
\ct,t1,t2,t3,...,t\ Combine Texts (save current)

The purpose of this function is to create a text, whose name is denoted by "t1" which will contain the "concatenation" of the texts whose names are denoted by "t2", "t3", ... "t".

The active form of the expression will also delete the current text named "t1" if any; the neutral form of the expression will not cause deletion of any pre-existing texts.

250 - |cwc,s1| Change Warning Character
\cwc, ... \ Change Warn. Char. (initial)

The purpose of this function is to allow the user to select other symbols for syntactic purposes than those initially defined in the language; see [qwc]. The active form of this function changes the current user tables while the neutral form changes the initial table; see [cnb].

261 - |cws,d| Change Work Space
\cws,x\

This function allows a respecification of the upper limit of space required for function execution and text storage; the neutral form is used to allow a binary (hex, octal ...) number denoted by "x" to be entered; the active form uses the decimal number system instead; see [qws].

171 - |cx,s0,s| Character to "X"

This function returns as a value a sequence of binary numbers, each preceded by the string denoted by "s0" which result from converting the character string denoted by "s"; the conversion is in the current specification for the "X" base; see [cxb, qxb, xc].

200 - |cxb,d| Change "X" Base (active)
\cxb,d\ Change "X" Base (initial)

Through use of this function, the user may change the specification of the radix (or base) for those functions which operate in the "X" base; "d" symbolizes the decimal value of the desired new radix, and this should normally be either "2" or "4" or "8" or "16"; active and neutral forms operate with respect to the "X" base in the same manner as described for the "cnb - change number base" function.

259 - |da,s0| Date

The value of this function is the current calendar date in the sequence of "day, month, and year"; each of these elements is preceded by the string of characters denoted by "s0".

The date may be either user entered (using the "sda" function) or may be automatically generated by a system clock or calendar; see [sda].

131 - |di,n1,n2,vz| Divide

The value of this function is the integer result of dividing the number denoted by "n1", by the second number denoted by "n2"; if the result of the division would be indeterminate - "n2" = zero - then the value of the expression is that string denoted by "vz".

Treatment of non numeric, signs, and null strings is identical to that described in the "ad" function.

208 - |dq,s| Define Quote

The purpose of this function is to enable the user to specify the character that is to be recognized by the scan algorithm as an unconditional protecting character. To be further defined.

- |dr,t,a,o,v| Define Relationship

To be defined and coded.

164 - |ds,d,s| Duplicate String

The value of this expression is a concatenation of that number of copies of the string "s" which is denoted by the decimal number "d".

This function may be quite properly used to "DO" something a desired number of times as illustrated below:

```
%ds,5,!%dt,x,%ad,3,%x/////=
```

This example shows a most awkward way of creating a text named "x" which contains the product of 5 and 3.

103 - |dt,t,s,d1,d2| Define Text (superseding)
\dt,t,s,d1,d2\ Define Text (save current)

This function is used to create a "text" whose name is denoted by "t", containing the string of characters symbolized by "s"; arguments denoted by "d1", "d2" are for undefined purposes.

The active form of the expression serves also to erase the text whose name is "t" if currently in the text area; the neutral form will not cause any text deletion.

173 - |dx,d,x| Decimal to "X"

The value of this function is that number which results from converting the decimal number denoted by "d" to its equivalent in the current "X" base; the argument symbolized by "x" is undefined; see [xd, cxb, qxb].

206 - |ea,t1,t2,...,t| Erase All excepting

This function is used to erase from the text area all defined texts; excepted from this erasure are the texts whose names are denoted by "t1", "t2", ... "t"; full erasure is accomplished as shown below:

```
%ea/=
```

207 - |ed,t,d1,d2,vz| Extract "D" characters

The value of this function is a string of characters extracted from the text whose name is denoted by "t"; the first character returned is that which is found distant to the right of the text divider by the number denoted by "d1", and the number of characters returned is denoted by the number "d2".

If there are absolutely no such characters available then the value of the expression is that string denoted by "vz" which is always treated actively; this is regardless of whether the original expression was used actively or neutrally.

224 - |ef,f1,f2,...,f| Erase Files

This null valued function is used to erase from auxiliary storage the files whose names are denoted by "f1", "f2", ... "f".

151 - |ep,t,p1,p2,...,p| Erase Partitions

This function serves to delete from the text named "t" any partitions that may be found in it to the right of the internal text divider.

If arguments "p1", "p2", and so on are specified, then only those partitions with values denoted by these arguments are deleted.

- |er, ... | Express Relationship

To be defined and coded.

104 - |et,t1,t2,...,t| Erase Text
\et,t1,t2,...,t\ Erase all occurences of Text

This function is used to erase from the text area texts whose names are denoted by the arguments of the expression; the active form of the expression erases only the latest version of the named texts, should more than one version exists in the text area.

249 - |etb,s| Erase Trailing Blanks

This function, which Neil Colvin insisted be made available, serves to delete from the character string symbolized by "s" all blanks which immediately precede the occurence of "new line" codes; particularly useful when reading punched cards with many trailing blank columns.

112 - |ex,f| Exit

This function is used to exit from the processor back to a system monitor; if the expression contains any arguments, such as "f" then a file is created in auxiliary storage which contains all of the user work space including variables, working area and text area.

This file is given the name denoted by "f", and may be reloaded on any other system and the action interrupted by the execution of this function is then resumed at the same point.

Successful resumption of action is subject to compatibility as determined by verification of the system authorship and version number.

226 - |fb,f,vt,vf| File Branch

The value of this function is the string symbolized by "vt" if the file whose name is denoted by "f" is to be found in external storage; if not the value of this function is that string symbolized by "vf".

137 - |fc,t,vz| Fetch Character

The value of this function is the character to be found immediately to the right of the internal text divider of the text whose name is denoted by "t"; if such a character exists then the text divider is advanced to point to the next available character - or alternatively to the end of the "text".

If no character is available then the value of the expression is that string denoted by "vz" which is always treated actively.

138 - |fdc,t,d,vz| Fetch "D" Characters

The value of this function is that decimal number of characters denoted by "d" which is to be found to the right or left of the internal divider of the text whose name is denoted by "t"; a positive number specifies to the right, and a negative number to the left; the text divider is then relocated to point to the next character available if to the right, or to the first character returned in the value string if to the left.

Should there be absolutely no characters (not even one) available for the value, then "vz" represents the string which will be returned as the value of the expression, and this value is always treated actively.

139 - |fde,t,d,vz| Fetch "D" Elements

The value of this function is that string of characters to the right or left of the text divider of the text whose name is denoted by "t" which comprises the number of elements specified by the decimal number "d"; an element is that which is found between partitions.

The divider is then moved either to the right or left to point either to the next character to be read (if to the right) or that character which corresponds to the first character of the value string (if to the left).

If absolutely no characters are returned, then the divider is not moved and the value of the expression is that string symbolized by "vz" which is always treated actively.

140 - |fdm,t,d,s,vz| Fetch "D" Matches

The value of this function is a character string taken to the right of the text divider of the text whose name is denoted by "t"; the contents of the text "t" is scanned searching for occurences of the string denoted by "s"; when that number of occurences equal to the number "d" is found then the divider is relocated to point to the next immediately following character, and the value string is that which lies between the old and the new divider locations.

If the required number of matches is not found, then the divider is not moved, and the value of the expression is that string denoted by "vz", which is always treated actively.

If the desired number is negative, then the action takes place to the left of the divider instead of to the right.

141 - |fe,t,vz| Fetch Element

The value of this function is that string of characters which is to be found immediately to the right of the current location of the divider in the text named "t" up to the next encountered partition - or end of text.

The text divider is moved to point to the immediately available next character.

If absolutely no characters are to be found, then the value is the string denoted by "vz", which is always treated actively.

142 - |ff,t,d,vz| Fetch Field

The value of this expression is that string of characters to be found to the right of the first occurence of a partition of value symbolized by "d" until the next partition is encountered, or to the end of the text named "t"; the divider is then moved to point to the next available character.

If absolutely no characters are returned, then the divider is not moved, and the value of the expression is the string symbolized by "vz" which is always treated actively.

143 - |fl,t,s,vz| Fetch Left match

The text named "t" is searched to the left of its internal divider for a string identical to that denoted by "s"; if such a string is found then the divider is moved to point to the first character of that string and the value of the expression is that group of characters to be found between the new location of the divider and its old location.

If no such matching string is found, then the divider is not moved, and the value of the expression is the string "vz" which is always treated actively.

145 - |fp,t,xl,...,x| Fetch Partition

The value of this function is the next partition to be found in the text whose name is denoted by "t"; the text divider is moved to point to the next ensuing character.

144 - |fr,t,s,vz| Fetch Right match

The text named "t" is examined starting at the current location of the text divider for a string identical to that denoted by "s"; if such a string is found, then the divider is moved to the next character which follows this string (or the end of the text) and the value returned is that string of characters which lies between the old and the new locations of the text divider.

If no such match is found, then the value of the expression is the string denoted by "vz", always treated actively, and the divider is not moved.

106 - |ft,t,s1,s2,...,s| Fetch Text

The value of this function is the contents to be found in the text named "t" to the right of its internal text divider.

Any partitions found in that portion of the text are replaced with the strings denoted "s1", "s2", ... "s", in such a manner that partitions with value "1" are replaced by "s1", value 2 with "s2" and so forth.

210 - |ftb,t,s,vz| Fetch To Break character

The value of this function is a string of characters taken from the current location of the internal text divider of the text whose name is denoted by "t" to the first ensuing character in text "t" which is found in the string symbolized by "s"; the text divider is moved to the first character which follows that which terminated the scanning action.

If no character is returned, then the divider is not moved and the value of the expression is that string symbolized by "vz" which is always treated actively.

211 - |fts,t,s,vz| Fetch To Span character

The value of this function is a string of characters taken from the current location of the internal text divider of the text whose name is denoted by "t" to the first ensuing character in text "t" which is not found in the string symbolized by "s"; the text divider is moved to point to the first character which follows that which terminated the scanning action.

If no character is returned, then the divider is not moved and the value of the expression is that string symbolized by "vz" which is always treated actively.

212 - |hc,s| How many Characters

The value of this function is the actual number of characters comprising the string symbolized by "s".

150 - |hm,t,s| How many Matches

The value of this function is that number of occurences of the string of characters symbolized by "s" in the text whose name is denoted by "t".

149 - |hp,t,d| How many Partitions

The value of this function is the number of partitions to be found to the right of the text divider in the text whose name is "t"; if there is no explicit third argument, then the value represents the total number of partitions; if argument three is specified, then the value is that number of partitions whose value is equal to the number symbolized by "d".

114 - |ht,t| Hide Text
\ht\ Hide all Texts

The purpose of this function is to put a screen or fence in the text area such that only those texts created after that whose name is denoted by "t" are available.

This fence is removed by execution of this function with no arguments; if it is desired to conceal all of the text area, the the neutral form is to be executed.

115 - |ic| Input Character

The value of this function is one character the source of which is the currently selected input device; this may be absolutely any one single character in the system character set.

116 - |id,d| Input "D" characters

The value of this function is a stream of characters from the current selected input device; the action is terminated when that number symbolized by "d" has been received; in the case of certain sources, an End of File will also terminate.

153 - |idt,d| Input "D" Texts

This function serves to place into the text area "texts" which are loaded from various types of auxiliary storage media and created through use of the "output text" functions.

136 - |ig,d1,d2,vt,vf| If Greater

This expression compares in an arithmetic sense the two strings denoted by "d1" and "d2"; if the value of "d1" is greater than that of "d2", then the value of the expression is that string symbolized by "vt", else the value is that string symbolized by "vf".

135 - |ii,s1,s2,vt,vf| If Identical

Strings symbolized by "s1", and "s2" are compared, if found to be absolutely identical, then the value of the expression is that string symbolized by "vt"; if not identical the value is the string symbolized by "vf".

117 - |im,s1,s2,...,s| Input to Match

The value of this expression is a stream of characters coming from the currently selected input device; termination of this stream occurs as soon as one of the strings symbolized by "s1", "s2", ... "s" has been encountered; the value includes the string that caused termination.

102 - |is,dev| Input String

The value of this function is a stream of characters coming from the currently selected input device; termination occurs as soon as the currently specified "activating character" is encountered; this activating character does not become part of the value string of the function.

152 - |it| Input Text

The purpose of this function is to place in the text area one text loaded from some auxiliary storage media, which is in the format generated through use of the "output text" functions.

213 - |iw,n| Input Wait

This function sets up an internal timer which becomes effective for the next ensuing input function, such as "is", "ic", "idc", "im"; automatic termination of the input process takes place at the end of the specified time interval even though normal terminating conditions specified for these functions may not have been met.

The duration of the time interval is specified by "n" which is in seconds.

- |lef,dev| Load External Function

To be defined and coded.

216 - |lf,s0,d1,...,d| List Files

The value of this function is a list of the names of the files in auxiliary storage; each file name is preceded by the string symbolized by "s0".

Additional arguments are to be defined.

- |lr, ... | List Relationship

To be defined and coded.

105 - |lt,s0,d1,d2,...,d| List Texts

The value of this function is a list of the names of the texts to be found in the text area; each name is preceded by the string symbolized by "s0".

Arguments symbolized by "d1", "d2", ... have not been defined.

110 - |mc,d| Multi-partition Character

The value of this function is the "multi-partition character" whose value is denoted by the decimal number "d".

146 - |md,t,d| Move Divider to pos. "d"
\md,t,d\ Move Divider "d" increments

This function is used to move the text divider in the text whose name is denoted by "t". Active form serves to move the divider the number of positions from the left end of the text if "d" is positive, or from the right end if negative; the neutral form is used to advance or retreat the divider from its current location that number of positions specified by the number "d".

It should be noted that these moves are positional and do not distinguish any difference between partitions and characters; this is the only function available to position the text divider other than immediately preceding a character.

109 - |mt,t,s1,s2,...,s| Multi-part Text all matches
\mt,t,s1,s2,...,s\ Multi-part Text next match

This function serves to replace in the text whose name is denoted by "t" the strings symbolized by "s1", "s2", "s" with multi-partitions of values respectively "1", "2", ...; the neutral form of this function only replaces the first occurence of these strings found in the text.

130 - |mu,n1,n2,vz| Multiply

The value of this function is the arithmetic product of the numbers symbolized by "n1", and "n2". The same rules as to sign and non numeric matter apply for this function as they do for the "ad" function.

Should there be an overflow condition arise as a result of the execution of this function, then the value of the expression is that string symbolized by "vz".

111 - |ni,vt,vf| Neutral Implied

This function returns the string symbolized by "vt" if the last previously executed implied "fetch text" was neutral. If that last previously executed implied "fetch" was active, then the value of this function is that string symbolized by "vf".

188 - |not,x| Not (complement) the bits

The value of this function is the complement of the binary number (in the current "X" base) symbolized by "x".

209 - |nu,s1,s2,...,s| Null

The purpose of this function is to cause execution of the function whose name is denoted by "s1", with appropriate arguments "s2", ... "s"; that function is executed but any resulting value string is suppressed.

246 - |oj,s,s1,d,s2| Output Justified lines

To be defined and coded.

248 - |op,s,s1,d,s2| Output Paddded lines

To be defined and coded.

186 - |or,x1,x2| Or the bits

The value of this function is the logical or of the two binary number strings "x1" and "x2" expressed in the current "X" base.

101 - |os,s| Output String

This function causes the selected output device to output the string of characters symbolized by "s".

154 - |ot,t1,t2,...,t| Output Texts

This function outputs to the selected channel, the texts whose names are denoted by "t1", "t2", ... "t", in a format which preserves all of the internal conditions of the text; this includes location of the internal text divider, and text partitions if any.

108 - |pc,d| Partition Character

The value of this function is a "partition character" of value symbolized by the decimal number "d".

174 - |pl,s1,s2,...,s| Plot

This is a general purpose plotter or display control function; details may vary in its use as a function of the system in which it is used. Certain subfunctions as described below have been standardized:

|pl,lf,s0|	List of Plotter subfunctions
|pl,as,port,time|	Assign output port and time delay
|pl,nm|	Output plotter control information
|pl,vm|	Return as value control information
|pl,cal|	Set up for "Calcomp" plotter
|pl,tek|	Set up for "Tektronix" display
|pl,mq,x|	Rotate to quadrant "x"

162 - |ps,d,s1,s2| Pad String

The value of this function is the string "s2" to which is added either to its right or to its left enough duplications of string "s1" so that the total size of the value string becomes equal to the number symbolized by "d"; positive values of "d" pad to its right, and negative values to its left.

107 - |pt,t,s1,s2,...,s| Partition Text all matches
\pt,t,s1,s2,...,s\ Partition Text next match

The purpose of this function is to replace in the text whose name is denoted by "t" the strings symbolized by "s1", "s2", ... "s" with partitions identified by numbers which correspond to the position of the strings in the expression; that is to say "s1" is replaced by a partition value "1", "s2" by partition value "2" &c.

In its active form, this function serves to replace all occurences of the desired strings; in its neutral form only the first occurence found is so replaced for each of the specified strings.

Action takes place starting at the current location of the internal text divider which is not moved by this function.

A particular example of interest might be:

```
&pt,animal,dog,dog,dog,dog/=
```

This example would in the text whose name was animal, replace the first match with "dog" with partition value "1", the second, "2", the third with "3" and the fourth with "4".

196 - |qfc,s0| Query Fill Character schema

The value of this function is the current specification for the optional fill character output immediately after each "new line"; the parameters are each preceded by the string symbolized by "s0"; see [cfc].

194 - |qin,s0,t1,t2,...,t| Query Id Number

To be defined and coded.

197 - |qld,t| Query Left of Divider

The value of this function is the (decimal base) number of positions - characters and partitions - located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

192 - |qll| Query Line Length

The value of this function is the current specification for the line length; this specification is changed through use of the "cll" function.

134 - |qnb| Query Number Base

The value of this function is the current value of the radix for arithmetic operations and functions.

202 - |qof| Query Over Flow conditions

The value of this function are characters which may be used to identify the cause of an interrupt or overflow condition.

167 - |qp,t| Query Partition

The value of this function is the decimal identification of the next partition to be found in the text whose name is denoted by "t"; the text divider is not moved by this function.

267 - |qpc,s0,t1,t2,...,t| Query Protection Class

To be defined and coded.

198 - |qrd,t| Query Right of Divider

The value of this function is the (decimal base) number of positions - characters and partitions - located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

204 - |qro| Query Rub Out char. schema

The value of this function describes the current specification for the use of the "rub out"; see [cro].

205 - |qta| Query Text Area used

The value of this function is representative of the amount of space consumed in the text area by user defined texts.

251 - |qwc,a2,a1,...,a| Query Warning Characters

The value of this function is a list of the currently specified and functional warning characters of the language.

262 - |qws| Query Work Space
\qws\

The value of this function is the upper limit of the user work space; active form of this expression yields the value as a decimal number; neutral form yields the value in the current "X" base.

201 - |qxb| Query "X" Base

The value of this function is the currently set radix for the functions which operate in a "binary" or "X" base.

215 - |ra,d,s1,s2,s3,...,s| Return Argument

The value of this function is that string symbolized by one of "s1", "s2", "s3", ... "s" whose position corresponds to the value of the decimal number symbolized by "d".

263 - |rcp,d1,d2,s| Return Character Picture

To be coded.

166 - |ri| Restart Initialized

Execution of this function causes immediate termination of any unexecuted scripts, and causes return to the idling condition, reinitializing all normally changed user specifications.

245 - |rj,s,s1,d,s2| Return Justified lines

To be defined and coded.

252 - |rn,n| Random Number

The value of this function is a number ranging between 0 and "n", randomly selected through a computational algorithm; see [srn].

189 - |rot,d,x| Rotate the bits

The value of this function is that binary number which results from a rotation of the binary number "x" expressed in the current "X" base; the number of bits rotated is specified by the decimal number "d"; the direction is

clockwise if the number is positive, and counterclockwise if negative.

247 - |rp,s,sl,d,s2| Return Padded lines

To be defined and coded and coded.

165 - |rr,sl| Return to Restart

This function causes immediate cessation of execution of any unexecuted script, forces a return to the idling level of the processor and then returns the value of the string symbolized by "sl"; if that string is an executable expression, then said expression is executed.

163 - |rs,s| Reverse String

The value of this function is the string of characters symbolized by "s" reversed, end for end.

228 - |saf,dev| Select All File function dev.

To be defined.

158 - |sar| "Auto Return" on line feed
\sar\ no Auto Return on line feed

Execution of this function serves to enable (if active) or disable (if neutral) the automatic generation of a "carriager return" code when a "new line" code is output.

260 - |sda,da,mo,yr| Set Date

This function enables the user to set into the system the current (r any other) desired date.

199 - |sem,dev| Set "Echoplex" Mode active
\sem,dev\ "Echoplex" Mode inactive

This function enables character echo from the system if executed actively; if neutrally, then echo generation is suppressed.

222 - |sf,f,tl,t2,...,t| Store File

This function is used to place into auxiliary storage under the file name denoted "f", those texts whose names are denoted by "tl", "t2", ... "t"; on completion of this action, the named texts are erased from the text area.

If no texts are named, then the assumption is made that the entire text area is to be placed into auxiliary storage.

157 - |sfd,fun,dev| Specify Function Device

To be defined and coded.

190 - |sh,d,x| Shift the bits

The value of this function is that binary number string which results from a logical shift to the right or left of the string whose binary value, expressed in the current "X" base is symbolized by "x"; the number of shifts is specified by the decimal number "d" and the direction by its sign, positive to the right and negative to the left.

253 - |srn,n| Seed Random Number

Through use of this function the automatic generation of random numbers is initiated; the user seed number is symbolized by "n" in the above expression.

258 - |sti,tl,t2,t3| Set Time

Through use of this function the correct current (or incorrect if desired) time of day is established in the system.

129 - |su,nl,n2,...,n| Subtract

The value of this function is the result of subtracting the number symbolized by "n2" from that symbolized by "nl"; all rules indicated for sign and non numeric matter in the "ad" function apply.

231 - |sw,sl,s2,s3,...,s| Switches

Special system dependent function for user definition.

232 - |sy,sl,s2,...,s| System functions

Special system dependent function for user definition.

127 - |tb,t,vt,vf| Text Branch

If the text whose name is denoted by "t" is to be found in the text area, then the value of the function is that string symbolized by "vt", else the value is string symbolized by "vf".

257 - |ti,sl,s2| Time

The value of this function is the current time of day derived from a system clock, in the format HH MM SS, where HH represent Hours, MM represent Minutes and SS seconds; these elements are separated in the value string by the string symbolized by "sl".

125 - |tm,d| Trace Mode activated
\tm\ Trace Mode deactivated

This function is used to enable step by step execution of the scripts and functions of the language; once activated each step is initiated by depression of the "new line" key ("line feed") or a specified number of steps may be specified by the number symbolized by "d".

124 - |tma| Trace Mode All activated
\tma\ Trace Mode All deactivated

Active execution enables full trace action; neutral execution allows display only of expressions about to be executed.

168 - |tr,t,s| Trim

This function is used to replace multiple adjacent occurences of the character symbolized by "s" with a single such occurence in the text named "t".

218 - |uf,f,tl,t2,...,t| Update File

This function combines the action of "erase file" and "store file" by first storing a file successfully before erasing the old file version.

169 - |ut,cc| User Trap active
\ut\ User Trap inactive

Activating of this trap through use of this function will cause an automatic execution of a text (or function) whose name is denoted by "cc"; this automatic execution takes

place whenever the user attempts to make use of system interrupt capabilities to escape out of executing "scripts" when it is desired that such escape be inhibited.

118 - |vt,t1,t2,...,t| View Texts

This function enables examination of defined texts, indicating visually the location of internal text divider, partitions and their values, as well as multipartitions.

181 - |wc,s1,s| Write Characters

This plotter or display function generates a graphic display from a font of vectors named "s1"; the string symbolized by "s" is the text string to be plotted or displayed.

175 - |wi,xn1,yn1| Write Initialize

This null valued function serves to create a reference value for the current position of the display pointer; the "x" and "y" coordinates of this position are set equal to the decimal integer strings found in the second and third arguments respectively.

179 - |wl| Width Left

The value of this function is the incremental value of the width of a character being plotted to the left of its center of gravity.

178 - |wr| Width Right

The value of this function is the incremental value of the width of a character being plotted to the right of its center of gravity.

180 - |ws,xn1,yn1,...,xn,yn| Write Straight Lines

This function causes the generation of the appropriate control information to cause the display of one or more line segments starting at the current position of the display pointer. Successive points are referenced by consecutive incremental x and y argument pairs. The x and y components are expressed as decimal integer strings. The sign of the number indicates direction.

Non numeric matter preceding the "x" value causes certain auxiliary functions to take place as follows:

```
U       "pen up" or invisible vector
S       Scale change for "x" and "y"
A       Absolute position vector pair
I       Incremental vector pair
Q       Quadrant rotation
         0 - 3 clockwise, 4 - 7 counter clockwise
W       Character width information
         "x" left and "y" right of center
```

```
{}--------------------------------------
{}  %dt,rect,(,8,0,0,5,-8,0,0,-5)/=
{}  %ws%rect//=
{}
--------------------------------------
```

176 - |wx| Write "X" displacement

The value of this function is equal to the algebraic difference between the initial value "X" of the display pointer position and the sum of any subsequent incremental displacement values.

177 - |wy| Write "Y" displacement

The value of this function is equal to the algebraic difference between the initial value "Y" of the display pointer position and the sum of any subsequent incremental displacement values.

An effective way to return the display pointer to its original location is illustrated as follows:

```
{}----------------------
{}  %ws,U%wx/,%wy//
{}
----------------------
```

Assuming it is desired to draw a rectangle,8 units wide, and 5 units high:

170 - |xc,x1,x2,...,x| "X" to Character

The value of this function is a string of characters whose value is represented by the numbers in "X" base symbolized by "x1", "x2", ... "x".

271 - |xcf,s,x| eXperimental Change Function

This function is used to assign a user defined machine adress symbolized by the "X" base number "x" for the built in function whose mnemonic is denoted by "s".

172 - |xd,x| "X" to Decimal

The value of this function is the decimal equivalent of the number in the current value of the "X" base symbolized by "x".

255 - |xi,port| eXperimental Input

The value of this function is that number in the current "X" base which results from an attempted input from the port whose designation in the "X" base is symbolized by "port".

123 - |xj,x| eXperimental Jump

Execution of this function cause a jump to the memory location symbolized in the "X" base by "x"; a return from programs encountered at that location will cause normal reentry into the system scanner.

256 - |xo,x,port| eXperimental Output

Execution of this function enable output of the "X" base value of "x" to the port whose "X" base id is symbolized by "port".

270 - |xqf,s| eXperimental Query Function

The value of this function is the machine adress in "X" base radix of the entry point for the function whose mnemonic is denoted by the string "s".

119 - |xr,x| eXamine Register

The value of this function is the contents of the memory location symbolized in the current "X" base by "x".

121 - |xrp,x| eXamine Register Pair

The value of this function is the combined contents of the two memory locations, whose first location is symbolized in the current "X" base by "x".

120 - |xw,x1,x2| eXperimental Write in reg.

Use of this function causes writing the value "x1" into the memory location "x2"; both of these are in the current "X" base.

122 - |xwp,x1,x2| eXperimental Write reg. Pair

Use of this function causes writing of the double word data symbolized in the "X" base by "x1" in the memory location whose first word is identified by "x2".

126 - |yt,t,s,vt,vf| Ys There

This function searches the text whose name is denoted by "t" for an exact match with the string symbolized by "s"; if such a match is found, then the value of the expression is that string symbolized by "vt", else it is that string symbolized by "vf".

182 - |zd,r,v-,v0,v+| "Z" reg. Decrement and branch

Execution of this function causes the contents of a special register identified by "r" to be decreased by a count of one.; if the resulting contents of that register are greater than zero, then the expression value is that string symbolized by "v+"; if equal to zero, then the value is symbolized by "v0", and if less than zero the value becomes the string symbolized by "v-".

183 - |zi,r,v-,v0,v+| "Z" reg. Increment and branch

The action of this function is identical to that described for "zd", except that the designated register "r" is increased by a count of one prior to testing.

184 - |zq,r| "Z" reg. Query

The value of this function is the current contents of the special register designated by "r"; execution of this function does not cause any change in said register.

185 - |zs,r,n| "Z" reg. Set

This function is used to preset designated special register "r" to the number symbolized by "n".

SAM 76 – Setting up the System

There are two versions available:

Availability

a. Z80 - little less than 8K
   Uses RST0 thru RST6

b. 8080 - about 9 K
   Does not use any RST locations

In addition the source code is available from a number of ACNJ members if special adressing requirements are encountered or if it is desired to configurate a system with fewer functions, or different use of the RST locations.

Except as indicated above both versions are virtually identical, and the information that follows applies to both of these versions; in consequence user defined texts and procedures may be successfully moved between systems using either of these two versions.

Possible incompatibility may be ascertained through use of the SAM76 function which reveals the version number thus:

```
{}------------------
{}  %@@n/=21
{}
~~~~~~~~~~~~~~~~~~
```

reveals the version number of the "romable object code".

a. Romable program loads at ^h8000.

b. "Once Only" start at ^h8000 initializes variables.

c. Restarts may be made at 8003 or 0.

d. System and user variables start at ^h400

e. Workspace is above ^h600.

2 Memory utilization

I/O specifications, and user options:

a. APPLE/ZAPPLE I/O Conventions apply.

b. Normally initialized to seek and take available RAM.

System Differences

c. Special use of control codes is as follows:

^C	Return to Monitor
^X	Cancel current material typed in
DEL	Rubout (changeable by user)
^N	Shift Out to alternate character set
^O	Shift In back to regular character set.

The Line Feed key is viewed by SAM76 as being equivalent to the "NULIN" code, as specified by ASCII; use of this key serves to go to the beginning of the next line - combining the functions of "Carriage Return and Line Feed". The Carriage Return serves only to return to the beginning of the same line; this permits underlining &c.

Modifying SAM76 tables and options:

4 Fiddler's Cuide

There are two levels of modifications which the user may apply in the process of tailoring the SAM76 code to suit his system and needs:

a. Permanent.

b. Local or temporary.

The type of changes are identical but the location of the table to be changed and the time at which the table is to be changed are different.

Effectively there is a main table of variables and options located in the "romable" or protected section of code; after having executed a "GO" to ^h8000 this table is copied into the user work space starting at location ^h40C.

After this initial entry at ^h8000 entry should be made only at ^h8003 or at location 0 if the user wishes to preserve any changes made in the duplicate table at ^h40C either with monitor facilities or through the execution of SAM76 functions which are available to establish a variety of user options.

Typical Permanent Changes:

5 Useful Fiddles

Normally changes in the table at ^h800C should be limited to necessary respecification of I/O adresses, and in some special cases to the location of the beginning and or end of the user work space.

In addition the user may make permanent changes to such parameters as the maximum length of line, and the specification of the number of fill characters after each "carriage return/line feed". (Note Line Feed is the proper character used for this combined function).

Table 1 lists the primary portion of the table at ^h800C with corresponding adresses of its duplicate at ^h40C.

Table 2 lists locations that may be of interest to the user as candidates for permanent changes; since these locations may all be changed within the framework of the SAM76 language, it is suggested that changes be made first in that manner, and that the modified duplicate table at ^h40C be examined before making the desired changes in the permanent table.

TABLE 1

```
 .SBTTL /Variables & once only code/
 .PAGE
;[VAR1]
VAR1:   .BLKB   1       ; [80/4-0C] JMP
BOOT:   .BLKW   1       ; [80/4-0D] Monitor Exit
MSCI:   .BLKB   3       ; [80/4-10] Console Input
MSRI:   .BLKB   3       ; [80/4-13] Channel 2 Input
MSCO:   .BLKB   3       ; [80/4-16] Console Output
MSPO:   .BLKB   3       ; [80/4-19] Channel 2 Output
MSLO:   .BLKB   3       ; [80/4-1C] Channel 3 Output
MSCSTS: .BLKB   3       ; [80/4-1F] Console Status
MSCHK:  .BLKB   3       ; [80/4-22] IO Check
MSSET:  .BLKB   3       ; [80/4-25] IO Set
MSMCK:  .BLKB   3       ; [80/4-28] Memory Check
;
$232:   .BLKB   3       ; [80/4-2B] %sy,x1,x2,.../
$231:   .BLKB   3       ; [80/4-2E] %sw,x0,.../
        .BLKB   3       ; [80/4-31] spare
;
;
OR2:    .BLKW   1       ; [80/4-33] start active zone
TOPMAX: .BLKW   1       ; [80/4-35] text area limit
TOPLOC: .BLKW   1       ; [80/4-37] future3
SPARE4: .BLKW   1       ; [80/4-39] future4

;
```

Notes with reference to Table 1:

a. The adresses indicated are those of the actual locations which would be changed as required to suit individual operating system requirements.

b. If it is desired to disable any of the monitor functions which would be adressed via the vectors of Table 1, replace the JMP with a RET - (for instance one might wish to disable the automatic RAM grabber by putting a "RET" at ^h8027" - If this is done it becomes necessary to actually stipulate the adress of the end of user workspace by entering said adress at ^h8037).

c. $232, and $231 are locations of jump vectors for the SAM76 functions "sy" and "sw" respectively. These are intended for user applications; if not required it is wise to change the adresses for these functions at ^h802B and ^h802E respectively to the adress found at ^h8031 - this causes the message <nav-xxx> to be returned should these two functions be tested.

d. Location labelled "TOPMAX" is usually shown as 00-00; this in combination with the availability of the MEMCHECK function of the ZAPPLE monitor triggers the automatic determination of the upper limit of user work space. A non zero value at that location takes precedence and serves to restrict the user work space to that specified.

e. Location labelled "OR2" defines the beginning of the user work space; this may be changed if the user wishes to have clear space for his own purposes at it's normally initialized value of 05FE.

Caution should be exercised in changing OR2 and TOPMAX so that TOPMAX be greater than OR2 by at least ^h400 bytes.

TABLE 2 (is continuation of TABLE 1)

```
TIR:    .BLKW   1 ; 0
TO:     .BLKW   1 ; TCO
TI:     .BLKW   1 ; TCI
TT:     .BLKW   1 ; TCT
TR:     .BLKW   1 ; TCR
TE:     .BLKW   1 ; TIH
RIX:    .BLKB   1 ; N12
BIX:    .BLKB   1 ; N10
MCH:    .BLKB   1 ; "=" EQUAL
TCE:    .BLKW   1 ; FNP1
CE:     .BLKW   1 ; SCE
TM2:    .BLKW   1 ; 0
TC3:    .BLKW   1 ; 0
TCECT:  .BLKW   1 ; 0
TCEAN:  .BLKB   1 ; 0
RUBOT:  .BLKB   1 ; N177
RUBO1:  .BLKB   1 ; N74
RUBO2:  .BLKB   1 ; N76
NULCH:  .BLKB   1; null character
NULCT:  .BLKB   1; filler count
PLIN:   .BLKB   1; line length
```

Notes regarding changes in Table 2:

a. PLIN contains the number of characters on a line at which SAM76 is required to automatically generate a "NULINE" on the console device. This is usually initialized in the permanent table at 72 decimal (^h48), and is changed in the duplicate table by using the "cll - Change Line Length" function thus:

```
{}  %cll,64/=
```

would set line length at decimal 64; the permanent table may be changed prior to being placed into protected memory using the neutral form of this function.

b. NULCT, and NULCH locations are used to specify the number and the type of "null" character the user wishes to have automatically generated on the console device when a "NULIN" is output to it. This is usually initialized at zero count, and ASCII-0 for the character; to change this requires the use of the "cfc - Change Fill Character" function thus:

```
{}  %cfc,*,4/=
```

puts 4 asterisks at beginning of each new line; the neutral form of the function is used to change the permanent table.

c. RUBOT, RUBO1, and RUBO2 are used to specify the character to be used for character deletion purposes, and the symbols that are desired to surround the string of consecutively deleted characters; initially this is set to be the ASCII - DEL code and the surrounding symbols are the < and > brackets. Changing this makes use of the "cro - Change Rub Out" function thus:

```
{}  %cro,$AB/=
```

sets $ to be the deleting character and the deleted string is bracketed by A and B; the neutral form of this function may be used to change the permanent table.

d. MCH is the location of the character used to terminate entry; this character is known as the "Activator". Initially this is set to be the = sign, and is changed via use of the "ca - Change Activator" function thus:

```
{}~~~~~~~~~~~~~~~
{}  %ca,Z/=
{}
~~~~~~~~~~~~~~~
```

changes activator to be "Z"; use the neutral form to change the permanent table.

e. RIX and BIX are the current specifications for the arithmetic and logical or binary bases respectively. These locations are initialized to provide base 10 for arithmetic and base 16 or hex for logical functions. Within the framework of SAM76 they are changed using the "cnb - Change Number Base" and "cxb - Change X Base" functions; for example:

```
{}~~~~~~~~~~~~~~~~~~
{}  %cnb,7/=
{}
~~~~~~~~~~~~~~~~~~
```

sets arithmetic base to be 7.

```
{}~~~~~~~~~~~~~~~~~
{}  %cxb,8/=
{}
~~~~~~~~~~~~~~~~~~
```

sets logical or binary base to octal.

The neutral form of these functions may be used to change the permanent initialization table.

SAM76 - Loading the object code tape

The two versions are available in compressed binary form for which a special loader is required; this loader is usually to be found in normal hex loader format ahead of the binary.

The normal procedure to load is as follows:

a. Load the hex formatted loader;

b. The loader should be at ^h1000 to ^h475; simply go to ^h1000 if using APPLE/ZAPPLE monitors, otherwise change the calls to F006 to reflect your own reader input routine before running.

c. If using APPLE/ZAPPLE just do a G8000 to start SAM76, otherwise make the appropriate changes as required with reference to Table 1 first.

If you are using a machine with output lights at port FF, then the progress of loading will be displayed. As each block is successfully loaded a check sum will appear for a few seconds then the adress of the new block being loaded will be displayed. If the check sum remains, an error is indicated and then it is necessary to stop the reader, back up the tape to the beginning of the previous block, start the reader and G400 again.

d. Test SAM76 by getting a list of available built in functions by typing in the following:

```
&@@f, /=
```

e. Three types of anomalies are flagged by output to the console:

T	Time out no tape or jammed tape
M	Memory error
P	Parity error in block

f. For the afficionado format is described below:

16 Bytes	^h80 or ^o200
SOH	^h1
DEL	^hFF or ^o377
1 byte	Number of data bytes (0 means 256)
2 bytes	Load adress (Low then High)
variable	data bytes
2 bytes	Check Sum

LOADIT - loader

```
.RADIX  8       ; added by editor

;
 .TITLE /loadit/

        .LOC    002000  ; ^h1000

ZAPMON= 1
ZAPPLE= 170000  ; ^hF000

SENSE=  377     ; ^hFF

;Control Codes
CCDLE=  20      ; Data Link Escape
CCENQ=  5       ; Enquiry
CCEM=   31      ; End Message
CCSOH=  1       ; Start of Header
CCETB=  27      ; End of Tx Block
CCACK=  6       ; Acknowledge
CCNAK=  25      ; Neg. Acknowledge
CCSTX=  2       ; Start Text

;
 .DEFINE SKIP2[R]=
        [.BYTE  001![[[R]&6]<3]]

GETLDR: CALL    GETBY
        CPI     CCSOH
        JZ      SOHFND
GETLDX: CPI     CCETB
        JZ      RSEXIT
        CPI     170
        JNZ     GETLDR
        CALL    GETAD
        PCHL    ;all done

SOHFND: CALL    GETBY
        CPI     377
        JNZ     GETLDX

LDR377: CALL    GETBY
        MOV     C,A
        MVI     E,0
        MOV     D,E
        CALL    GETAD

LOAD:   CALL    GETBY

RCVOK:  MOV     M,A
        MOV     B,M
        CMP     B
        JNZ     MEMERR
        INX     H
        DCR     C
        JNZ     LOAD
```

```
CALL    PARTST

JMP     GETLDR

NIXNIX: MVI     C,"?"
        SKIP2   D
MEMERR: MVI     C,"M"
        SKIP2   D
PARERR: MVI     C,"P"
        SKIP2   D
TIMERR: MVI     C,"T"

EREXIT: CALL    CO
RSEXIT: JMP     RESTART

;
GETAD:  CALL    GETBY
        MOV     L,A
        CALL    GETBY
        MOV     H,A
        CMA
        OUT     SENSE
        RET

GETBY:  CALL    RI
        JC      TIMERR
        PUSH    PSW
        ADD     E
        MOV     E,A
        JNC     GETBYR
        INR     D
GETBYR: POP     PSW
        RET

;
PARTST: PUSH    D
        CALL    GETAD
        POP     D
        MOV     A,H
        CMP     D
        RNZ
        MOV     A,L
        CMP     E
        RET

;

        .END            ;Pseudo op added by editor

;
```

Quick Reference Information for the SAM 76 Language

Mnemonics

An attempt was made, in the selection of the mnemonics of the resident functions, to be consistent in the use and meaning of words or terms represented by the function names; generally, whenever reasonable, letters used in the function names may be expanded as follows:

letter	position 1	positions 2 and/or 3	
a	alphabetic	all	activator
b	bring	branch	base
c	change	character	c3
d	define	decimal	divider
e	erase	element	e3
f	fetch	file	field, function
g	gl	greater	g3
h	how many	h2	h3
i	input, if	identical	initialize
j	jl	jump	j3
k	kl	k2	k3
l	list	left	length
m	ml	match	mode
n	neutral	number	n3
o	output	o2	o3
p	partition	p2	p3
q	query	q2	q3
r	return	right	register
s	set, select	string	space
t	trace	text	t3
u	update, user	u2	u3
v	view	v2	v3
w	write	warning	work
x	eXperimental	"X" number base	x3
y	yl	y2	y3
z	Zpecial	z2	z3

In the above table only the more frequently or major terms are indicated; positions marked with a letter number pair are for possible reference.

It should be noted that the meaning of the words used in the above table are consistent; that is to say that the same connotation may be applied - for instance "text" always means; "a named string of symbols to be found in the text area".

Quick reference resident function list

```
238 - |@f,s0|              wh@ are Functions
239 - |@n|                 wh@ is processor ser. Number
237 - |@t|                 wh@ is processor Title
159 - |ab,s1,s2,vt,vf|     Alphabetic Branch
128 - |ad,n1,n2,n3,...,n|  Add
160 - |ai,s0,s1,s2,...,s|  Alphabetic Insertion
187 - |and,x1,x2|          And the bits
161 - |as,s0,s1,s2,...,s|  Alphabetic Sort
220 - |bf,f,vz|            Bring File
113 - |ca,s|               Change Activator (current)
      \ca,s\               Change Activator (initial)
195 - |cfc,d1,s|          Change Fill Character schema
      \cfc,d1,s\           Change Fill Char. (initial)
193 - |cin,t1,d1,...,t,d|  Change Id Number
148 - |cld,t|             Characters Left of Divider
191 - |cll,d|             Change Line Length (active)
      \cll,d\              Change Line Length (initial)
133 - |cnb,d|             Change Number Base (active)
      \cnb,d\              Change Number Base (initial)
```

```
266 - |cpc,t1,d1,...,t,d|     Change Protection Class
147 - |crd,t|                 Characters Right of Divider
203 - |cro,s1|                Change Rub Out char. schema
      \cro,s1\                Change Rub Out (initial)
132 - |ct,t1,t2,t3,...,t|    Combine Texts (superseding)
      \ct,t1,t2,t3,...,t\     Combine Texts (save current)
250 - |cwc,s1|                Change Warning Character
      \cwc, ... \             Change Warn. Char. (initial)
261 - |cws,d|                 Change Work Space
      \cws,x\
171 - |cx,s0,s|               Character to "X"
200 - |cxb,d|                 Change "X" Base (active)
      \cxb,d\                 Change "X" Base (initial)
259 - |da,s0|                 Date
131 - |di,n1,n2,vz|           Divide
208 - |dq,s|                  Define Quote
    - |dr,t,a,o,v|           Define Relationship
164 - |ds,d,s|                Duplicate String
103 - |dt,t,s,d1,d2|         Define Text (superseding)
      \dt,t,s,d1,d2\          Define Text (save current)
173 - |dx,d,x|                Decimal to "X"
206 - |ea,t1,t2,...,t|       Erase All excepting
207 - |ed,t,d1,d2,vz|        Extract "D" characters
224 - |ef,f1,f2,...,f|       Erase Files
151 - |ep,t,p1,p2,...,p|     Erase Partitions
    - |er, ... |            Express Relationship
104 - |et,t1,t2,...,t|       Erase Text
      \et,t1,t2,...,t\        Erase all occurences of Text
249 - |etb,s|                 Erase Trailing Blanks
112 - |ex,f|                  Exit
226 - |fb,f,vt,vf|           File Branch
137 - |fc,t,vz|              Fetch Character
138 - |fdc,t,d,vz|           Fetch "D" Characters
139 - |fde,t,d,vz|           Fetch "D" Elements
140 - |fdm,t,d,s,vz|         Fetch "D" Matches
141 - |fe,t,vz|              Fetch Element
142 - |ff,t,d,vz|           Fetch Field
143 - |fl,t,s,vz|           Fetch Left match
145 - |fp,t,x1,...,x|       Fetch Partition
144 - |fr,t,s,vz|           Fetch Right match
106 - |ft,t,s1,s2,...,s|    Fetch Text
210 - |ftb,t,s,vz|          Fetch To Break character
211 - |fts,t,s,vz|          Fetch To Span character
212 - |hc,s|                How many Characters
150 - |hm,t,s|              How many Matches
149 - |hp,t,d|              How many Partitions
114 - |ht,t|                Hide Text
      \ht\                    Hide all Texts
115 - |ic|                   Input Character
116 - |id,d|                 Input "D" characters
153 - |idt,d|                Input "D" Texts
136 - |ig,d1,d2,vt,vf|      If Greater
135 - |ii,s1,s2,vt,vf|      If Identical
117 - |im,s1,s2,...,s|      Input to Match
102 - |is,dev|              Input String
152 - |it|                   Input Text
213 - |iw,n|                 Input Wait
    - |lef,dev|             Load External Function
216 - |lf,s0,d1,...,d|      List Files
    - |lr, ... |            List Relationship
105 - |lt,s0,d1,d2,...,d|   List Texts
110 - |mc,d|                Multi-partition Character
146 - |md,t,d|              Move Divider to pos. "d"
      \md,t,d\                Move Divider "d" increments
109 - |mt,t,s1,s2,...,s|    Multi-part Text all matches
      \mt,t,s1,s2,...,s\      Multi-part Text next match
130 - |mu,n1,n2,vz|         Multiply
111 - |ni,vt,vf|            Neutral Implied
188 - |not,x|               Not (complement) the bits
209 - |nu,s1,s2,...,s|      Null
246 - |oj,s,s1,d,s2|        Output Justified lines
248 - |op,s,s1,d,s2|        Output Paddded lines
186 - |or,x1,x2|            Or the bits
101 - |os,s|                Output String
154 - |ot,t1,t2,...,t|      Output Texts
108 - |pc,d|                Partition Character
174 - |pl,s1,s2,...,s|      Plot
162 - |ps,d,s1,s2|          Pad String
107 - |pt,t,s1,s2,...,s|    Partition Text all matches
      \pt,t,s1,s2,...,s\      Partition Text next match
196 - |qfc,s0|              Query Fill Character schema
```

```
194 - |qin,s0,t1,t2,...,t|  Query Id Number
197 - |qld,t|               Query Left of Divider
192 - |qll|                 Query Line Length
134 - |qnb|                 Query Number Base
202 - |qof|                 Query Over Flow conditions
167 - |qp,t|                Query Partition
267 - |qpc,s0,t1,t2,...,t|  Query Protection Class
198 - |qrd,t|               Query Right of Divider
204 - |qro|                 Query Rub Out char. schema
205 - |qta|                 Query Text Area used
251 - |qwc,a2,a1,...,a|     Query Warning Characters
262 - |qws|                 Query Work Space
      \qws\
201 - |qxb|                 Query "X" Base
215 - |ra,d,s1,s2,s3,...,s| Return Argument
263 - |rcp,d1,d2,s|         Return Character Picture
166 - |ri|                  Restart Initialized
245 - |rj,s,s1,d,s2|        Return Justified lines
252 - |rn,n|                Random Number
189 - |rot,d,x|             Rotate the bits
247 - |rp,s,s1,d,s2|        Return Padded lines
165 - |rr,s1|               Return to Restart
163 - |rs,s|                Reverse String
228 - |saf,dev|            Select All File function dev.
158 - |sar|                 "Auto Return" on line feed
      \sar\                   no Auto Return on line feed
260 - |sda,da,mo,yr|       Set Date
199 - |sem,dev|            Set "Echoplex" Mode active
      \sem,dev\               "Echoplex" Mode inactive
222 - |sf,f,t1,t2,...,t|   Store File
157 - |sfd,fun,dev|        Specify Function Device
190 - |sh,d,x|             Shift the bits
253 - |srn,n|              Seed Random Number
258 - |sti,t1,t2,t3|       Set Time
129 - |su,n1,n2,...,n|     Subtract
231 - |sw,s1,s2,s3,...,s|  Switches
232 - |sy,s1,s2,...,s|     System functions
127 - |tb,t,vt,vf|         Text Branch
257 - |ti,s1,s2|           Time
125 - |tm,d|                Trace Mode activated
      \tm\                    Trace Mode deactivated
124 - |tma|                 Trace Mode All activated
      \tma\                   Trace Mode All deactivated
168 - |tr,t,s|             Trim
218 - |uf,f,t1,t2,...,t|   Update File
169 - |ut,cc|              User Trap active
      \ut\                    User Trap inactive
118 - |vt,t1,t2,...,t|     View Texts
181 - |wc,s1,s|            Write Characters
175 - |wi,xn1,yn1|         Write Initialize
179 - |wl|                  Width Left
178 - |wr|                  Width Right
180 - |ws,xn1,yn1,...,xn,yn|  Write Straight Lines
176 - |wx|                  Write "X" displacement
177 - |wy|                  Write "Y" displacement
170 - |xc,x1,x2,...,x|     "X" to Character
271 - |xcf,s,x|            eXperimental Change Function
172 - |xd,x|                "X" to Decimal
255 - |xi,port|            eXperimental Input
123 - |xj,x|                eXperimental Jump
256 - |xo,x,port|          eXperimental Output
270 - |xqf,s|              eXperimental Query Function
119 - |xr,x|               eXamine Register
121 - |xrp,x|              eXamine Register Pair
120 - |xw,x1,x2|          eXperimental Write in reg.
122 - |xwp,x1,x2|         eXperimental Write reg. Pair
126 - |yt,t,s,vt,vf|      Ys There
182 - |zd,r,v-,v0,v+|     "Z" reg. Decrement and branch
183 - |zi,r,v-,v0,v+|     "Z" reg. Increment and branch
184 - |zq,r|               "Z" reg. Query
185 - |zs,r,n|             "Z" reg. Set
```

Editorial Note: Object tapes for the 8080 and Z-80 versions of SAM 76 are available from Computer Mart of New Jersey, 501 Rt. 27, Iselin, NJ 08830. Telephone: (201) 283-0600. Tentative price will be $6.00. Computer Mart of New Jersey will also be publishing a book containing all available documentation on SAM 76. Available in early Spring, it will sell for $9.95. – *TRW*

SAM 76 – Additional comments

Dear Jim:

I just got around to reading your issue 10, volume 2, and was pleased to see the article by Frits Van Der Wateren which describes an implementation of Strachey's GPM for the 6800.

The first purpose of this letter is to point out to the relatively uninitiated that the SAM76 language, described in your first issue of volume 3 has its roots in GPM, and that the structure and philosophy are virtually identical between the two languages.

GPM - SAM76

Mimicking

The main difference between the two languages is the scope of available resident functions, as well as certain features of SAM76 which enable precise user definition in the SAM76 language of functions which might also be defined in terms of assembly language instructions - in other words "exact mimicking capability".

This feature is important to permit portability or transferability of user programs or, as I prefer to call them, scripts. Should some user script make use of a function not resident in some one else's SAM76 interpreter, it is relatively simple to create a user defined function identical in its functioning to the resident function originally used.

SAM76 - 6800

A version of the SAM76 language coded for the 6800 is in process of development by the "notorious" Wayne Loufbrow youngster; it is expected that this version will be functionally identical to the 8080/Z80 implementation, thus permitting absolute transferability of user scripts.

The second purpose of this letter is to respond to some queries from users as to the problem of formatting user script for appearance and legibility, and yet not cause these formatting characters to become part of the evaluation process.

Addendum to SAM76 description

Because of the celerity with which you published the description of the SAM76 language, I was not able to make the appropriate addition which responds to this problem; this addition is described below, as it will appear in the forthcoming "SAM76 Language Handbook".

The table under the heading "Quoting or Protection" on page 20 should be ammended by the addition of the "IC" warning character as illustrated by the excerpt shown below.

AS	,	77	Argument separator
QC	@	100	Quote single character
IC	`	140	Ignore single character

The following specification for this warning character should be concatenated, with scissors and glue, at the end of the SAM76 language description just ahead of the birds on page 22 of your magazine.

SIERRA DIGITAL'S PDP8 X8 CROSS ASSEMBLER SERIES

News Release *Received: 77 Dec 30*

Sierra Digital Systems has announced the addition of four new microprocessor cross-assemblers to its X8 cross-assembler series for the Digital Equipment Corporation PDP8 mincomputer. The X8 series cross-assemblers now cover the Z80, 1802, SC/MP and 8048 microprocessors in addition to the previous 6502, 6800, 8080, F8 and 2650 version.

By using an X8 series cross-assembler, assembly language programs are converted into object code which may be loaded into a microprocessor system, or put into ROM of PROM memory. X8 cross-assemblers feature a Universal Assembler Format of common assembler directives and techniques. This standardization of features combined with a quantity discount schedule makes the series especially attractive to users of more than one microprocessor type. New special package offers at very substantial discounts are available to Educational Institutions for microprocessor course development.

Sierra Digital's X8 cross-assemblers run in 8K words of memory under the OS/8 operating system. The X8 assemblers are written in PDP8 assembly language to provide very fast operation and minimum memory requirements. Pseudo-ops and runtime options provide for conditional assembly and extensive listing control. Generated object code may be output in the microprocessor's standard loader format, or BNPF for ROM generation.

Each cross-assembler is priced at $400 and distributed in PDP8 binary format on paper tape, Dectape, or Dec floppy diskette. Source files are also available for an additional $250. Sierra Digital Systems is located at 13905 Rancheros Drive, Reno, Nevada 89511, (702) 329-9548.

Ignoring next character

Oftentimes the user wishes to arrange his script for convenient legibility using "nulin", "tabs" or other characters; this is strictly for formatting purposes. Under normal circumstances these characters will be part of the expressions, and be returned in the value string unless deleted in the scanner.

One way of achieving this is to place these characters in unused arguments of primitives such as the third argument of an "os" expression. This could be a problem if at some later time the additional unused arguments acquire a purpose.

Under initialized conditions the "`" character is used to signify that the immediately next ensuing character is to be deleted in the scanner.

Final erratum

Another bright youngster, Karl Nicholas, pointed out to me that expression "k" under < skim > on page 23 was unnecessarily complicated, and suggested the following simplification which after being checked out showed the correctness of his observation:

```
< k >
%pt,,[1]/%i/![1]/
```

Closing comments

Plaudits and bouquets to your staff who did a great job of laying out in an esthetic and artistic manner the rather complicated material you received from me.

I hope that the growing body of users of the SAM76 language will contribute algorithms and scripts to your pages, and in that manner provide an escape from the "basic" syndrome.

Cordially,

Ancelme Roichel

SAM76 Language Update

Source Code Availability

The current position with respect to availability of the source code is very simple: it is available in machine readable form under the following conditions:

1. - A relatively complex test must be passed with flying honors to demonstrate the requestor's understanding of both the language philosophy and pragmatics; one of the ten test problems might be:

"Write a procedure named "FETCH" that precisely and completely mimics the "FT" function".

2. - The requestor must demonstrate skill and knowledge with respect to assembly language programming as it relates to the version of the source code desired; if the source code is desired as a model for implementation in a new machine then skill in the target machine must be demonstrated - team work is encouraged in this instance.

3. - The requestor must be prepared to agree to contribute free time towards the advancement of the language; this can be in a number of ways including improving the efficiency of existing source code, publishing algorithms and examples, giving talks to school and other educational groups &c.

4. - The requestor must agree to make available for publication at no charge any programs written either for other machines, or as additions to the source version requested. This does not, however, include any application programs, normally referred to as scripts or procedures written in the SAM76 language itself.

The foregoing requirements are an attempt at preserving the current integrity and portability of the language until enough implementations have been made that a "DE FACTO" standard is established; the benefits to users of such a position should be obvious.

I/O Vector Patch

Through an oversight all of the I/O calls were not made through the vector table created at ^h4Ø0 in early versions of the object code distributed at San Jose; this anomaly was brought to our attention the second day of the WEST COAST FAIRE, and the changes listed below will correct the problem for users with other than "ZAPPLE" monitors:

Version	location	from	to
Z8Ø	8ØC6 & 8116	CD 15 FØ	CD 21 Ø4
	811F & 81CD	CD 18 FØ	CD 24 Ø4
8Ø8Ø	8ØE7 & 813B	CD 15 FØ	CD 21 Ø4
	8144 & 81F7	CD 18 FØ	CD 24 Ø4

Location of object code

The normal location for the interpreter object code was assembled to start at ^h8ØØØ; this was a compromise selection due to the desire to have all of the user work space in one continuous area starting at Ø for the Z8Ø, and ^h4Ø0 for the 8Ø8Ø; the idea was to permit saving the user work space as an executable load module that would be portable from machine to machine, regardless of the location of the object code itself (be it below or above ^h8ØØØ).

Since then it has become painfully obvious that such hope could not be realized with regard to the POLYMORPHIC and HEATHKIT systems; consequently changes have been made in the object code to normalize certain pointers at the time the "EX" Exit function is executed, and restores these same pointers as appropriate to the machine in which the saved user work space might be reloaded.

version <Ø022>

Active and neutral execution of the "%N" function reveals the creation date and version number respectively; versions <Ø022> with a date equal to or greater than 14CC have been modified as required to permit portability of the actual work space. The creation date code may be decoded using the following procedure:

```
< date>
%ad,1,%su,%xd,%@@n//,%mu,31,%di,%xd,%@@n//,31////
[1]%ad,1,%su,%di,%xd,%@@n//,31/,
%mu,%di,%di,%xd,%@@n//,31/,12/,12///
[1]%ad,1964,%di,%di,%xd,%@@n//,31/,12//
```

Note that scripts output with the "OT" output text function, are not in question and that the new version is only required if the user desires to move his work space to machines where that space does not start at ^h4Ø0.

New function "XQS"

This function was added to delineate the limits of the user work space; it may be used at any time, but generally prior to executing the "EX" exit function; formal definition follows:

```
272  |xqs,sØ|                    X Query Save
```

The value of this function are two numbers, each preceded by the string "sØ" representing in the normal number base of the machine the starting and ending adresses of the user work space that should be saved.

Normally the "cws" change work space function would be first executed to reduce the work space to a minimum size, then the "xqs" function executed to ascertain the limits of the required save, prior to doing an "exit".

failure of "XI" in 8Ø8Ø code

Version <Ø022> 14CC corrects the non functioning of the "XI" experimental input in the 8Ø8Ø versions of the SAM76 language processor. The following shows the changes that can be made in early versions to correct this problem:

Early version coding:

```
XI:     MVI     A,KINP
        STA     CHA1
        MVI     A,KRET
        STA     CHA1+2
        CALL    MSXX
        STA     CHA1+1
        CALL    CHA1
        JMP     MSXRB
```

Corrected coding:

```
XI:     MVI     A,KINP
        STA     CHA1
        CALL    MSXX
        MOV     L,A
        MVI     H,KRET
        SHLD    CHA1+1
        CALL    CHA1
        MOV     C,A
        JMP     MSXRB
        NOP
```

```
CHA1=   Ø4DE    |       KRET=   C9      |       KINP=   DB
```

XI, MSXX, and MSXRB locations may vary between versions.

"CFC" superfluous argument

It was pointed out that the "cfc" change fill character funtion did not function as defined - requiring an extra argument; this may be readily corrected by replacing with NOP the call to the subroutine that moves the argument pointer in the expression being evaluated:

```
CFC:    MVI     A,NULCT-TIR     ; 3E 1E
        CALL    STORNO
        IJSAS   ; replace in z8Ø with one NOP, 8Ø8Ø 3 NOPs
        DCX     H
        CALL    MSR9
        MOV     M,A
        IJRQ
```

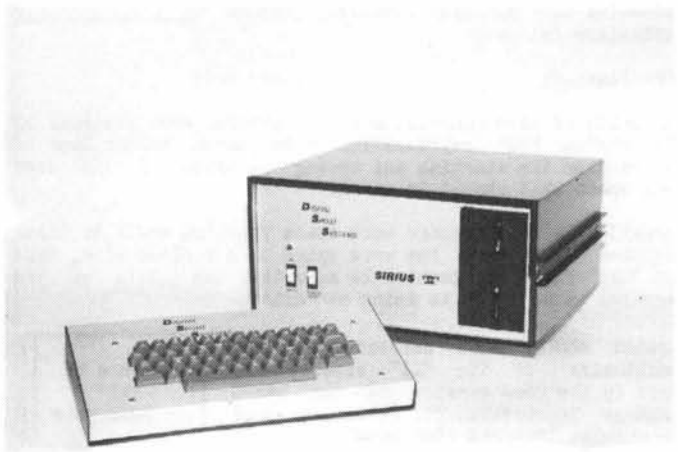
In the Z8Ø version IJSAS is an RST2, in the 8Ø8Ø version IJSAS is assembled as a CALL to MSAS; the locations of CFC, STORNO, MSAS, and MSR9 may vary between versions.

SAM76 language afficionadoes might like to receive advance copies of material scheduled for publication. At this time it is suggested that those people wishing to receive preprints may send to SAM76 a self addressed envelope containing one dollar; this dollar will be replaced with preprints as available and the envelope mailed - then to continue if so desired %PREPRINTS/.

<PREPRINTS>

An object code version tailored to the POLYMORPHIC should be available by the time this is published. The price is $6.00 postpaid for this as well as for version <0022>. The 190 page textbook is priced at $12, and a system loader information booklet is $2; all of these are postpaid. Specify loose-leaf or bound preference for the text book.

SAM76 Inc., Box 257, R.R.1, Pennington, N.J., 08534



THIS IS SIRIUS

News Release *Received: 78 April 13*

Sirius II comes complete, using a unique combination of two micro processors, the Mostek Z80 (for main computations), and a Fairchild 3870, that handles all keyboard and TV interface overhead. Also included are 32,768 characters of read/write memory, RS-232 interface for I/O, 8,192 character PROM board with 1,000 character monitor supplied, mini-floppy disc drive, a 64 key keyboard with alpha/numeric and graphic capabilities, and TV interface. Other features included are a full disc operating basic interpreter, monitor and software programs (ranging from home management, personal finance, educational learning programs, process control, to games and more games). The Sirius II is also capable of running the full complement of business software that its big brother (Sirius IV) runs.

Presently available hardware includes additional RS-232 I/O, additional parallel I/O, add-on 16,384 BYTE RAM cards, A/D-D/A converters, and additional mini-floppy disc drives.

The Sirius II is manufactured by Digital Sport Systems, and has a suggested retail price of $1,850. Contact Digital Sport Systems, 7th and Elm Streets, West Liberty, Iowa 52776; (319) 627-4211.

A BELL FOR YOUR SWTPC TERMINAL

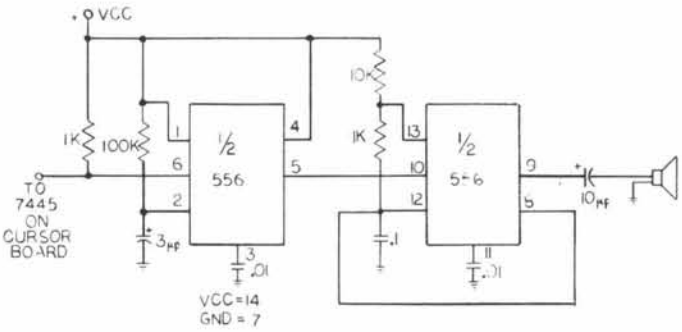
Dear Sir: *Received: 78 Feb 5*

Enclosed is a circuit for adding a "bell" function to a SWTPC CT-1024 terminal. The circuit consists of a 556 (dual 555) timer IC, four resistors, five capacitors, and a speaker. I built the entire circuit on the 14-pin Wire-wrap socket that the 556 is plugged into. The bell is triggered from an unused output pin of the 7445 decoder on the computer controlled cursor board in the CT-1024.

The left portion of the circuit is a one-shot with a duration of about ½ second. The right portion is an astable oscillator with a frequency of about 1Khz. The output of the one-shot is connected to the reset of the astable. When the one-shot is triggered by a low-going pulse from the 7445 decoder on the cursor board, its output goes high for ½ second. This high pulse allows the astable to oscillate at 1Khz for ½ second. The 1K resistor at the trigger for the one-shot is required to pull-up the output of the open-collector 7445. The astable is coupled to the speaker through a 10 mfd. capacitor. The circuit I built plugs in "piggy-back" onto another 14-pin IC (with Vcc=14, Gnd=7). The cost is less than $4.00.

Sincerely,
Stuart Brown

128½ Taber St.
Ft. Wayne, IN 46803



TIME-SHARE FOR NORTH STAR

News Release *Received: 78 April 6*

A Time-Share Disk BASIC System is now available for users of the North Star Floppy Disk System. Designed to operate with either 8080 or Z-80 processors, NORTHSHARE™ provides up to four independent users with selectable memory partitions and buffered terminal outputs.

Minimum memory requirements for operation are 24K bytes. There are no special hardware requirements outside of additional terminals and I/O ports to support the multiple users.

System includes one Diskette and Release 3 North Star Basic and DOS with NORTHSHARE™ Supervisor and Documentation Package. Price is $48. Byte Shop of Westminster, 14300 Beach Blvd., Westminster, CA 92683; (714) 894-9131.

SAM76 Language Update No.3-July 1978

BY ANCELME ROICHEL

version <0023>

This new version is upward compatible with previous versions and is the result of user requests for means to facilitate access to major pointers. In addition linking of user programs to the processor is simplified; provisions are made for up to four overlay zones defined by the user. These enhancements were made without, however, increasing the size of the processor. The new functions are defined below:

- |@cn,current,new| Change function Name

This function allows user to rename any of the resident function "current" names to any desired "new" name; these new names must consist of two or three alphabetic characters; in changing names the user should be particularly careful in his command sequence. At this time "os" and "is" should not be changed.

- |qio| Query current IO assignments

The value of this function is in the current "X" base the value of "IOBYTE" as defined by the monitor - if such a facility is provided.

- |sio,iobyte| Set IO assignment

This null valued function is used to reassign the monitor "IOBYTE" if provided for. It should be noted that a "cold" start entry into the processor carries with it the current monitor assignment; if this function is used then a "control C" or "ex" exit causes restoration back to the original assignment. A "warm" reentry, or a "hot" continue causes reassignment to the setting in force at the time of exit.

- |xll,sØ| Xamine Label List

The value of this function is a list of internal system labels available for examination by name; these labels may be actual values, identify locations of program entry points or tables, or give addresses of memory locations in which system pointers may be found. Each label name is preceded by the string symbolized by "sØ". (Label names and their purpose will be described later).

- |xal,label,offset| Xamine Address of Label

The value of this function is the value of the "label" to which the value of the "offset" is added. Both the value returned and the "offset" are in the current "X" base.

- |nud,function,arguments| Null Display mode

The "nud" function serves to modify the scanner to the end that instead of a value being returned by the function whose name is symbolized by "function" (with its appropriate arguments), output of what would have been the value is sent to the currently assigned "console" device.

"ex" Exit function

The exit function in version <0023> has been enhanced to automatically reduce the work space to a suitable minimum prior to returning to the operating system. Reentry automatically restores the work space to its maximum allowable size. Next enhancement will be to implement the passing of a file name and other information back to the operating system.

"cws" Change Work Space

This function has now been enhanced so that execution of %cws/ without arguments serves to reduce the work space to a suitable minimum.

"ot" Output Text

The neutral form of this function may be used to "re-specify" size of inter record gaps and length of leader and trailer; the two forms of this function are:

%ot,n1,n2,...,nN/= which serves to "output texts" whose names are symbolized by "n1", "n2", "nN"; the overall output is bounded by 64 bytes of leader, and individual texts are separated by 16 bytes of gap.

&ot,leader,gap,n1,n2,...,nN/= first resets the specifications for leader and gaps to the new desired decimal values symbolized by "leader" and "gap", and then outputs the texts in the normal manner.

Generalized structure - labels and points of interest

The accompanying figure shows a typical arrangement of the user work space, as well as associated pointers and their names. This work space may be located either below or above the main system program, can be saved after having performed the "ex" exit function, and this saved file can be reloaded in another system with its workspace in a different area of the machine; a "hot" reentry permits the process terminated by the "ex" function to resume with the next and other subsequent unevaluated expressions.

System Labels

Typical system labels as determined by the "xll" Xamine Label List function are given below with their characterization.

```
"AST"  ;  Address System subroutine Table
"ASV"  ;  Address System subroutine Vec4ors
"ENC"  ;  Entry Continue "hot start"
"ENR"  ;  Entry Restart Level "warm start"
"ESL"  ;  Entry Subroutine Links
"OPV"  ;  Offset Permanent - Variable tables
"POA"  ;  Pointer overlay A - "extra"
"POB"  ;  Pointer overlay B - "disk"
"POC"  ;  Pointer overlay C - "plot"
"POD"  ;  Pointer overlay D - "user"
"PSE"  ;  Pointer service entry
"PSW"  ;  Pointer function "SW"
"PSY"  ;  Pointer function "SY"
"PUL"  ;  Pointer user link table
"PWA"  ;  Pointer Work Area start
"TAE"  ;  Text area end
"TAH"  ;  Text area high
"TAL"  ;  Text area low
"TAM"  ;  Text area max
```

Details of the mechanism of linking the overlay areas to the system function table, and details of user application of system subroutines will be described in a later write up.

Exercise for the adventurous

The inquisitive person may begin to investigate the system subroutine system by finding the code for the "xr", "xrp", "xw", and "xwp" functions. The entry point for functions can be found using the "xqf" function. The function ends with either a "jump" or "return" instruction. In the body of the code are some subroutine calls; the addresses called are also found in two other places: somewhere beyond the start of the table located at "AST" (said table locates some fifty of the system subroutines) and also somewhere - the same index distance (but modulo three) in the table created at address starting at "ASV".

The first table is used if an instruction in one of the two following forms is encountered in a user program:

RST 6	CALL ESL
index number	index number

Alternatively a "call" to the proper location in the table of jump vectors which starts at "ASV" will achieve the same task.

In like manner other subroutines may be identified through exploration of other functions.

"sw" and "sy" linking

The user may make use of the "sw" and "sy" functions by storing at locations determined by %xal,PSW,offset/ and %xal,PSY,offset/, respectively, the memory addresses which execution of these functions should reach. If the offset is Ø, then the temporary table address is given, if the offset is that value returned by executing %xal,OPV/, then the permanent table address is given. The "xrp" and "xwp" functions may be properly used to read and write in these locations.

The value of this function is the contents, in the current "X" base, of the identified sector. Each "byte" is preceded with the string symbolized by sØ.

- |xws,unit,track,sector,X| X Write Sector

This null valued function places the string symbolized by X in the designated sector. If that string is greater than one sector then only as much as will fit will be transferred.

special feature of bf and uf

The "bf" bring file, and "uf" update file functions operate as expected when used with the active function marker - namely: %bf,filename/, and %uf,filename/. If the neutral form is used then any file may be brought in regardless of its type (provided there is room) and may be put back on disk. Care must be used in doing this since strange files will not have the necessary pointers to be compatible with the SAM76 processor. This is a very effective means for moving files from disk to disk - particularly on one disk drive systems.

For example:

```
&bf,xdir.com/=

Now change diskettes

&uf,xdir.com/=
{nav-uf-aef}

Ho-hum an already existing file message - well no matter

%ef,xdir.com/&uf,xdir.com/=
```

Anomaly Indications

Whenever a non fatal anomaly occurs during the execution of file functions, a value is returned by these normally null valued functions. These values are in the form:

{nav-function-anomaly}

In the above indication, the function being executed is symbolized by "function", and the anomaly type may be one or more of the following:

"dsk" disk
"nrm" no room
"err" error
"fnf" file not found
"dir" directory
"clo" close
"ext" extension
"aef" already existing file
"rud" reading unwritten data

CP/M compatible overlay

This overlay is nominally 1K and provides at this time the following functions:

prim	Id No	Description
BF	220;	Bring File
EF	224;	Erase File
FB	226;	File Branch
LF	216;	List Files
QFA	QFA;	Query File Attributes
QFE	QFE;	Query File Extension
RF	243;	Read File
SDU	SDU;	Select Directory Unit
SFE	SFE;	Select File Extension
UF	218;	Update File
WF	244;	Write File
XRS	XRS;	X Read Sector
XWS	XWS;	X Write Sector

Those functions which in the preceding table have a numeric "Id No" have been described previously in Doctor Dobbs as well as in the SAM76 language manual; the other functions are still in the experimental evaluation phase and are described below.

- |qfa,filename,sØ| Query File Attributes

The value of this function is a list of file characteristics derived from the disk operating system directory; each attribute is preceded by the string symbolized by "sØ". One of the attributes is the file size expressed as a decimal number of 256 byte pages.

- |qfe| Query File Extension

The value of this function is the current setting of the file extension; note that there is a difference between "blanks" or "spaces" and "null" or zero.

- |sdu,d| Select Directory Unit

This null valued function allows the user to designate the Directory unit to be accessed by the various file functions; a Directory unit is any device which has a directory structure and could be cassette as well as disk.

- |sfe,extension| Select File Extension

This null valued function is used to preset a desired file extension so that the extension need not be used in any of the file functions. In effect this may be used as a classification means. The extension may be set to any desired three (or less) symbols, including spaces and ???, or to the null string (viz: %sfe/) with varying effects.

- |xrs,unit,track,sector,sØ| X Read Sector

Availability

This new enhanced version is available on paper tape or TDL cassette as follows for a postpaid price of $8.00 - ($2.00 credit will be given to original purchaser of earlier version returned with order); POLY-88 version for 8080 only on cassette $8.00; CPM version on standard diskette $15.00.

Addresses in table below are all in HEX.

Type	ASV	VAR	TAM	START	END
Z80 High	0100	0400	7FFF	8000	9FFF
8080 High	0100	0400	7FFF	8000	A3FF
Z80 Low	0100	3400	FFFF	1000	2FFF
8080 Low	0100	3400	FFFF	1000	33FF
8080 Poly-88	4800	5000	FFFF	2000	43FF

Notes: VAR = START - OPV; all version with ASV at 0100 are CPM compatible; Z80 versions make use of RST0 thru RST6 while processor is running and restores RST0 on exit; TAM=FFFF is theoretical maximum reduced by size of operating system used; CPM diskette contains low versions; the disk overlay is at C00, and other overlays are at 400 and 800; it is possible in the CPM system to run programs which are smaller than 2K without affecting the SAM76 processor.

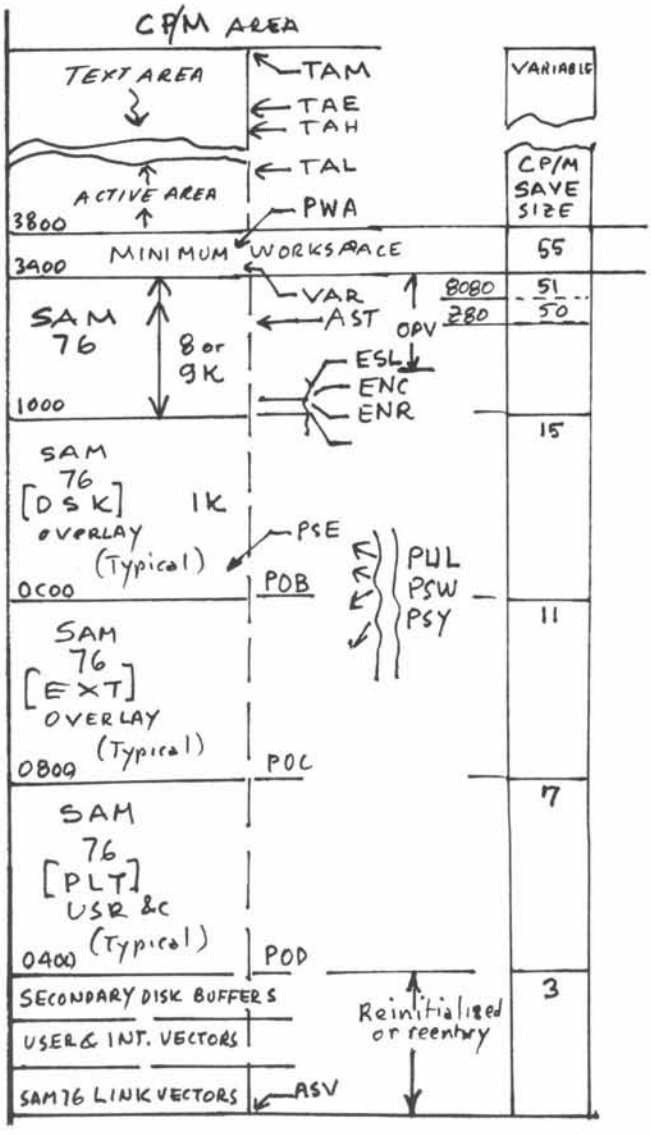
Text book describing the language in simple terms is available for a price of $12 - postpaid. (Specify bound or loose leaf). System loader booklet appropriate to object code format is $2.

Source for CP/M Disk functions

The source program for the disk functions as implemented in the CP/M version will be published in Dr. Dobbs Journal, however copies may be purchased for the implementation of the moment - no guarantee that it will match the object code furnished - but to serve as a model for implementation in other disk systems; price is $10.00 postpaid; available as xerox printed copy or in machine readable form on standard diskette or paper tape [TDL macro format].

SAM76 Inc., Box 257, R.R.1, Pennington, N.J., 08534

Typical System Layout



(cont'd from page 18)

Recently, PASCAL has been hailed as the "next" standard language.[6] Besides the arguments above against standardization, I am not very impressed with PASCAL as a language. Some syntactic kluges have been pointed out already,[7] but the main problem with PASCAL is that it is one of the class known as "structured" languages; a misnomer, as SNOBOL4, for example, has a much neater, more elegant, and simpler syntax. That PASCAL is structured means essentially that the programmer is limited with respect to the control structure that he can use. This makes the source code easier to understand and debug, but it restricts the programmer and may discourage creativity. The controversy over structured versus nonstructured languages that raged in *Communications of the ACM* and similar publications for several years seems now to be over, structured programming techniques having emerged victorious; but this does not mean that you, or I, or anyone else has to agree.

Rather than making PASCAL or some other single language a standard, what is needed is a total lack of standards, allowing the programmer to choose which language to use for a given application based on the problem's nature and his personal preference. Each microcomputer manufacturer could write one or several languages that had not yet been implemented for their processor. Transportability between machines could be provided by device-independent I/O: the user, or, better, the maker of the user's mainframe, would write a set of I/O subroutines for the user's hardware configuration and I/O devices that would be called by the language interpreters and compilers, these being written with calls to user-provided routines only. Relocatability of code would be very helpful. Alternatively, the writer of the language might spend the small amount of extra work needed to modify the basic program for use with a variety of common systems. Some software is already being offered this way.

Some obvious first choices for languages to implement on microcomputers are PASCAL, ALGOL, LISP or one of its many variants, SNOBOL4, RPG II, GRASS, SKETCHPAD, APL, MUMPS, C (anyone for MULTICS on a micro?), GPSS, SIMSCRIPT, and a TRAC*-like language.

A lack of a high-level programming language standard would not only give the user the benefit of a wide variety of languages to choose among, it would also be to the advantage of the computer makers: there is certainly more money in being the first to introduce a SNOBOL4 than in being the hundred and first to produce a BASIC.

References

1. See, for example, Marc LeBrun, "Tilting at Windmills, Or, What's Wrong with BASIC," *People's Computer Company* vol. 2 no. 1 (December 1972) p. 5.
2. John McCallum, "The Altair (S-100) Bus Forum: PCC 77," *BYTE* vol. 3 no. 3 (March 1978) pp. 148-151.
3. Robert Suding, "Why Wait? Build a *FAST* Cassette Interface," *BYTE* vol. 1 no. 11 (July 1976) p. 46.
4. Glen A. Taylor, "Language Development: A Proposal," *BYTE* vo.2 no. 11 (November 1977) pp. 190-191.
5. Peter Skye, "The 8080 High Level Language Project of Peter Skye, Continued," *BYTE* vol. 2 no. 5 (May 1977) pp. 68-70.
6. Carl Helmers, "Is PASCAL the Next BASIC?" *BYTE* vol. 2 no. 12 (December 1977) pp. 6-8 and 184-185.
7. Leigh Janes, "Reactions to Previous Comments," *BYTE* vol. 3 no. 2 (February 1978) p. 159.

*Trademark of Rockford Research, Inc.