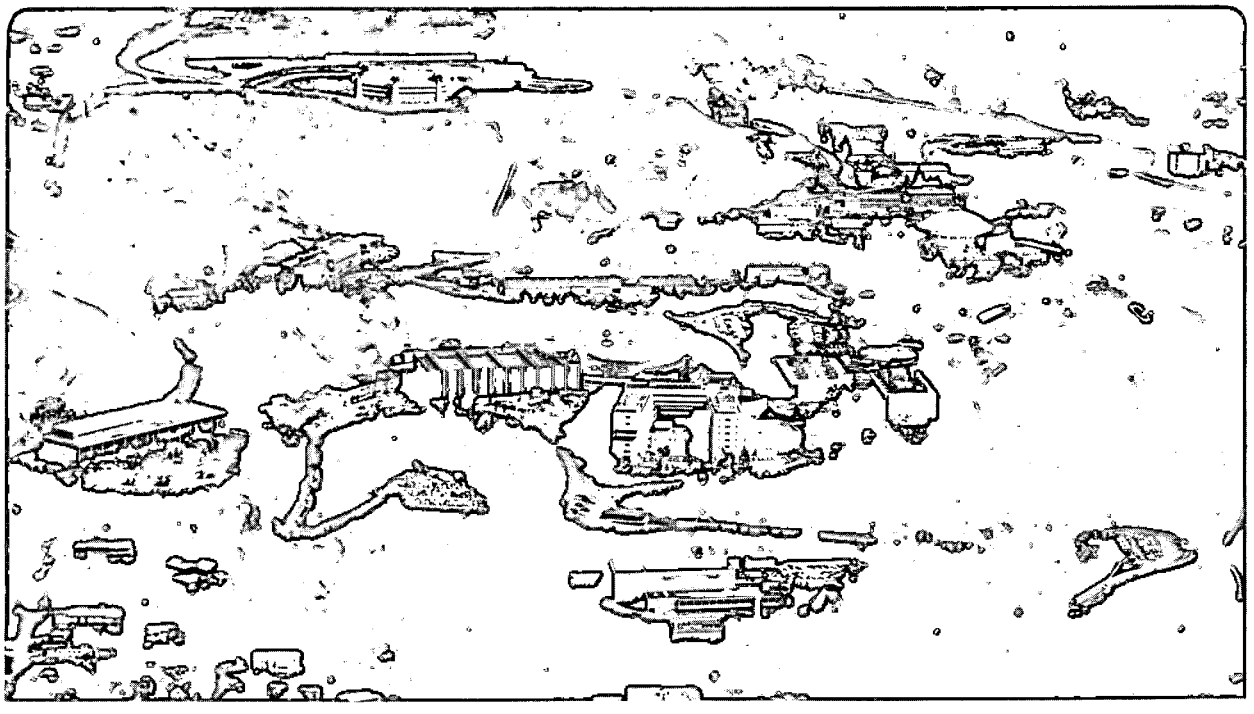
Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

Engineering Division

***EVA* REFERENCE MANUAL**

R.A. Belshe

June 1987



Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

DISCLAIMER

EVA Reference Manual

Robert A. Belshe

Real Time Systems Department
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

June 1987

This work was supported by the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

Table of Contents

1. INTRODUCTION 1
2. EVA VERSION 6J 1
    2.1. ON-LINE MONITORING 2
    2.2. ONE-DIMENSIONAL SPECTRA 2
    2.3. TWO-DIMENSIONAL SPECTRA 2
    2.4. VIRTUAL 2-D SPECTRA 2
    2.5. DECLARATIONS 2
    2.6. RESERVED NAMES 2
    2.7. DATA TAPES 2
    2.8. AUXILIARY REGISTERS 2
    2.9. SUBSCRIPTS 3
3. HOW TO START EVA 3
4. CONTROL COMMANDS 3
    4.1. CLEAR 3
    4.2. COMPILE,NAME[,LO] 3
    4.3. DO NAME 4
    4.4. EXIT 4
    4.5. HELP 4
    4.6. LOAD SPEC,FILE 4
    4.7. READ U,KEY 4
    4.8. RESTORE 4
    4.9. REWIND U 4
    4.10. SAVE 4
    4.11. SORT U,NF,NB 4
    4.12. TEST,U 5
    4.13. TERMINAL,DEV 5
    4.14. USL,NAME 5
    4.15. UNLOAD U 5
    4.16. WRITE U 5
5. THE EVAL LANGUAGE 6
    5.1. COMMENTS 6
    5.2. DECLARATIONS 7
    5.3. DATA AND SPECTRUM MANIPULATION 8
    5.4. BIT MANIPULATION: 11
    5.5. PROGRAM FLOW CONTROL 11

6. APPENDIX A: PROGRAMMING EXAMPLES 14
6.1. SAMPLE PROGRAM 14
6.2. 2-DIMENSIONAL GATES 16
6.3. VARIABLE LENGTH EVENTS 18
7. APPENDIX B: PROGRAM OPTIONS 21

1. INTRODUCTION

EVA is a program used for creating one-dimensional (1-D) and two-dimensional (2-D) spectra from event tapes. In order to create these spectra an algorithm is written in the EVAL language which describes both the structure of a single event and the analysis to be done on each event. The process of sorting an event tape is then simply to repeat this algorithm for each of the millions of events on the tape. The EVA language has been defined to easily describe such algorithms, and a compiler has been written to produce optimum machine code for each individual set of sorting conditions. This allows the user full control over the handling of the events and produces code which runs much faster than a generalized FORTRAN program.

EVAL (EVent Analysis Language) was developed by Anders Holm at the Niels Bohr Institute. The original MODCOMP implementation of EVA was done at LBL by Martin Neiman in 1980.

This manual describes the implementation of sorting programs using the EVA compiler for the Modcomp CLASSIC computers at LBL. When used with the program GATES, it allows free-form 2-dimensional regions to be used as gates.

EVA can be described as a program which transforms the computer into a programmable calculator with a large memory. The data tape appears to the calculator as a stream of data words. The user must write a program, in the EVAL language, which specifies how words of this stream are grouped into events and how each event is to be treated. The main register of the calculator is the accumulator, which can contain either integer or floating point numbers. There are also three integer registers (X, Y, Z) which are dedicated to special functions, but can also be used as general purpose quick access memory. The calcuator can test the value in the accumulator, do arithmetic with it, and use it as a channel number of a spectrum in memory to be incremented. While it can do both integer and floating point arithmetic, integer arithmetic is much faster.

The major input from the user is the EVAL program for the calculator. Most of this manual is concerned with how to write such programs. Following this there are several examples of complete EVAL programs, with a discussion of each particular program. In reading the programs the following points should be noted:

- While a name may be any length, only the first eight characters are significant.
- The delimiters space, comma, and equals are equivalent and may be used interchangeably to improve readability.
- A line beginning with a 'C' followed by a blank is treated as a comment.
- The pound sign '#' indicates that the remainder of the line is to be treated as a comment.
- Blank lines may be inserted anywhere in the program.
- The line numbers are added to aid in the discussion of the program. They are *not* a part of the program itself.
- Indentation of statements in the program is ignored by the compiler. When a listing of the program is requested, the output will be indented automatically to show the structure of the program.

2. EVA VERSION 6J

Version 6J contains all the features of the previous versions including the facility to address channels in any 1-D spectrum with the load and store commands. This allows 1-D spectra to be treated as lookup tables, which can greatly reduce the number of instructions required to make some logical decisions. Other recently added features are:

- NZINC command. This command is exactly like INC, except channel zero of the selected 1-D spectrum is never incremented.
- More sorting areas. The number of addressable sorting areas has been increased to 16. The number of usable areas depends on the amount of disk space available.
- Virtual 2-D spectra may be specified, and viewed with the EVA monitor program, CLOOK. From one to 32 virtual spectra may be displayed on the screen at one time.
- When a program saves 2-D spectra on disc, any 1-D spectra are saved on the large disc also. Programs which create only 1-D spectra now use the

shared save area EM0.

2.1. ON-LINE MONITORING

While sorting is in progress, 1-D and virtual 2-D spectra may be monitored with the program CLOOK. This program allows the user to select either a single spectrum or a list of spectra and have them displayed on the Tektronix 4014 terminal.

2.2. ONE-DIMENSIONAL SPECTRA

During sorting, one-dimensional spectra are located in memory. Up to 800,000 words of memory may be allocated to 1-D spectra. When sorting of a tape is completed, if the program has only 1-D spectra, they are saved on the shared save area EM0. If the program has 2-D spectra as well, both the 2-D and 1-D spectra are saved in the selected area.

2.3. TWO-DIMENSIONAL SPECTRA

Two-dimensional spectra are created in reserved areas on the large disc. These spectra may be as large as 16 million channels with the amount of disc space currently available. There are 16 user-selected areas for the storage of 2-D spectra. Because 2-D spectra are not in memory, the time to increment one channel is much larger. A simple sorting program can increment over 100,000 1-D channels per second, but only about 9000 2-D channels per second.

The algorithm used to update 2-D spectra uses a 256 word presort buffer in memory for each disc track, and a 8192 word presort buffer on disc for each disc track in the spectra. The presort buffers in memory reduce the amount of space available for 1-D spectra. Because the disc presort buffers are so large, a significant amount of sorted data may be lost in the case of a computer malfunction. The data in memory cannot be recovered after the computer is restarted, but the large presort buffers which are on the disc can be recovered when EVA is restarted, if no other sorting has been done.

2.4. VIRTUAL 2-D SPECTRA

Virtual 2-D spectra are displayed on the screen of the Tektronix 4014 in real time, but are not stored in memory. All data is lost whenever the screen is erased. Display of virtual spectra is controlled by the monitor program, CLOOK.

2.5. DECLARATIONS

All formats, data, spectra, and variables must be declared. Declarations may be in any order but must all appear *before* the first executable statement in the program.

2.6. RESERVED NAMES

There are two reserved variable names, 'SYNC' and 'EVSIZE'. SYNC is predefined, and represents the position of the current event in the tape buffer. EVSIZE must be declared in each program, either as a DATA constant or as an integer variable. When the EVAL program completes its processing of an event, it must set EVSIZE to the number of 16 bit words in the event. At the end of the EVAL program, the event pointer is incremented by EVSIZE, SYNC is incremented by one, and the program is reentered at the top. When the event pointer passes the end of the tape buffer, new tape record is read and the variable SYNC is set to zero.

2.7. DATA TAPES

Data tapes have a fixed record length, so an event will usually be partly in one record and partly in the next. The input record buffering routine in EVA does a 'look ahead' which allows the EVAL program to always see complete events. This feature relieves the user from having to deal with split events, and ensures that all events on the data tape will be processed. EVA will process data tapes with record lengths up to 5000 words, and it expects all records in a file to have the same length.

2.8. AUXILIARY REGISTERS

Three auxiliary registers, 'X', 'Y', and 'Z', are available to hold integer values. The 'X' and 'Y' registers are used to hold the X and Y channel numbers when incrementing a two-dimensional spectrum. The 'Z' register is also used to contain the channel number when 1-D spectra are addressed with the load and store commands. The 'Y' register also has a special function when 2-dimensional gates are used. Otherwise the 'X', 'Y', and 'Z' registers may be used for any general purpose. These registers may be used as 'rapid-access' temporary locations, which can save

significant amounts of sorting time in some programs.

2.9. SUBSCRIPTS

The value of a subscript is used as an offset from the named item. As in FORTRAN, a subscript value of one is the same as having no subscript. Unlike FORTRAN, a subscript in EVAL is used to reference the items which follow the subscripted item. A subscript of 2 references the item immediately following the subscripted item, a subscript of 3 references the 2nd item after the subscripted item. For example, if the following three variables are defined in consecutive statements;

```
IVAR  AA
IVAR  BB
IVAR  CC
```

then AA(1) is the same as AA
and AA(2) is the same as BB
and AA(3) is the same as CC

The same rules apply to subscripted SPECTRA, FORMAT, and DATA.

When a spectrum is subscripted, a number of spectra with different names are created. For example, the statement:

SPEC GAMMA1(3) 1024,2

creates three consecutive spectra named **GAMMA1(1), GAMMA1(2),** and **GAMMA1(3).** The user's program will reference these spectra as **GAMMA1(IVAR)** where IVAR has a value of 1, 2, or 3.

Note that reference to GAMMA1(0) or GAMMA1(IVAR), when IVAR equals zero, results in a reference to whatever item was declared just before GAMMA1. This is a common error which can be very difficult to detect. The values of subscripts are not validated by the program because of the additional execution time that would be required.

3. HOW TO START EVA

To run EVA the user types CTRL-C on the DECwriter, then types:

```
/EVA/EXE,,xx
```

xx is the name of the file where the loadable image of EVA is stored (usually TT).

EVA will announce its name, then the user is asked (on the DECwriter):

EVA>>

This is EVA's prompt for a command, which must be from the following list. Only the first three letters of any command need to be typed.

Until the COMPILE command has been given, only the HELP, USL, TERMINAL, and EXIT commands can be executed.

4. CONTROL COMMANDS

4.1. CLEAR

Resets the accumulated spectra to zero. 1-D spectra are always cleared, if any 2-D spectra are present, the user is asked if they should be cleared.

4.2. COMPILE,NAME[,LO]

This command is used to convert the user's EVAL program into a executable subroutine. Because it only takes a few seconds to compile most EVAL programs, the object module is not saved; EVAL programs are re-compiled each time they are used. NAME is usually the name of the program as cataloged on the RSL partition. Other partitions may also be used (see the USL command). The option parameter LO is included if a listing of the EVAL program on the VERSATEC printer is wanted. At the end of compilation, statistics are printed which show the amount of memory and disk space used by the program.

Occasionally, to find an obscure bug, it may be necessary to look at the assembly language code generated by the compiler. This can be done by setting option bit 2, and then compiling with the LO parameter.

If the selected EVAL program contains any 2-D spectra, the user will be asked which of the 16 disc sorting areas will be used, and is asked if a recovery of lost 2-D data is to be made. If the recovery option is selected, the program will report the number of counts recovered. If the previous sort ended normally there is no need to perform a recovery, and if done, the number of counts recovered will be zero.

4.3. DO NAME

When this command is issued, command input is switched to the file 'NAME' on the users USL file. When the last command in 'NAME' has been executed, command input is switched back to the DECwriter.

4.4. EXIT

The EVA task exits, returning all memory to the system.

4.5. HELP

Prints a list of the possible commands.

4.6. LOAD SPEC,FILE

This command loads the spectrum 'SPEC' with data from a previously prepared text file. 'FILE' is the name of the file on the current USL. This file contains a list of integers, separated by commas or blanks, which will be loaded into consecutive channels of the spectrum. When the character '#' is found in this file, it and the remainder of the line are treated as a comment. When the end of file is reached, a message is printed on the control console which shows the number of channels loaded with data. This command allows the loading of look up tables which can be used to make the process of making decisions involving multiple gates much simpler. The program 'IGATE' can be used to create text files for the LOAD command.

4.7. READ U,KEY

This command allows the user to reload spectra from a tape which was created with the WRITE command. At the time of the WRITE, a 'data set KEY' is printed on the DECwriter. The READ command finds the spectra on tape 'U' which correspond to 'KEY' and loads them into memory. Sorting may then proceed with the next data tape.

4.8. RESTORE

After each sort command, EVA automatically saves all accumulated spectra on the top disc (partition RD0). If, for some reason, a sorting session has been interrupted, this command will restore the contents of all spectra to the values they had after the last completed SORT command. Sorting may then proceed with the next data tape.

4.9. REWIND U

Rewinds the tape at any of the possible tape stations. 'U' must be one of the numbers 1, 2, 3, 6, 7, or 9. These numbers correspond to the logical unit names MT1, MT2, MT3, etc.

4.10. SAVE

This command may be used to force saving of the spectra on disc. It is only needed if sorting has been terminated by option 7, and the user wants to save the spectra.

4.11. SORT U,NF,NB

Begin sorting tape 'U' at file 'NF'. If NF is positive the tape is positioned to the indicated file. If NF is zero, the tape is not repositioned; sorting will begin with the next block on the tape.

'NB' is the number of blocks (records) to be sorted. If NB is positive then NB *blocks* will be sorted, unless an end file is reached first. If NB is zero, one complete *file* will be sorted. If NB is negative then NB *files* will be sorted.

The heading of the file will then be typed and sorting is started. Sorting will continue until the requested number of blocks or files have been processed or until a double end of file is reached, or until the reflective marker at the end of the tape is sensed. Sorting may also be terminated by setting program option 7.

At the end of sorting the number of blocks sorted and the number of error blocks will be printed. Error blocks (those containing tape errors) are ignored when sorting.

If 150 tape errors are detected during a sort pass, sorting is terminated automatically. The tape is left at the end of the last record read.

Sorting can be terminated at any time by setting program option bit 7 as described in Appendix B.

If the tape should go off during sorting, the message 'TAPE IS OFF-LINE' will appear. When this happens you can continue by putting the tape back on-line, or you can terminate sorting with program option bit 7.

The message 'SAVE COMPLETED' will appear when the 1-D spectra have been copied to disc and the presort buffers, if any, have been emptied into the 2-D spectra. This step is skipped if the sort was terminated by option 7, see the 'SAVE' command.

4.12. TEST,U

Prints the run number and run title as read from tape 'U' (at its current position).

4.13. TERMINAL,DEV

Assign the operator terminal to a different device. When EVA is first activated, the operator terminal is assigned to the device 'AT1'.To shift control to the DECWRITER the command is 'TER,TY0'.

4.14. USL,NAME

Change the name of the file which will be searched to find the EVAL source program. The default file name is RSL. To change to the global user file MSL, the command is 'USL,MSL'. File manager file names may also be used; in this case the command is 'USL,USL:xxx', where 'xxx' is the user's login name.

4.15. UNLOAD U

Tape 'U' is rewound and unloaded or put off-line.

4.16. WRITE U

The user is first asked if this is a new tape. If the reply is 'No', the tape will be positioned at the logical end of tape before writing begins, otherwise the tape is rewound. The user is then asked to type a 32 byte title which will be written into the spectrum header record. Finally, all the spectra which exist in the current EVAL program are added to the tape. Spectra are written on the tape in the same order as they are declared in the EVAL program.

5. THE EVAL LANGUAGE

Each EVAL statement occupies a single line and begins with a mnemonic command. Following the command are possibly some parameters. The commands are divided into 5 groups.

1. Comments
2. Declarations
3. Data and spectrum manipulation
4. Bit manipulation
5. Program flow control

The entire EVAL program must be contained in a BRA-KET pair, i.e. the first line of the program must be a BRA and the last line must be a KET. The symbols '[' and ']' may be used in place of the words 'BRA' and 'KET'.

The following notation is used in the description of EVAL commands:

< >	Indicates a required parameter. The type is described by the word enclosed in the < >, i.e. <NAME> means a text name.
[ ]	Indicates an optional parameter.
(*)	Indicates an optional subscript. The subscript itself must be of the type INUM, IDATA, or IVAR.
|	A vertical line is read as 'or' i.e., <DATA | VARIABLE> means either data or a variable is required.
NAME	The name to be assigned to a variable, data item, format, or spectrum. Names may be any length but only the first eight characters are significant.
NUM	Integer or Real number. A REAL number includes a period, an INTEGER number does not.
INUM	Integer number.
DATA	Integer or Real data.
IDATA	Integer data.
VAR	Integer or Real vairable.
IVAR	Integer variable.
J	Integer variable (optional) used to declare an array of variables or an array of spectra.

5.1. COMMENTS

C A line beginning with a 'C' followed by a blank is treated as a comment.

The '#' symbol and all following characters are treated as a comment.

5.2. DECLARATIONS

DATA <NAME> <VALUE>
Assigns 'VALUE' to 'NAME'. 'VALUE' must be either an integer or a real number. The value of 'NAME' cannot be changed by the program.

SPEC <NAME(J)><NUMCHAN><WORDS/CHANNEL>
Defines a spectrum called 'NAME', containing 'NUMCHAN' channels. All spectra start with channel zero. 'WORDS/CHANNEL' must be either 1 or 2, which will overflow with 65,535 or 2,147,483,647 counts respectively. If the integer subscript 'J' is present, then 'J' identical spectra are created, and designated NAME(1) through NAME(N). These spectra will usually be incremented using the subscripted form of the INC statement.

DSPEC <NAME(J)><XMAX><YMAX><WORDS/CHANNEL>
Defines a two-dimensional spectrum called 'NAME'. XMAX and YMAX are the number of channels on each axis. The remaining details are the same as in SPEC, above.

LSPEC <NAME(J)><XMAX><YMAX><WORDS/CHANNEL>
Defines a virtual two-dimensional spectrum called 'NAME'. XMAX and YMAX are the number of channels on each axis. The number of words per channel must be present, even though it is not used.

FORMAT <NAME><WORD><FIRST BIT><LAST BIT>
Defines a format called 'NAME' which can be used to unpack a parameter from the event buffer. It describes the inclusive interval from 'FIRST BIT' to 'LAST BIT' of word number 'WORD' in the event. The bits are numbered from 0 to 15 with 0 being the most significant. The first word of the event is word 1, not word_0.

GATE <NAME>CHANNEL><HIGH CHANNEL>
Defines a gate called 'NAME' which extends from 'LOW CHANNEL' to

G2D <NAME>
Defines a 2-dimensional gate called 'NAME'. The values for this gate will be read from disk partition 'RR1' where they have been placed by the program 'GATES'. The gates on disk are associated with NAMES in order, i.e. the first gate on disk is associated with the first G2D statement in the EVAL program. Each 2-dimensional gate requires 256 words of memory.

HEX <NAME> <VALUE>
Assigns 'VALUE' to 'NAME'. Value must be a hexadecimal number of not more than 4 digits. The value of 'NAME' cannot be changed by the program.

IVAR <NAME(J)>
Creates an integer variable called 'NAME'. If the integer subscript 'J' is present, then an array of length 'J' is allocated. The elements of this array are accessable by using the subscripted form of the variable.

RVAR <NAME(J)>
Creates a real (floating-point) variable called 'NAME', with optional subscript as in IVAR, above.

5.3. DATA AND SPECTRUM MANIPULATION

ADD <NUM | DATA(*) | VAR(*) | FORMAT(*)>
Adds 'NUM', 'DATA', 'VAR', or 'FORMAT' to (from) the accumulator. If either the accumulator or the operand is a real number the result will be a real number, otherwise it will be a 16 bit integer. (A 16 bit integer can assume all values from –32768 to +32767.

CALL <FUNCTION> <NUM | DATA(*) | VAR(*)>
The CALL statement operates on the value currently in the accumulator, and leaves the result in the accumulator. The mathemetical routines used are the from the standard FORTRAN library. Of the ten functions are available, only the PWR function requires a parameter.

SIN	Compute sine of angle (radians) in accumulator.
COS	Compute cosine of angle (radians) in accumulator.
TAN	Compute tangent of angle (radians) in accumulator.
ASIN	Compute arc sine of accumulator.
ACOS	Compute arc cosine of accumulator.
ATAN	Compute arc tangent of accumulator.
EXP	Compute e**x, where x is value in accumulator.
ALOG	Compute the natural log of the accumulator.
SQRT	Compute the square root of the accumulator.
PWR	Compute A**X, where A is the value in the accumulator, and X is the parameter following the function name.

CHS
Changes the sign of the accumulator.

COUNT
Add one to the event counter. This command is used in the online version of EVA (CSORT). It is ignored by EVA.

DIV <NUM | DATA(*) | VAR(*) | FORMAT(*)>
Divides the accumulator by 'NUM', 'DATA', 'VAR', or 'FORMAT'. If either the accumulator or the operand is a real number the result will be a real number, otherwise it will be a 16 bit integer.

FETCH [IVAR(*)]<FORMAT(*)>
The ADC systems at LBL produce numbers between 4030 and 4095 when the input pulse falls below the lower threshold. These values should usually be treated as zero. This can be done by defining a spectrum to be 4029 channels, or merely by ignoring the last 66 channels. In some applications (for example when testing for zero, or doing arithmetic) it is desirable to have a binary zero. FETCH is similar to GET, but will assume that the data was produced by such an ADC and will set all values greater than 4029 to zero. FETCH takes about 50% longer than GET to execute.

ABS
Converts the accumulator to its absolute value (fixed or floating).

FIX
Converts the contents of the accumulator (by truncation) from a real number to a 16 bit integer. FIX will be ignored if the accumulator already contains an integer.

FLOAT
Converts the contents of the accumulator from an integer to a real number. FLOAT will be ignored if the accumulator already contains a floating point number.

FRAC
Set the integer part of the accumulator to zero. For example, 11.345 becomes 0.345 .

GET [IVAR(*)]<FORMAT(*)>
Unpacks a parameter described by 'FORMAT' into the accumulator. If the optional parameter 'IVAR' is present, the parameter will also be stored in the variable 'IVAR'.

GETX [IVAR(*)]<FORMAT(*)>
GETY [IVAR(*)]<FORMAT(*)>
GETZ [IVAR(*)]<FORMAT(*)>
Unpacks a parameter described by 'FORMAT' into the 'X', 'Y', or 'Z' register. The value will remain in the register until changed.

INC [VAR(*) | FORMAT(*)]<SPEC(*)>
The one-dimensional spectrum 'SPEC' is incremented by one in the channel number contained in the accumulator. If one of the optional parameters 'VAR' or 'FORMAT' is present. the accumulator is first loaded with the parameter. If the number is outside the limits of the spectrum, the spectrum is not incremented.

INC <DSPEC(*)>
INC <VSPEC(*)>
The two-dimensional spectrum 'DSPEC' or 'VSPEC' is incremented by one in the channel described by the present contents of the 'X' and 'Y' registers. These registers are loaded via the LDX, LDY, GETX, GETY, or COPY commands.

LDA <NUM | DATA(*) | VAR(*) | SPEC(*)>
Loads the accumulator with the 'NUM', 'DATA', or 'VAR' indicated. If 'SPEC(*)' is specified, the accumulator is loaded with the value of channel 'n', where 'n' is the current contents of the 'Z' register.

LDX <INUM | IDATA(*) | IVAR(*) | SPEC(*)>
LDY <INUM | IDATA(*) | IVAR(*) | SPEC(*)>
LDZ <INUM | IDATA(*) | IVAR(*) | SPEC(*)>
Loads the 'X', 'Y', or 'Z' register with the 'INUM', 'IDATA', or 'IVAR' indicated. If 'SPEC(*)' is specified, the register is loaded from channel 'n' of SPEC(*), where 'n' is the current contents of the 'Z' register. The value will remain in the register until changed.

MUL <NUM | DATA(*) | VAR(*) | FORMAT(*)>
Multiplies the accumulator by 'NUM', 'DATA', 'VAR', or 'FORMAT'. If either the accumulator or the operand is a real number the result will be a real number, otherwise it will be a 16 bit integer.

NZINC [VAR(*) | FORMAT(*)]<SPEC(*)>
This command is identical to INC except that channel zero of the selected spectrum is never incremented.

RAN <RVAR> <RVAR | DATA>
This command is used to increment a pseudo-random value and then add the value to the accumulator. The first parameter is the pseudo-random variable and the second parameter is a fraction which is to be added to the random variable. Whenever the random value becomes greater than 1.0, the random value is decremented by 1.0.

The purpose of this command is to allow a random distribution of counts when a parameter is multiplied by a gain correction constant. The random variable is always a fraction from 0.0 to 0.99999 which is added to the value of the parameter, for example:

SCALED_PARAMETER = (RAW_PARAMETER + RANDOM_VALUE) * GAIN_CONSTANT

In EVAL code this becomes:

```
LDA   RAW_PARAMETER
RAN   RANDOM_VALUE,RANDOM_FRAC
MUL   GAIN_CONSTANT
STA   SCALED_PARAMETER
```

SET <IVAR(*)><INUM | IDATA(*) | IVAR(*)>
Set the first parameter equal to the second parameter. This operation does not change the value of the accumulator.

STA <VAR(*) | SPEC(*)>
Stores the contents of the accumulator in 'VAR' or 'SPEC(*)'. If 'SPEC(*)' is specified, the accumulator is stored into channel 'n', where 'n' is the current contents of the 'Z' register.

STX <VAR(*) | SPEC(*)>
STY <VAR(*) | SPEC(*)>
STZ <VAR(*) | SPEC(*)>
Stores the contents of the 'X', 'Y', or 'Z' register in 'VAR' or 'SPEC(*)'. If 'SPEC(*)' is specified, the register is stored into channel 'n', where 'n' is the current contents of the 'Z' register.

STEP <IVAR(*)><INUM | IDATA(*) | IVAR(*)>
Set the value of the first parameter to the sum of itself and the second parameter. This operation does not change the value of the accumulator.

SUB <NUM | DATA(*) | VAR(*) | FORMAT(*)>
Subtracts 'NUM', 'DATA', 'VAR', or 'FORMAT' to (from) the accumulator. If either the accumulator or the operand is a real number the result will be a real number, otherwise it will be a 16 bit integer. (A 16 bit integer can assume all values from –32768 to +32767.

COPY COMMANDS

The following instructions copy the integer in one register into another register. If the source register is the accumulator, it must contain an integer. Copy commands execute in 0.2 microseconds, interchange commands execute in 0.6 microseconds.

CAY Copy the accumulator to the 'Y' register.
CAX Copy the accumulator to the 'X' register.
CAZ Copy the accumulator to the 'Z' register.
CXA Copy the 'X' register to the accumulator.
CYA Copy the 'Y' register to the accumulator.
CZA Copy the 'Z' register to the accumulator.
CXY Copy the 'X' register to the 'Y' register.
CXZ Copy the 'X' register to the 'Z' register.
CYX Copy the 'Y' register to the 'X' register.
CYZ Copy the 'Y' register to the 'Z' register.
CZX Copy the 'Z' register to the 'X' register.
CZY Copy the 'Z' register to the 'Y' register.
IXY Interchange the 'X' and 'Y' registers.
IXZ Interchange the 'X' and 'Z' registers.
IYZ Interchange the 'Y' and 'Z' registers.

5.4. BIT MANIPULATION:

ASH <INUM | IDATA>
Performs an arithmetic shift of the accumulator. In a left shift (value from 1 and 15) zeroes are shifted in, while in a right shift (value from −1 to −15) the sign bit is propaged into the number. This operation is not allowed if the accumulator contains a real number.

LSH <INUM | IDATA>
Performs a logical left or right shift of the accumulator. Zeroes will be shifted in and the bits shifted out will be lost. A value from 1 and 15 means a shift to the left, a value from −1 to −15 means a shift to the right. This operation is not allowed if the accumulator contains a real number.

AND <INUM | IDATA(*) | IVAR(*) | FORMAT(*)>
OR <INUM | IDATA(*) | IVAR(*) | FORMAT(*)>
XOR <INUM | IDATA(*) | IVAR(*) | FORMAT(*)>
Performs a logical AND, OR, or XOR (exclusive OR) between the accumulator and 'INUM', 'IDATA', 'IVAR', or 'FORMAT'. The result is stored in the accumulator. These operations are allowed only between integers.

5.5. PROGRAM FLOW CONTROL

BRA, KET, [,]
Used in conjunction with IF and ELSE to define program flow. The left and right bracket symbols '[' and ']' may be used interchangably with 'BRA' and 'KET'. The entire EVAL program must be contained in a BRA-KET pair, i.e. the first non-comment line of the program must be a BRA and the last line must be a KET.

IF [VAR(*) | FORMAT(*)] <LOGOP> <NUMBER | DATA(*) | VAR(*) | FORMAT(*)>
IF [VAR(*) | FORMAT(*)] [NOT] <GATE(*) | G2D(*)>
If the optional parameter 'VAR' is present, the variable 'VAR' is fetched into the accumulator before any test is performed. If the optional parameter 'FORMAT' is present, a word from the tape buffer will be unpacked into the accumulator. Otherwise, the current contents of the accumulator is tested.

When the second form of the IF command is used, the accumulator is compared to the lower and upper limits of a GATE constant. If the GATE constant is a two-dimensional gate (G2D), the accumulator is compared to the Yth element of the named 2D gate array, where Y is the current value of the Y register.

The possible LOGOPs are:

EQ: equal
NE: not equal
LT: less than
GT: greater than
LE: less than or equal to
GE: greater than or equal to

If the logical test is true, or the value is within the specified gate, program continues with the next statement. If it is false, the program jumps to the *next visible* ELSE (i.e. one not enclosed in a BRA-KET pair beginning below the IF), or if none is found, to the *next visible* KET below the IF statement. Execution continues with the statement immediately following the ELSE or KET.

The six LOGOPs above are reserved names, and may not be used to name FORMATS, SPECTRA, DATA, or VARIABLES.

ELSE
Used in conjunction with IF statements to indicate an alternative program path. When an ELSE statement is reached by any path *other* than a failed IF test, the program jumps to the next visible KET.

FOR <IVAR> <FIRST> <LAST> [<STEP>]
This statement initializes a FOR-NEXT loop. The variable IVAR is set to the value specified by FIRST. The values LAST and STEP are saved for use by the associated NEXT statement. FIRST, LAST, and STEP must be positive integer numbers, integer data, or integer variables. If STEP is missing, a value of one is used. This statement does not change the value of the accumulator.

NEXT <IVAR>
This statement adds STEP to IVAR, then compares the value of IVAR with the value of LAST specified in the associated FOR statement. If IVAR is less than or equal to LAST, the program jumps back to the statement following the FOR statement. If IVAR is greater than LAST, the program continues with the statement following the NEXT. This statement does not change the value of the accumulator.

GOTO <NAME>
This statement causes the program to jump to the label statement containing NAME. NAME is a string of up to 8 characters which has not been used to declare a format, spectrum, data or variable.

:LABEL

A label statement is a line containing a colon in column one, followed by a NAME of up to 8 characters. Characters following NAME on a label statement, if any, are ignored.

```
# EXAMPLE OF A LOOP USING GOTO STATEMENT

    BRA
      SET N=1
    :LOOP
      ( statements )
      STEP N,1
      IF N LT NMAX
        GOTO LOOP
    KET
```

REPEAT

This statement causes a jump to the BRA statement which is at the beginning of the BRA-KET pair in which the REPEAT is enclosed.

```
# EXAMPLE OF A LOOP USING REPEAT STATEMENT

    SET N=1
    BRA
      ( statements )
      STEP N,1
      IF N LT NMAX
        REPEAT
    KET
```

6. APPENDIX A: PROGRAMMING EXAMPLES

6.1. SAMPLE PROGRAM

In the following example, a three parameter event is sorted. The coincidences of two GELI detectors are examinded and six spectra are produced. There are two GELI singles spectra, two concidence spectra of GELI2 with gates on GELI1 for both TRUE and RANDOM gates on the TAC. Each event consists of four words as follows:

word 1: GELI # 1
word 2: GELI # 2
word 3: TAC
word 4: end of event flag (hexadecimal F000)

The outermost BRA-KET pair, lines 3 and 73 are required because the closing KET marks the end of the program.

In lines 7–9 the three parameters are given names and declared to be in words 1, 2, and 3 of the event. The ADC's used in this experiment produce 12 bits (4096) channels) which are placed in bits 4 to 15 of each word. Lines 7 and 8 include all 4096 channels while line 9 drops the 2 least significant bits resulting in a 1024 channel parameter. Line 10 declares the end of event flag to be all bits of word 4.

In lines 14–19 the six spectra to be sorted are declared. Spectra S1 and S2 will be used for singles, so they are declared to have 2 words per channel. The others are coincidence spectra and are not expected to accumulate more than 65,535 counts in any one channel, so they are declared as single word spectra. This saves both memory and sorting time.

In lines 23–26 the gates are defined.

In line 30 the variable EVSIZE is declared. The value of EVSIZE will be set by the program on line 39 or line 72.

In lines 31 and 32 the variables X and Y are declared. These variables are used to avoid the overhead of unpacking the raw data from the event each time the GELI values are needed. Line 33 defines a constant EFLAG, and sets its value to hexadecimal F000.

Lines 37–73 contain the executable part of the program. Line 37 compares the value of END to EFLAG. This is to ensure that we are properly synchronized with the event and to handle any incomplete events which may appear in the data. If the end flag does not match, the program jumps to line 72 where EVSIZE is set to one. This sequence causes the program to search ahead one word at a time until it is properly aligned with the event boundary.

When END does match the end flag, we proceed to line 39, where EVSIZE is set to four.

In line 42 the parameter described by the format GE1 is both loaded into the accumulator and stored in the variable X. In line 43 the spectrum S1 is incremented in the channel contained in the accumulator, thus creating a singles spectrum. Lines 44 and 45 do the same thing for GE2.

In line 46 the TAC is loaded into the accumulator. Line 47 tests the accumulator against the gate TRUE. If the accumulator contains a number from 300 to 380 (see line 25) the IF statement is satisfied and the program

proceeds to the next line. If not, the program jumps to the next visible ELSE or KET, in this case to the ELSE on line 58.

Lines 50–56 are a group of statements which will be executed only when the IF statement of line 47 is satisfied. The BRA-KET pair is necessary because there is an ELSE statement in this group which we do not want to be visible to the IF on line 47. On line 51 the variable X is brought into the accumulator and tested against gate G1. If X is within the gate, line 52 is executed which fetches Y into the accumulator and then increments spectrum ST1. The program would then jump to the end of the BRA-KET (line 56). When the IF in line 51 is not satisfied the program jumps to the ELSE on line 53 and then proceeds to line 54 where the accumulator is compared to gate G2. If it is within the gate, line 55 is executed which will bring Y into the accumulator and increment spectrum ST2, otherwise the program will jump to the end of the BRA-KET (line 56) and the treatement of the event will be terminated.

The ELSE statement on line 58 is reached only in the case where the IF statement on line 47 is not satisfied. The program proceeds to line 59 where the accumulator (which contains the TAC value loaded by line 46) is tested against the gate RANDOM. Lines 62–68 contain a BRA-KET similar in structure to the BRA-KET in lines 50–56.

Line 72 is executed only when the IF test on line 37 fails and jumps to the ELSE on line 71. The BRA-KET on lines 38 and 69 are necessary to make the ELSE statement on line 58 invisible to the IF on line 37.

```
 1   #  SAMPLE 1 #    A SORTING PROGRAM
 2
 3   BRA
 4
 5     # DEFINE THE WORDS IN AN EVENT
 6
 7     FORMAT  GE1   1   4   15       # GELI IS 4096 CHANNELS
 8     FORMAT  GE2   2   4   15
 9     FORMAT  TAC   3   4   13       # TAC IS 1024 CHANNELS
10     FORMAT  END   4   0    3       # END OF EVENT FLAG
11
12     # DEFINE THE SPECTRA
13
14     SPEC    S1   4029   2
15     SPEC    S2   4029   2
16     SPEC    ST1  4000   1
17     SPEC    ST2  4000   1
18     SPEC    SR1  4000   1
19     SPEC    SR2  4000   1
20
21     # DEFINE THE GATES
22
23     GATE    G1     1238   1242
24     GATE    G2     2318   2336
25     GATE    TRUE    300    380
26     GATE    RANDOM 400    480
27
28     # DECLARE VARIABLES AND CONSTANTS
29
```

```
30      IVAR EVSIZE           # EVENT SIZE
31      IVAR   X
32      IVAR   Y
33      HEX    EFLAG,F000     # END OF EVENT FLAG
34
35      # EVENT ANALYSIS BEGINS HERE
36
37      IF END EQ EFLAG               # TEST FOR COMPLETE EVENT
38        BRA
39          SET EVSIZE=4
40
41          # GET GELI VALUES AND INCREMENT SINGLES
42          GET  X=GE1
43          INC  S1
44          GET  Y=GE2
45          INC  S2
46          GET  TAC
47          IF TRUE
48
49             # HERE IF TAC IS IN 'TRUE' GATE
50             BRA
51                IF X G1
52                   INC Y ST1
53                ELSE
54                IF G2
55                   INC Y ST2
56             KET
57
58           ELSE
59           IF RANDOM
60
61              # HERE IF TAC IS IN 'RANDOM' GATE
62              BRA
63                 IF X G1
64                    INC Y SR1
65                 ELSE
66                 IF G2
67                    INC Y SR2
68              KET
69         KET
70
71      ELSE
72      SET EVSIZE=1
73    KET
```

6.2. 2-DIMENSIONAL GATES

Sometimes, one desires to use a region in 2-dimensional space as a gating condition. One can do this by using a 2-dimensional gate (G2D) which is tested in the same way as a 1-dimensional gate (GATE). 2-D gates are first

defined by using the program GATES which writes the gates on disc partition RR1 where EVA then finds them. A 2 D gate is merely a region of a plane with 256-channel X and 256 channel Y resolution. The number in the accumulator is used on the x coordinate, while a special Y register contains the Y coordinate. A simple example of a sort using a 2-D gate follows.

Lines 4–5 are the formats which will be used to get the parameters for the X and Y axes. Note that they are both 8 bits (256 channels) in length. In line 14 a 2-dimensional gate named R is declared. EVA will use the first gate that it finds on the disc for R. Line 23 gets EY into the Y register. In line 26 the gate R is tested. If it is true, i.e., the point EX,EY lies within region R, the program continues to line 27 where the gated spectrum, DG, is incremented. Line 29 increments the singles spectrum, DS. This example illustrates how a 2-dimensional gate can be used to select a region in 2-parameter space.

```
 1   # SAMPLE 2 #    A PROGRAM USING A 2-DIMENSIONAL GATE
 2   BRA
 3     # DEFINE THE WORDS IN THE EVENT
 4     FORMAT EX 1,4,11
 5     FORMAT EY 2,4,11
 6     FORMAT DF 3,4,15
 7     FORMAT END 4,0,3
 8
 9     # DEFINE SPECTRA
10     SPEC DS 4096,2
11     SPEC DG 4096,2
12
13     # DEFINE GATES AND VARIABLES
14     G2D R                # R IS 2-D GATE ARRAY
15     IVAR D               #
16     IVAR EVSIZE          # EVENT SIZE
17
18     # EVENT ANALYSIS BEGINS HERE
19     IF END EQ 15
20        BRA
21           SET EVSIZE=4
22           GET  D DF
23           GETY EY
24           GET  EX
25           BRA
26              IF R
27                 INC D DG
28           KET
29           INC D DS
30        KET
31     ELSE
32        SET EVSIZE=1
33   KET
```

6.3. VARIABLE LENGTH EVENTS

This program uses the subscripting capabilities of EVA to process zero-skipped data. In this example the raw event data is assumed to have a GELI detector in word 1, a TAC in word 2, followed by 1 to 6 NAI counter values, with a final end of event FLAG word. Each counter value has a 4-bit TAG in bits 0–3. The FLAG word has a tag value of 15. Any of the NAI counters will be missing if the corresponding counter did not fire. The GELI, TAC, and end of event FLAG word will always be present.

The program will create the following spectra:

SINGLE(1) through SINGLE(8), a singles spectrum for each counter.

CSPEC, a spectrum which shows the number of NAI counters present per event.

GS(1) through GS(6), a spectrum for each NAI counter which will be incremented only when the GELI and TAC are within specified gates.

In this example we use the subscripted form of the SPEC and IVAR statements. The subscript N is used to access the different words in the event and the subscript C is used to access the different spectra. Since the number of words per event is variable, EVSIZE must be recomputed for each event.

This program uses the symbolic form of the BRA-KET statements which some users may find easier to read. Lines 45–55 are an example of a loop constructed from STEP and REPEAT statements. Lines 71–76 are an example of a FOR-NEXT loop.

```
 1   # SAMPLE 3 #  A PROGRAM TO SORT VARIABLE LENGTH EVENTS
 2
 3   [
 4
 5     # ===== DEFINE THE EVENT STRUCTURE =====
 6
 7     FORMAT TAG 1,0,3          # COUNTER NUMBER OR END EVENT FLAG
 8     FORMAT ADC 1,4,15         # COUNTER VALUE
 9
10     DATA GELI .1              # FIRST WORD IS GELI
11     DATA TAC  2               # 2ND WORD IS TAC
12
13     # ===== DEFINE SPECTRA =====
14
15     SPEC SINGLE(8) 4096,2     # 8 SINGLES SPECTRA
16     SPEC CSPEC   6,2          # NAI COINCIDENCE SPECTRUM
17     SPEC GS(6) 4096,2         # GATED SPECTRA FOR THE 6 COUNTERS
18
19     # ===== DEFINE VARIABLES =====
20
21     # RAW PARAMETERS SAVE HERE TO AVOID THE OVERHEAD OF
22     # UNPACKING EACH PARAMETER TWICE
23
24     IVAR ADC1(8)
25
```

```
26      # TAG VALUES ARE SAVED HERE TO AVOID THE OVERHEAD OF
27      # UNPACKING EACH PARAMETER TWICE
28
29      IVAR TAG1(8)
30
31      IVAR N                      #  WORD NUMBER IN EVENT
32      IVAR C                      #  COUNTER NUMBER
33      IVAR NMAX                   # SUBSCRIPT OF LAST COUNTER IN EVENT
34      IVAR EVSIZE                 # EVENT SIZE
35
36      # ===== DEFINE GATES =====
37
38      GATE GELIGATE 3000,3500
39      GATE TACGATE  1000,2500
40
41      # SORTING BEGINS HERE
42
43      IF TAG EQ 1                     # ENSURE BEGINNING OF EVENT
44
45        SET N=1                       # INITIALIZE LOOP TO
46                                      # FORM SINGLES SPECTRA
47        [
48          GET C TAG(N)                # C IS COUNTER NUMBER
49          IF NE 15                    # TEST FOR END OF EVENT
50            STA TAG1(N)               # SAVE TAG VALUE
51            INC ADC(N) SINGLE(C)
52            STA ADC1(N)               # SAVE PARAMETER VALUE
53            STEP N,1                  # INCREMENT N TO NEXT WORD IN EVENT
54            REPEAT                    # GO BACK FOR NEXT WORD
55        ]
56
57        SET EVSIZE=N                  # SET EVENT SIZE
58        SUB 1
59        STA NMAX                      # SAVE LAST COUNTER'S SUBSCRIPT
60
61        [
62          # IF GELI AND TAC ARE WITHIN GATES, AND THERE
63          # ARE ONE OR MORE NAI COUNTERS PRESENT, INCREMENT
64          # THE COINCIDENCE SPECTRUM AND THE GATED SPECTRA
65
66          IF ADC1(GELI) GELIGATE
67            IF ADC1(TAC) TACGATE
68              IF NMAX GT 2
69                SUB 2
70                INC CSPEC             # INCREMENT COINCIDENCE SPECTRUM
71                FOR N=3,NMAX          # INITIALIZE 'FOR' LOOP
72                  LDA C TAG1(N)
73                  SUB 2
74                  STA C               # FORM SUBSCRIPT OF GATED SPECTRUM
75                  INC ADC1(N) GS(C)   # INCREMENT GATED SPECTRA
76                NEXT N
```

```
77        ]
78
79      ELSE                    # HERE IF FIRST WORD DID NOT HAVE TAG = 1
80      SET EVSIZE=1            # SET EVSIZE TO SEARCH FOR NEXT EVENT
81    ]
```

7. APPENDIX B: PROGRAM OPTIONS

A number of the program option bits are tested by various routines in EVA. Except for options 7 and 8, they are primarily intended for debugging the EVA compiler.

When EVA is executed, all options are automatically set to zero (off).

To set any option(s) the user must first start EVA in the normal way, then type CTRL-C (on the DECwriter or any CRT) and then type:

/EVA/POP N,N,N....

where the N's are the options to be set.

for example, to set options 2 and 7 type CTL-C, then:

/EVA/POP 2,7

Table of Program Option Bits	
Bit	Action
0	Print dump of compiled program in hexadecimal
1	Print dump of 2 dimensional gates
2	Print compiled instructions for each line
3	Print symbol table at end of compilation
4	Print spectrum descriptors at end of compilation
5	Dump each spectrum on printer (during WRITE)
6	Dump buffer pointers during sorting
7	Immediately terminate tape search or sort in progress
8	Pause sorting while this option is set.

Option bit 8 is used by the analysis program SUSIE to give priority to the interactive user. The bit is set while SUSIE is calculating or reading the disk, and is reset while SUSIE is waiting for the next keyboard command. If SUSIE is terminated by an abort, it may leave this bit set which will cause EVA to remain paused. When this happens, you can restart EVA by typing CTRL-C, then:

/EVA/POP NO8

LAWRENCE BERKELEY LABORATORY
TECHNICAL INFORMATION DEPARTMENT
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720