SHARE PROGRAM LIBRARY AGENCY

PROGRAM NUMBER

172006

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA
(305) - 284-6257

CERN SUMX

A Program for Analyzing Statistical Data

. (Originally developed by CERN)

Michael J. Beniston
Hugo R. Penafiel

July, 1967

Dr. M. J. Beniston
IBM Corporation - 63G
P. O. Box 10500
Palo Alto, CA 10500

360D-17.2.006

TABLE OF CONTENTS

*See first page of this Manual for a detailed index.

SECTION 1 - Program Abstract

SUMX - A Program for Analyzing Statistical Data

This program analyzes information about large numbers of events, usually kept on magnetic tape, and produces histograms, two-dimensional scatter diagrams, lists, and ordered lists. SUMX will also find mean values and variances.

SUMX was originally written at Berkeley, but the present version was completely rewritten at CERN in Geneva, Switzerland. (SUMX 466, version 5.25). The present program for the S/360 is written for the FORTRAN IV H-level compiler. It requires at least 256K of core memory without overlays, at least three tape drives, and a recommended minimum of three disks (1 system, 1 scratch, 1 program).

This program and its documentation have been submitted to the Program Information Department for general distribution in the expectation that they may prove useful to other members of the data processing community. The program and its documentation are, essentially, in the author's original form and have not been subjected to any formal testing. IBM only serves as the distribution agency in supplying this program. It is the user's responsibility to determine the usefulness and technical accuracy of the program in his own environment. This program is not part of the IBM product line.

Questions concerning the use of the program should be directed to the author or other designated party. Any changes to the program will be reflected in the appropriate Catalog of Programs; however, the changes will not be distributed automatically to users.

Section 2 - Tape Key to Distributed Program

The SUMX tape provided with this writeup has been set up to run as is under OS on any 360 with sufficient core (256K or more). Detailed operating instructions are given in Section 3. At this point one should note only that the tape contains control card images as well as source decks and data.*

The job control language included on the tape does not make reference to catalogued procedures, thus precluding possible difficulties at different installations from those at which the program was tested. The default options taken conform with those indicated in appropriate OS/360 manuals.

No preliminary copying, punching, or printing of the tape should be necessary for an installation using OS. If another monitor is to process the program, the first three files can be punched out, the cards with // in columns 1-2 removed (these are always at the beginning of each file, in front of the program or data cards), and the resulting decks run as normal Fortran jobs for the installation.

The tape has four files - three to run the program and one of sample printout. These files are described in detail in figure 2-1.

*Tape must be unblocked. See Appendix A.

Figure 2-1

Format of SUMX Tape after Unblocking

File No.	Format*	No. of Logical Records	Contents	Job Step Name	Function of Job Step
1	F	7674	JCL** + SUMX Fortran source deck	STOSX	Put source deck on disk in data set SOR(SUMX)
2	F	330	JCL + data	STOCDS	Put data on disk in SXINPT
3	F	38	JCL (for 5job steps) + control cards for RUNSX step	FORT	Compile all SUMX Fortran routines, putting object modules on tape
				ASM	Assemble TIMER, put object module on same tape
				LKED	Linkedit SUMX with TIMER, saving load module in SXCMOD (SUMX)
				RUNSX	Execute sample SUMX run, printing long and short output, (no binary output)
			JCL	PUNCH1	Punch SUMX source deck
4	UA	2138	Sample printout		Illustrate printout obtained from SUMX

*F format: unblocked 80-character records. UA format: maximum 132 character records first character carriage control.

**JCL: job control language for Operation System/360.

SECTION 3 - Operating Instructions

The following operating instructions apply to the initial run of SUMX from the unblocked tape. After the initial run the user can readily adapt the program to his own uses.

3.1 - Running Under OS

a) Setup - the following materials will be required

1. A 2311 disk pack or other direct access device with at least 500 tracks available to this program (diskname dddddd will be needed in (b) below)

2. An unlabeled tape for object modules, to be mounted when requested on some 2400

3. About four boxes of cards, if a punched source deck is desired

4. The SUMX unblocked tape, of course, mounted on any convenient 9-track drive

5. The punched cards for the deck in (b) below

b) Preliminary run

The following little deck should be run under OS before SUMX to catalog a reference data set. This data set is never used, but its (disk) volume is referred to by all job steps. The uncataloging is to assure that no duplicate names will be encountered while processing:

```
//FIXCAT   JOB   (as per your installation)
//   EXEC   PGM=IEHPROGM
//SYSPRINT   DD   SYSOUT=A
//DD1   DD   VOLUME=SER=dddddd UNIT=2311, DISP-OLD
//SYSIN   DD   *
     UNCATLG  DSNAME=SXREF
     UNCATLG   DSNAME=SOR
     UNCATLG   DSNAME=SXINPT
     UNCATLG   DSNAME=SXLMOD
     CATLG     DSNAME=SXREF,    VOL=2311=dddddd
     SCRATCH   DSNAME=SOR,        .       .
     SCRATCH   DSNAME=SXINPT,     .       .
     SCRATCH   DSNAME=SXLMOD,   VOL=2311=dddddd
/*
```

You may wish to substitute a SCRATCH VTOC for all the SCRATCH statements above if - only if - you are using a scratch or empty disk pack. In either case, be sure the above program has run successfully before you go on.

c) Running SUMX

1. Mount the unblocked SUMX source tape on any convenient 9-track unit (xxx in step 3).

2. Get a "ready" message from OS on the console, either by reading an end of file on the card reader (e.g., after the end of job in (b) above), or by an "REQ" command.

3. Type in

```
START   RDR,  xxx
START
```

where xxx is the unit number the source tape is mounted on.

4. Let the program run. See figure 2-1 for job steps. You should not have to mount any volumes except a tape for steps 3-4 (since the disk should still be mounted from the preliminary job in (b)).

5. If you do not want a punched source deck (it takes about 45 minutes to punch one), cancel the job as soon as it requests or begins using the punch. The jobname is SUMX.

d) After the SUMX Run

1) The following data sets are cataloged in the system catalog: SXREF, SOR, SXINPT.
Uncatalog them if necessary for your installation.

2) The following data sets are on the disk dddddd:

Name	Type	Approx. Size(trks)	Contents
SOR(SUMX)	PDS	200	Fortran source decks for SUMX and preprocessor
SXINPT	Seq.	16	Input data for SUMX
SXLMOD (SUMX)	PDS	50	Load module library

3) The object decks are on the tape mounted for steps 3-4. They may be punched out, or the tape itself saved for future linkedits.

4) The printed output may be compared to the sample execution listed as file 4 of the unblocked tape. File 4 may be printed by a stand-alone program such as DEBE.

3.2 - Running Under a Monitor Other than OS

a) Preliminary Processing

Punch out the two three files of the tape, discarding the 6 or 8 OS control cards at the front of each file. However, since file 1 is so long, it may be easier to copy the Fortran cards (only) onto another tape, if your compiler accepts such input.

b) Run SUMX itself

Compile the Fortran deck (from file 1 of tape). Linkedit and execute program using SUMX control cards as given in second file.

Appendix A

Unblocking Distributed Tape

The following operating instructions apply to the initial run of SUMX the distributed tape. After the initial run the user can readily adapt the program to his own uses.

The distributed tape is in a blocked format. To be utilized under the instructions in Section 3, it must first be unblocked. This may be accomplished by mounting the blocked distributed tape on 9 track tape drive bbb, and an unlabeled tape on unit uuu. Then run the following program:

```
                Col. 16                                                    Col. 72
//JOBNAME  JOB  (as per your installation)
//  EXEC  PGM=IEBGENER
//SYSPRINT  DD  SYSOUT=A
//SYSUT1  DD  UNIT=bbb,VOLUME=(,RETAIN,,,SER=INTAPE),                        x
              LABEL=(1,NL),DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400),           x
              DISP=(OLD,PASS)
//SYSUT2  DD  UNIT=uuu,VOLUME=(,RETAIN,,,SER=OUTAPE),                       x
              LABEL=(1,NL),DCB=(RECFM=F,BLKSIZE=80),DISP=(NEW,PASS)
//SYSIN  DD  *
  GENERATE
/*
```

Now copy

```
//  EXEC  PGM=IEBGENER
```

through

```
/*
```

two more times, replacing the underlined 1 in the above LABEL parameters with 2 through 3.

Thus, you should have a deck of 31 cards of the form:

```
//JOBNAME ...
//  EXEC ...
      :          LABEL=(1,NL) ...
      :            :
      :          LABEL=(i,NL) ...
/*
```



```
//  EXEC ...
      :          LABEL=(2,NL) ...
      :            :
      :          LABEL=(2,NL) ...
/*

      :
      :
      :
//  EXEC ...
      :          LABEL=(3,NL) ...
      :            :
      :          LABEL=(3,NL) ...
/*
```

Finally, add the following cards

```
                Col. 16                                                    Col. 72
//  EXEC  PGM=IEBGENER
//SYSPRINT  DD  SYSOUT=A
//SYSUT1  DD  UNIT=bbb,VOLUME=(,RETAIN,,,SER=INTAPE),                        x
              LABEL=(4,NL),DCB=(RECFM=UA,BLKSIZE=132),DISP=(OLD,PASS)
// SYSUT2  DD  UNIT=uuu,VOLUME=(,RETAIN,,,SER=OUTAPE),                       x
              LABEL=(4,NL),DCB=(RECFM=UA,BLKSIZE=132),DISP=(NEW,PASS)
//SYSIN  DD  *
  GENERATE
/*
```

The final deck should contain 40 cards.

Note: All subsequent instructions refer to the SUMX unblocked tape now mounted on uuu.

*** SUMX SECTION - 01.06.66 ***

(1)

*** SUMX SECTION - 01.06.66 ***

(2)

Section : SUMX
Date : 15.3.1966
J. ZOLL

A 001

Foreword : Purpose of the program

SUMX is a program for analysing statistical data. Normally, these data are kept on magnetic tapes, event by event, and SUMX is called upon to compile this information into histograms, two-dimensional scatter diagrams, lists and ordered lists, or to find mean values and variances. Facilities are provided to select subsets of events according to criteria defined on control-cards, and to allow the user to add routines for computing, event by event, quantities not immediately available.

The program has been written primarily for digesting the information about large numbers of events in the bubble-chamber. The raw measurements are processed through the programs THRESH and GRIND. The output from GRIND, on magnetic tape, contains detailed information about the various possible interpretations of each event. This output is fed to the program SLICE which enables the physicist to take decisions as to the nature of each event and to extract the important quantities from the flood of data for producing the Data Summary Tape, DST, which is the input to SUMX.

Although SUMX originated in the context of high-energy physics data-processing, there is nothing about the program which limits its use to this field - it is a completely general program for producing histograms etc. Only the jargon in this manual betrays the origins.

The program is written in basic Fortran IV. Some very few machine-dependent operations are insulated into small routines.

Section : SUMX
Date : 15.3.1966
J. ZOLL

A 002

Foreword : History of the Program

The version of SUMX presently running on the CERN machines belongs to the third generation of the program. An account of the history may be useful for people who have to switch program.

SUMX was originally written at the University of California, Berkeley (UCRL Physics Note 309) in Fortran II and FAP. This program was introduced to CERN, amended and documented by E. Kalaaud in 1963 (CERN DD/Exp/63/15). It became at once very popular throughout the whole of CERN, a considerable fraction of the available computer time was used for SUMX. Soon it became clear, that a more powerful, efficient and elegant program of the SUMX type would be needed.

Therefore, SUMX was completely rewritten in 1964, only the name and the good ideas were retained - the modular structure of the program and the Test-vector. The program was still in Fortran II and a little FAP for the CERN 7090. Dynamic storage in Common was introduced for the new program : The old program stored its variables in fixed arrays of maximal size local to the various processors ('blocks'), a wasteful programming habit strongly favoured by Fortran. In the new program all processors store their information into one common vector in a continuous fashion, so that each histogram etc. uses only as much space as it really needs, and any space not needed by one processor is available to all the others. As a result, all information became easily accessible and numerous restrictions, such as number of histograms per pass or number of tests available could be dropped. All these individual restrictions coalesced into the single limitation imposed by the total amount of storage available for blank Common. On the 7090 with the Fortran II Monitor system it was possible for the program itself to include all unused store into blank common, thereby using the whole machine. - To save the user a tiresome, risky and nearly impossible calculation, SUMX was programmed to treat a job too large for the store by sections. The control-cards to the various processors were changed, new processors were added, in particular CHARM, multiple entries to what is nowadays called the tactical routines were made possible, and a number of new facilities were brought in, in particular very simple diagnostic facilities were added. To have a name, the new program was called CERN-SUMX.

The TC programs were re-written in Fortran IV and tried on the CERN 7090 under IBSYS.

/...

Section : SUMX
Date : 15.3.1966

A oo2

CERN-SUMX was mechanically translated into Fortran IV by the SHARE program SIFT.

People at the TC division got very definite ideas about machine-independent programming and CERN-SUMX certainly did not qualify for it. Also, a year's running of CERN-SUMX gave a lot of experience and served to show the short-comings of the program. Therefore, a second major rewrite was undertaken in June 1965, yielding 'Interim SUMX', which gradually evolved into SUMX 466 by internal polishing and adding missing pieces. This SUMX 466 will be put onto the CERN machines shortly after this manual has been published.

The following objectives have been sought:

1. To produce a Fortran IV program which can be made to work with a minimum of trouble on any machine of the size of the 7090, and under the more sophisticated operating systems of the present day, which allow much fewer tricks to be played than the old Fortran II Monitor system. In order to achieve this, programming had to be done in elementary Fortran IV, machine-dependent problems had to be excapsulated into small routines of well defined functions.

2. To lessen the burden on the people who prepare the control-cards for SUMX. Over the last two years the use of SUMX has increased enormously, so that large quantities of control-cards are being prepared. In the absence of a real break-through in technique, it was felt worthwhile to put some effort into increasing the intelligence of the program. The use of mnemonics has been introduced for designating Tests and quantities on the Data Summary Tape, the rules for punching the Control-cards have been eased, the necessity for terminator-cards which existed in CERN-SUMX has been done away with - all the time staying backward compatible as far as reasonable.

3. Finally, a more comprehensive system for debugging the control-cards was definitely needed. CERN-SUMX would run until it found the first faulty card and give up there. SUMX 466 continues to process the control-cards, so that hopefully all mistakes can be spotted with a single run.

Section : SUMX
Date : 15.3.1966
J. ZOLL

A oo3

Foreword : Organisation of the SUMX Write-up

This section SUMX of the TC Program Library Manual is divided into 5 sub-sections, lettered from A to E. Sub-sections A, B and C are meant to be read by the normal user; sub-sections D and E contain technical information for the people who want to play with the program.

Section A is a description of the program in general terms, for the user to get a feeling for the program - what it can do and what it can't - so that he may use it efficiently.

Section B gives the cards for a small number of example passes of increasing complexity, not so much to show what the control-cards to the individual blocks look like, but to show how to stack them with each other and with the SUMX-system cards.

Finally, section C gives the detailed specifications for all control-cards, and any other technical information which the user must have, such as formats of the Data Summary Tape.

The instructions on how to run the program are found in the papers at the very end of Section C.

Section : SUMX
Date : 15.3.1966
J. ZOLL

A 004

Principles of operation

The Structure of SUMX is this : the actual data-processing is done by a number of processors. The orderly sequential operation of the various processors is regulated by the master program SUMX. The master does not care about the processing done by any individual processor, this provides for flexibility in that the processors are plug-in units which one can physically put in or leave out, depending on whether or not one wants the kind of operation of any individual processor.

So as to be a little more concrete, we enumerate here the various processors, operational or planned :

TAPE	looks after the input of the DST (data summary tape)
Block 4	manipulates histograms gathered by block 6
Block 6	accumulates histograms and ideograms
Block 7	accumulates 2-dimensional histograms
Block 10	finds mean values and their variances
Block 11	compiles counter matrices
Block 14	produces lists
Block 15	produces ordered lists
SELECT	evaluates the truth-values of conditions
CHARM	allows the user to perform auxiliary computations
IGNOR	discards certain events
MINIMX	selects groups of DST-words
REPORT	rearranges groups of DST-words

The 'blocks' produce the results one is finally interested in; the 'tactical routines' (SELECT, etc.) enable the user to manipulate the data.

The operation of SUMX separates clearly into 3 stages : During stage 1 the program is initialized from control-cards which specify the details of the data-processing wanted. Stage 2 is the actual operation of the program, a scan through the DST to extract and accumulate the information wanted according to the instructions taken in during stage 1. Stage 3 is the output stage which prints the information gathered during stage 2. A complete cycle of the 3 stages is called a 'pass'. A SUMX job may consist of several consecutive passes, either because the user specified this on the control-cards (logical passes), or because the amount of work asked for is so large that the program has to do it in sections on its own initiative (physical passes).

During the initial stage the user talks to the program to inform it of his desires. To begin with, he has to deal with the master by means of the 'SUMX-system cards', but then he can ask to be put through to the individual processors with 'Request cards'. The master takes note of which processors have been contacted, but does not listen into the conversation which

Section : SUMX
Date 1.11.1967

A 004

instructs the processor about every detail of the operations needed, e.g. the control-cards to block 6 specify, for each histogram, which words are to be histogrammed, which conditions have to be satisfied by any event for it to be included into the subset of events for this histogram, what the parameters of the histogram are - such as number of channels and channel width.

During the operation stage the DST is processed. The data on the DST are grouped into 'events', each event consists of all the numbers relevant to the event in the bubble chamber, or whatever the statistical unit may be. The processor TAPE reads one event after the other into core-store, ready for the other processors to do their work. The current event is held in the vector BOUT, such that the first number is stored in BOUT(1), the second in BOUT(2) and so on. The physical meaning of a given number is determined by its relative position within the event and hence in BOUT. When specifying a given physical quantity on the control-cards, its address in BOUT is given.

Terminology : In this manual the processors are normally called 'blocks'; if differentiation is necessary the term 'proper blocks' is used, as opposed to tactical routines.

Section : SUMX
Date : 15.3.1966
J. ZOLL

A 006

Selection of Subsets

In practically all applications of SUMX it is necessary to select many different subsets of events. This is made possible by the Test-vector and the tactical routine SELECT. On the control-cards to SELECT the user specifies the conditions which any event must satisfy for the various 'tests' to be true or false. During the operation stage, SELECT evaluates for each event the truth-values of all defined tests and stores them in the test-vector.

Most operations of the blocks can be made conditional on the truth-values of the tests by quoting a test on the control-card calling for a given operation. For example, with block 6 selection can take place at three different strata :

a) A given event can give rise to several entries into a single histogram. Each such entry can be made conditional on an associated test.

b) With any histogram, a principal test may be quoted - a given event will contribute to this histogram only if it also causes this test to be true.

c) With the EVALUATE card one or several master-tests can be associated with a batch of histograms, a given event enters this batch only if all master-tests are true.

Section : SUMX
Date : 15.3.1966
J. ZOLL

A 007

Data-editing with CHARM

It often happens that the information on the DST does not quite have the form one needs for a given SUMX run - basically the information is there, but the quantity one would like to histogram has to be derived from the numbers coming from the DST.

The tactical routine CHARM has been written to provide the user with a facility for inserting routines of his own which do the necessary computations and store the results back into BOUT ready to be used by the blocks. These private routines are subroutines to CHARM and are to be called CHARM1, CHARM2,.... CHARM9. The package of CHARM with its dependent subroutines looks from the outside just like an ordinary processor, it is steered by control-cards and the various facilities are normally available. But inside, the operation of the package is split, CHARM performs all the necessary red-tape operations, the CHARM sub-routines do the computations.

It is true that this set-up reduces the users troubles to the logical minimum - each Charm-subroutine is exactly as simple as his problem - but it must not be overlooked that he is writing routines which become part of a complex program. This makes it necessary that he has a vague idea of the internal working of the program and a clear idea of the common storage area he is working with : BOUT in blank common. For details see C 200 and A 010.

It should be realized that data-editing in SUMX can be wasteful, both in time and in space. The computations have to be repeated with every physical pass; the Charm-subroutines, with the library-routines they possibly use, can consume considerable amounts of core-store. So one should be careful when deciding on the information-content of the DST and not too readily rely on CHARM.

Section : SUMX
Date : 15.3.1966
J. ZOLL

A 008

The Dictionary facility

With this facility it is possible to assign numerical values to alphabetic identifiers, which can then be used to give symbolic BOUT-addresses and symbolic Test-numbers on the control-cards to the blocks. The specifications for the facility are to be found in C 910 and C DICTIO.

The purpose of the facility is two-fold. The use of names is mnemonically easier than the use of numbers. However, this is partially spoiled by the fact that the blocks, when printing the control-cards during the input stage, can only print the numerical values of the symbols used, but not the symbols themselves. This is unfortunate, but there is nothing reasonable one can do about it. The second purpose is to enable the user to change allocation of BOUT-addresses and of Test-numbers without having to change the control-cards to the blocks. This enables the various groups of a collaboration to club together the control-cards prepared independently. For this to work, it is necessary that a set of symbols is reserved for each group; because, several symbols may be assigned the same value, but the same symbol cannot have different values for the same block.

Section : SUMX
Date : 15.3.1966
J. ZOLL

A 009

Sequencing of the blocks

The order in time in which the individual processors handle each event is of some importance, because normally some processors depend on the result of some other processor ; for example, the blocks rely on SELECT having set up the test-vector.

The connection between the master-program and the processors may be visualized as a switch with 35 positions. During the operation stage, the switch is turned over all its positions for each event and control goes to the processor connected to each position, provided the position is alive, or active. The positions are activated during the initial stage from the control-cards.

The switch has the following 35 positions :

pos. 1	block 0	is permanently connected to TAPE
pos. 2	block -1	to those positions the tactical routines are connected in the order in which the requests arrive during the initial stage.
pos. 3	block -2	
...		
pos. 20	block -19	
pos. 21	block 1	to those positions the proper blocks are connected, provided they are activated during the initial stage. For these, the order in which the requests arrive is immaterial.
pos. 22	block 2	
...		
pos. 35	block 15	

This numerical order of the positions is what is loosely called the 'numerical order of the blocks' in various places of this manual; it is the order in which the blocks are called up for processing during the operation stage and the output stage. To avoid mishap, the control-cards should also be stacked in this order.

Within a given pass, a given tactical routine, but not a proper block, may be requested several times. The specimen pass in section B shows this, also the pass sketched in C 912 contains a multiple request: 3 tactical routines are requested, causing 3 blocks to be activated: SELECT (block -1), CHARN (block -2) and again SELECT (block -3). This enables one to define some tests in block -1 which can be used to control the operation of CHARN (block -2), whose results may then in turn be used in block -3 to evaluate some more tests.

Section : SUMX
Date : 15.3.1966
J. ZOLL

A o10

Data Storage allocation

Labelled common blocks are used to communicate program parameters between the routines of SUMX 466.

Blank common is used for the dynamic store, the larger this store the more histograms etc. can be produced in any one physical pass. The actual size of blank common is set by the user in the main program, in the routines of SUMX a dummy size of 4500 words has been declared to keep the compiler happy.

general working space	normal BOUT	special BOUT	area available to the blocks used	free	100	super dynamic store
1 5 0 TEMP 0	1 L N CM B O U T	L E B O U T		I I D O N E	I I T O P	I I M A X

EQUIVALENCE (TEMP, ITEMP), (CM, ICM, BOUT, IBOUT)

This sketch shows the employment of blank common. The first 500 words, called the vector TEMP-ITEMP, are reserved as temporary working space and volatile communication between routines. TEMP is freely available for any non-standard block or any CHARM-routine. The rest of blank common is occupied by the dynamic store, called CM, ICM, BOUT, IBOUT. CM has 3 main parts : The BOUT-area, the storage area for the blocks and the superdynamic storage area.

BOUT holds the data summary words, these come from the DST, modified or augmented by the results of the user's Charm computations. BOUT has 2 parts : normal and special BOUT. The only thing special about special BOUT is that it is never touched by SUMX, while normal BOUT is cleared to zero for every event, to prevent left-over from the last event. Also, SUMX uses normal BOUT as general purpose working space during the initial and the output phase of each physical pass.

The sizes of normal and special BOUT are set by the user with the control-card **&NEW PASS** (see C o11). Because SUMX needs normal BOUT for other purposes, the size of normal BOUT is tacitly stepped up to 1000 words, should the user declare a shorter length. As a result, special BOUT cannot start before BOUT (1001).

/...

Section : SUMX
Date : 1.11.1967
(amendement)

A o10

The user must be careful during Charm-computations to stay within the limit of total BOUT, because immediately behind BOUT starts the area containing the control-information of the blocks. If some of this is destroyed, this will lead to an apparent mal-functioning of SUMX, which is sometimes traced back to the original fault only with difficulty. For protection see C 200 et seq.

Immediately behind BOUT starts the area which is available to all blocks for them to store anything they like. The subroutine MANAGE with its satellites MSIN and MLOCAT looks after an orderly use of this store. It tells a given block where it may start using space ('sign-on'), it expects to be informed by the block how much space the block has used when it is through ('sign-off'). MANAGE keeps a list of where the information of each block is and this may be consulted by the blocks ('connect').

The top end of the dynamic store is occupied by the so-called super-dynamic store. This holds the test-words, the dictionary, the control-cards read but to be saved in the event of pass-interruption and the information for the listing-pool. This storage is super-dynamic in the sense that the information is shifted around to make room if needed. In front of the super-dynamic store is a glacis of 100 words for safety.

The common cell IIDONE monitors the amount of space consumed by the blocks, it increases as more space is used up. The common cell IITOP indicates the space occupied by the super-dynamic store, it decreases as the requirement grows. When the two overlap, all space is used up (store-full condition) and pass-interruption occurs, see A o11.

Section : SUMX
Date : 15.3.1966
J. ZOLL

A oll

Break up of a logical into physical passes

Due to the finite size of the core-store it may well happen that there is not enough space in the dynamic store to compile all the information required by the user in a single pass.

SUMX with its standard blocks has been programmed to cope with this situation, by automatically breaking down a logical pass into as many physical passes as may be needed. As a given block reads its control-cards, it reserves space in this dynamic store for each histogram etc., as it comes along. The standard blocks are programmed to detect the Store-full condition, i.e. when the dynamic store is full up. This condition is signalled to the main program (common cell IROOM) which proceeds to stages 2 and 3. At the end of this physical pass it returns to stage 1 and sends control to the block which continues from where it was interrupted.

This facility allows the user to forget about the finite size of the machine when preparing control-cards (except for BLOC4). He does however have to know about it so as not to get confused with the output which comes from the several physical passes.

It is this distinction between logical and physical passes which causes the difference between proper blocks and tactical routines. A tactical routine remains active for the whole of a logical pass, whilst a block is active only during the physical pass in which it has been requested.

Section : SUMX
Date : 15.3.1966
C. LETERTRE/J. ZOLL

A ol2

Production of Lists with the Listing Pool

The listing pool is designed to handle an indefinite number of lists coming from different sources : lists compiled by Block 14 or Block 15, lists of events which passed or failed specified tests in SELECT, the list of events mentioned in IGNOR, the list of events which could not be read properly because of magnetic tape failure. The option exists for an individual list to be printed on the line-printer, to be punched or to be written onto a binary tape.

The listing pool needs magnetic tapes for intermediate data storage - the user has to indicate to the operator as well as to SUMX which tapes are available (see *LIST and *BINARY, C ol7); also the listing pool becomes very inefficient if a vast amount of information is to be handled during any one pass. For these reasons, the user must unfortunately have some understanding of how the listing pool works.

During the initial phase the processors contact the listing pool to 'book' their lists. Each list is identified by 2 words, a BCD word usually giving the name of the processor (e.g. BLOC 14, BLOC 15, SELECT) and an integer word. For organising efficient operation the listing pool relies on the fact that during stage 3 the lists will be called up in the same order in which they have been booked during stage 1.

During stage 2, the operation stage, the processors compile the information from the current event and hand it together with the 2-word label identifying the list to the listing pool. The pool writes the 'list entries' piecemeal onto the scratch-tape ITSCRT. To reduce write/read time several list-entries are packed into a record. The shorter the list-entries are, the more of them will fit into a record; to prevent inefficient use, each entry is restricted to having a maximum of 40 words, not counting the 4 words of book-keeping information such as the label.

At the end of stage 2 the listing pool rewinds the tape ITSCRT, prints out a summary of what has been happening and squeezes out empty lists. The lists are copied from the tape ITSCRT onto the N buffer tapes ITLIST(1),.....ITLIST(N) in a cyclic fashion, i.e. all entries of the first list, of the lists N + 1, 2N + 1 ... are written onto the tape ITLIST(1).

/...

Section : SUMX
Date : 15.3.1966

A o12

This is to reduce idle read time and rewind time during the output stage. The user has indicated N, the number of buffer tape , and their unit-numbers on the SUMX-system card *LIST. For normal production runs N should at least be 2, normally 3, or more if a lot of data has to be handled. The use of many tapes is awkward in several respects, and unnecessary for short test-runs of some hundred events. Therefore, for such runs N = 0 or 1 is allowed. The meaning of 'such runs' is given more precisely in C o17. For N = 0 all lists simply remain on the tape ITSCRT, for N = 1 every second list is copied onto the tape ITLIST(1).

During the output stage the processors call on the listing pool to have their lists put out one by one. The pool finds the list on one of the buffer tapes, selects the entries for the current list and transmits them onto the appropriate medium, i.e. printer, cards or binary tape. Having reached the end of the list, the buffer-tape is rewound. The next list is bound to be on another tape (except if N = 0), so that the pool does not normally have to wait for the rewind, unless the lists happen to be of very different lengths. This danger is reduced by having N = 3 or more.

The processors specify the output-medium when they call for the output of each list, and also give the formats in case of BCD-output. With the lists of Blocks 14 and 15 the information content and the output formats are defined by the user on the control-cards to these Blocks. This is different for an important group of lists, the so-called LFAIL-lists. So far, 3 standard processors give rise to these lists: in SELECT the user can ask for lists of events which pass or fail given tests, IGNOR gives a list of the events mentioned on the control-cards to allow cross-checking, TAPE gives the list RTT of the events which have not been given to SUMX for processing because a read-error occurred. These lists answer the question 'which events ?'. The information content and the output format for these lists are defined by the subroutine LFAIL. This is a problem-dependent routine which any user has to re-write in accordance with what is on his DST. The SUMX-deck contains a 'standard' LFAIL which is good for DST's produced by SLICE in the normal mode. Information on how to write LFAIL is found in C 300.

Section : SUMX
Date : 15.3.1966
J. ZOLL

B o11

Example Pass 1 : The Bare Minimum

```
(1)  *NEW PASS 24,0
(2)  MINIMAL EXAMPLE PASS
(3)  *DISCARD  0
(4)  *TAPE
(5)  6          LABEL      2593 / L26
(6)  *BLOCK 6
(7)  'DISTRIBUTION OF BEAM MOMENTUM
(8)  100        .1         670
(9)  24
(10) *ALL DONE
```

These are the control-cards for a very simple pass, giving a histogram of the 24th word of each event on the DST.

Card 1 is the SUMX-system card initiating a new pass, the 24 indicates that only the first 24 words of each event are of interest. Special BOUT has length 0. Details : see C o11.

Card 2 gives a 72-column pass-title which is printed at the top of each page of the output.

Card 3 indicates that the records on the DST do not have a record label (label of 0 words). It is very important for the user to be clear about the purpose of this card, see C o14, C 100 et seq.

Card 4 is the request-card calling for TAPE, the processor which is responsible for the reading of the DST; details see C TAPE.

Card 5 is read by TAPE. It indicates the use of logical tape unit 6 for the DST. The first 'event' on the tape is a label, the name of the tape is 2593/L26; this is only used for printing - no checking is done by SUMX.

Card 6 is a SX-system card calling Block 6 into operation, the subsequent cards are read by Block 6 and instruct it about the processing wanted, see C BLOCK 6.

Card 7 is the 80-column hollerith title of the first (and only) histogram, it is used to identify the histogram if there are several.

Card 8 specifies the histogram parameters: 100 channels, channel width of 0.1, starting at 670.

Card 9 says that BOUT (24) is to be histogrammed.

Card 10 signals the end of the control-cards.

Section: SUMX
Date: 15.3.1966
J.ZOLL

B o12

Example Pass 2: Selection

```
≠NEW PASS 430,0
SECOND EXAMPLE PASS
≠TAPE     20
10                  DST EXP 93

≠SELECT
TEST      3
          124       BETWEEN   12        15

≠BLOCK    6
'TITLE OF FIRST HISTOGRAM
30        1         9.5
210
220
'TITLE OF SECOND HISTOGRAM
50        2.5       -10       3
430
≠ALL DONE
```

(1) The card ≠TAPE carries '20' in col. 11: only 20 events are read from the DST, this is used for debugging. C TAPE).

(2) The card ≠HEADCARD is missing: the records on the DST have a record-label of 2 words. (C 100).

(3) The control-cards to SELECT define TEST 3 to be true if HOUT (124) lies in the open interval (12.,15).

(4) All events give 2 entries into the first histogram, HOUT (210) and HOUT (220) are histogrammed. This histogram is said to have 'multiplicity 2'.

(5) Only those events which satisfy TEST 3 contribute to the second histogram.

Section : SUMX
Date : 15.3.1966
J. ZOLL

B o13

Example Pass 3 : Dictionary

```
≠NEW PASS 430,0
THIRD EXAMPLE PASS
≠DICTIONARY
3         LUCIEN
124       FRANZ
210       JEAN/10
430       LIZZY
≠SELECT
TEST      LUCIEN
          FRANZ     BETWEEN   12        15

≠BLOCK 6
'TITLE OF FIRST HISTOGRAM
30        1         9.5
JEAN
JEAN2
'TITLE OF SECOND HISTOGRAM
50        2.5       -10       LUCIEN
LIZZY
≠ALL DONE
```

(1) The Request-card ≠TAPE is missing, hence no DST has been specified, therefore SUMX will not execute this pass. This is used for checking the control-cards.

(2) The Request-card ≠DICTIONARY calls for operation of the context-routine DICTIO, which is not a processor. It reads its control-cards which assign the numerical values on the left to the identifiers on the right, see C DICTIO and C 910.

(3) The instructions received by SUMX from the remaining control-cards are identical to those in B o12, but they are written differently. Only HOUT-locations and Test-numbers can be represented symbolically.

TC PROGRAM LIBRARY

SECTION SUMX
DATE 15.3.1966

C 014

EXAMPLE PASS 4 CHARM + LISTS IN SELECT

```
      SUBROUTINE CHARM1
C-------------    SUMX CDE  27-JAN-1966    ----------------------------------
      DIMENSION    ITEMP(5001, ICM(40001,BOUT(40001,IBOUT(40001            BASIC 1
      EQUIVALENCE (TEMP,ITEMP),(CM,ICM,BOUT,IBOUT)                        BASIC 2
      COMMON TEMP(5001,CM(40001 /CHIEF/ITA,ITB,ITPU,JSTAGE,IROOM,ITEMS    BASIC 3
      COMMON /CHARMP/NREC,NWORDS,I1,L1,L2,L3,L4,L5,L6,ISECT                 CHARM
C-------------    END CDE  ----------------------------------------------------
      BOUT(512)= BOUT(8121**2
      RETURN
      END
C-----------------------------------------------------------------------------

      SUBROUTINE CHARM2

C-------------    SUMX CDE  27-JAN-1966    ----------------------------------
      DIMENSION    ITEMP(5001, ICM(40001,BOUT(40001,IBOUT(40001            BASIC 1
      EQUIVALENCE (TEMP,ITEMP),(CM,ICM,BOUT,IBOUT)                        BASIC 2
      COMMON TEMP(5001,CM(40001 /CHIEF/ITA,ITB,ITPU,JSTAGE,IROOM,ITEMS    BASIC 3
      COMMON /CHARMP/NREC,NWORDS,I1,L1,L2,L3,L4,L5,L6,ISECT                 CHARM
C-------------    END CDE  ----------------------------------------------------
      LOC= IBOUT(3) + IBOUT(L1)
      CALL CMSET (LOC, SQRT(SIN(BOUT(L2))**2+COS(BOUT(L3))**2))
      RETURN
      END
C-----------------------------------------------------------------------------

      SUBROUTINE CHARM3

C-------------    SUMX CDE  27-JAN-1966    ----------------------------------
      DIMENSION    ITEMP(5001, ICM(40001,BOUT(40001,IBOUT(40001            BASIC 1
      EQUIVALENCE (TEMP,ITEMP),(CM,ICM,BOUT,IBOUT)                        BASIC 2
      COMMON TEMP(5001,CM(40001 /CHIEF/ITA,ITB,ITPU,JSTAGE,IROOM,ITEMS    BASIC 3
      COMMON /CHARMP/NREC,NWORDS,I1,L1,L2,L3,L4,L5,L6,ISECT                 CHARM
C-------------    END CDE  ----------------------------------------------------
      IF (JSTAGE-2) 100,200,300

C--               INITIAL STAGE

  100 A= 0.
      RETURN

C--               OPERATION STAGE

  200 A= A + BOUT(L1)
      RETURN


C--               OUTPUT STAGE

  300 CALL PTITLE
      A= A+A
      WRITE (1(0,9000) L1,A
      RETURN

 9000 FORMAT (*0SQUARED SUM OF BOUT(*I4,* FOUND BY CHARM3=*F10.4)
      END
C-----------------------------------------------------------------------------
```

```
*MEN PASS 1200.0
FOURTH EXAMPLE PASS
*LIST   12.13.14.15
*DISCARD  1

*TAPE
25        LABEL     DST MAX
26        LABEL     DST MORITZ

*SELECT
TEST      2
          1         BCD       MAX
LIST
TEST      4
          1         BCD       MORITZ
LIST
TEST      1
          2         TRUE
          4         TRUE
AND       -12       BETWEEN   .5        12.5
LIST                FAIL

*CHARM
SQUARE              1
CORRECT   4         2         7         1025      1029
          4         2         8         1026      1030
TO SUM              -3        1200

*SELECT
TEST      24
          928       BIGGER    937
LIST                FAIL
******
```


Section : SUMX
Date : 15.3.1966

C 014

1. The card *LIST tells the listing pool to use tapes 12, 13, 14 and 15 as scratch tapes, see C 017.

2. *DISCARD 1, the first work of each DST-record is to be ignored.

3. There are 2 DST's to be processed, they both carry a label as first 'event'.

4. SELECT is instructed to give separate lists of the events which pass tests 2 and 4, and of the events which fail the 'main test', test 1. These lists are lists of identification, the quantities which identify the event and the formats for printing them are defined by the user through the subroutine LFAIL. The standard LFAIL on the library copes with event-identification as produced by SLICE. For details on LFAIL see C 300.

5. *CHARM requests the processor CHARM which reads the control-cards following. The Charm-subroutines used in this pass are listed and must be included in the deck with the main program, they are discussed in C 200. Col. 21-30 of the control-cards specify to which Charm-subroutine the call refers, cols. 31-80 give the 5 arguments of the call, and the call takes place only if the test mentioned in cols. 11-20 is true, for each event. On the 4th control-card the Charm-number is -3, this instructs CHARM to make an extra call to CHARM3 during the initial stage, immediately after the card has been read, and also during the output stage.

   For each event, the calls to the various Charm-routines are made in the order of the control-cards, so these routines can talk to each other, more precisely, any Charm-routine can use the results of the ones called earlier.

6. SELECT is called a second time to operate on the results of the Charm-routines.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C ool

Control-cards to SUMX 466 : Generalities

The ensemble of the SUMX control-cards consists of 2 quite distinct sets :

1) The control-cards for any block specify the detailed operation wanted from this block. In principle each block reads its own control-cards, hence their formats could be anything. In fact, the standard blocks call on the SUMX-system (routine CARDIN) to have their cards read for them. This requires strict conventions for the formats, which are needed anyway if anyone is to remember anything. These conventions are described in C 900 et seq., the cards for the blocks are described in the various papers of subsection C, e.g. the cards for SELECT in the paper C SELECT.

2) The SUMX-system cards regulate the overall operation of the SUMX system and provide context information. All SUMX-system cards carry the symbol ± in column 1. Col's 2-10carry the text identifying the card. Parametric information starts at Column 11, blanks are ignored. The SUMX-system cards are not read by the same mechanism as the other control-cards, and no particular conventions apply beyond those already mentioned.

   The 'Request-cards' are a subset of the SUMX-system cards. They call for the operation of the blocks, e.g. ±BLOCK 6 or ±SELECT. These cards are described along with the control-cards of the relevant block. The other SUMX-system cards are described in C oll et seq.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C oll

±NEW PASS
±NEW RUN
±ALL DONE

SUMX-system cards : ±NEW PASS, etc.

The first card of any pass must be ±NEW PASS (or possibly ±NEW RUN), the second card must specify a 72 column pass title.

The ±NEW PASS (or ±NEW RUN) card has a double purpose: it heads all cards of a pass, specifying certain parameters for this pass; it also terminates the cards for the previous pass, if any.

The ±NEW RUN card has exactly the same function and formats as ±NEW PASS, except that it signals the fact that the DST's used for the previous pass are no longer needed and may be unloaded in good time.

The card ±ALL DONE must be the last card in the deck of control-cards. It acts as terminator for the very last pass and the job.

Examples :

10	20
±NEW PASS	1250,590

This allocates dimensions 1250 to normal BOUT and 590 to special BOUT, i.e. BOUT is 1840 words long.

10	20
±NEW PASS	600,0

Special BOUT may have zero length. Normal BOUT must have at least 1000 words, because it is used as working space during the initial and the output phases. Therefore, the present example does not allocate 600 but 1000 words to normal BOUT.

But the information '600' is retained, its interpretation depends on the DST format used. With DST Format 1 (C 101) it signals that, of any event, only the first 600 words are to be read. With DST Format 2 (C 102) this can be used to indicate the size of the DST records. For details see C 100 et seq. and D 006, common cell FTK. Note that special BOUT cannot start before BOUT (1001).

/...

Section : SUMX
Date : 15.3.1966

C o11

*NEW PASS
*NEW RUN
*ALL DONE

10	20
*NEW PASS	

If no dimensions for BOUT are given, the values from the previous pass are used. If there is no previous pass, the program assumes 1000,200.

The three examples so far cause a new pass to start from scratch, except that DST-allocation and possibly BOUT-dimensions are carried over from the previous pass. This is different for the example to follow. Using the card *SAVE (see C o12) it is possible to dump the state of tactical routines request during some pass and to pick it up again for some later pass with -

10	20
*NEW PASS	KEEP 2

this restores level 2 of tactical routines. See C o12 for more details.

Section : SUMX
Date : 15.3.1965
J. ZOLL

C o12

*SAVE

SUMX-system cards : *SAVE

Take a schematical example job :

```
(1) *NEW PASS      2000,50                                  )
    ---                                                     )
    *TAPE                                                   )
    ---                                                     )
    *DICTIONARY                                             )
    ---                                                     )
    *SELECT                (block -1)                       ) level 1
    ---                                                     )
    *CHAR                  (block -2)                       )
    ---                                                     )
    *SAVE          save level 1                             )
    *SELECT                (block -3)                       ) level 2
    ---                                                     )
    *SAVE          save level 2                             )
    *--------                                               )
    ---
    ---
    ---

(2) *NEW PASS      KEEP 2                                   restore level 2
    ---                                                     destroy higher levels
    ---
    *SELECT                (block -4)
    ---
    ---
    ---

(3) *NEW PASS      KEEP 1                                   )  restore level 1
    ---                                                     )  destroy higher levels
    ---                                                     )
    *SELECT                (block -5)                       ) level 2
    ---                                                     )
    ---                                                     )
    *SAVE                                                   )  save new level 2
    *--------                                               )
    ---
    ---

(4) *NEW PASS      1500,0                                   destroy all levels
```

Section : SUMX
Date : 15.3.1966

C o12

*SAVE

The card *SAVE saves the present state of the tactical routines, the contents of special BOUT and the dictionary onto a scratch tape (more details - E SAVE). This may be done several times, thereby creating several levels (up to 45).

The card *NEW PASS KEEP 2 in the example starts the 2nd pass by restoring level 2, i.e. the dictionary, 2 calls to SELECT and 1 call to CHARH. The effect of this is as though the cards preceding the second card *SAVE had been read again.

Any card *NEW PASS always destroys all levels higher than the level it restores.

SUMX refuses to execute an *SAVE if, in the current pass, a request for a proper block (i.e. block 1-15) has already been made.

The prime use of this facility is internal to SUMX with breaking down a logical pass into physical passes. *SAVE and KEEP give the user access to this facility.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C o13

*OPTION

SUMX-system cards : *OPTION

This card enables one to set various options.

10	20	
*OPTION	FALSE	undefined Tests are false

10	20	
*OPTION	TRUE	undefined Tests are true

A given test may be always undefined just because it has not been defined; it may also be undefined for a particular event because its definition occurs within an EVALUATE-skip in SELECT which is skipped for this particular event. One may argue whether such a test should be interpreted to mean 'true' or 'false'. The user is free to have it either way for any one pass using either of the above cards. If neither card is given, FALSE is assumed. The option is not reset for a new pass.

10	20	
*OPTION	NOBLANK	ignore blank cards

10	20	
*OPTION	BLANK	read blank cards

Blank cards can easily slip into a deck unnoticed, causing havoc, so one would like SUMX to ignore them. On the other hand it happens in SUMX that some control-cards contain nothing but zeros and some people are too lazy to punch a zero when blank is just as good. Again you can have it either way. If nothing is specified CARDIN ignores blank cards. But as soon as *OPTION BLANK appears, blank cards are taken to be intentional.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 014

*DISCARD

*TAKE

SUMX-system cards : *DISCARD, *TAKE

10	20
*DISCARD	0

This card is very important for people using a DST not produced by SLICE. It directs the DST read-routine to ignore the n = 0 first words of each DST-record. To know what this is all about, you should read C 100. The integer in cols. 11-20 must be in the range $0 \leq n \leq 500$.

10	20
*TAKE	600

This card provides another way to specify the size of the records on the DST. For details see C 101 et seq.

Example :

*NEW PASS	1250
*TAKE	600

The card *NEW PASS reserves 1250 words for normal BOUT and indicates for the time being a record size of 1250 words. This is reset to 600 words by the card *TAKE occurring subsequently.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 015

*OUTPUT
*INPUT
*REWIND
*UNLOAD
*FORWARD
*BACKWARD

SUMX-system cards : Tape handling

These control-cards are not normally needed for simple SUMX runs. They enable one to get the SUMX output onto magnetic tape or to read the SUMX control-cards from magnetic tape, and to manipulate the tapes.

10	20
*OUTPUT	4

switches the output stream to logical unit 4.

10	20
*OUTPUT	

switches the output stream back to the system output.

10	20
*INPUT	4

switches the input stream to logical unit 4.

10	20
*INPUT	

switches the input stream back to the system input.

10	20
*REWIND	4

rewinds logical unit 4.

/...

Section : SUMX
Date : 15.3.1966

C o15

	10	20
*UNLOAD	39	

does a rewind-unload on unit 39.

	10	20
*FORWARD	4,12	

forespaces logical unit 4 over 12 files.

	10	20
*BACKWARD	4,12	

backspaces logical unit 4 over 12 files.

These tape-handling cards are executed by the SUMX-system as soon as they are read. They may be placed anywhere, even in the middle of the control-cards for a standard block.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C o16

*GO ANYWAY

SUMX-system card : *GO ANYWAY

Normally, SUMX thinks that a mistake has occurred, if the user asks for a pass without requesting any proper block. Sometimes however, this may be intentional, e.g. a pass merely to compile some lists from SELECT may be wanted. SUMX can be made to execute this pass by giving :

	10	20
*GO ANYWAY		

as the last control-card for this pass.

Section : SUMX
Date : 15.3.1966
C. LETERTRE/J. ZOLL

C o17

±LIST

SUMX-system cards: ±LIST

The card ±LIST informs SUMX of which tapes are available for the listing-pool: the scratch-tape ITSCRT and the buffer-tapes ITLIST(1), ITLIST(2) ... ITLIST(N). The operation of the listing-pool and the significance of the tapes is explained in A o12.

SUMX needs another logical tape, ITDYN, as scratch-tape for dumping the contents of the dynamic store if pass-interruption occurs or if the card ±SAVE is used. To economize on the number of tapes, ITDYN is set to the same unit-number as ITSCRT, and the two types of information are separated by an End-of-File.

1)

	11
±LIST	12, 13, 14, 15

This sets : ITDYN = ITSCRT = 12
ITLIST(1) = 13
ITLIST(2) = 14
ITLIST(3) = 15 and N = 3

(As far as the program is concerned, the upper limit for N is 20)

2)

	11
±LIST	12, 13

In this example N = 1. This is allowed if not more than 30 lists have been booked, which together to not contain more than 100,000 words. This limit is just reached e.g. by 20 lists of 500 entries each with 10 words/entry. The house-keeping information of the listing-pool going with each entry need not be counted. Note that the list RTT (faulty events), IGNOR and all lists from SELECT must be included in the count. With N = 1 only short lists for block 15 can be processed (see C BLOCK 15).

/...

Section : SUMX
Date : 15.3.1966

C o17

3)

	11
±LIST	12

In this example N = 0. This is allowed if not more than 10 lists have been booked which together do not contain more than 20,000 words. With N = 0 no lists for block 15 can be processed.

4)

	11
±LIST	0

This card switches the listing-pool off, ITDYN retains the value it had before this card was read.

5) The absence of the card ±LIST in a given SUMX-run is interpreted as ±LIST 12.

In a given SUMX-run, the card ±LIST should appear only once, placed after the title of the first pass, because trouble can result if the unit-number of ITDYN is changed after some information has been written onto ITDYN.

If the limits on the number of lists or the total number of words are reached during the pass, the listing pool switches itself partially off: the book-keeping continues, so that the summary printed after the pass will be correct, but writing onto ITSCRT stops and no lists will be produced. However, if Blocks 14 or 15 are active, the program stops there and then with a diagnostic, because the chances for a useless pass are high.

T.C. PROGRAM LIBRARY

Section : SUMX
Date : 15.3.1966
C. LETERTRE/J. ZOLL

C 018

*BINARY
*PUNCH

SUMX-system cards : *BINARY, *PUNCH

These cards specify the logical numbers of the tape-units for receiving binary output and card-image output from the listing-pool. They should be placed just after the card *LIST, and, of course, only if the tapes are needed.

Binary lists are by-passed if the card *BINARY is absent. Punched lists go onto the on-line punch if the card *PUNCH is absent.

Examples:

10	20
*BINARY 18	

10	20
*PUNCH 19	

PS/4447/472/dmh

T.C. PROGRAM LIBRARY

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 100

TC Convention for Library Tapes

1) So as to minimize dependence on any particular computer, TC put up the following convention for its library tapes :

   Of each logical record,

   the first word contains the length of this record minus 1.

   the second word contains a label giving the record type.

   the third word contains the 1st word of information.

   the fourth word contains the 2nd word, etc.

2) The use of this convention overcomes in principle the problem of reading records of indefinite length. But only if all is well, most terrible things can happen if there are tape-faults. Therefore, the SUMX versions operational on the CERN machines solve this problem in a machine-dependent fashion (subroutines XREAD of the General Section).

3) As a result, SUMX is independent on this, or any other convention. But it had to be programmed to ignore the first 2 words of TC tapes. It was reasonable to do the programming in a general fashion, such that the first n words (with $0 \leq n \leq 500$) of each logical record can be ignored. The relevant bits of program are in the subroutine TAPE and its subsidiaries. SUMX itself is only concerned with reading the control-card *DISCARD <n> and loading the common cell NDISCD with the integer n. (C 014)

Note : NDISCD is preloaded with 2. If it is to have a different value, in particular zero, a control-card *DISCARD <n> must appear in the deck. This is not very logical, but convenient for TC users, people using DST's not derived from SLICE must look out.

PS/4447/361/dmh

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 101

DST Format 1
(Standard)

1) Usage

DST's with this format contain ordinary logical Fortran records of arbitrary length on magnetic tape.

Such a tape is read by version 1 of TAPE, which uses version 1 of READST as a subroutine, which in turn uses XREAD of the General Section. At CERN all routines needed to read DST's with Format 1 are on the system-library.

The use of Format 1 requires tape-compatibility between the reading and the writing Fortran-system; i.e. a 7090 Fortran tape can be read on a 7090 or a UNIVAC 1107 with IIIc tape-units, but not on a CDC 3600.

2) Tape structure

The first logical record can, but need not, be a binary tape-label. If it is, TAPE writes the contents of the first 10 words onto the system-output in BCD and integer interpretation, but not in octal or floating-point. The existence of the label is signalled on the control-cards to TAPE (cf. C TAPE).

The events follow with 1 event per logical record.

The tape is terminated with at least 1 ordinary end-of-file mark.

3) Record label

The version 1 of READST ignores the first NDISCD words of each logical record. NDISCD is a common cell, it is preloaded with 2 and can be altered with the control-card *DISCARD <n>, for details see C 100, C 014.

4) Record length

Each record may be of arbitrary length; if the record is longer than the declared length of normal BOUT, the excessive words are lost.

Some time is gained in reading fewer words than BOUT can accommodate. Therefore READST asks for the reading of NTK words (NTK is a common cell) if NTK is shorter than the capacity of normal BOUT. NTK is loaded with the declared length of normal BOUT (this may be shorter than 1000, even though BOUT must have at least 1000 words, cf. C 011), NTK can also be set to be n with the control-card -

*TAKE <n> (cf. C 014)

NTK must exclude the discarded words.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 102

DST Format 2

1. Usage

DST's with this format contain ordinary Fortran records of restricted length, several complete events per record, and with it, control-information specifying the arrangement of the events within the record. The use of this format is strongly recommended for short events (~ 100 words); not to use it for very short events (~ 10 words) would be criminal.

2. Tape Structure

As with DST format 1, the first 'event' may be a tape label. As far as reading is concerned, there is no special treatment for this label, hence the label must be contained in a proper record of format 2, with the necessary control-information. It is alright to have the label in a record by itself, with the control-information signalling an event-content of 1. The tape must end with an end-of-file mark.

3. Record Structure

The lay-out of the events within each record must follow the conventions for PAKIN, see C 120. In particular, the first word must indicate the packing-mode used and the number of events contained in the record.

This first word of information may be preceded by a record label of NDISCD word, see C 100. Note that NDISCD is assumed to be 2 unless reset by *DISCARD 0.

4. Record Length

The program requires each logical record to be shorter or equal to an upper limit, because a buffer is needed to hold the record whilst the individual events are being processed. Little stands to be gained by having more than 1 physical record in a logical record, in which case full buffering would be lost on some machines and a tape-fault would destroy more events than necessary. Hence the upper limit (set in READST, version 2) is the size of a physical record, 255 words on the 3400/3800 and 512 words on the 6600.

Version 2 of READST asks PAKIN to read records of NTK words (not counting the record-label of NDISCD words), if NTK is shorter than this maximum length. NTK is loaded with the declared length of normal BOUT (C 011) and can be reset with the card *TAKE <n>, see C 014.

Section : SUMX
Date : 1.3.1967
L.D. JACOBS

C 103

DST Format 3

Introduction

The original SUMX as used in the Alvarez Group in Berkeley read compressed tapes derived from the encyclopaedic DST's for an experiment. Two notable features of these tapes were the following:

i) Several events were packed per record

ii) Only information relevant to a particular study e.g. effective masses and particle four-vectors, statistical data or unfitted momenta was stored.

Since it was felt that there was a need at CERN for tapes with these advantages a third version of READST has been written to read tapes with the format described below.

1. Usage

Where large numbers of events (10-20K or larger) with about 100 words of useful information per event are involved, it is recommended that a condensed and edited version of the DST's be employed. This together with the use of 800 b.p.i. density will reduce tape reading time considerably. READST version 3 also reads standard one record DST's.

2. Tape Structure

The first event may or may not be a label. A program TOAST which edits and condenses a DST and contains a charmlike routine MYDST, is available. This makes the "first event" a label. It is hoped to include in SLICE an option which will write DST's directly in format 3.

3. Record Structure

The TC convention implying NDISCD = 2 is followed. In order to make the records collatable, the first three words of the record are identical with BOUT(1), BOUT(2) and BOUT(3) of the first event in the record. The fourth word is the number of events in the logical record. The fifth word is the coded signal word which tells READST that the tape is packed. It is

{(070707070707)0707|0707 for the 6600 {3800} (7094)

which is in BCD a word with ten, eight or six 7's. This word separates the events in the record. The record is terminated by two of these words:-

EXPT. No.	ROLL. FRM.MN	blank	EVTS/REC	070707...07

FIRST EVENT	070707...07	SECOND EVT.	0707...07	THIRD EVT.

LAST EVT.	0707...07	0707...07

Section : SUMX
Date : 1.3.1967

C 103

4. Event Structure

Strings of two or more zeroes are collapsed out. The advantages here should be obvious. Often a numbering scheme for BOUT locations, which is consistent between one event type and another gives rise to tape-wasting strings of blank locations. With the present event structure, convenience and time-saving are combined at the expense of more computation and a more complex tape structure. An event will consist of several blocks of the form

M1	M2	(M2-M1 + 1) words

1st. 2nd. 3rd. (M2-M1 + 3)rd word

where M1 and M2 are integers defining locations BOUT(M1), BOUT(M1 + 1).. ..BOUT(M2). There is scope for error here. If things go wrong when reading the tape, a test is made on the values of M1 and M2, a packing control error is printed, NDUD is upped by one and READST quits reading the record. The largest value of M2 is used for zeroing BOUT prior to storing the next event in memory.

5. Record Length

Assumed to be 1000. The record length can be set smaller in TOAST and NTK adjusted accordingly, cf. C 101.

T.C. PROGRAM LIBRARY

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 120

Record lay-out for PAKIN

PAKIN allows reading of packed tapes, where several units of information, events, are grouped into one Fortran record.

2 packing modes are available : the 'fixed mode' and the 'stringy mode'.

Fixed mode

word 1	$N*(1 + 512 + 512^2) = N* 262657$
	N events are contained in this record. The +ve sign signals the 'fixed mode'. Multiplication with this factor creates redundant information, which is used if a tape-fault occurs.
word 2	n
	n words contained in each event.
word 3	1st word of first event.
word n + 2	last word of first event.
word n + 3	1st word of second event.
etc.	

Stringy mode

word 1	$-N* 262657$
	N events in this record; the -ve sign signals the stringy mode.
word 2	i_2, (address of 2nd address.)
word 3	1st word of 1st event.
...	
word i_2	i_3, (address of 3rd address.)
word i_2 + 1	1st word of 2nd event.
...	
word i_N	$i_{(N+1)}$, (address of final address.)
word i_N + 1	1st word of last event.
...	
word $i_{(N+1)}$ - 1	last word of last event.
word $i_{(N+1)}$	$i_{(N+2)}$, address of the would-be next address. This last word should be present only if the record can hold it. It should give an address definitely outside the range of the record.

/...

PS/4447/364/dmh

Section : SUMX
Date : 1.11.1967
(amendement)

C 120

Examples in the stringy mode

Where a physical record holds 511 words, and say we want a record label of 2 words which PAKIN will be directed to ignore. This leaves a capacity of 509 words for the record proper.

In the first example, there is some room left after the last event, but not enough for the next event :

word -1 :	label 1
word 0 :	label 2
word 1 :	-787971 (= $-3*262657$)
word 2 :	203 (= i_2)
word 3 :	first word of 1st event with 200 words
word 203:	304 (= i_3)
word 204:	first word of 2nd event with 100 words
word 304:	505 (= i_4)
word 305:	first word of 3rd and last event with 200 words
word 505:	706 (= i_5) or not on tape
(word 506-509	rubbish or not on tape)

In the second example, the 1 st event finishes exactly with the last word of the record :

(words -1 to 303	the same as with the first example)
word 304:	510 (= i_4)
word 305:	first word of 3rd and last event with 205 words
word 509	205th and last word of last event.

PS/4447/364/dmh

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 200

CHARM Routines Part 1

This paper gives information on the programming of simple Charm-routines.

The paper B ol4 contains 3 Charm-routines as examples and you really must look at it, or else the present paper will remain obscure. The first 2 routines are of the simple type, or 1-stage routines; the third routine is a 3-stage routine, this type is discussed in the next paper.

The name of a Charm-routine is CHARM1, CHARM2,... or CHARM9; up to 9 routines may be present at any one run; which routine is to be used for a given call, is indicated by giving the integer 1, 2, ... or 9 on the control-card controlling the call (see C CHARM).

The 'Common-Dimension-Equivalence Deck' contains the necessary declarative statements :

a) The vectors of Blank Common are defined (see A ol0). TEMP is a short-term working space, the integer name for the same vector is ITEMP. The vector BOUT is the dynamic store, IBOUT its integer name. In SUMX this dynamic store is usually called by its other name, CM, the use of the symbol BOUT is reserved for designating the Data Summary Words. But since the BOUT area is merely the beginning of the dynamic store, there is really no difference between CM and BOUT as far as the machine knows, and the card BASIC 2 contains the due equivalence. The dimension of BOUT is given as 4000, this is a mere dummy, the actual length is defined in the main program and is bound to be longer than 4000.

b) The labelled common block /CHIEF/ contains 3 program parameters the user is likely to be interested in :

ITA is the logical number of the unit containing the control-cards

ITB is the logical number of the line-printer. ITA and ITB are usually the system-input and system-output units, but not necessarily, see C ol5.

JSTAGE specifies the stage SUMX is currently operating, it is 1, 2 or 3 during the initial, operation or output stage. This is of interest only for 3-stage Charm-routines.

/...

Section : SUMX
Date : 15.3.1966

C 200

c) The labelled common block /CHARMP/ contains the following parameters :

NREC is the position of the current event on the current DST

NWORDS are the number of words contained in the current event

ii is the pointer to the information of the current Charm-call in the dynamic store. This is needed by the user only for 3-stage Charm-routines which need more parametric information than can be accommodated in L1, L2 ... L5. (see C 202).

L1 - 5 The 5 cells L1 ... L5 are loaded with the integers appearing in the 5 parameter fields of the control-card controlling the current Charm-call (C CHARM).

L6 contains the last location used with CHSET.

ISECT contains an integer set by the user : the section-number.

L6 and ISECT are used for diagnostics only. They are initially cleared to zero by CHARM for each Charm-call. If an error occurs, the diagnostics given by SUMX contain a print-out of the section-number. Apart from the initial clearing, this number is entirely under the control of the user, by setting it in a given Charm-routine of some size, he can define sections within the routine. L6 is discussed below.

In the example CHARM2 of B ol4 the following happens : an expression is computed : A = SQRT ($\sin^2$ + $\cos^2$) and this is stored into BOUT(LOC). The address LOC itself depends on the summary words. This situation is very dangerous - though maybe unavoidable - because if LOC turns out to be outside the range of BOUT some information somewhere in the store will be clobbered, and more likely than not this will lead to a malfunctioning of SUMX which is very difficult to track down to the source of the trouble. This is avoided by using the routine CHSET with the following specifications :

CALL CHSET (LOC,A)

1) loads L6 = LOC for diagnostics
2) checks if LOC is a legitimate BOUT-address
3) if yes, performs BOUT(LOC) = A
   RETURN
4) if not it takes SQRT(-1), the square-root of -1, to connect into the diagnostic system, followed by RETURN.

People are strongly urged to use CHSET whenever there is the slightest danger, i.e. always except in the simplest case, as shown in CHARM1 of B ol4.

T.C. PROGRAM LIBRARY

Section : SUMX
Date : 20.6.1966
C. LETERRIER/J. ZOLL

C 300

Subroutine LFAIL

This subroutine defines the information content and the printing formats for the so-called LFAIL lists. These lists can be generated in SELECT by asking for the lists of events which pass or fail given tests. The identification of the events depends on the lay-out of the information on the DST, and has to be programmed in accordance with it.

The SUMX deck contains 2 versions of LFAIL, 'SLICE' and 'EXAMPLE', both are listed on page 2 of this paper. The library on the 6600 contains a dummy, which suppresses all LFAIL lists, unless it is overruled by a live LFAIL coming in through the card reader.

SELECT contains the following calls (symbolically) :

ITEMP (2) = 6H SELECT
ITEMP (3) = ± MT (test number)
CALL LFAIL
CALL LIST 1, 2, or 3 for stage 1, 2, 3.

This places the BCD and integer label of the list into TEMP (2) and TEMP (3), calls LFAIL to complete the information and then calls the listing pool to digest the information.


```
CERN TC LIB  -- SUMX 466 --   V5.25    1-JUN-66   SLICE 0/9   LFAIL    PAGE  1
C+
      SUBROUTINE LFAIL                                                      39+
C--       LFAIL FOR STANDARD DST COMING FROM SLICE
C--       RECORD NUMBER,EXPERIMENT,ROLL/FRAME,GMIND DATE-TIME
C---------------     SUMX CDE  27-JAN-1966          ------------------------
      DIMENSION    ITEMP(500), ICM(4000),BOUT(4000),IBOUT(4000)          BASIC 1
      EQUIVALENCE (ITEMP,ITEMP),(CM,ICM,BOUT,IBOUT)                      BASIC 2
      COMMON TEMP(500),CM(4000) /CMIEF/ITA,ITB,ITPU,JSTAGE,IRDOM,ITEMS   BASIC 3
      COMMON /CHARMP/NREC,NWORDS,I1,L1,L2,L3,L4,L5,L6,ISECT               CHARM
C---------------     END CDE                        ------------------------
      EQUIVALENCE   (LH,CM(4001)),   (LE,CM(800))
      EQUIVALENCE (FMTH,CM(4013)), (FMTE,CM(808))
      DATA (KEY=5HLFAIL)


      GO TO (100,200,300), JSTAGE

C--               INITIAL PHASE. MARK LFAIL LIST BY SETTING THE KEY

  100 ITEMP(1)= KEY
      RETURN

C--               OPERATION PHASE. COMPILE LIST
C--               THE LIST LABEL HAS BEEN SET INTO TEMP(2),TEMP(3)
C--                              BY THE CALLING ROUTINE

  200 ITEMP(1)=9
      ITEMP(4)=NREC
      ITEMP(5)=IBOUT
      ITEMP(6)=IBOUT(2)/100000
      ITEMP(7)=MOD(IBOUT(2),100000)
       TEMP(7)=FLOAT(ITEMP(7))/10.
      ITEMP(8)=IBOUT(4)
      ITEMP(9)=IBOUT(5)
      RETURN

C--               FINAL PHASE. PRINT LIST

  300 ITEMP(1)= 0
      LH= 1
      CALL UHOLLR  (FMTH,61,
     + 41H(1HM 3HM RECORD  EXP ROLL/FRAME GMIND DATE -  TIME    LFAIL LIST))
C**************    512   86 4953/1024.1    660101 1234567

      LE= 1
      CALL UHOLLR  (FMTE,41,
     + 4(1HM  1X    I6, I5, I9,1H/F6.1,      I10,I8)                    )
C**************    512   86 4953/1024.1    660101 1234567
      RETURN
      END                                                                   39
C*************************************************************************
```

```
CERN TC LIB  -- SUMX 466 --   V5.25    1-JUN-66   EXAMPLE   LFAIL    PAGE  1
C+
      SUBROUTINE LFAIL                                                      40+
C--       LFAIL, AN EXAMPLE, IT GIVES
C--       RECORD NUMBER,IBOUT(1),IBOUT(2),IBOUT(3)  AS INTEGERS
C---------------     SUMX CDE  27-JAN-1966          ------------------------
      DIMENSION    ITEMP(500), ICM(4000),BOUT(4000),IBOUT(4000)          BASIC 1
      EQUIVALENCE (ITEMP,ITEMP),(CM,ICM,BOUT,IBOUT)                      BASIC 2
      COMMON TEMP(500),CM(4000) /CMIEF/ITA,ITB,ITPU,JSTAGE,IRDOM,ITEMS   BASIC 3
      COMMON /CHARMP/NREC,NWORDS,I1,L1,L2,L3,L4,L5,L6,ISECT               CHARM
C---------------     END CDE                        ------------------------
      EQUIVALENCE   (LH,CM(600)),   (LE,CM(800))
      EQUIVALENCE (FMTH,CM(601)), (FMTE,CM(801))
      DATA (KEY=5HLFAIL)


      GO TO (100,200,300), JSTAGE

C--               INITIAL PHASE. MARK LFAIL LIST BY SETTING THE KEY

  100 ITEMP(1)= KEY
      RETURN

C--               OPERATION PHASE. COMPILE LIST
C--               THE LIST LABEL HAS BEEN SET INTO ITEMP(2),ITEMP(3)
C--                              BY THE CALLING ROUTINE

  200 ITEMP(1)=7
      ITEMP(4)=NREC
      ITEMP(5)=IBOUT(1)
      ITEMP(6)=IBOUT(2)
      ITEMP(7)=IBOUT(3)
      RETURN

C--               FINAL PHASE. PRINT LIST

  300 ITEMP(1)= 0
      LH= 1
      CALL UHOLLR (FMTH,nn,
     + 60H(1HM RECORD  EXP ROLL/FRAME GMIND DATE -  TIME    LFAIL LIST))

      LE= 1
      CALL UHOLLR (FMTE,II,41H(1HM  1X    I6, I5, I9,1H/F6.1,      I10,I8))
      RETURN
      END                                                                   4
C*************************************************************************
```

Section : SUMX
Date : 20.6.1966

C 300

Comments

1. The CDE deck provides the stage number in the Common cell JSTAGE, this is 1, 2 or 3 for the initial operation or output stage. NREC contains the position of the current event on the current DST. Also, the vectors TEMP and BOUT are defined.

2. During stage 1, LFAIL merely has to supply the key-word, this makes the listing pool realize that there is a live LFAIL in operation, so it accepts the booking of the list.

3. During stage 2 it compiles the list entry, starting with TEMP(4), in TEMP(1) it indicates the length of the list by giving the last address in TEMP containing something. Please, keep this information down to the essentials.

4. During the stage 3, the formats for printing are loaded. TEMP(1) must be cleared to zero, never mind why. LH and LE indicate the number of lines caused by each of the 2 formats. The calls to UHOLLR load the BCD-text in the 3rd agrument into the vector specified in the 1st argument, the 2nd argument indicates the length of the text. Note the equivalences for LH, LE, FMTH, FMTE with BOUT, this is where the listing pool expects the information.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 900

Control-cards to the Blocks : Conventions

1) The 'SUMX convention'

Throughout this manual we adopt a notation for the relation between a number coming from the DST and its address in BOUT, thus :

X = BOUT (Xloc)

W = BOUT (Wloc)

As an example, the momentum of the beam track might be the 65th word of the event, in which case we have : Xloc = 65

and p_B = BOUT (65).

If SUMX is to use the numbers in BOUT in arithmetic operations (as it usually does, simple listing is the exception), then SUMX must know whether this number is an integer or a floating point number. To cope with this, we use the age-old 'SUMX convention' :

> X on the DST is an integer if Xloc on the control-card is given a negative sign.

All integers are changed into floating point form before any arithmetic operations are performed on them. (the one exception is the equality test in SELECT).

2) Lay-out of the control-cards (excepting SUMX-system cards)

All control-cards are sub-divided into 8 fields-of-ten. Each piece of information occupies one or several complete fields-of-ten. The rules for punching the information in these fields are these :

Floating point number. It may be placed anywhere in the appropriate field-of-ten. If the number has a fractional part, the decimal point must be given; if it has not, the decimal point may be omitted. Significant zeros must be punched, blanks are ignored.

Integer number. It may be placed anywhere in the appropriate field-of-ten. It may, but need not, be followed by a decimal point; fractions are ignored. Significant zeros must be punched, blanks are ignored. If the integer is a BOUT-location or a test-number, an alphabetic identifier may be punched instead, see C 910.

Hexadecimal number. This occupies 1 field-of-ten. Significant zeros must be punched, non-significant zeros may be punched, the minus sign is not allowed, blanks are ignored, the result is right-adjusted.

/...

BCD text. (e.g. histogram title or commentary) may occupy up to all 8 fields-of-ten. It must not contain illegal characters.

BCD information. (e.g. 'TEST' or 'BETWEEN' in SELECT) must be left-justified in the appropriate field-of-ten, only the first 3 characters need be present. Note particularly that this does not apply to SUMX-system cards, the ones with '*' in col. 1.

BCD option is yes-no information to the program. The option is 'false' if the whole field-of-ten is blank, it is 'true' if at least one of the first 6 characters is non-blank. The third case is undefined, what happens depends on the machine.

3) Interpretation of the control-cards

In the papers of Section C which give the specifications for the control-cards to the blocks, the 'formats' used to interpret each field-of-ten are indicated. The following codes are used :

F floating point number

U floating point number, zero is replaced by 1.

I fixed point integer - V as I, but zero replaced by 1

L fixed point integer, BOUT-address

T fixed point integer, Test-number. Fields interpreted with L and T may contain a symbolic BOUT-address or test-number

O octal number (occupies 2 fields-of-ten)

A3 BCD-information

B BCD-option, blank is 'false'

A BCD-text, single field-of-ten

R,S BCD-text on the rest of the card (several fields-of-ten)

Note, that though some of these codes resemble FORTRAN format codes, the control-cards of SUMX are not read through FORTRAN formatted input, but CARDIN reads and interprets them character by character.

Control-cards to the Blocks : Card-types

The various control-cards can be subdivided into the following classes:

1) SX-system cards, serious kind: e.g. *SELECT or *NO PASS.
These cards normally call a routine into operation which itself reads more control-cards, all Request-cards come under this heading.

2) SX-system cards, frivolous kind:
These cards have immediate action, setting flags or handling tapes.
(*OPTION, *GO ANYWAY, *DISCARD, *TAKE, *INPUT, *OUTPUT, *REWIND, *BACKWARD, *FORWARD, *UNLOAD)

3) Heading card: This is always the first card of the unit for any block, e.g. the heading card for a histogram. With this type of card, the first field-of-ten always contains BCD-information (normally with ' in col. 1).

4) Parameter card: This carries parametric information, its interpretation depends on its position relative to the heading card. Example: the card specifying the number of channels in a histogram.

5) Multiplicity cards always come in crowds, at least potentially. e.g. with block 6, the multiplicity-cards specify which words are to be histogrammed. With any one histogram, there is always an indefinite number of multiplicity-cards. Information on all multiplicity-cards is arranged such that the first field-of-ten cannot meaningfully contain zero.

6) Zero-card: a blank card with 0 in col. 1. This is not its only form, but this standard form is used everywhere in this manual. What matters is that the first field-of-ten read as an integer contains 0. This card terminates the multiplicity-cards.

7) Skip-cards: EVAL <NT> and FINISH <LN>, the control-cards for the skipping-facility, see C 911.

8) FINISH in col. 1-6 and empty second field-of-ten. This indicates the end of the control-cards to a block or a context-routine.

All cards to SELECT are heading cards, Block 10 and CHAN do not have any Multiplicity cards, nor any Parameter-cards, because all the information is on the heading cards.

In the various papers of this section, giving the specifications for the control-cards to the blocks, the card-types are indicated by the letters REQ, HEAD, PAR, MULT, ZERO, SKIP, FIN.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 902

Control-cards to the Blocks ; Termination

There are 3 kinds of indefinite card sequences where SUMX needs an indication of the end:

1) end of cards for a pass
2) end of cards for a block
3) end of multiplicity-cards

1) End of pass is signalled by any one of ≡NEW PASS, ≡NEW RUN, ≡ALL DONE. The first 2 cards serve the double purpose of terminating the last and opening the next pass (cf. C 011).

2) The end of cards for a block is normally signalled by the occurrence of the Request-card for the next block, or a pass-terminating card, or any other SX-system card of the serious kind (group 1 of C 901). The card FINISH (group 8) can also be used for terminating, but is never needed.

3) The end of a set of multiplicity-cards is normally signalled by the occurrence of the heading-card (group 3 of C 901) of the next unit, or by the occurrence of a card which terminates the block, or by the occurrence of a skip-card (EVAL, FIN, group 7 of C 901). The Zero-card (group 6 of C 901) can also be used for terminating, but is not normally needed.

To be sure that SUMX recognises a heading-card as such, rather than mistake it for a badly punched multiplicity-card, the heading-card should carry in col. 1 the apostrophe (4/8 punch). This is not needed with CHARM and Block 10, because those do not have multiplicity-cards.

For backward compatibility, i.e. to enable using cards prepared for the previous SUMX versions, the following may be noted:

a) If the zero-card is used (it was obligatory with the old version) there is no problem, since the zero-card terminates, and the next card is automatically taken as the heading-card, with or without apostrophe. If the zero-card is represented by a blank card, one has got to look out, because nowadays SUMX ignores blank cards, unless it is told not to (≡OPTION BLANK, C 013).

b) If the dictionary-facility is not used, any alphabetic character in the first field-of-ten is sufficient to ensure the character of the heading-card.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 910

Use of the Dictionary Facility

This facility allows one to use symbolic BOUT-addresses and symbolic test numbers on the control-cards to all standard blocks, provided that the value and the dimension of the identifier used in the symbol has been declared prior to the symbol appearing on a control-card. (cf. C DICTIO).

An arbitrary number of symbols can be derived from any identifier. Take the following example: ≡DICTIONARY (cf. C DICTIO).

```
127     JACK
501     TRACK/7
```

This assigns the values 127 to the identifier JACK and 501 to the identifier TRACK. At the same time TRACK is declared to be a 2-dimensional array, whose 'weak' index has the range 1 to 7. This means, the information on the tracks of an event is stored in an array with 7 words per track, the first track starts at 501, the second at 508, the third at 515, etc.

There are 3 ways to derive symbols from an identifier.

Addition:

JACK+3	:	130
JACK-12	:	115
JACK	:	127
TRACK+100	:	601

Vector: The symbol JACK/4 means the 4th element of the JACK-vector, hence it has the value 130.

Matrix: The symbol TRACK3 means the first element of the 3rd track; it has the value 515 since each track is 7 words long.

The symbol TRACK3/6 denotes the 6th element of the 3rd track, hence it has the value 520.

Unlike in FORTRAN, maximum dimensions of 1 and 2-dimensional arrays do not have to be declared in the dictionary, because SUMX is not interested. In particular, the dictionary does not need to be informed of your intention to use an identifier for a vector. For a matrix, the effect of the 'strong' index depends on the declared range of the 'weak' index, hence it must be given.

/...

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 910

These are the rules for using identifiers on the control-cards :

1) An identifier consists of up to 6 letters. Figures and special characters will not work.

2) On the control-cards, a symbol must be inside the proper field-of-ten. A 6-character identifier leaves only 4 columns for the -ve sign and the indices, so you had better have short names for 2-dimensional arrays.

3) Integer content of the designated BOUT-word is flagged according to the SUMX-convention by preceding the whole symbol with a minus sign, i.e. -JACK+2 means - (JACK+2). (The brackets are implied, they must not be punched.)

4) The use of a symbol standing for an integer works only for BOUT-addresses and Test-numbers, but not for other integer parameters, such as the number of channels in a histogram.

5) Blanks anywhere in the field-of-ten are ignored.

PS/4447/368/dmh

Section : SUMX
Date : 15.3.1966
J. ZOLL

C 911

Use of the Skipping facility

1) This facility enables a specified sequence of units (e.g. histograms) to be conditionally skipped, depending on the truth-value of a test. The string of units skipped is preceded by an EVALUATE card, and followed by a FINISH card. Any number of sequential skips can be defined; skips may also be nested, and nesting may continue to the 19th level.

2) Skipping has several uses:

   a) The evaluation of tests in detail is avoided if there is some overriding reason, why they will all be false, thus saving computer time.

   b) When n tests have been defined in SELECT, all n^2 "AND-combinations" of them can be used without further test definitions.

   c) Since the skipping facility is available in SELECT itself, it is possible to define a test in one of several ways, or not at all, depending on the truth of a previously defined test.

3) An evaluate card has in

   col. 1-3 EVA

and in col. 11-20 <MT> the test-number on which the skip is conditional.

The skip takes place if test MT is false.

A finish card has in

   col. 1-6 FINISH

and in col. 11-20 <LN> the level-number, which is defined below. The final finish card with blanks in col. 11-20 closes all skips, as well as serving its usual function of terminating the control-cards for the block.

4) Whilst ℓ evaluate cards are needed to define a nest of level ℓ, a single finish card terminates a nest of arbitrary level.

The level number LN is defined to be zero initially, and to increase by 1 every time an evaluate card is encountered. A "FINISH <LN>" card terminates all skips on level LN and higher, and resets the level to LN-1, skips on this and lower levels still continuing.

PS/4447/368/dmh

T.C. PROGRAM LIBRARY

Section: SUMX
Date: 15.3.1966
J.ZOLL

C BLOCK 4

BLOCK 4 for Handling Histograms

Block 4 combines histograms of Block 6 by addition, multiplication or division into 'compound histograms' during stage 3, the output stage of SUMX. The spill-channels are always ignored. Before being used in the compounding operation each individual histogram can be transformed in 3 ways: scaling, reflection and reduction.

For each compound histogram the histograms are used in the order and with the mode of operation given on the multiplicity cards. Each histogram is first pulled out of the store of Block 6 and dumped into a buffer-vector so that the original is not disturbed. In the buffer the required transformations are performed in this order:

tra 1) multiplication by a factor: each channel of the histogram is multiplied by the factor W given on the multiplicity card, W may be negative.

tra 2) reflection: the histogram is turned over in the buffer, such that the last channel becomes the first and vice-versa. The lower and the upper limit are interchanged, the channel-width is set to be negative. Folding for symmetries can be achieved by judicious use of this option.

tra 3) reduction: the transformed histogram is derived from the old by adding groups of r channels together, starting with channels 1,2,...r, to form 1 new channel each. The histogram parameters are transformed to have $(\Delta x)_{new} = r * (\Delta x)_{old}$ as channel width and $n_{new} = (n_{old} - 1)/r + 1$ as number of channels. r = 0 on the multiplicity card cuts out this option.

The transformed histogram is used to operate on the 'compound'. The compound is cleared to zero initially and contains at any stage the result of the operations up to this stage - clearly the card for the first histogram has to specify the mode 'addition'. The 4 possible modes of operation - depending on the multiplicity card for the histogram - are these :

op 1) addition : the histogram is added, channel by channel, to the compound.

op 2) multiplication (*) : the content of each channel of the compound is multiplied by the content of the corresponding channel of the histogram.

op 3) division (/) : each compound-channel is divided by the corresponding histogram-channel.

op 4) inverse division (ø) : each histogram-channel is divided by the corresponding compound-channel

If division by zero occurs for a given channel, the result is set to zero and a flag is raised causing all further operations with this channel to be by-passed and an * to be printed in the line labelled 'FLAGS' for this channel.

Example: Say we want to form $a/(\frac{b}{c} - \frac{d}{e})$, where a, b ... e are the contents of any channel of histograms 1, 2 ... 5 in block 6, also we want to multiply the compound by 1000 to scale for plotting. The expression needs to be reshuffled to enable serial operation :

$$\frac{c}{(\frac{cb}{c} - d)} * a$$

and the multiplicity cards contain this information in this order :

	Histgr.	W	action
(e)	5	1.	+
(b)	2	1.	*
(c)	3	1.	/
(d)	4	-1.	+
(e)	5	1.	ø
(a)	1	1000	*

Block 4 cannot execute all arithmetic expressions you can think of, because it has no buffers for holding intermediate results apart from the compound, but hopefully the more frequent ones are covered.

Section : SUMX
Date : 15.3.1966

C BLOCK 4

At any stage after the first histogram has been loaded into the compound, there are 3 parameters associated with the compound :

NX, number of channels - taken from the shortest histogram so far.

ΔX, channel width - taken from the first histogram; this may be negative if this histogram needed reflection.

XL, lower edge - also taken from the first histogram.

The histograms are combined channel by channel, irrespective of what the variable value at the channel edges may be. This can well be nonsense, and therefore a check is performed for each histogram, to see whether its 3 parameters NX, ΔX, XL agree with those of the compound. This check gets rather complicated because of the reflection-option and it assumes that the first histogram was not reflected. If the check fails, the only action taken is to print a warning DSCRP in the line labelled COMMENT for all histograms showing a discrepancy. It is up to the user to check whether the compounding operation is indeed what he intended it to be. Spurious discrepancy warnings will appear if the first histogram was reflected, so one should avoid this if possible.

Note the following peculiarities of block 4 :

Unlike a normal block it does nothing during the operation stage of SUMX, because its raw-material is not the summary data but the result of block 6. This result has to be in the store when block 4 does its job. With big SUMX-runs there is a risk that the pass gets interrupted in the middle of block 6 because the store is full, thereby possibly missing some histograms which are needed by block 4. This danger is reduced by grouping the histograms needed by block 4 at the beginning of block 6, and by putting the control-cards for block 4 ahead of those for block 6 - but this is good practice anyway.

Section: SUMX
Date : 15.3.1966

C BLOCK 4

Block 4 - Lay-out of Control-cards

	col.	11	21	31	41	51	61	80
REQ	*BLOCK 4	<jbloc>						
	•	•	•					
HEAD	' <Hollerith title of compound histogram>							
MULT	<Nhgr 1>	<Weight 1>	<r1>	REFLECT	<action>	SKIP		
	<Nhgr 2>	<Weight 2>	<r2>	REFL	<action>	SKIP		
	...	...	...	...	...	...		
	<Nhgr n>	<Weight n>	<rn>	REFL	<action>	SKIP		
ZERO	0		(not necessary, see C 902)					
	•	•	•					
FIN	FINISH		(not necessary, see C 902)					

indicates 'BCD-option'

<....> means: "value of..."

CAPITAL means: literal

For formats see C 900

For card-types see C 901

Section: SUMX
Date: 15.3.1966

C BLOCK 4

card type	format	cols.	Information to be punched
REQ		1	*-asterisk, identifies SX-system card
		2-10	BLOCK 4
		11-20	normally blank; give the block-number if the histograms have not been compiled by block 6.
HEAD	S	1	'-apostrophe, assures character of heading card
		2-78	Hollerith title of compound histogram
		80	normally blank, punch L or anything to obtain 'log' plotting (see E PLID)
		1-3	must not have FIN
MULT	I	1-10	Nhgr, serial number of the constituent histogram
	U	11-20	W, weight of the constituent hgr., -ve for subtraction For zero or blank, W = 1 is assumed
	I	21-30	normally blank, to obtain reduction, punch r, the number of small channels which are to form a big channel.
	B	31-40	normally blank, to obtain reflection punch REFL or anything.
	A	41	(42-50 blank) action: (blank means +) + addition: comp. = compound + histogr. * multipl: comp. = compound * histogr. / division: comp. = compound / histogr. ⌀ inverse: comp. = histogr. / compound
	B	51-60	if this field is not blank, a flag is set into the information of the histogram which causes BLOCK 6 to by-pass the output of the histogram. The histogram remains available for use in BLOCK 4.
ZERO			this card is not normally needed, see C 902
FIN			this card is never needed, see C 902

T.C. PROGRAM LIBRARY

Section : SUMX
Date : 15.3.1966
J. ZOLL

C BLOCK 6

Block 6 for Histograms and Ideograms

This block accumulates, prints and plots histograms and ideograms from variables with associated weights and possibly root-variances, all originating on the Data Summary tape.

The multiplicity for each histogram is practically unlimited.

An ideogram is accumulated if a location of the root-variance is specified on the control-cards; but if the appropriate field is left blank, a histogram results. In this description the term histogram is taken to include ideograms, but not vice-versa.

Entry of a given event into a particular histogram can be made conditional at 3 different strata:

a) A multiplicity-element is excluded if it has an associated test MT and if this test is false.

b) An event is excluded from the whole histogram if this has a principal test MPT and if this is false.

c) Or if the event is excluded from the group of histograms of which the particular one is a member (cf. C 911)

The accumulation of an ideogram proceeds as follows:

a) The abscissa is - for the interval (XL, XU) - subdivided into NX channels, each of width ΔX. The j-th channel spans the range $(Xj, Xj + \Delta X)$ with $Xj = XL + (j - 1)\,\Delta X$, $j = 1, 2, \ldots NX$. XL is the lower edge of the ideogram, XU the upper edge. The underflow channel is defined as channel j = 0 and spans $(-\infty, XL)$; similarly the overflow channel has j = NX + 1 and spans $(XU, +\infty)$.

/...

b) Now, take a given event with quantity X_o, root-variance σ and weight W. For this event an amount W is put into the ideogram distributed over n channels, beginning with the channel for $X_o - 3\sigma$, say channel k_1, and ending with the channel for $X_o + 3\sigma$, say k_n. Each of those channels receives an amount

$$\frac{1{\cdot}0027 \cdot W}{\sigma\sqrt{2\pi}} \int_{X_k}^{X_k + \Delta X} \exp\left\{-1/2 \left(\frac{X - X_o}{\sigma}\right)^2\right\} dX$$

except if $k = k_1$, the lower integration limit is $X_o - 3\sigma$

if $k = k_n$, the upper integration limit is $X_o + 3\sigma$.

The factor in front of the integral looks after correct normalization to W.

c) Block 6 works correctly if $\sigma = 0$. If σ is negative, the absolute value is used.

d) In the normal course of operation a certain fraction of an event will be put into the spill-channels if this event with its 6σ-span reaches beyond the lower or upper edge (or both) of the ideogram, XL or XU. This would be nonsense if, say the lower limit is a physical limit, e.g. -1 for a cosine. To cope with this situation, flags are available to indicate the physical nature of the bounds. If either or both of these flags are raised, the appropriate spill-channel receives nothing and the total to be distributed is renormalized. The result of this operation e.g. for underflow would be a Gaussian, cut at XL and $X_o + 3\sigma$ of total area W. However, should the event with its 6σ-span be entirely outside the range of the ideogram, something must be wrong and the event does go into the spill-channel.

The distribution-function for an ideogram is not accumulated automatically. But this can be done with a second ideogram as follows:

Take an Xloc among the Special Summary words and place into BOUT(Xloc) the centre value of X in the resonance, see C IMTOST. An ideogram may now be compiled specifying this Xloc together with Wloc, dloc and dscale of the original ideogram.

/...

The output of block 6 is printed by subroutine PLID (cf. E PLID), except for the self-explanatory page heading.

An example of cards and output for block 6 can be found in the examplepass in section B.

Each histogram occupies at most (2N + 5M + 35) words in the dynamic store, where M is the multiplicity. In this formula 20 words are allowed for the title, if the title is shorter, less space is needed.

The histograms of block 6 can be manipulated (addition, subtraction) using block 4. The individual histograms are identified by their serial number for this purpose. In this context it is sometimes necessary to rigg the serial number of some histogram by inserting dummy histograms. To ease this annoying operation, a special card is accepted by block 6:

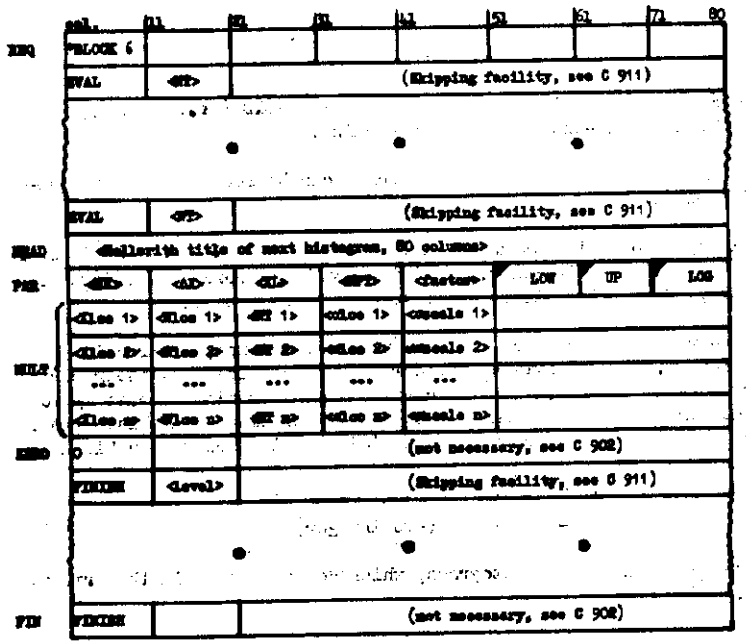
cols. 1-6 : 'DUMMY

11-20 : n (+ve integer)

This generates n dummy-histograms, which do not appear in the printed output.

/...

Block 6 - Lay-out of Control-cards



...	indicates "SKIP-option"
<...>	means : "value of ..."
CAPITAL	means : literal

For formats see C 900

For card-types see C 901

card-type	format	cols.	Information to be punched
REQ		1	*-asterisk, identifies SA-system card
		2-10	BLOCK 6
HEAD	S	1	'-apostrophe, assures character of heading card
		2-80	Hollerith title of histogram
		1-3	must not have EVA or FIN
PAR	I	1-10	NX, number of channels
	F	11-20	ΔX, channel width
	F	21-30	XL, x-value at the lower edge of the histogram
	I	31-40	NPT, number of the principal test
	U	41-50	factor to scale histogram for printing + plotting
	B	51-60	normally blank, punch LOW or anything if lower edge physical
	B	61-70	normally blank, punch UP or anything if upper edge physical
	B	71-80	normally blank, punch LOG or anything to get 'log' plotting (cf. E PLID)
MULT	L	1-10	Xloc, location of X, quantity to be histogrammed
	L	11-20	Wloc, location of W, weight attached to X; Wloc = 0 signals W = 1.
	I	21-30	NT, test associated with the multiplicity element. If this field is blank or zero, NT is taken from the previous multiplicity-card. Only if all previous cards for this histogram have blank or zero NT, does the acceptance of the entry become unconditional.
	L	31-40	σloc, location of σ, the error on X; σloc = 0 signals σ = 0, to give a histogram
	U	41-50	σscale, scale-factor for σ; σ' = σ x σscale is taken as the root-variance of X for ideogramming. If zero, 1 is substituted.
ZERO			this card is not normally needed, it is mentioned here only for people using old control-cards. See C 902 for an explanation. Also, one should be aware of the fact that Xloc = 0 causes termination of the cards for the current histogram.
FIN		1-3	FINish
		11-20	blank or ≠zero, terminates the cards for the block. This card is never needed, see C 902.

PS/4447/371/dmh

T.C. PROGRAM LIBRARY

Section : SUMX
Date : 15.3.1966
J. ZOLL

C BLOCK 7

BLOCK 7 for scatter diagrams

This block accumulates and plots 2-dimensional histograms from variables originating on the Data Summary tape. In addition the 1-dimensional histograms of the projections onto the 2 axes are accumulated, plotted and printed.

The information displayed in each cell of the diagram may be either an integer up to 30, or an integer up to 8190. The former is arbitrarily called a scatter diagram, and the latter a 2-dimensional histogram.

Contours may be superimposed on the diagrams and phase space curves drawn into the histograms of the projection. These may be standard contours or routines specifying them may be supplied.

Options are available to suppress the plotting of the 1-dimensional histograms and the display of phase space.

The contents of one cell are stored on an integral number NBITS of binary digits, which number can be supplied independently for each diagram on its control-cards. The maximum number of bits which can be accommodated in a given cell is 2**NBITS-2. Overflow is lost, but is flagged in the output.

The amount of store used up by 1 diagram depends on its dimensions and the number NBITS.

The multiplicity of each diagram is practically unlimited.

An appendix gives some elucidation about the meaning of the output from block 7.

/...

PS/4447/372/dmh

Block 7 - Lay-out of Control-cards

	col. 1	11	21	31	41	51	61	71	80
REQ	*BLOCK 7								
	EVAL	<NT>	(Skipping facility, see C 911)						
	• • •								
	EVAL	<NT>	(Skipping facility, see C 911)						
HEAD	' <Hollerith Title of next diagram, 80 columns>								
PAR 1	<NPT>	<j-kontur>	<Nbits>	FOLD	NOTHIST	NOTPHASE			
PAR 2	<NX>	<NY>	<ΔX>	<ΔY>	<XL>	<YL>			
PAR 3	(Contour control-card, must not be given if j-kontur = 0)								
MULT	<Xloc 1>	<Yloc 1>	<NT 1>						
	<Xloc 2>	<Yloc 2>	<NT 2>						
	...	...	...						
	<Xloc n>	<Yloc n>	<NT n>						
ZERO	0		(not necessary, see C902)						
	FINISH	<level>	(Skipping facility, see C 911)						
	• • •								
FIN	FINISH		(not necessary, see C 902)						

Dalitz contour : j-kontur = 1 or 2

PAR 3	<Mass X>	<Mass Y>	<Mass 3rd>	<tot.Energy>

Peyrou contour : j-kontur = 3

PAR 3	<P Max>	

Chew-Low contour : j-kontur = 4

PAR 3	<Mass 1>	<Mass 2>	<Mass 3>	<Min.Mass 4>	<tot.Energy>	TNEG	MSQ	

indicates "BCD-option"

<...> means : 'value of ...'

CAPITAL means : literal

For formats see C 900

For card-types see C 901

Section : SUMX
Date : 15.3.1966

C BLOCK 7

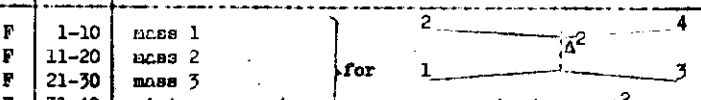
card-type	format	cols.	Information to be punched
REC		1	*-astorisk, identifying SK-system card
		2-10	BLOCK 7
HEAD	S	1	'-apostrophe, ensures character of heading card
		2-80	Hollerith title of histogram
		1-3	must not have EVA or FIN
PAR 1	T	1-10	NPT, number of principal test
	I	11-20	j-kontur, contour number, leave blank if no contour
	I	21-30	Nbits, binary digits to be used for each cell. If blank or zero, 3 is substituted. Nbits = 2, 3, 4, 5 allows a maximum of 2, 6, 14, 30 hits to be recorded in any one cell. Nbits > 6 gives a 2-dimensional histogram.
	B	31-40	normally blank, punch FOLD or anything for folding, i.e. each entry(a|b) gives (x=a|y=b) and (x=b|y=a)
	B	41-50	normally blank, punch NOTHIST or anything to suppress plotting of the 1-dimensional projection histograms
	B	51-60	normally blank, punch NOTPHASE or anything to suppress plotting and printing of phase-space.
PAR 2	I	1-10	NX, number of x-channels : ≤ MAXROW - 14
	I	11-20	NY, number of y-channels : ≤ MAXROW - 9
	F	21-30	ΔX, channel width in x
	F	31-40	ΔY, channel width in y
	F	41-50	XL, x-value at the lower edge of the abscissa
	F	51-60	YL, y-value at the lower edge of the ordinate. Some or all of these numbers may be left blank if a contour is called which will substitute calculated values, see below. If all fields are blank, zero should be punched into col. 1, because blank cards are usually ignored. MAXROW is a program parameter, MAXROW = 120 or 130 for the narrow or the wide line printers.

Section : SUMX
Date : 15.3.1966

C BLOCK 7

card-type	format	cols.	Information to be punched
PAR 3			Contour control-card, must be omitted if j-kontur on card PAR 1 is zero or blank. This card is not read by block 7 but by the contour routines.
for DALITZ j-kontur 1 or 2	F	1-10	Mass X, mass of the particle whose kinetic energy or opposing (effective mass)^2 appears on the abscissa
	F	11-20	Mass Y, ditto mass of the ordinate particle
	F	21-30	Mass 3rd, mass of the third particle
	F	31-40	total energy of the system (cf. E DALITZ)
for PEYROU j-kont=3	F	1-10	maximum momentum
for CHEW j-kont=4	F	1-10	mass 1
	F	11-20	mass 2
	F	21-30	mass 3
	F	31-40	minimum mass 4
	F	41-50	total energy
	B	51-60	normally blank, punch TNEG if $-\Delta^2$ on the abscissa
	B	61-70	normally blank, punch MSQ if M_4^2 on the ordinate (cf. E CHEW)
MULT	L	1-10	Xloc, location of x-value of the point to be plotted
	L	11-20	Yloc, location of y-value
	T	21-30	NT, test associated with the multiplicity element. If this field is blank or zero, NT is taken from the previous element. Only if all previous cards for this diagram have NT = 0, does the acceptance of the entry become unconditional.
ZERO			this card is not normally needed, see C 9o2 for explanation
FIN			this card is never needed, see also C 9o2

for



abscissa : Δ^2

ordinate : M_4

If any or all numbers on card PAR 2 are zero (or left blank, SUMX does not see the difference) the following substitutions occur by the standard contour routines :

i-kont. = 0 this will give trouble;

i-kont. = 1 Dalitz plot, kinetic energies

if NX or/and NY zero → NX = 80, NY = 50

XL, YL not touched

if ΔX or/and ΔY zero → ΔX = (EKX - XL)/NX, ΔY = (EKY - YL)/NY where EK maximum possible kinetic energy;

i-kont. = 2 Dalitz plot, $(\text{effective masses})^2$

if NX or/and NY zero → NX = 80, NY = 50

if XL zero → $XL = (M_y + M_3)^2$, $YL = (M_x + M_3)^2$

if ΔX or/and ΔY zero → $\Delta X = (ET - M_x)^2 - XL\ /NX$, $\Delta Y = (ET - M_y)^2 - YL\ /NY$;

i-kont. = 3 Peyrou plot

if NX zero → NX = 100

if NY zero → NY = 3NX/10

if XL zero → XL = -P

if YL zero → YL = 0

if ΔX zero → ΔX = (P - XL)/NX

if ΔY zero → ΔY = (P - YL)/NY;

i-kont. = 4 Chew-Low plot

if NX or/and NY zero → NX = 100, NY = 50

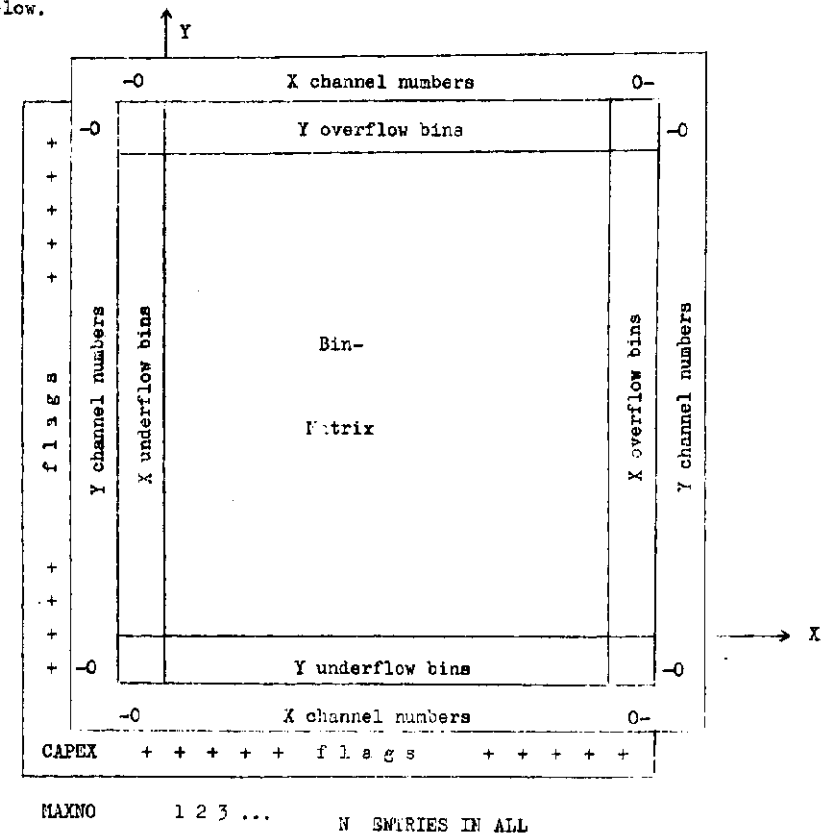
if XL and YL zero → $XL = \Delta^2_{min}$, $YL = M_{4min}$ or M^2_{4min} (option HSQ)

if ΔX or/and ΔY zero → $\Delta X = (\Delta^2_{max} - XL)/NX$, $\Delta Y = (M_{4max} - YL)/NY$

or $= (M^2_{4max} - YL)/NY$ (option HSQ)

This is a guide to the output of the results from block 7. It is meant to be read together with the specimen output included in the example pass in Section B.

1) The diagram consists of the 2-dimensional array of bins, the bin-matrix, surrounded by three frames, as indicated schematically in the figure below.



Section : SUMX
Date : 15.3.1966

C BLOCK 7

BIN MATRIX The occurrence on the DST of a pair of numbers giving rise to an entry into a certain bin will be termed a "hit". The following symbols are used to specify the bin contents :

- * The contour passes through this empty bin.
- \+ 1 hit has occurred at this bin.
- <N> with N = 2, 3,...8, 9, A, B, C... indicates multiple hits (cf. MAXNO below).

SPILL The first frame around the bin matrix consists of the 4 spill channels. Overflow, arising from values outside the limits of the diagram, is accumulated in the bins of the spill channels. For example, a bin at the top of the diagram has X inside the limits of the diagram whilst Y is too large. The bin at the top right corner has both X and Y too large.

CHAN. NOS The second frame is a print of the channel numbers, both for X and Y. -0 or 0- indicates the spill channels.

CAPEX The third frame carries "+" as flags, indicating which rows and which columns contain at least 1 cell, having recieved so many hits, that the exact number could not be recorded (cf. MAXNO below).

MAXNO This line contains +23456789ABCDEFG...., placed underneath the channel numbers in such a fashion, that it can be used as a dictionary to look up the numerical value of the letters. The last character indicates the capacity of any bin in the diagram. E.g. in diagram 1 in the example pass the last character is "E". This means: in any cell 13 hits can be recorded (D). 14 or more hits will give rise to "E" and to the CAPEX flags appearing on the outer frame.

ENTRIES IN ALL This gives the total number of entries anywhere into the diagram, including the spill channels. This number also includes the entries that have been lost because they met with a full bin.

2) <u>The 1-dimensional histograms</u> of the projections are accumulated during the operation phase independently from the diagram, so they are not affected by the CAPEX condition.

The output is printed by subroutine PLID, and is the same as if the histogram came from block 6, except:

a) The phase space may be superimposed on the plot with "*".

b) For each channel, the percentage of the phase space intercepted by this channel is printed. The sum of all these numbers is 100.0.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C BLOCK 10

<u>Block 10 for Statistics</u>

1) This block finds the minimum, the maximum, the mean and the standard deviation of a given Data Summary word belonging to all the events for which given tests are true.

2) Weighting may be done, to get the weighted mean and the weighted standard deviation.

3) A facility is included to combine the results from several Data Summary words to get a grand total, this will be termed "pooling"

4) To reduce the bother in preparing control-cards, it has been arranged that a single control-card can call for the results from several Data Summary words. The set of words specified by the same card will be termed a "batch". If pooling is required, the whole batch enters the same pool. A given pool may receive none, one or several batches.

5) The formulae used for getting the statistical quantities are these :

a) <u>For a single DST word</u>

say N events are accepted; during the pass block 10 accumulates

$$Ws = \sum_{1}^{N} i\ W_i \qquad M = \sum_{1}^{N} i\ W_i X_i \qquad MS = \sum_{1}^{N} i\ W_i X_i^2 \ ;$$

and in the output phase block 10 prints the minimum, the maximum, the weight-sum, the average $\bar{X} = M/Ws$ and the standard deviation

$$\sigma = \sqrt{MS/Ws - \bar{X}^2}.$$

b) <u>For a non-empty pool</u>

one or several batches are to enter the pool, giving a total of, say, K DST words to be combined. We have :

$$Wsp = \sum_{1}^{K} j\ Ws_j \qquad Mp = \sum_{1}^{K} j\ M_j \qquad MSp = \sum_{1}^{K} j\ MS_j$$

and the average $\bar{X} = Mp/Wsp$

the standard deviation $\sigma = \sqrt{MSp/Wsp - \bar{X}^2}$

Section : SUMX
Date : 15.3.1966

C BLOCK 10

Block 10 - Lay-out of Control-cards

	col.	11	21	31	41	51	61	71	80
REQ	*BLOCK 10								
	EVAL	<NT>	(Skipping facility, see C 911)						
	•		•		•				
	EVAL	<NT>							
HEAD	<HdLTitle>	<Xloc>	<Wloc>	<NT>	<Mult>	<Xstep>	<Wstep>	<Pool>	
	FINISH	<Level>	(Skipping facility, see C 911)						
	•		•		•				
FIN	FINISH		(not necessary, see C 902)						

<...> means : 'value of ...'

CAPITAL means : literal

For formats see C 900

For card-types see C 901

Block 10 has no control-cards of type PAR or MULT.

Section : SUMX
Date : 15.3.1966

C BLOCK 10

card-type	format	cols.	Information to be punched
REQ		1	*-asterisk, identifies SX-system card
		2-10	BLOCK 10
HEAD	D	1-10	Hollerith title of the batch
		1-3	must not have EVA or FIN
	L	11-20	Xloc 1, location of X1, the first DST-word of the batch
	L	21-30	Wloc 1, location of W1, the weight for X1, Wloc = 0 signals W = 1.
	T	31-40	NT, block number
	I	41-50	Mult = n, number of DST-words in this batch, words in Xloc 1, Xloc 2,... Xloc n are to be processed
	I	51-60	Xstep, step length to go from one DST-word to the next Xloc i = Xloc 1 + (i-1) *Xstop This formula is to be taken literally : Xstep may need to be negative if Xloc 1 is negative, to indicate integer content.
	I	61-70	Wstep, step length for the associated weights : Wloc i = Wloc 1 + (i-1) *Wstep This formula is to be taken literally : Wstep may need to be negative if Wloc 1 is negative, to indicate integer content. Wstep must be = 0 if W = 1 for the whole batch.
	I	71-80	NP, the batch is to enter into pool NP, 0 < NP ≤ 200. If NP = 0, the batch is not pooled
FIN			This card is never needed, see C 902

Section : SUMX
Date : 15.3.1966
C. LETERTRE/J. ZOLL

C BLOCK 14

Block 14 for Straight Lists

This block uses the listing-pool (cf. A 012) to produce lists of DST words from events which satisfy certain tests. On the control-cards the user has to specify which DST words are to make up a list-entry, the medium onto which the list is to be output (printer, punch or binary tape) and the formats for BCD output or the list label for binary output.

If any binary lists are to be produced with blocks 14 or 15, the logical number of the tape unit for output must be specified with ±BINARY, see C 018. Lists for punching are written onto the on-line punch (Fortran statement PUNCH), unless a tape unit has been specified with ±PUNCH, see C 018, in which case, card images are written onto tape, ready for off-line punching.

For BCD output, the user has to give 2 formats: the 'heading format' and the 'entry format'. The heading format must be pure hollerith (no I, A or F), it is used to print a page heading (printer) or to punch the first card(s) of the list (punch).

The entry format is used to print or punch each list-entry. The formats must be proper Fortran formats, except that the initial and final brackets must be omitted, as they are supplied by the program, also, in the case of printing, the carriage control character is supplied by the program. Formats specifying several lines are alright, the carriage control characters for the second and later lines must be supplied by the user. If multi-line formats are used, the number of lines caused by each of the two formats must be indicated, for the program to get the break-up into pages right (printer). To avoid performing a pass and fouling up against a faulty format when printing the results, the formats are given a trial-run on dummy data during the initial stage.

In the case of binary output, the 'entry format' is ignored, and the contents (in A6) of the first 6 cards or less specifying the 'heading format' are written into the list label.

A list on tape (binary or punch) is terminated by an end-of-file mark, the tape is terminated by 2 successive end-of-file marks.

Section : SUMX
Date : 15.3.66

C BLOCK 14

Blocks 14 and 15 - Lay-out of control-cards

	col. 1	11	21	31	41	51	61	71 80
REQ	±BLOCK 14							
	EVAL	<NT>	(Skipping facility, see C 911)					
	•	•	•					
	EVAL	<NT>						
HEAD	'PRINT	<NT>	<NCD>	<LH>	<LE>	<mode>	<Kcypos>	
PAR	< Heading format, NCD cards >							
PAR	• • •							
MULT	<Xloc 11>	<N 1>	<Nstep 1>	cumulative Entry Format				
	<Xloc 21>	<N 2>	<Nstep 2>	cumulative Entry Format				
	• • •							
	<Xloc nl>	<N n>	<Nstep n>	cumulative Entry Format				
ZERO	0	(Not necessary, see C 902)						
	FINISH	<level>	(Skipping facility, see C 911)					
	•	•	•					
FIN	FINISH		(Not necessary, see C 902)					

<•••> means : 'value of ...'

CAPITAL means : literal

For Formats see C 900

For card-types see C 901

With blocks 14 and 15, Xloc should never be negative.
(i.e. don't signal integer content, nobody is interested)

PS/4447/473/dmh

Section : SUMX
Date : 15.3.1966
C. LETERTRE/J. ZOLL

C BLOCK 14

Block 14 for Straight Lists

This block uses the listing-pool (cf. A 012) to produce lists of DST words from events which satisfy certain tests. On the control-cards the user has to specify which DST words are to make up a list-entry, the medium onto which the list is to be output (printer, punch or binary tape) and the formats for BCD output or the list label for binary output.

If any binary lists are to be produced with blocks 14 or 15, the logical number of the tape unit for output must be specified with *BINARY, see C 018. Lists for punching are written onto the on-line punch (Fortran statement PUNCH), unless a tape unit has been specified with *PUNCH, see C 018, in which case, card images are written onto tape, ready for off-line punching.

For BCD output, the user has to give 2 formats: the 'heading format' and the 'entry format'. The heading format must be pure hollerith (no I, A or F), it is used to print a page heading (printer) or to punch the first card(s) of the list (punch).

The entry format is used to print or punch each list-entry. The formats must be proper Fortran formats, except that the initial and final brackets must be omitted, as they are supplied by the program, also, in the case of printing, the carriage control character is supplied by the program. Formats specifying several lines are alright, the carriage control characters for the second and later lines must be supplied by the user. If multi-line formats are used, the number of lines caused by each of the two formats must be indicated, for the program to get the break-up into pages right (printer). To avoid performing a pass and fouling up against a faulty format when printing the results, the formats are given a trial-run on dummy data during the initial stage.

In the case of binary output, the 'entry format' is ignored, and the contents (in A5) of the first 6 cards or less specifying the 'heading format' are written into the list label.

A list on tape (binary or punch) is terminated by an end-of-file mark, the tape is terminated by 2 successive end-of-file marks.

Section : SUMX
Date : 15.3.1966

C BLOCK 14

card-type	format	cols.	Information to be punched
REQ		1	*-asterisk, identifies SX-system card
		2-10	BLOCK 14 or BLOCK 15
HEAD	A3	1-3	'PRINT - output on printer 'PUNCH - output on punch (cf. C 018) 'BINARY - output on binary tape (cf. C 018)
	T	11-20	NT, test number
	V	21-30	NCD, number of cards used for the heading format
	V	31-40	LH , number of lines caused by the heading format
	V	41-50	LE , number of lines caused by the entry format
	V	51-60	Block 15 only: mode of the key blank, 0 or 1 - floating point 2 - integer 3 - BCD
	V	61-70	Block 15 only: key position N, the Nth word of the list entry is the key for sorting.
PAR	S	1-80	NCD such cards: the heading format without the brackets and the carriage-control character for the first line (printer and punch) or the BCD text for the list-label (binary)
MULT	L	1-10	XLOC I1, location of first DST word in this group
	V	11-20	NI, number of DST words in this group
	V	21-30	NSTEP I, step-size: XLOC Ij = XLOC I1 + (j - 1) *NSTEPI
	S	41-80	entry format, cumulative over all cards MULT Note: the listing-pool imposes the restriction that a list-entry must not contain more than 40 words.
ZERO FIN			These cards are never needed (cf. C 902)

(Format V means: integer, zero or blank is replaced by 1)

Section : SUMX
Date : 15.3.1966

C BLOCK 14

Lay-out of a list on binary tape

List label	1	-31	integer, marker
	2	JLBIN	serial number of the list
	3	NENTR	number of entries
	4	N	number of words per entry
	5-84	BCD text	from the control-cards in A6

NENTR is the number of entries made into this list, fewer may appear on the tape if some have gone lost due to read errors during manipulation in the listing-pool.

List entry	1	marker	0 - OK, 1 - read error
	2	1st word	information
		. . .	
	N + 1	last word	

The marker indicates whether or not a read error occurred with this entry during manipulation in the listing-pool.

End-of-file — to mark the end of the list.

Section : SUMX
Date : 15.3.1966
C. LETERTRE/J. ZOLL

C BLOCK 15

Block 15 for Ordered Lists

This block uses the listing pool to accumulate lists of DST words, very much like block 14 - in fact it uses the same routines as block 14 for the initial and operation stage. But during the output stage each list is ordered for increasing keys before the list is handed back to the listing pool for output. Except for the sorting process, the lists are handled by the listing pool, which recognises the label BLOC15 and gives somewhat special treatment to these lists.

The control cards for block 15 are identical to those of block 14, and are described in C BLOCK 14, except that 2 more quantities have to be specified: which word of the list entry is the key for sorting, the first, second etc. - if nothing is specified, the first word is assumed to be the key. Further, the mode of the keys - whether floating, integer or BCD - has to be signalled.

The lists of block 15 cannot be of arbitrary length, because all the keys have to be in store at the same time for the sorting process. The maximum number of entries which can be handled is :

$$N = IWA/2,$$

where IWA is the space available :

$$IWA = iiMAX - 1000 - 4L - NINF - LGREC$$

- iiMAX is the number of words in the blank common vector CM
- 1000 is the length of minimal BOUT
- L is the number of lists for block 15
- NINF is the length of information for block 15 in the dynamic store (say 100 words per list)
- LGREC is an I/O buffer of as many words as a physical record needs, 512 on the 6600, 255 on the 3800 and the 7090.

A list which is longer than this maximum is simply ignored. For a list which has a chance to be longer than the maximum, one should first take a histogram (with block 6) of the keys. This will allow one to break down the long list into several lists, each one safely shorter than the maximum, which is printed by Block 15, note that this maximum will be less when NINF, the space needed by block 15 in the dynamic store is bigger.

/...

Section : SUMX
Date : 15.3.1966

C BLOCK 15

To be able to operate for block 15, the listing-pool needs at least one buffer-tape ITLIST(1) and the scratch-tape ITSCRT, even then only short lists can be handled, lists where all the information can be in store simultaneously, i.e. lists with less than N = IWA/(NWPE + 2) entries, where NWPE is the number of words per entry. For lists longer than this, an extra tape is needed for dumping the sorting directory. (see the card *LIST, C 0l7) .

Programming note:

The operation of block 15, in particular its interaction with the listing-pool, is not very obvious, and probably it will be helpful to have a rough idea of just what is happening. During the operation stage all list entries are written onto the tape ITSCRT, clearly a pass becomes uneconomic if so many lists have been asked for that the information content of ITSCRT becomes comparable with that of the DST, and by the way, if the end of the tape ITSCRT is hit, that is bad luck. At the end of the operation stage, all lists not belonging to block 15 are distributed onto the available buffer tapes and output. The space available for block 15 to work with is computed, and with it the maximum number of entries which can be handled. All lists of block 15 are inspected, to see whether they are too long or whether an 'index-tape' is needed. If yes, one of the buffer tapes is reserved as index tape, provided at least 2 buffer tapes are available (in the drum version this is simpler), if not, all lists needing the index-tape are killed, along with any list which is too long anyway.

When control reaches block 15 during the output phase, the output for all other blocks has been done - block 15 has the whole machine to itself: the buffer tapes and the complete dynamic store. First, the lists of block 15 are extracted from the tape ITSCRT and are distributed onto as many buffer tapes as are available to reduce idle read time (this is not economic if the only list in the whole pass is one long list for block 15, maybe one day something will be programmed for this special case). Next, the information for block 15 standing somewhere in the middle of the dynamic store is shifted to its lower edge to get one big working area, after which each list is processed in turn:

Block 15 reads through the appropriate buffer tape, collecting the keys in the store and writing the list onto ITSCRT with the serial number of each entry added into the entry, this is a safeguard against getting out of step in case of read errors. The subroutine SORTZV is called which returns the 'index' indicating the positions which each entry should occupy in the list. If multiple passes are needed, the index is written onto tape. For each pass, the appropriate section of the index is taken into storage, and the tape ITSCRT is read to build up a batch of entries in the store, which when complete is handed to the listing-pool for output.

T.C. PROGRAM LIBRARY

Section : SUMX
Date : 15.3.1966
J. ZOLL

C CHARM

CHARM, Tactical Routine for Data Editing

1) This subroutine enables the user to control, by means of control-cards, computations performed with the Data Summary words, whose results are to be stored into BOUT, ready for use by subsequent blocks.

2) Any number of CHARM operations may be asked for, by as many cards. The skipping facility is available, and each operation has an associated test, so that the operation will only be performed if this test is true.

3) With each CHARM operation a number jTypo has to be specified, indicating the type of operation to be performed. Nine such types are permitted.

4) During the operation stage CHARM calls any of nine subroutines, called CHARM1, CHARM2,... CHARM9, depending on jTypo. The user is free to supply CHARM subroutines which perform the calculations which he needs, without his having to program any rod tape.

5) On the control-cards the user may further specify up to 5 integer numbers Loc1, Loc2,... Loc5. They usually specify the Data Summary words for computation. These five numbers are transmitted by CHARM to the appropriate CHARM subroutine as arguments, held in the labelled common block /CHARMP/.

6) If the number jTypo on the control-card is preceded by a minus-sign, a call to the CHARM subroutine takes place during the initialization stage, thus enabling it to do some initializing, such as reading extra control-cards. Also, an extra call occurs during the output phase.

7) Examples of both the "simple" and the "complex" kind of CHARM subroutines (jTypo +ve or -ve) are given with the specimen pass in section B.

8) To economise on the number of CHARM routines, one argument L can be used for a computed GOTO in the CHARM subroutine.

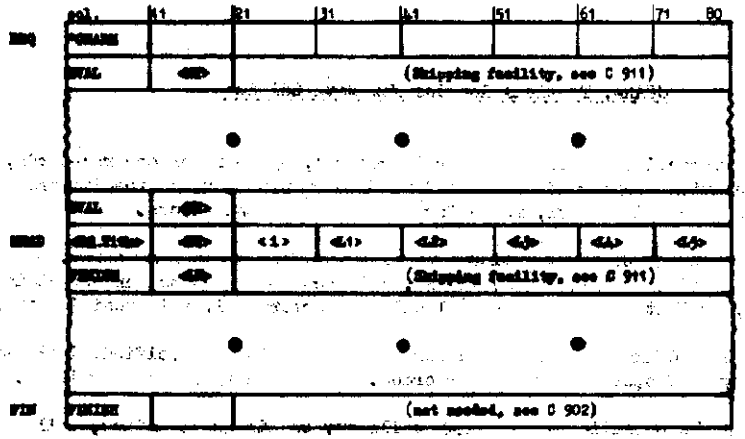
9) Suggestions on writing CHARM routines are given in C 20C.

/...

Section : SUMX
Date : 15.3.1966

C CHARM

CHARM - Layout of Control-cards



<...> means : 'value of ...'

CAPITAL means : literal

For formats see C 900

For card-types see C 901

Section : SUMX
Date : 15.3.1966

C CHARM

card-type	format	cols.	Information to be punched
REQ		1	*-asterisk, identifies SX-system card
		2-10	CHARM
HEAD		1-10	Hollerith title of Charm-call
		1-3	must not have EVA or FIN
	T	11-20	NT, test number; the call occurs only for those events for which test NT is true.
	I	21-30	i, subroutine CHARMi is to be called
	L	31-40	L1, first argument
	L	41-50	L2, etc. The 5 numbers L1 - L5 are loaded into
	L	51-60	L3 the 5 cells L1 - L5 in the labelled
	L	61-70	L4 common block /CHARMP/ before the
	L	71-80	L5 routine CHARMi is called.
			The dictionary facility can be used for the L's.
FIN			This card is never needed, see C 902

Section : SUMX
Date : 15.3.1966
J. ZOLL

C DICTIO

Constructing the Dictionary

The dictionary is constructed by means of control-cards to the context-routine DICTIO , which is called into operation by the Request-card *DICTIONARY.

In C 910 we have discussed an example of a very simple dictionary. This way of writing it is perfectly proper, but it soon becomes tedious if there are many identifiers to be defined.

The following example demonstrates a more compact way of writing the dictionary :

```
*DICTIONARY
127         JACK,JOHN,JAMES,JERRY/4,JULIAN3/10,JIM
            JOSEPH,JILL
JOHN        ROSS;RUDY+6,REX
JULIAN2/7   MAX/10
```

The first 2 cards show pretty obvious assignment of values : JACK = 127, JOHN = 128, JAMES = 129, JERRY = 130, JULIAN = 134, JIM = 164, JOSEPH = 165, JILL = 166. The indices can be used to obtain steps larger than 1; this is much like storage allocation in Common with Fortran. Continuation from the previous card occurs if the first field-of-ten is blank.

The last 2 cards show equivalencing : ROSS is another name for JOHN, and RUDY for JAMES, ROSS = 128, RUDY = 129, REX = 136. A somewhat trickier equivalence is given on the last card: MAX is the same as JULIAN2/7, and MAX2 = JULIAN3/7.

Note that if you wanted to use JERRY in this example as a 2-dimensional array, 4 is the declared range of the 'weak' index. Similarly 10 for JULIAN and 7 for RUDY. Of course, you don't have to. You should clearly realise this double purpose of the 'weak' index appearing in the dictionary.

The left-hand side of each dictionary-card is the first field-of-ten. This may contain an integer, or an already defined symbol, or nothing (for continuation). The right-hand side starts at col. 11, symbols are separated by a comma; double commas give trouble, a final comma is optional with or without continuation, blanks anywhere are ignored.

/...

Section : SUMX
Date : 15.3.1966

C DICTIO

The dictionary cards are terminated by the optional FINISH card (FIN in col. 1-3) or by the next SUMX-system card (* in col. 1).

The dictionary is clear initially, it is also cleared by a new pass starting at level zero. A pass starting at some higher level restores the state of the dictionary in this level (cf. C 912).

A second call to DICTIO does not remove the old dictionary, but adds on to it. If an identifier is redefined, the new definition overwrites the old one; a message is printed.

Everywhere the indexing '/(n+1)' is equivalent to '+n', except that both /0 and /1 are equivalent to +0. The service-routine DIGNAL which analyses every symbol on the control-cards, both to the blocks and to DICTIO, loses the difference between the two forms.

Warning : Never use an identifier which starts with EVA or FIN.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C IGNOR

IGNOR, Tactical Routine for Event Selection by Event Number

1) This tactical routine rejects or accepts events whose serial numbers have been mentioned on the control-cards.

2) The tape-label (if any) has serial number 1. This is not handed on by TAPE. Hence 1 must not then appear on the control-cards, or else IGNOR would get stuck waiting for event 1 to come along.

3) The serial-numbers must be ordered, in ascending sequence. The very simple program operates on the queue principle.

4) If continuation tapes exist, the sequences for the different tapes have to be separated by the dummy record number -1 (or any negative).

5) The serial numbers may be punched 8 to a card as positive integers, each having 10 columns to itself. Zeros or blanks are always ignored, except in col. 1-10. This terminates.

6) The REQUEST-card carries =IGNOR in col. 1-7. If col. 11-16 are blank ('BCD-option' see C 900), the specified events will be rejected. If not blank, they will be accepted whilst all others are rejected.

7) The events mentioned on the cards are listed by IGNOR via LFAIL when found (cf. A 012, C 300)

8) An example may be found in the example pass in section B.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C INTOBT

Pre-Loading Special BOUT

Special BOUT can be preset from control-cards to the context-routine INTOBT, which is called into operation with the Request-card =SPECIAL BOUT

Each control-card to INTOBT carries Xloc in the first and X in the second field-of-ten. This sets BOUT (Xloc) = X. If X is a fixed point integer, Xloc has to be preceded by the minus sign according to the SUMX-convention. If Xloc is not within the range of special BOUT, full execution of the pass is cut.

The control-cards are terminated by the next SX-system card, or FINISH, or a Zero-card.

No provision exists to load Hollerith or octal into special BOUT, but this can be done with a CHARH-routine or with a RESOH.

T.C. PROGRAM LIBRARY

Section : SUMX
Date : 15.3.1966
J. ZOLL

C SELECT

SELECT for Defining Tests

The tactical routine **SELECT** allows one to define by control-cards the circumstances under which tests are true. These cards are read during the initialization stage and are stored, suitably encoded, into the dynamic store, giving a string of card images. During the operation stage this string is scanned for each event and the truth-value for each test is stored into the test-vector as soon as it is computed. Whenever a Test 1, a 'main test', is encountered, the event is abandoned there and then if the test fails. Hence the user defines the sequence of operations by the order of his control-cards, as a result the truth-value of an earlier test can be used in the definition of a later test; the test-number has only the function of a telephone-number, but does not imply any ordering.

Pages 2 and 3 of this paper give an example of the control-cards to SELECT and the resulting output during the initial stage for easy orientation.

A test consists of an indefinite number of test-components, all of which have to be true for the test to be true. Each component, except the first, is started by a test-element having AND in cols. 1-3. Each test-component consists of an indefinite number of test-elements, which are combined by the inclusive OR.

Test 410 of the example shows the 3 arithmetic test-elements available. BETWEEN checks that the DST-word is in an interval given on the card, SMALLER and BIGGER compare the DST-word with another DST-word plus a literal. If the address of the other DST-word is zero, the comparison is with the literal only. If an integer content is signalled for either DST-word, the contents are floated before the computation; this is an exact operation. Note that these tests are open interval tests. For floating point numbers one should not rely on this, unless they are known to be exact integers, and even then it is easier to give an extra decimal.

Test 402 shows the 3 exact-equality test-elements. EQUAL may be used for fixed and floating point exact integers, but never for ordinary floating point numbers (cf. Test 64). BCD and OCTAL are machine dependent; for BCD the first n characters starting with col. 31 are taken, where n = 6 (7090) or n = 8 (3600) or n = 10 (6600) and blanks are significant. For OCTAL the last n characters are used, where n = 12 (7090) or n = 16 (3600) or n = 20 (6600), and blanks are ignored. For BCD the literal is left-adjusted, for OCTAL it is right adjusted. If integer content of the DST-word is signalled for an EQUAL test-element, the literal is fixed once for all. For all four cases (fixed EQU, floating EQU, BCD, OCT) the comparison is done by a fixed point subtraction, followed by a test for zero using the arithmetic IF in Fortran.

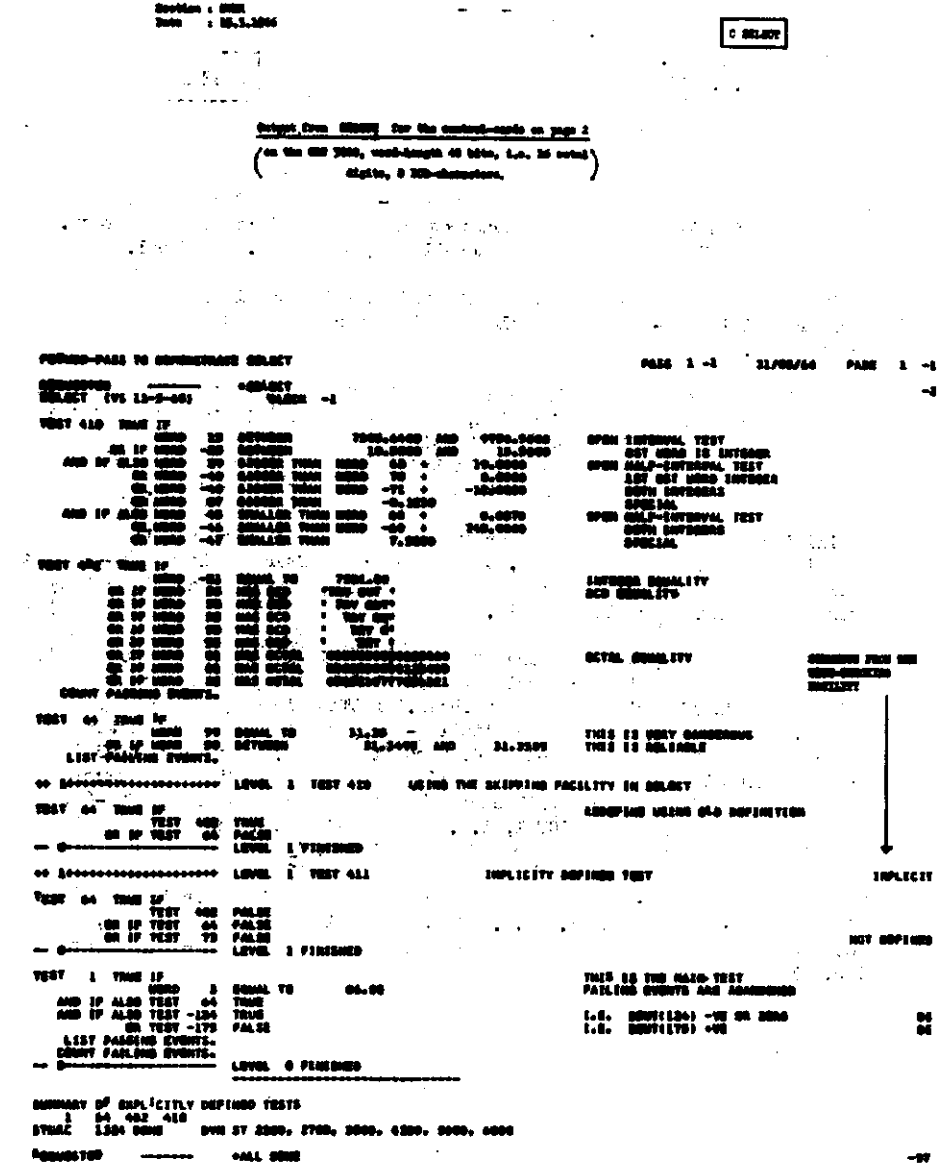
PS/4447/378/drh

SECTION SUMX
DATE 15.3.1966

C SELECT

LISTING OF CONTROL-CARDS

```
*SELECT
TEST    410
        15      BETWEEN   7395.64    9758.5      OPEN INTERVAL TEST
        -25     BET       10.5       15.5             DST WORD IS INTEGER
AND     39      BIGGER    63         19          OPEN HALF-INTERVAL TEST
        -40     BIG       70                           1ST DST WORD INTEGER
        -40     BIG       -71        -12              BOTH INTEGERS
        87      BIG                  -.123            SPECIAL
AND     45      SMALLER   68         .037        OPEN HALF-INTERVAL TEST
        -46     SMA       -69        745              BOTH INTEGERS
        -47     SMA                  7.5              SPECIAL

TEST    402
        -51     EQUAL TO  7531                   INTEGER EQUALITY
        55      BCD       TRY OUT                BCD EQUALITY
        55      BCD        TRY OUT
        55      BCD         TRY OUT
        55      BCD          TRY OUT
        55      BCD           TRY OUT
        81      OCTAL     123400                 OCTAL EQUALITY
        81      OCT             123400
        88      OCT       077765432107776543 21
COUNT

TEST    64
        59      EQU       31.35                  THIS IS VERY DANGEROUS
        59      BET       31.3495    31.3505     THIS IS RELIABLE
LIST            FAIL

EVAL    410     USING THE SKIPPING FACILITY IN SELECT
TEST    64                                       REDEFINE USING OLD DEFINITION
        402     TRUE
        64      FALSE
FINISH  1

EVAL    411             IMPLICITY DEFINED TEST
TEST    64
        402     FALSE
        64      FAL
        73      FAL
FINISH  1

TEST    1                                        THIS IS THE MAIN TEST
        3       EQU       86                     FAILING EVENTS ARE ABANDONED
AND     64      TRUE
AND     -124    TRU                              I.E.  BOUT(124) -VE OR ZERO
        -175    FAL                              I.E.  BOUT(175) +VE
LIST
COUNT           FAIL
*ALL DONE

COL.    11      21        31         41          51                        80
```



Section : SUHX
Date : 15.3.1966

C SELECT

The second test 64 shows the use of the test-elements TRUE and FALSE, whereby previously defined tests are used in the definition of a new test. For the use of TRUE and FALSE with negative test-numbers, see E col.

A given test may be re-defined, and the redefinition may involve the earlier definition - in the example this is the case with the 2nd and the 3rd test 64.

The definition of test n also causes the implicit definition of test (n+1) to be the logical opposite of test n, unless test (n+1) is itself explicitly defined (more details in E col). This and the use of the skipping facility is shown with the 2nd and the 3rd definition of test 64, which is defined one way if test 410 is true and another if it is false.

Facilities are provided to count or list the events passing or failing any given tests. The cards LIST, COUNT must come after the last test-element defining the test. This enables one to obtain the number or details of the events satisfying certain criteria.

COUNTing is entirely the business of SELECT, it simply clocks up the number of events failing or passing the specified tests; the result is printed by SELECT during the output phase. LISTing is only stocred by SELECT, but it is executed by the listing-pool, the formats for printing are defined by the experiment-dependent routine LFAIL. Each card LIST or LIST FAIL gives rise to a separate list in the pool. With the example of pages 2 and 3, the first LIST card generates the list named SELECT-64, the second a list named SELECT 1, i.e. the test-number is used to identify the list, the minus sign flags the FAIL condition.

The size of the test-vector in the superdynamic store is determined by the highest test-number defined. E.g. if test 2004 is explicitly defined, 2005 words are reserved for the test-vector to hold the truth-values for tests 1 - 2005.

Section : SUMX
Date : 20.5.1966

C SELECT

Addendum

Consider the following sketch of cards to SELECT :

```
*SELECT
TEST  3      \
TEST  5       } level 0
TEST  7      /
EVAL  3      \
TEST 12       \
TEST  5        } level 1
TEST  8       /
TEST 13      /
```

For any one event, Test 12 and the following cards are executed only if Test 3 is 'true', hence Test 12 is either explicitly defined or undefined. Test 5 clearly is always explicitly defined, in level 1 or in level 0. Test 8 is either explicitly defined in level 1 if Test 3 is true, or implicitly in level 0 otherwise - it is never undefined. This consequence of the basic philosophy had not been appreciated and caused a lot of worry recently. In the new version of SUMX (5.25), the test-checking has been extended to monitor this condition, it prints *PI* on the right-hand side of the page. One might note that, with Test 13 there is no danger, because it is defined in the same skip bracket as test 12.

Section : SUMX
Date : 15.3.1966
J. ZOLL

C TAPE

Version 1

TAPE for reading DST's

The routine TAPE with its subroutines looks after the input of the data to be processed. It has to be programmed to match the medium and the lay-out of the data.

Version 1 of TAPE reads a DST on magnetic tape with formats described in C 101 and C 102. The DST may physically consist of several tapes, one control-card has to be given for each tape. TAPE can accommodate up to 50 such control-cards, whether that many tapes can be processed is another matter and depends on the machine.

TAPE uses LFAIL and the listing-pool to compile the list RTT o containing the presumed event-numbers of those events with a read-error. Such events are not returned to SUMX for processing, but TAPE goes straight for the next event instead.

For testing TAPE can be instructed to read only a certain number of events from each tape.

If the card * TAPE does not occur in the deck of control-cards, stages 2 and 3 of SUMX are skipped: this can be used to check out the control-cards without needing the DST.

Section : SUMX
Date : 15.3.1966

C TAPE

TAPE - lay-out of control-cards

	col.	11	21	31 40
REQ	*TAPE	<N>		
MULT	<log.unit>	LABEL	<name of 1st tape>	
	<log.unit>	LABEL	<name of 2nd tape>	
		. . . .		

card-type	format	cols.	Information to be punched
REQ		1	*-asterisk, identifies SX-system card
		2-10	TAPE
		11-20	normally blank, punch N if only N events are to be read from each tape.
MULT	I	1-10	number of the logical unit for the tape
	B	11-16	normally blank, punch LABEL or anything if the first 'event' is some information identifying the tape. This information is merely printed and the 'event' is not processed.
	S	21-40	BCD-text which is printed whenever reading of the particular tape is started.
			These multiplicity cards should be given in the order in which the tapes are to be processed.

PS/4447/379/dmh

ALPHABETIC INDEX OF COMMON CELLS

BCDWD	/CHA/	BCD REP OF CHARACTERS ..BLANK,0,1,2,3,4,5,6,7,8,9,M	SX 8
DATUM	/TITLE/	BCD DATE	TITLE
DSTS	/RUN/	INFORMATION AREA FOR TAPE	SX 7
GANZ	/CHA/	FLOATING POINT INTEGERS FOR LABELLING ABSCISSA	SX 8
HDLLER	/TITLE/	BCD PAGE TITLE	TITLE
IBLANK	/CHA/	WORD FULL OF BLANKS	SX 8
II	/CHARMP/	STORAGE POINTER FOR CHARM ROUTINES	CHARM
IIDONE	/LOG/	UPPER EDGE OF DYNAMIC STORE CURRENTLY CONSUMED	SX 5
IIMAX	/DYN/	UPPER EDGE OF CM	SX 4
IITOP	/DYN/	UPPER EDGE OF DYNAMIC STORE = LOWER EDGE SUPERDYN ST	SX 4
IMBEG	/LOG/	STARTING ADDRESSES OF INFORMATION IN DYNAMIC STORE	SX 5
IMEND	/LOG/	STOPPING ADDRESSES OF INFORMATION IN DYNAMIC STORE	SX 5
IMSNIT	/LOG/	TACTICAL ROUTINES CONNECTION DICTIONARY	SX 5
INBLOC	/PAR/	BLOCK BEING INPUT	SX 6
INIT	/PAR/	ERROR FLAG FOR INPUT	SX 6
IOPBLC	/PAR/	BLOCK OPERATIONAL	SX 6
IPAGE	/TITLE/	GLOBAL PAGE NUMBER	TITLE
IPASS	/TITLE/	PHYSICAL PASS NUMBER	TITLE
IROOM	/CHIEF/	'STORE FULL' FLAG	BASIC 3
ISECT	/CHARMP/	CHARM SECTION NUMBER	CHARM
ISYSIN	/TAPES/	SYSTEM INPUT	SX 9
ISYSOT	/TAPES/	SYSTEM OUTPUT	SX 9
ISYSPU	/TAPES/	SYSTEM PUNCH	SX 9
ITA	/CHIEF/	BCD INPUT, LOGICAL UNIT NUMBER	BASIC 3
ITB	/CHIEF/	BCD OUTPU, LOGICAL UNIT NUMBER	BASIC 3
ITBIN	/LISTA/	TAPE FOR BINARY OUTPUT, LISTING POOL	LIST 1
ITDYN	/TAPES/	SCRATCH TAPE FOR DUMPS	SX 9
ITEMS	/CHIEF/	CLOCK ACTIVITY FLAG	BASIC 3
ITLIST	/LISTA/	BUFFER TAPES FOR LISTING POOL	LIST 1
ITOVLY	/TAPES/	TAPE FOR HOLDING SEGMENTS IN OVERLAY-MODE	SX 9
ITPU	/CHIEF/	CARD OUTPUT, LOGICAL UNIT NUMBER	BASIC 3
ITSCRT	/LISTA/	SCRATCH TAPE FOR LISTING POOL	LIST 1
JFREE	/LOG/	NEXT NEGATIVE BLOCK AVAILABLE + 20	SX 5
JGOOD	/RUN/	NUMBER OF GOOD EVENTS	SX 7
JPAGE	/TITLE/	LOCAL PAGE NUMBER	TITLE
JSTAGE	/CHIEF/	STAGE NUMBER (INITIAL,OPERATION,OUTPUT)	BASIC 3
KARD	/CHA/	80A1 REPRESENTATION LAST KARD - NOT TO BE DESTROYED	SX 8
KRDBUF	/CHA/	80A1 REPRESENTATION LAST KARD - FOR MANIPULATION	SX 8
KTIGHT	/CHA/	A-MAX REPRESENTATION LAST CARD (=KRDBUF(81))	SX 8
L1 ETC	/CHARMP/	CHARM - PARAMETERS	CHARM
L6	/CHARMP/	LAST ARGUMENT IN CHSET	CHARM
LEBOUT	/DYN/	LENGTH TOTAL BOUT	SX 4
LETHE	/MACHIN/	MACHINE - DEPENDENT PARAMETERS	SX10
LEVDP	/PAR/	CURRENT DUMP LEVEL (FOR *SAVE)	SX 6
LEVEL	/PAR/	SKIP LEVEL CURRENTLY OPEN	SX 6
LEXDYN	/DYN/	LOCATION SUPER-DYNAMIC STORAGE AREAS	SX 4
LGMAX	/LISTA/	MAX. NUMBER OF WORDS PER LIST-ENTRY, LISTING POOL	LIST 1
LNBOUT	/DYN/	LENGTH NORMAL BOUT	SX 4
LPASS	/TITLE/	LOGICAL PASS NUMBER	TITLE
LREQ	/RUN/	LAST REQUEST	SX 7
LTEMP	/RUN/	= 501, LENGTH OF TEMP + 1	SX 7
MAXPAG	/MACHIN/	MAX. NUMBER OF LINES PER PAGE ON L/P	SX10
MAXROW	/MACHIN/	NUMBER OF ROWS PER LINE ON L/P	SX10
MINDYN	/DYN/	PARASITIC USE OF SUPER-DYNAMIC STORE	SX 4
NAMERQ	/PAR/	LAST REQUEST IN BCD	SX 6
NCARDS	/LOG/	NUMBER OF CONTROL CARDS HELD	SX 5
NCARDT	/LOG/	NUMBER OF CONTROL CARDS TO BE RE-READ NEXT PHYSICAL PASSSX 5
NDISCE	/RUN/	NUMBER OF WORDS TO BE IGNORED	SX 7
NDUD	/RUN/	NUMBER OF DUD RECORDS	SX 7
NEXDYN	/DYN/	WORDS OCCUPIED IN SUPER-DYNAMIC STORAGE AREAS	SX 4
NOBLAN	/FLAGS/	OPTION FOR BLANK CARDS	SX11
NOMORE	/FLAGS/	.TRUE. IF +VE BLOCK REQUEST OCCURRED	SX11
NORPAG	/MACHIN/	NORMAL NUMBER OF LINES PER PAGE ON L/P	SX10
NREC	/CHARMP/	RECORD NUMBER OF CURRENT EVENT	CHARM
NREQ	/RUN/	NEW REQUEST	SX 7
NTAPES	/RUN/	NUMBER OF DST'S TO BE USED	SX 7
NTK	/RUN/	MAX. NUMBER OF WORDS TO BE READ, EACH EVENT	SX 7
NTLIST	/LISTA/	NUMBER OF BUFFER TAPES + 1, LISTING POLL	LIST 1
OUH	/CHA/	BCD REP OF CHARACTERS BLANK,0,1,2,3,4,5,6,7,8,9,+,-,.	SX 8
NWORDS	/CHARMP/	LENGTH OF CURRENT EVENT	CHARM
UDEF	/FLAGS/	OPTION FOR UNDEFINED TEST	SX11

Section: SUMX
Date : 15.3.1966
J. ZOLL

D 002

Common block /CHIEF/

ITA, ITB	SUMX input, output unit-numbers set by LOADCN, altered by DECIPH for ±INPUT, ±OUTPUT.
ITPU	SUMX punch unit-number, set by LOADCN, altered by DECIPH for ±PUNCH
JSTAGE	Stage number; 1, 2, 3 for initial, operation, output stage. The blocks set JSTAGE = 0 to signal failure of the "main test" to the main program. JSTAGE = 4 is used for a special call to TAPE. TAPE sets JSTAGLE = 3 to signal end of operation stage. BLOCK sets JSTAGE = 5 to signal the end of output stage.
IROOM	nowadays this is merely a flag. Set positive by MSIGN. Set negative by MANAGE or CARDIN if the dynamic store is full.
ITEMS	Counts how many e.g. histograms have been asked of Block 6. ITEMS is set to zero by MANAGE before each call to a block during stage 1; if ITEMS is still zero after return from the block, the block is de-activated.

Section: SUMX
Date : 15.3.1966
J. ZOLL

D 003

Common block /DYN/

LNBOUT	Length of normal BOUT
LEBOUT	Length of total BOUT, preset in LOADCN, reset in SUMX from the card ±NEW PASS.
IITOP	Upper limit of the dynamic store = lower limit of the superdynamic store. Set in MLIMIT which is called by MANAGE or MDUMP for a new pass, updated in MPLACE and MSQUEZ.
LEXDYN (5)	Starting addresses for the 5 areas of superdynamic storage. Preset by MANAGE or MDUMP for a new pass, updated in MPLACE and MSQUEZ.
IIMAX	Number of words available with CM in blank common. In some DO-loop this cell is also addressed as LEXDYN (6).
NEXDYN (5)	Number of words actually used in each of the 5 super-dynamic areas. Set as LEXDYN.
MINDYN (5)	Indicators for parasitic storage requirements in the super-dynamic region. Set by the listing - pool, used in MLIMIT to determine the lower edge: iiTOP.

Section: SUMX
Date : 15.3.1966
J. ZOLL

D 004

Common block /LCG/

IIDOMM — Address of next word available in the dynamic store. Set for new pass in MANAGE (5HINITL), updated in MANAGE (5HCHECK), also in EVALIN.

IMBEG (35) — Starting addresses of the information in the dynamic store for the blocks, 19 negative blocks, 15 positive blocks (MANAGE).

IMEND (20) — Address of first word after the information in the dynamic store for the negative blocks (MANAGE).

IMNIT (19) — Is a dictionary indicating which tactical routine is connected to which negative block (MANAGE).

JFREE — Next available negative block number + 20. (MANAGE, MDUMP).

NCARDS — Number of cards held in the super-dynamic region 1. Reset in CARDIN, updated in SAVE.

NCARDT — Number of cards to be read from super-dynamic region 1, before continuing on the input-stream. Set by MANAGE (5HCHECK) when the interruption of the logical pass occurs (NCARDT = NCARDS), used by CARDIN, updated by RESTOR.

Section: SUMX
Date : 15.3.1966
J. ZOLL

D 005

Common block /PAR/

INBLOC — Number of block whose input is in progress (SUMX).

IOPBLC — Number of block whose operation is in progress (BLOCK).

NAMERQ — BCD - text on last request-card (SUMX).

INIT — Trouble flag: zero if no trouble during the input stage, otherwise BCD name or block-number of the last block with errors.

LEVEL — EVALUATE skip level currently open. LEVEL = 0 in LOADCD updated in EVALIN.

LEVDP — Last SAVE level. Only used in MDUMP, but in common for overlay.

Section: SUMX
Date : 15.3.1966
J. ZOLL

D oo6

Common block /RUN/

LREQ	Integer representation of last request. LREQ = NREQ in CARDIN, cf. D o20 for these integers.
NREQ	Integer representation of request card being handled. Set by DECIPH, used by CARDIN, cf. D o20.
NDISCD	Number of words to be ignored of each DST record. Preset by LOADCN, reset by DECIPH from *DISCARD.
NTAPES	Number of DST's called for, set by TAPE, preset NTAPES = 0 in LOADCN.
TIME	(reserved for use with timing).
JGOOD	Number of events not having failed any "main test". (SUMX).
NDUD	Number of faulty records read, for use with reading DST's and the scratch tapes of the listing pool.
LTEMP = 501	Number of words in TEMP plus 1.
NTK	Max. number of words to be read of each DST record. Loaded from *NEW PASS in SUMX. Reset by DECIPH from *TAKE.
DST(202)	Memory region for use by TAPE, in common because of overlay.

PS/4447/384/mk

Section : SUMX
Date : 15.3.1966
J. ZOLL

D oo7

Common Block /CHA/

IBLANK	word full of blanks.
KARD (80)	80A1 representation of last card read by CARDIN, this must not be destroyed.
KRDBUF (80)	KARD is copied by CARDIN into KRDBUF, this is then used for manipulation.
KTIGHT (14)	EQUIVALENCE (KTIGHT, KRDBUF(81)) (This is written this way simply because there is physically no space on the card SX 8 to put KTIGHT). CARDIN reads a new card into KTIGHT with format 8A10 on the 6600, 10A8 on the 3600 or 13A6,A2 on the 7090. The content of KTIGHT is then exploded into KARD with UBLOW.
NUM (14)	BCD representation (A1) of : blank, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *
BCDMD (13)	BCD representation (A1) of : *, blank, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, M
GANZ (140)	contains the floating point integers for labelling the abscissa of histograms. Constructed in LOADCN.

PS/4447/385/dmh

Section : SUMX
Date : 15.3.1966
J. ZOLL

D oo8

Common Block /TAPES/

Tape, drum or disk units set in LOADCH

ISYSIN	system input unit
ISYSOT	system output unit
ISYSPU	system punch unit
ITDYN	scratch-unit for dumping the contents of the dynamic store.

Common Block /MACHIN/

Machine dependent parameters set in LOADCH

MAXROW	rows available on the line-printer. Because parameter information cannot be used in FORMAT-statements, the program must cater for the various values of MAXROW. This it does only for MAXROW = 120 and 130.
MAXPAG	maximum number of lines available on a page.
NORPAG	normal number of lines available on a page.
LETHE (1)	number of binary digits per word.
LETHE (2)	number of octal digits per word.
LETHE (5)	number of BCD-characters per word.
LETHE (6)	number of words needed to hold the BCD-text from one card (80 columns).
LETHE (10)	number of words in a physical Fortran record.

Section : SUMX
Date : 15.3.1966
J. ZOLL

D oo9

Common Block /FLAGS/

UDEF	=	.TRUE.	undefined tests are true
		.FALSE.	" " " false
NOBLAN	=	.TRUE.	blank cards are ignored
		.FALSE.	" " " accepted
NOMORE	=	.TRUE.	: a request for a block proper has occurred in the current pass, no more SAVE is to be honoured.

ZZZ is a reserve, ZZZ(7) is used in SUMX.

Common block /TITLE/

used for printing page heading

HOLLER (12)	hollerith pass title 12A6, from the card after NEW PASS
LPASS	number of the current logical pass
IPASS	number of the current physical pass (negative)
DATUM (2)	the date in BCD
IPAGE	continuous page number
JPAGE	page number for current physical pass (negative)

Section : SUMX
Date : 15.3.1966
J. ZOLL

D o10

Common Block /CHARMP/

This block mediates communication between SUMX and the CHARM-subroutines written by the user.

NREC	Serial number of the current event on the current tape
NWORDS	number of words in the current event
ii	base address of the information in the dynamic store for the current CHARM-call
L1 - L5	current CHARM parameters, taken from the corresponding control-card.

Section : SUMX
Date : 15.3.1966
J. ZOLL

D o20

Deciphering Request-cards

Any card with * in col. 1 is handed by CARDIN to the routine DECIPH for inspection. For the Request-cards DECIPH stores an integer into the common cell NREQ :

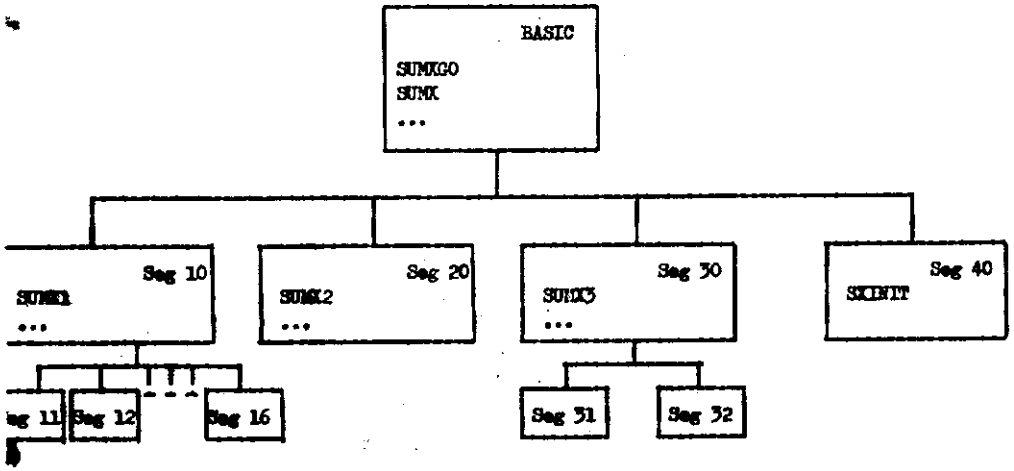
Card	NREQ
*NEW PASS	-99
*NEW RUN	-98
*ALL DONE	-97
*SAVE	-50
*TAPE	-49
*SPECIAL BOUT	-48
*DICTIONARY	-47
*IGNORE	-1
*SELECT	-2
*CHARM	-3
*MINIMAX	-4
*REPORT	-5
*TCT 6	-6
*TCT 7	-7
*BLOCK 1	1
*BLOCK 2	2
....	
*BLOCK 15	15

Section : SUMX
Date : 1.3.1966
J. ZOLL

D 030

Overlay Scheme for SUMX

The following overlay scheme is used on the CDC 3400 and 3800 at CERN :



Primarily one overlays the 3 stages of SUMX : SUMX1, SUMX2, SUMX3 which are the master-routines of the 3 stages. Into the same memory space one overlays SKINIT, this is called right at the beginning of the job and never again.

At the second level one overlays the processors, the blocks and tactical routines with each other. The longest branch in the tree is SUMX - SUMX1 - BL7A. The other segments are so arranged as to have as few of them as possible without any becoming larger than the one containing block 7 with its contour-routines.

PS/4447/483/jc

Section : SUMX
Date : 1.3.1966

D 030

The segments contain the following routines (entry points) (routines marked * appear in more than 1 segment).

Basic segment

SUMX34V, LFAIL, RESON, Charm-routines

SUMX	HIST1	PTITLE	TIME1	ULTIMO
EVALUE	ADD	READST	TIME2	USHIFT
IUCHAN	PAKIN	TEST	TIME3	CHMSW
MLOCAT	PAKOUT	TESTCL	TRACE	CHSET

IUCOMP, UBLANK, UZERO, UBLOW, UBUNCH, UCOPY, UHOLLR
XEOF, XRDIN, XILLEG, XPMT, IXLONG
XREAD, XTPBW, XTPFW, XTPOFT, XREW, XBACK
all the routines coming from the system library.

Segment 10

SUMX1	EVALIN	IUOCT	MLIMIT	SAVE
BLOCKA	FAULTY	LETTER	MPLACE	TESTCH
CARDIN	IRRTOL	LIST1	MSQUEZ	TESTDF
CATRAN	IUKOMA	MANAGE	PREQ	TESTPD
DECIP1	IULOOK	MSIGN	PSTOR	
DIANAL	IUNEXT	NDUMP	RESTOR	

Segment 11 DICTIO, INTOBT, TAPE1

Segment 12 CHARMA, SELEC1

Segment 13 IGNOR1, MINUK1

Segment 14 BL4A, BL6A

Segment 15 BL7A, KONT1, SWKT1, CHEM1, DALIT1, PEYRO1

Segment 16 BL10A, BL14A, *LISTPF

PS/4447/483/jc

Section : SUMX
Date : 1.8.1966

D 030

Segment 29

SUMX2	*LIST2D	TESTIN	BLIOB	MINMX2
BLOCKB	*LIST2V	UFETCH	BL14B	SELEC2
LIST2	TAPE2	BL6B	CHARMB	FREQ
LIST2B	TAPEX	BL7B	IGNOR2	

Segment 30

SUMX3	LISTIN	VPRINT
BLOCKC	LISTP	
EVALPR	*LISTPF	
LIST3	PLID	

Segment 31

CHARMC	BL6C	CHEW3
IGNOR3	BL7C	DALIT3
SELEC3	KONT3	PSYRO3
BL4C	SMKT3	

Segment 32 *LIST2D, *LIST2V, SORTZV, BL10C, BL14C, BL15C

Segment 40 SXINIT

Section : SUMX
Date : 15.3.1966
J. ZOLL

E 001

Routines of the Test-Facility

The test-vector contains the truth-values for the tests, one word per test, starting with Test 1 and continuing up to Test NTH + 1, where NTH is the highest test number defined. The test-vector is kept in the super-dynamic area 4, i.e. LEXDYN (4) contains the starting address and LEXDYN (4) the length of the test-vector : C.(J) with J = LEXDYN (4) + NT contains the truth-value of test NT, with NT > 0.

The truth-value is held as a floating-point number :

-31	explicitly true	+31.	explicitly false
- 6	implicitly true	+ 6.	implicitly false
	0 undefined		

The routines of SUMX, including SELECT, do not address the test-vector directly, but make use of the service-routines of the test-facility:

CALL TESTDF (NT) is called during stage 1, usually by SELECT, to register test NT as explicitly defined and test NT + 1 as implicitly defined. TESTDF assures (using MPLACE) that the test-vector is at least NT + 1 words long. The corresponding cells of the test-vector are marked for the test-checking facility (TESTCH below), test NT + 1 is marked as implicitly defined, unless it is already explicitly defined.

CALL TESTCH (NT) is called during stage 1 by all blocks whenever the use of test NT is indicated on the control-cards. TESTCH's only action is to print a comment if NT is not the number of a previously explicitly defined test:

IMPLICIT	test has previously been defined, but only implicitly
DS	if NT < 0, test-word is a DST-word (see below)
NOT DEFINED	test has not been defined.

/...

Section : SUMX
Date : 15.3.1966

E oo1

CALL TESTED is called during stage 1, usually by SELECT, to print the test-numbers of all tests defined so far.

CALL TESTCL is called by the main program during stage 2 for each event, to erase all tests before reading the next event.

CALL TESTIN (NT,± 31) is called during stage 2, usually by SELECT, to set test NT to be true or false. TESTIN transmits the truth-value into the test-word NT, and sets the test-word NT + 1 to contain ± 6. unless it contains already ± 31., i.e. unless test NT + 1 is already explicitly defined.

ITEMP = NT
CALL TEST
IF (ITEMP) true,true,false. This rather clumsy, but historical, calling sequence is used by all blocks to interrogate test NT. TEST replaces the test-number in ITEMP by the truth-value of test NT.

There are 3 possible cases :

NT = 0 : ITEMP = -11. (unconditional operation)

NT < 0 : ITEMP = BOUT (-NT), the test-word is a BST word.

NT > 0 : ITEMP = Test-word NT + δ,

where δ = 0. if xOPTION TRUE

δ = 1. if xOPTION FALSE (cf. C oo3)

Section : SUMX
Date : 15.3.1966
J. ZOLL

E oo2

Routines of the skipping facility

The skipping facility operates with 'units' and does not care what they are. With the blocks the following are 'units' :

1 test with SELECT, 1 operation with CHARM, 1 histogram with blocks 6 and 7, 1 batch with block 10, 1 list with blocks 14 and 15.

The skipping facility relies on the standard use of the dynamic store by the blocks:

> The first word of any unit, ICM(ii), in the dynamic store contains IINEXT, the address of the first word of the next unit - such that the store pointer ii is stepped from one unit to the next by the statement ii = ICM(ii).

The skipping facility gets into this continuous string of units by inserting 3 words of skip-control at the appropriate points. These are :

> word 1 : -4 if skip. (or 0 to signal 'end of block')
>
> word 2 : ii of the first unit not to be included in the skip : iiSTOP
>
> word 3 : NT, the number of the test on which the skip is conditional.

The skip controls are placed into the dynamic store by the routine EVALIN, which is normally called by CARDIN :

CALL EVALIN (ii, 2, NT) on having read a card **EVAL <NT>**

> This causes EVALIN to bump the level by 1, to store words 1 and 3 of the skip-control, to remember the origin of the new skip-level, and to step ii = ii + 3. ii is the base-address of the current unit under input.

CALL EVALIN (ii, 4, LN) on having read a card **FINISH <LN>** or on simulating a card **FINISH 0.**

> This causes EVALIN to set the ii from the argument as iiSTOP into word 2 of all unclosed skips of level LN or higher, and to reset the level to LN - 1. If LN is zero, indicating the end of control-cards to the block, ICM (ii)= 0 is set to mark the end and the dynamic store pointer is reset: iiDONE = ii + 1.

/...

Section : SUMX
Date : 15.3.1966

CALL EVALIN (ii, 5, 0) is called from the blocks when pass-interruption occurs.

This has the same effect as the call with 4 as second argument, except for an extra print. It closes all levels still open. These levels will be re-opened in the next physical pass, because CARDIN is programmed to keep exactly the right EVAL cards in the card-buffer (superdynamic region 1) for re-reading, preceding the control-cards of the last, unfinished but partially read unit.

During the operation stage the blocks call on the routine EVALUE to execute the skips :

Let ii be the base-address for the unit about to be executed, the following statements check and execute a skip :

```
201  CALL EVALUE  (ii, IND)
     GO TO (202, 201, 99), IND
202  CONTINUE
```

EVALUE inspects the content of ICM(ii) and informs the block about what happened:

ICM (ii) > 0 :	IND = 1	nothing happened, do normal processing
ICM (ii) = -4 :	IND = 2	a skip-control has been seen, the skip has or has not been executed, depending on the test, ii has been up-dated, call back in case there is a nest.
ICM (ii) = 0 :	IND = 3	end of block, quit.

During the output stage the blocks also call on EVALUE to step ii over skip-controls, for convenience. EVALUE knows that it is in stage 3 and does not skip.

Section : SUMX
Date : 10.7.1966
J. POLL

Utility Routines

*CALL UCOPY (A, B, N)	copy vector A into vector B, N words (the beginning of B must not overlap with the end of A)
*CALL UZERO (A, J1, J2)	clear A(J1) until A(J2) to zero
*CALL UBLANK (A, J1, J2)	" " " " to BCD-blank
CALL UHOLLR (A, N, nH···)	load the N-character BCD-text nH··· into vector A
*J = IUCOMP (ITEXT, IVEC, N)	table look-up (compare) : the word ITEXT is the same as the word IVEC(J). If J = 0, the word ITEXT is not an element of the set IVEC of N elements.
*CALL UBLOW (BU,BL, N)	The vector BU contains N characters of continuous BCD-text (in A6, A8, A10 - depending on the machine). This text is blown into the vector BL of N elements in A1 form with blank-fill.
*CALL UBUNCH (BL, BU, N)	The N element vector BL contains N characters in A1 form. These are to be compressed into continuous BCD text in the vector BU in A6, A8 or A10. If N is not a multiple of 6, 8 or 10, the last, incomplete word must get blank-fill.

KCH contains BCD characters in A1 form with blank-fill

J = IUKOMA (KCH, J1, J2)	KCH(J) is the first element containing '1H,' or '1H.' J = J2 + 1 is none of the elements KCH(J1)....KCH(J2) contain this.
ITEXT = IULOOK (N, KCH, J1, J2)	The first N (or less) non-blank characters contained in KCH(J1)....KCH(J2) are packed into ITEXT
J = IUTEXT (KCH, J1)	KCH(J) is the first non-blank element starting from KCH(J1).

Section : SUMX
Date : 18.7.1966

E 003

M=IUHEX (TEMP)

TEMP(1) ...TEMP(10) contain the BCD representation in A1 of a hexadecimal number. M is to contain this number.

mA = UFETCH(J)

returns BOUT(J) if J > 0
1. if J = 0
FLOAT(IBOUT(-J)) if J < 0

CALL USHIFT (JR, JL, N)

shifts the words CM(JL) ... CM(JR) by N words, to the left if N < 0, or to right if N > 0, or not at all if N = 0. [Shift to the right means : CM(JL + N) = CM(JL)]

The routines marked * are used a lot, they are being re-coded in ASCENT for speed.

PS/4447/480/dmh

Section : SUMX
Date : 15.3.1966
J. ZOLL

E 100

The skeleton of a Block

This paper gives an outline of the structure of a block, mainly to show how it uses the various SUMX-facilities. It is meant to be read together with a listing of block 6, the listing of the control-cards and the output from block 6 contained in the specimen pass, section B. See also the 'key' at the end of the listing of block 6.

Initial Stage

SUBROUTINE BL6A name of the block, stage 1.

CDE statements
the 'complete deck' may be obtained from the TC Library on special request, normally only the short version for CHARM is given. This deck is a pulling-file, everything not needed should be taken out.

100 WRITE (ITB,9001)
This prints some BCD to show that the block has been reached, this is very useful when debugging the control-cards.

ii = HSIGN (6)
This is a request to the management for space in the dynamic store, ii is set to contain the address of the first word available, i.e. the block may use CM(ii), CM(ii + 1), etc. Throughout the block, ii contains the base-address of the histogram currently being processed.

Read control-cards for 1 histogram

101 J1 = ii + JHL

CALL CARDIN (3HDHS,J1,J2,IND,ii)
This reads the heading card of the histogram. J1 is the address in CM for CARDIN to store the first word of the information read, CARDIN sets J2 to contain the address of the last word stored. The first argument conveys the following instructions:

1st Action : D, read next card, unless the buffer holds an unprocessed card.

2nd Type : H, heading card to be read.

3rd Format: S, read the whole card in maximal A format, i.e. A6, A8, A10 depending on the machine.

/...

PS/4447/392/dmh

The last argument conveys the base-address of the current histogram. If a card EVAL is read, the skip information is placed into CM(ii), CM(ii + 1), CM(ii + 2) and ii is bumped up by 3 (this is done in EV.LIN, called by C.RDIN). If a FINISH card is read, ii specifies the address on which the skip is to end. The 4th argument communicates back to the block a number of conditions :

GO TO (102, 101, 120, 170), IND

IND = 1 the heading card has been read

IND = 2 EV.L or FINISH has been read, ii is updated, call back to read the heading card

IND = 3 the dynamic store is full, quit

IND = 4 end of control-cards, quit

102 WRITE (ITB, 9002) list print the heading card
As a matter of principle, SUMX prints out every card it has read to make error-spotting easy.

109 JJ = J2 + 1
address for the information from the next MULT-card

J1 = ii + 2

CALL C.RDIN (10HNPIFFTUBBB,J1,J2,YY,YY)
read the P.Rameter-card into CM(J1)

1st-.ction : N, read next card

2nd-Type : P, parameter-card

3rd-Field 1 : I, integer (number of channels)

4th-Field 2 : F, floating (lower limit)

5th-Field 3 : F, floating (channel width)

6th-Field 4 : T, test-number (principal test)

7th-Field 5 : U, floating (scale factor)
zero replaced by unity

8th-Field 6 : B, BCD option (lower limit physical)

9th-Field 7 : B, BCD option (upper limit physical)

10th-Field 8 : B, BCD option (log plotting)

/...

Arguments 4 and 5 are not used, some compilers need a calling sequence with the right number of arguments, damn them.

CALL T.STCH (CM(ii + 5))
let the test-checking facility have a look at the test-number just read; if something unusual it will print a diagnostic warning without further interfering.

110 CALL C.RDIN (WERELETLUR,JJ,J1,YY,YY)
IF (ICM(JJ).EQ.0) GO TO 118
Read next MULTiplicity card into CM(JJ).

1st-.ction : N, next card

2nd-Type : M, MULT

3rd-Field 1 : L, Bout-adr. (Xloc)

4th-Field 2 : L, (Wloc)

5th-Field 3 : T, test-number (NT)

6th-Field 4 : L, Bout-adr, (sloc)

7th-Field 5 : U, floating (sscale)
zero replaced by unity

8th-rest : R, rest in A6 (commentary)

ICM(JJ) = 0 indicates the end of multiplicity cards. This normally happens because C.RDIN has seen the heading for the next histogram. But also a Request-card, a card EV.L or FINISH, or a Zero-card, or the store-full condition cause this.

End of cards for the current histogram

118 ICM (ii + 1) = MULT
ICM (ii + 11) = JJ + 1
store the multiplicity of the histogram and the address of the storage-area for the bins.

IINEXT = JJ + NX + 5
ICM (ii) = IINEXT
Now the total storage requirements are known, the base-address for the next histogram is worked out and stored in the first word of the current histogram. This is a convention which must be obeyed by all blocks using the skipping facility.

/...

```
120  CALL MANAGE (5HCHECK, IINEXT)
     IF (IROOM.LE.0) GO TO 169
```
This books the storage up to and excluding CM(IINEXT) from the management, which however may raise an objection if 'store full' by setting IROOM negative, in this case, quit.

```
     ITEMS = ITEMS + 1
     CALL UZERO (CM,JJ,IINEXT)
```
now that it is sure that the current histogram can be done, the counter is bumped and the storage area is zeroed or otherwise initialized.

```
     II = IINEXT
     GO TO 101
```
step the base-address for the next histogram and go to do it.

End of cards for the block

```
169  CALL EVALIN (ii,5,0)
```
call on the skipping facility to temporarily close all skip levels for pass interruption.

```
170  RETURN
```
normal exit via 170.

Operation Stage

```
     SUBROUTINE BL6B      name of the block, stage 2.

200  ii = MLOCAT (6)
```
get the storage address for block 6 from the management

```
201  CALL EVALUE (ii, IND)
     GO TO (205,201,99),IND
```
Call on the skipping facility to execute a skip, if skip-information is in store at CM(ii). On return, ii is updated and IND contains the following instruction :

IND = 1 no skip seen

IND = 2 a skip has been seen, call back because there might be a nest.

IND = 3 all histograms have been done, quit.

/...


```
205  CALL UCOPY (CM(ii + 1),MULT,11)
```
This loads the control information for the current histogram into the common block /KONTCM/, which is a working space available to any block on short term.

```
     IF (MULT.LE.0)    GO TO 244
```
A histogram of multiplicity zero has no execution. (Dummy)

```
     ITEMP = MPT
     CALL TEST
     IF (TEMP) true,true,false
```
Load the principal test-number into TEMP and call TEST. TEST replaces it by the truth-value of the test. Don't execute this histogram if the test is false.

```
207  Continue the calculations for the current histogram.

244  ii = ICN (ii)
     GO TO 201
```
Load the base-address of the next histogram into ii and go to do it.

Output Stage

```
     SUBROUTINE BL6C      name of the block, stage 3.

300  ii = MLOCAT (6)
```
get the storage address

```
     CALL EVALPR (-ii,YY)
```
Initialize the printing of the 'master tests'.

```
301  CALL EVALUE (ii,IND)
     GO TO (302,301,99),IND
```
Call on the skipping facility. During stage 3 EVALUE never skips, but it steps ii over the skip-information in the dynamic store. The significance of the IND is the same as during stage 2.

```
302  Continue

     CALL PTITLE
```
throw the page and print the pass-title

```
     WRITE (ITB,9101) list
```
print the histogram title

/...

Section : SUMX
Date : 15.3.1966
J. ZOLL

E CARDIN

Subroutine CARDIN

1. Purpose

The primary purpose of CARDIN is to read 1 card for the routine calling CARDIN. The calling routine has to specify which card it wants to be read (last card or next card), what type of card it expects (request-card, heading-card, parameter-card, multiplicity-card), the format for reading this card and where it wants the information to be stored.

To keep CARDIN within reason, only a very limited range of formatting is accepted : any card is divided up into 8 fields-of-ten columns, and each field may contain 1 piece of information. The exceptions are BCD-text which may start with some field-of-ten and occupy the rest ofthe card; also an octal number occupies 2 fields-of-ten.

Apart from this principal use, CARDIN also takes its own initiatives: the most glaring case is that of the skipping-facility which it handles with the block hardly noticing (using EVALIN); it fabricates terminator-cards if they are not in the deck and saves control-cards which in principle have been dealt with, but may be needed again in case the logical pass gets interrupted, in particular the EVAL cards of all currently open skip-levels are kept in the card-buffer (super dynamic region 1).

Whenever CARDIN reads a card with '*' in col. 1, this card is given to DECIPH for inspection. There are 3 possible replies from DECIPH :

a) the card is not a SX-system card, CARDIN proceeds normally as though the asterisk did not exist,

b) the card is a SX-system card of the 'frivolous kind', it has been executed by DECIPH, CARDIN forgets it and takes the next card instead,

c) the card is a SX-system card of the '... ious kind' (see C 301), DECIPH has set NREQ (cf. D o20). To the calling block CARDIN delivers blank cards if types PAR or MULT are wanted, and a FINISH card if type HEAD is wanted (flag BEFIRM) - until CARDIN is asked for type REQ: this generates any necessary terminator cards.

For card types MULT, CARDIN keeps a check on the 'store full' condition - if it occurs a blank card is delivered which shuts up the block.

/...

Section : SUMX
Date : 15.3.1966

E CARDIN

2. Calling sequence

CALL CARDIN (FORM, iiST, iiEND, IND, ii)

FORM — input parameter, consists of up to 10 characters and specifies the format, see 3.)

iiST — input parameter, the first word of information is to be stored in CM(iiST), the rest to follow.

iiEND — output parameter, the last word of information has been stored in CM(iiEND).

IND — output parameter, used only with heading cards (type H, K, S):

- IND = 1 heading card has been read
- = 2 EVA or FIN has been read and the information has been stored, ii is up-dated; call back to try again.
- = 3 dynamic store is full, go away to interrupt the logical pass
- = 4 end of control-cards for the current block.

ii — input-output parameter, used only with heading-cards, (type H, K, S)

input : ii = base-address of the current unit (e.g. histogram) in CM. If EVAL comes along, this information is to be stored in CM(ii).

ii = 0 signals not to use the skipping facility.

output : ii is up-dated if EVAL did come along.

3. Formats

The input parameter FORM consists of up to 10 characters, at least 3. The first letter specifies the action, the second the card-type, the remaining 8 the formats for each of the 8 fields-of-ten.

First character : Action

N	(next)	read next card
A	(again)	re-read last card
D	(depending)	read next card, unless the buffer holds an unprocessed heading card. (flag HAVE)

/...

CALL EV.LPR (ii,NPT)
print the master-tests: i.e. the test-numbers controlling all skip-levels unclosed at ii and the principal test for the current histogram

Do the output for the current histogram

360 ii = ICH(ii)

GO TO 301
Load the base-address of the next histogram into ii and go to do it.

- 6 -

Section : SUMX
Date : 15.3.1966

E 100

CALL EV.LPR (ii,NPT)
print the master-tests: i.e. the test-numbers controlling all skip-levels unclosed at ii and the principal test for the current histogram

Do the output for the current histogram

360 ii = ICH(ii)

GO TO 301
Load the base-address of the next histogram into ii and go to do it.

PS/4447/392/dmh

Second character : Type

R	bring the next request-card, ignore any other
T	bring the next pass-terminator-card, ignore any other
H	read heading-card
P	read parameter-card
M	read multiplicity-card
S	read a card for SELECT, CARDIN at CERN does not distinguish between H and S.
K	read a heading-card, but keep NCARDS, i.e. do not reset it.

Other characters : Formats

F	floating point number
I	integer number
T	test number : integer or alphabetic identifier
L	BOUT location, CARDIN does not distinguish between T and L.
U	floating point number, if 0. replace by 1.
V	integer number, if 0 replace by 1
Z	hexadecimal, read 8 digits
B	option: if field blank, set zero if non-blank, set 3BITES
A	read in maximal A format. i.e. A4 on S/360
3	read with A3, 1X
D	read with A3, A4 , 2 words result
X	skip
R	remaining card to be read in A4
S	remaining card to be read in A-max
blank	end of format, stop reading.

/...

4. Interpretation

The actual 'reading' of the card is done by the subroutine CATRAN. This interprets within the given field-of-ten each character individually and pieces them together to get whatever is wanted.

As a general rule, CATRAN ignores blank characters except in BCD-fields, hence significant zeros must always be punched - people's minds are not usually so twisted that they don't punch them anyway.

Section : SUMX
Date : 15.3.1966
O. CZYŻEWSKI

E CHEW

Chew-Low Plots with Block 7

1. The Chew-Low plot is a graph of the mass of 'particle' 4 versus the 4-momentum transfer in the following reaction :

Fig. 1.

$$\Delta^2 = (\bar{p}_1 - \bar{p}_3)^2 - (E_1 - E_3)^2 = (\bar{p}_2 - \bar{p}_4)^2 - (E_2 - E_4)^2.$$

M_i, p_i, E_i stand for the rest-mass, the 3-momentum and the total energy of 'particle' i.

2. Subroutine CHEW computes the kinematical contour of any such reaction, if it is given the parameters of this reaction. These are :

   1-3) The rest-masses of particles 1, 2 and 3.

   4) The minimum mass of the compound particle 4. This is usually the sum of the rest-masses of the particles, into which the compound under study breaks up.

   5) The total energy of the system in its centre of mass.

   Two options are available :

   6) 'THEG' indicates to CHEW that $-\Delta^2 = t$ is used on the X-axis.

   7) 'MSQ' indicates that M_4^2 is on the Y-axis.

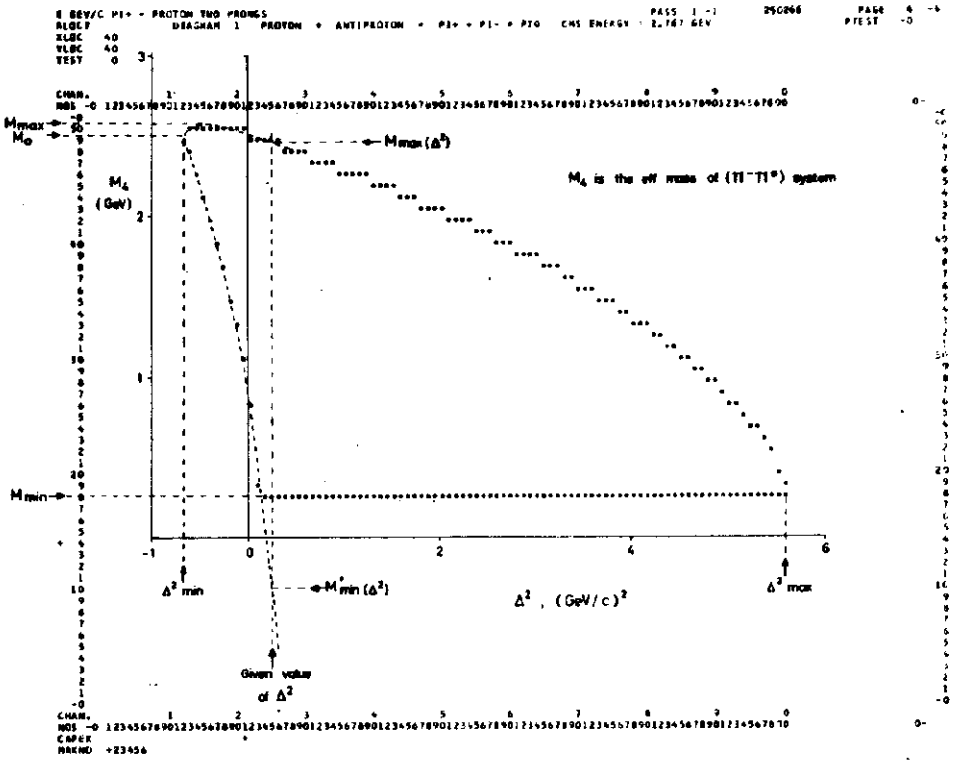
3. In the more obvious example of the reaction in fig. 1 (e.g. any inelastic peripheral collision), the momentum transfer Δ^2 is a positive quantity. There is however also the possibility of Δ^2 being negative, e.g. if the reaction involves an annihilation. CHEW has been designed to cope with this situation. Fig. 2 is a Chew-Low plot of the $\pi^-\pi^0$ system in the reaction $p + p \to \pi^+ + \pi^- + \pi^0$, produced with CHEW and Block 7.

4. On the next pages the formulae used in CHEW are given.

/...

Section : SUMX
Date : 15.3.1966

E CHEW



THE FRAME E-05 CONTAINS THE OVERSPILL - 3105 ENTRIES IN ALL

Section : SUMX
Date : 15.3.1966

E CHEW

Kinematical Limits

1. Mass of particle 4

M_{min} is an input parameter.

$M_{max} = E_T - M_3$ E_T is the total energy in CMS.

2. Four-momentum transfer Δ^2

One calculates M_o, i.e. the mass of the particle 4 corresponding to the smallest possible value of Δ^2 :

$$M_o = \left(\frac{M_1 E_T^2 - M_3 E_T^2 - M_3 M_1^2 + M_3^2 M_1 + M_3 M_2^2}{M_1} \right)^{1/2}$$

There are two possible cases :

a) $M_o < M_{min}$,

$$\Delta^2_{min} = 2E_1 E_3' - 2p_1 p_3' - M_1^2 - M_3^2$$

where $E_3' = \frac{E_T^2 + M_3^2 - M_{min}^2}{2E_T}$, $p_3' = \sqrt{E_3'^2 - M_3^2}$.

b) $M_o > M_{min}$,

$$\Delta^2_{min} = 2E_1 E_{3o} - 2p_1 p_{3o} - M_1^2 - M_3^2$$

where $E_{3o} = \frac{E_T^2 + M_3^2 - M_o^2}{2E_T}$, $p_{3o} = \sqrt{E_{3o}^2 - M_3^2}$.

The maximum value of the four-momentum transfer is in both cases :

$$\Delta^2_{max} = 2E_1 E_3' + 2p_1 p_3' - M_1^2 - M_3^2.$$

/...

Section : SUMX
Date : 15.3.1966

E CHEW

Calculation of the coordinates of the Points on the Contour

For a given value of Δ^2 we have to calculate $M_{min}(\Delta^2)$ and $M_{max}(\Delta^2)$, i.e. the smallest and largest values of the mass of the particle 4.

The smallest, kinematically permitted value of this mass for a given Δ^2 is :

$$M'_{min}(\Delta^2) = \left(E_T^2 + M_3^2 - \frac{2E_T[E_1(T + M_3) + p_1\sqrt{T^2 + 2TM_3}]}{M_1} \right)^{1/2}$$

$$T = \frac{\Delta^2 + (M_3 - M_1)^2}{2M_1}$$

In fact, if M_1 is the mass of the target particle and M_3 is the mass of the recoil, T is equal to the kinetic energy of the recoil particle in the Lab.

There are two possibilities :

a) $M'_{min}(\Delta^2) > M_{min}$, then $M_{min}(\Delta^2) = M'_{min}(\Delta^2)$.

b) $M'_{min}(\Delta^2) \leq M_{min}$, then $M_{min}(\Delta^2) = M_{min}$.

The maximum value of the mass of particle 4 is equal to :

$$M_{max}(\Delta^2) = \left(E_T^2 + M_3^2 - \frac{2E_T[E_1(T + M_3) - p_1\sqrt{T^2 + 2TM_3}]}{M_1} \right)^{1/2}.$$

T.C. PROGRAM LIBRARY

Section : SUMX
Date : 15.3.1966
J. ZOLL

E DALITZ

Contours for Dalitz-plots with Block 7

Well known facts about Dalitz plots may be found in the Berkeley Memo. 439, dated 14.3.1963 by J. Kirz. The formulae used in DALITZ are given here :

Definitions :

1) Given a final state with total energy E_t of 3 particles, masses A, B, C; momenta p_A, p_B, p_C; energies E_A, E_B, E_C; kinetic energies K_A, K_B, K_C; all in the centre of mass of the final state.

2) The 3-body state is considered the decay product of a parent 2-body state of particles, A and μ_A, say. The parameters of 'particle' μ_A are :

Energy $\quad \epsilon_A = E_T - E_A = E_B + E_C$

Momentum $\quad - p_A = p_B + p_C$

(Mass)^2 $\quad \mu_A^2 = \epsilon_A^2 - p_A^2 = (E_t - A)^2 - 2E_tK_A.$

This last equation gives the transformation between the kinetic energy of A and the (eff. mass)^2 of its opposing particle.

Note : The contour, the kinematic limit, consists of all those configurations of the 3-body state where the 3 momenta are collinear,

i.e. $\quad p_A + p_B + p_C = 0$

/...

PS/4447/395/dmh

Section : SUMX
Date : 15.3.1966

E DALITZ

Problem 1 : Given K_A, hence p_A, ϵ_A, μ_A; what is the range of K_B?

a) Eliminate E_C and p_C using 4-momentum balance :

$$\left.\begin{aligned} E_C &= \epsilon_A - E_B \\ -p_C &= p_A + p_B \end{aligned}\right\} \text{ or } \begin{aligned} E_C^2 &= \epsilon_A^2 + E_B^2 - 2\epsilon_A E_B \\ p_C^2 &= p_A^2 + p_B^2 + 2p_Ap_B \end{aligned}$$

Subtract to $\quad C^2 = \mu_A^2 + B^2 - 2\epsilon_A E_B - 2p_Ap_B$

or $\quad p_Ap_B = -\epsilon_A E_B + \gamma$ with $\gamma = \frac{1}{2}(\mu_A^2 + B^2 - C^2)$.

b) Eliminate p_B and solve the quadratic :

$$E_B = (\gamma\epsilon_A \pm p_A\sqrt{\gamma^2 - \mu_A^2B^2}) / \mu_A^2.$$

c) Reshuffle the square-root to get the properly symmetric form :

$$E_B = \frac{\epsilon_A(\mu_A^2 + B^2 - C^2) \pm p_A\sqrt{[\mu_A^2 - (B + C)^2][\mu_A^2 - (B - C)^2]}}{2(\mu_A)^2}$$

K_A itself is not in the physical region if $\mu_A < B + C$, obviously.

Problem 2 : The extreme values of K_A?

Trivially : 1) $K_A^{min} = 0$

2) $\mu_A^{min} = B + C$, from which K_A^{max} follows:

$$K_A^{max} = \left((E_T - A)^2 - (B + C)^2\right) / 2E_T.$$

PS/4447/395/dmh

Section : SUMX
Date : 15.3.1966
J. ZOLL

E NIMM

NIMM + ADD for Packing Bins

The routines NIMM and ADD pack a vector of bins into a Fortran vector of words for block 7. Fortran versions of these routines exist, but for speed it is much better to use machine-language routines. On the 6600 the Fortran versions cannot be used, because integer arithmetic cannot be done on the full word. These are the calling sequences :

N = NIMM (LF, IBIN, NBITS, INWORD, NBASE)

CM(LF)	is the first word of the Fortran vector
IBIN	is the number of the bin : 0, 1, 2, 3....
NBITS	is the number of bits occupied by 1 bin
INWORD	is the number of bins per word, it is arranged in block 7 that the sign-bit is never used.
NBASE = 2**NBITS,	the packing base

CALL ADD (J) adds the integer J into bin IBIN, which must be defined by a preceding call to NIMM

Section : SUMX
Date : 15.3.1966
J. ZOLL

E PLID

Subroutine PLID for Plotting Histograms

This subroutine plots and prints the results for the blocks 4, 6 and 7. The output is as follows :

1) The plot of the histogram. This is essentially integer-type information about the absolute value of the contents of each channel, excluding the spill-channels. The height of each column in the plot represents the content of a channel. Each X in a column stands for an amount equal to the increment in the vertical scale on the left. The top character of a column can be a number less than 10, or it may be H in which case this represents at least 10, but less than an amount which would have given rise to an X.

   For block 7 phase-space may be superimposed with "*".
   This plot may optionally be "logarithmic". However, to avoid trouble with zero, the square is used rather than the exponential to accelerate the scale.

2) The line labelled SIGN contains a "-" for each channel with negative content.

3) The contents of each channel are printed vertically in floating point. (Subroutine VPRINT).

4) The channel numbers are printed.

5) The variable value at the left edge of each channel and at the right of the last channel is printed, with a minus-sign above if negative.

6) CONTENTS ALL CHAN is the sum of the contents of all the channels appearing in the plot.

7) The contents of spill-channels and the number of times an entry has been made anywhere into the ideogram with any weighting factor (even with weight zero), is printed optionally.

8) For block 7 the percentage of phase-space intercepted by each channel may be printed.

T.C. PROGRAM LIBRARY

Section : SUMX
Date : 18.7.1966
J. ZOLL

E TAPE

Processor TAPE

The task of this routine is to deliver to SUMX the events for processing. The standard version of TAPE fetches them from the DST, but this is none of SUMX's concern. A TAPE could be written which reads the events from cards, or which generated events in the Monte Carlo manner.

A common storage area of 202 words, the vector DSTS in the common block /HUN/ is reserved for the exclusive use by TAPE and its subsidiaries.

For overlay, TAPE is split up into 2 routines TAPE 1 and TAPE 2. The following calls occur from SUMX to TAPE, with the common cell JSTAGE indicating the nature of the call:

a) JSTAGE = 1 — the Request card *TAPE has been read
CALL TAPE 1

b) JSTAGE = 4 — at the beginning of the Operation stage
CALL TAPE 2

c) JSTAGE = 2 — load the next event into BOUT; if there is no next event, TAPE replies by setting JSTAGE = 3. (end of pass)
CALL TAPE 2

Communication with SUMX : much of this is as it is for historical reasons - this is to say : that is what it has grown into, it is reasonable in the set-up for the standard TAPE - but if one were to draw up from scratch the external specifications for the processor TAPE, one would do it in a much more logical, consistent and elegant way.

INIT — the flag is handled by TAPE, if non-zero standard TAPE allows execution with 100 events [call (b)]

NTAPES — standard TAPE counts in this cell how many DST's have been asked for. SUMX (in MANAGE(IFGO)) suppresses the operation stage if NTAPES ≤ 0. [call (a)]

NTK, NDISCD — in these cells SUMX holds the contents of the cards *TAKE and *NDISCD.

PS/4447/479/dmh

Note on the Conversion of CERN SUMX to S/360

Pages B013, C910, C DICTIO:

On the use of the dictionary facility which is referred to on these pages: The symbolic names for BOUT locations and text numbers now consist of up to four alphanumeric characters. Longer identifiers are truncated to four characters by the program.

Example:

The symbolic names FRANZ and FRANCE are both identically equal to FRAN in this version of SUMX, due to the fact that only the first 4 characters are kept to designate the identifier.