INITIATION TO HYDRA

R.K. Böck[*]
CERN, Geneva

1. INTRODUCTION

1.1. About this paper

The HYDRA conventions and support programs, developed for use with analysis programs in high energy physics, have found a wide distribution. This note gives a short introduction to and the raisons d'être for the HYDRA system in a somewhat more casual style than the existing HYDRA system manual offers. Calling sequences of user routines and examples are included in the hope that this paper may also serve as a reference for the unsophisticated user. For any more detailed information, the reader is referred to the HYDRA system manual (which can be obtained from Mrs. K. Gieselman/TC).

1.2. About the implementation of HYDRA in FORTRAN

Programming languages at high level have made computers accessible to users with very little specialized training. They have also reduced the impact of developments in computer hardware or operating systems on existing user programs.

Scientific programming is predominantly done in FORTRAN as the oldest and most widely implemented high-level language. None of the alternatives, like ALGOL or PL/1, offers the same degree of portability between

[*] The many authors of HYDRA software and concepts are not responsible for this note ; obviously their contributions are vital beyond the point of simple acknowledgement.

different computers for programs, or is understood by an equally high number of computer users.

HYDRA conventions are therefore tailored to the common understanding of FORTRAN and the corresponding support packages are "embedded" in FORTRAN, i.e. they are FORTRAN subroutines or functions. They are themselves mostly written in FORTRAN as defined by the American National Standards Institute or ANSI. They resort to using non-FORTRAN statements only in very limited and well defined ways, and for three different reasons:

(a) To fill in very few obvious loopholes of FORTRAN by subroutines (bit-byte-character handling, transfer addresses).

(b) To make critical subroutines more efficient by hand-coding.

(c) To achieve computer and system independence by use of switches and non-FORTRAN statements recognised by the precompilation editor PATCHY (itself a FORTRAN program).

1.3. About the problem HYDRA attempts to solve

FORTRAN contains all ingredients to express basic algorisms adequately. At the elementary level, variables, mnemonics, assignment statements with expressions and branching instructions are sufficient. Larger programs can be structured by introducing the FUNCTION or SUBROUTINE notion with formal parameters ('calling sequence') constituting the interface between calling and called program.

With growing program size, subroutines become numerous, and their logical relation may become non-trivial. The accompanying data often have to be grouped and stuctural relations between data groups have to be introduced. Also, memory efficiency dictates overlapping use of storage. To attack such problems, FORTRAN makes available only the simple hierarchical subroutine and the COMMON storage blocks organisable in arrays. The burden of organising program and data structures is left entirely to the programmer.

HYDRA can not take over the structuring of program and data. It introduces notions, though, which alleviate this burden and which allow one to express and to communicate structures more easily. Program parts can then be more readily defined in terms of the interface data and the transformations applied to them between 'input' and 'output'. Hence, program parts are more easily understood, replaced, rearranged and documented.

These problems of communication between program parts are characteristic of a multi-user environment in which many programmers may contribute to the writing of a program, and in which the program's objectives evolve with time; in short, an environment which makes the 'black-box' approach for large programs near-impossible.

In order to achieve these objectives economically, supporting HYDRA 'system routines' and user written programs make use of concepts such as data blocking and structuring, dynamic memory, program modules, and trapping. The corresponding terminology and rules together with the support package interfaces constitute a learning threshold that one has to pass to obtain access to HYDRA.

2. HYDRA CONCEPTS

2.1. Blocking and structuring of Data

The organisation of data is vital to the writing of a program. FORTRAN gives the programmer some tools, such as mnemonics, arrays and multiple subscripts. Such variable assignments are static, i.e. the corresponding memory space is reserved at load time. Programmers use data arrays often in a relation defined implicitely by using them in a related way, or by introducing relating variables like pointers. The normal FORTRAN tools are enough to solve such problems locally.

If, however, data are communicated through many stages of a program, their structuring has to be worked out with more care. Also, mnemonics for variables become a burden rather than a help for long-range communication;

they cannot be remembered any more, they clash with local variables, and they do not usually show the structural qualities of data.

2.1.1. Data Banks

HYDRA therefore makes the distinction between shortlived data, for which no rules other than FORTRAN's are needed, and longlived data, for which the following terminology and rules are applied:

(a) Data elements are single precision floating point words.

(b) Data elements are grouped according to logical affinity (e.g. all data elements describing a particle track) and stored into contiguous storage words. This area is called the data-part of a bank.

(c) The bank carries a one-word BCD identifier and is referred to by a pointer called a link. Locally one uses the identifier as a mnemonic for the integer variable containing that link.

(d) Logical relations between banks, or rather between the data entities they represent, are expressed by links which themselves are also part of a bank. The simplest link is the pointer allowing one to go from a bank to 'the next bank of the same type'.

(e) Single-bit information concerning the group of data elements and associated links in a bank is stored in a status word, which is also part of the bank. We have by convention allowed 15 bits of this word to be freely assigned a meaning by the user.

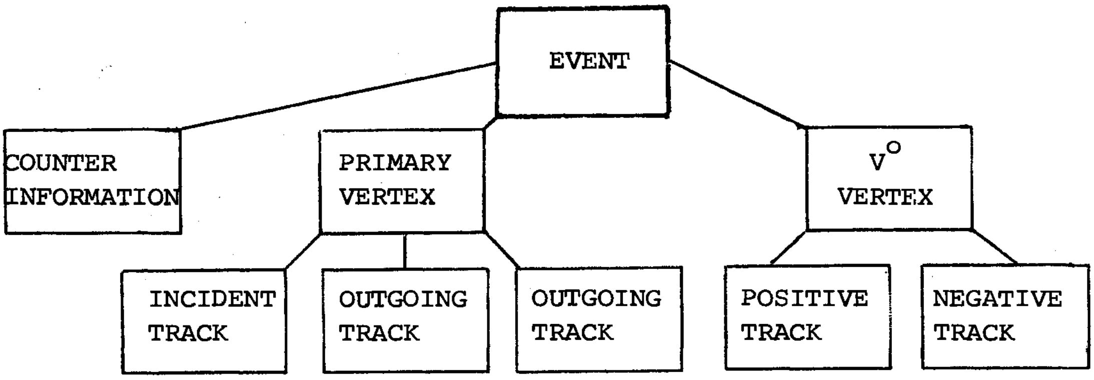
2.1.2. Data Structures

Links allow the construction of 'graphs' of any kind, in which the nodes are data banks and the (directed) edges are links. Such pictorial graph representation of data can be used to visualise the logical relation of data. Data banks interrelated in a well defined way are called a data structure.

The simplest hierarchical data structure is called a tree structure. It is so common and useful that some HYDRA system routines support it specifically. In a tree structure, banks exist at distinct levels of hierarchy, and each bank may be one of a set of identical banks, all with the same identifier and all depending in the same way from the hierarchically next higher bank. Each bank may be the starting node of one or several tree structures at lower level.

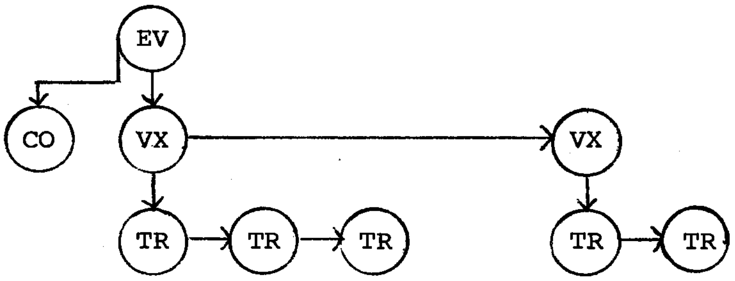
Example: High Energy Physics events are made up of vertices, every vertex has tracks associated to it. Also, to each event is associated a bank of information concerning electronic counter information to be used later. Assume the event to be a two-prong with an associated V^o.

The pictorial graph for this event information is then

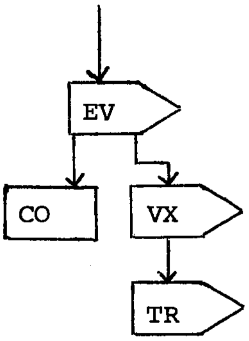


This information is structured into three levels, and vertices and tracks constitute sets of banks. HYDRA, for storage simplicity, replaces the 'fan-out' links from 'event' to the 'set of vertices' and from 'vertex' to the 'set of tracks' by one (hierarchically) downward link to one member of the set, and by one link per member joining this member to the next member of the set. The chain of links ends at the last member with a zero link.

The HYDRA representation of the above information is then



or in a frequently used shorthand allowing for several events



Associated to each bank are bank descriptions. Highly simplified they might look for our example like this:

EV:	1	data word	:	event number
	3	links	:	EV, VX, CO
	no status bits			
VX:	6	data words	:	X, Y, Z and errors
	2	links	:	VX, TR
	1	status bit	:	primary vertex
TR:	9	data words	:	1/p, λ, ϕ, and errors and correlations
	1	link	:	TR
	3	status bits	:	incident, +, -
CO:	data words only			
	no links			

2.1.3. Linkage Conventions

All links connecting data banks into a tree structure are called structural links in HYDRA. The example shows that they can be divided into hórizontal links, i.e. links from one member of a bank set to the next, and vertical links, i.e. links from a bank to a hierarchically lower level bank, which may again be a member of a set. In any given bank there may only be one horizontal link, but any number of vertical links.

Data structures are not always of a tree type, and banks in a tree structure may, for programming convenience, make reference to banks in some other data structure. Such non-structural links are called reference links in HYDRA. Any number of reference links is permitted in a bank. The storage convention for links in banks is to store structural links before reference links, and to have the first structural link always reserved for the horizontal link. 'First' and 'before' are defined with respect to the addressing inside the bank, described in 2.2.1. below.

2.2. Dynamic Memory

HYDRA relies for storage of all data, shortlived or longlived, on a dynamic store. The dynamic store is a FORTRAN array, arbitrarily called Q or IQ (in FORTRAN word-by-word equivalence) and usually assigned to blank common.

2.2.1. Bank Storage

For longlived data, bank space is obtained by calling the HYDRA subroutine MQLIFT, which returns an address relative to Q. At this address room for data elements, links and status word has been provided. The address points to the status word of the bank. Data elements of the bank are referenced by adding a bias to this address, links in the bank are referenced by subtracting a bias. Thus, for instance, the 'first link' and the 'fifth data word' of the bank VX are referred to by IQ(LVX-1) and Q(LVX+5). MQLIFT will fill up the array Q dynamically starting at the high-address end.

2.2.2. Working Space

The same FORTRAN array Q is used, in its low-address part, to contain shortlived data, and this part is called working space. As its origin is not fixed to keep flexibility for the HYDRA system, it must never be referred to by the name Q, but by local mnemonics. They are introduced by appending them to a COMMON statement invoked by a PATCHY macro (CDE-card) which defines Q, as well as some more system-user interface variables (see para. 4).

There are two conventions to be observed in organising the working space:

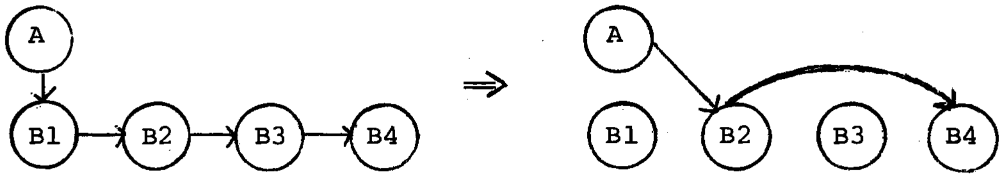
(a) links, which are relocatable integer variables, must be assigned space at lower addresses than other data variables.

(b) the extent of link and data working space must be announced to the system by CALL MQWORK, whenever their limits change.

2.2.3. Dropping of banks and Garbage Collection

Working space no longer needed is released by announcing new limits to the system. Bank space is released by dropping banks: If the information contained in a bank is no longer needed, the bank is marked as dropped by setting its drop bit, reserved for this purpose in the status word of the bank. One uses CALL QDROP which can also drop entire tree structures, or CALL SBIT1 (Q(L), IQDROP). Dropped banks stay in memory until bank space and working space clash during a new request. When this happens, the HYDRA system executes a garbage collection squeezing out dropped banks and shifting live ones. This moving of banks in store will require updating of links, done automatically by the garbage collector. Those links pointing to live banks will be relocated, i.e. a bias is added to each link so that the old address with respect to Q is converted into the new one. Links pointing to dead banks will be set to zero, unless the link is structural and the dead bank has a horizontal link connection to a live bank. In this case the link will be bridged, i.e. be replaced by the link to that live bank.

Example for bridging: (B1 and B3 are dropped)



Garbage collection is not normally under user control! The user has to be concerned only with the total extent of Q so that too frequent garbage collection and memory overflow are avoided. The obedience to storage rules for links in banks and working space ensures smooth functioning of the garbage collection mechanism.

2.3. Program Sturcturing and Interfacing

Program units written according to the modularity aims of HYDRA must be defined in such a manner that they may constitute the building blocks of larger programs. On the elementary programming level, FORTRAN functions and subroutines are defined by their calling sequence and by the operations they perform to transform 'input data' to 'output data'.

HYDRA attempts to facilitate the definition of higher level program units called processors by prescribing an interface definition in terms of data structures only. The definition of 'longlived' for data that are grouped and structured in banks thereby takes a meaning: Longlived data serve for inter-processor communication. Conversely working space serves only for intra-processor use.

2.4. Logical communication between processors

2.4.1. Call Banks

Processors will frequently be designed to operate on parts of existing structures, so that their 'calling sequences' must include indicative information about more complete data structures in store. This part of the processor interface is also stored in banks, and these banks are called call banks. Call banks are created and used in a way very similar to ordinary data banks. A subroutine JQBOOK is used for creating them, which routine automatically links call banks into the call bank data structure(*). Standard link names are used in processors to refer to call banks. There must be two such links: LQUP referring to the call bank generated at higher lever to call the current processor, and possibly LQDW which points to the call bank generated in the current processor for calls to lower levels.

2.4.2. Processor Calling

HYDRA processors are groups of ordinary FORTRAN subroutines: One is the primary routine which may invoke others, if there are more than one. Calling and returning conventions inside a processor are those of ordinary FORTRAN. For communication with the primary routine of the processor, a HYDRA calling and returning procedure is used passing through the system routines JQJUMP and JQBACK. This is necessary for correct handling of the call bank structure. Processors can thus easily be intercepted, for instance for overlaying. Also, the content of the call banks includes system status information and the return address, so that processors are 'reentrant', i.e. they can be interrupted whenever calling other processors, and during this interruption they may be called in a different context without destroying the information necessary for resumption of the original task.

(*) The call bank structure is a 'stack' i.e. a linear chain of banks. Its entrance link, LQUP, points to the last created call bank corresponding to the lowest level in the processor calling hierarchy. From there links point in chronological order to previously existing call banks. For every processor call (return), the stack is augmented (reduced) by a corresponding call bank.

2.4.3. Abnormal Communication : Trap Returns

Normal calling and returning conventions assume normal behavior of programs. In other words, they assume the invoked procedures can transform by the programmed method the input data structure into the output data structure according to the interface definition.

In many cases, programs at any level can find themselves in an abnormal situation which makes continuation impossible, or changes at least the procedures to be adopted.

Classically, signalling of such conditions is performed by including in the data interface of subroutines markers or flags which may determine the subsequent logic.

HYDRA attempts to single out two problems in this area, thus separating more clearly data and logic interfacing of processors.

(a) When an abnormal condition arises, the program flow can be directed back to a processor at a higher level, which has previously signalled its capability to resume operation after detection of the particular trouble. This process is called trapping. A trap is set by a call to a HYDRA system routine RQTRAP; signalling an abnormal condition is a call to RQTELL. Trapping may be imposed by the calling sequence of RQTELL, if the condition detecting program , which calls RQTELL, has no programmed recovery.

(b) Traps may also be requested for certain conditions during the initialisation phase, even though a recovery is foreseen in the program. To make this possible, all interesting conditions are given unique condition ID-numbers and signalled to the system by calling RQTELL, regardless whether recovery is programmed or not. All conditions occurring can thus be counted and summarised at characteristic points in the program, usually at the end of the run. For test and production runs, this provides a powerful diagnostic and checking tool. Causing a certain amount of overhead, though, condition ID's should not be used for general accounting purposes.

3. LIMITATIONS OF HYDRA

3.1. Limits of the modularity concept

HYDRA concepts provide ways for an orderly construction of large data and program structures. The pool of application programs using HYDRA which CERN is accumulating now, is, hopefully, more easily described, reconfigured and extended than programs using many different local conventions.

Processors taken out of a larger program are not, however, very general toy bricks that can be used in any place desired. HYDRA provides a means of avoiding to have pieces of a jigsaw puzzle which fit only in one unique place. For large analysis programs, though, standard and proven processor combinations are commonly used with a high-level steering program operating very much like any conventional black-box FORTRAN program. The HYDRA conventions allow to extract the desired parts of such programs and rearrange them or to reuse them in a new context, after careful checking or adaptation of data interfaces.

3.2. Programming discipline

HYDRA is an 'embedded language'; there are no compile-time checks for misuse of concepts nor for plain programming mistakes, as long as these do not contradict FORTRAN. HYDRA support routines are reasonably well protected against basic inconsistencies in the data they are given, but diagnostics occur at execution time and necessarily are not as revealing as a compiler's check list.

The common use of explicit pointers into an anonymous dynamic store necessitates programming discipline which, if violated, may lead to clobbering of the bank storage. A number of debugging aids exist, in particular the routine DQSNAP, which can, under a control of option-letters, map and dump all parts of the dynamic store. Some non-vital conventions on link names facilitate writing and reading of programs. Links to banks in the dynamic store are given variable names which are the identifier of the bank preceded by the letter L, e.g. LVX for the link to bank VX. Links which point to a cell of the dynamic store which itself contains a link (a frequent ocurrence), are given the name of the bank which that link points to, preceded by the letter K; thus, LVX = IQ (KVX). Links also known to the HYDRA system, such as LQUP which points to a processor call bank, all carry variable names beginning by LQ or KQ.

3.3. Some examples of common mistakes

- A bank is created with certain limits on the number of links and data words. The subsequent use of a bias exceeding those limits may destroy vital information in other parts of Q.

- A status word is entirely replaced by the statment IQ (L) = (expression). This is illegal, because status words contain system information. Only use CALL SBIT or CALL SBYT to enter information into status words.

- It is false to assume that dropping a bank by CALL SBIT1 (Q(L), IQDROP) removes it from the data structure. The removal and bridging happens only when garbage collection occurs. CALL QDROP, instead, does remove a bank logically from the indicated structure, but again any number of reference links to this bank continue to appear 'live'.

- Unwanted data structures or parts thereof are often not dropped upon return to higher level programs; this is a particularly frequent error for 'returns' by trapping. Trap handling therefore should be programmed only at high level and must take care of the data corresponding to this level.

- The effect of improper structuring of banks is frequently noticed by the necessity to loop in a complicated way in processors, and particularly when data are to be dropped. Banks should be part of one data structure only, and this structure should be of the tree type whenever possible.

- Most HYDRA system packages must be initialised at the beginning of the run, and in a specific order. The correct order of CALL statements is that of the description (para 5) below.

- On any computer installation exist various abort possibilities which should be properly trapped. Any abnormal end of program should still execute a CALL QFATAL producing a dump of the dynamic memory. This is the only way to make sure a diagnostic can be construed for an otherwise meaningless illegal situation.

4. THE HYDRA SUPPORT MATERIAL

CERN distributes HYDRA system support routines in the form of PAM-files, i.e. card images which the precompiler program PATCHY transforms into compilable material. On any given installation, HYDRA-routines will usually be made available as load libraries or modules, the details of which depend heavily on the computer system and the person(s) responsible for HYDRA.

In addition, a set of very few COMMON-, DIMENSION- and EQUIVALENCE-statements are necessary, which can be inserted, where PATCHY is available, by a PATCHY statement + CDE, Z = Q. These statements give processors access to several common system links and to the dynamic store variables Q and IQ. They must, of course, correspond to the CDE-statements used in compiling the system support routines.

For the technically interested reader, the expansion of the PATCHY macro + CDE, Z = Q is actually the following:

```
COMMON / QBITS / IQDROP, IQMARK, IQGO, IQGONE, IQSYS, IQCRIT
DIMENSION IQUEST(30), Q(99), IQ(99)
EQUIVALENCE (QUEST, IQUEST), (LQUSER, Q, IQ)
COMMON //  QUEST(30), LQUSER(7), LQMAIN, LQSYS(22), LQPRIV(7),
LQ1, LQ2, LQ3, LQ4, LQ5, LQ6, LQ7, LQSV, LQAN, LQDW, LQUP
```

It shows how variable names for some bit positions in the status word (e.g. IQDROP), for system links (e.g. LQUP), for user available links (LQUSER, LQPRIV) and for the dynamic store (array Q, IQ) are introduced.

The HYDRA support routines are largely independent of each other, but it is inevitable that some routines necessitate a preceding calling of others. Routines come in packages; only one of these packages (M, for memory management) is necessary, all others should be called upon only if their functions are desired. Some routines (RQTELL, QTITLE) are called both frequently and from other system routines, so that short (dummy) versions of these are supplied in case the corresponding package is not otherwise desired. Some of the packages (M-, T-, J-, R-package) must be initialised at the beginning of the program by calling a routine xQINIT (x for M, T, etc.). Routines that need no initialisation and are not part of any specific package, are called HYDRA utility routines.

5. THE MORE IMPORTANT CALLING SEQUENCES OF HYDRA

5.1. Memory Management (M-package)

(For details see paper B MQ of the HYDRA systems manual).

Example sequence: (using bank conventions of the example in 2.1. above).

```
+ CDE, Z = Q
      +, LINKS(24), LB, K(3), LVX, LEV
      +, FIRSTD, A(9,9), B, C(13)
      +, LASTD
      +, SPACE(4000), EOMEM
         DIMEMSION MVX(4)
         DATA MVX/2HVX, 4, 3, 6/
```

CALL MQINIT (EOMEM)

only once at beginning of run, to initialise the memory manager for entire dynamic store length; here, total length is 4128 words plus an unknown number of system links

CALL MQWORK (FIRSTD, LASTD)

whenever working space limits are changed. Here: 30 links and 97 data words are made available

CALL MQLIFT (LVX, LEV, - 2, MVX)

a bank with the identifier VX, with 4 links, 3 of which are structural, and with 6 data elements (words) is created. Its address (wrt Q) is returned in LVX, i.e. address limits are IQ(LVX - 4) to Q(LVX + 6). The meaning of LEV,-2 is the following: if LEV ≠ O, the bank will be appended to an already existing tree structure by the statements

IQ(LVX - 1) = IQ(LEV - 2)

IQ(LEV - 2) = LVX

If LEV = O, such linking actions as necessary have to be performed by the calling program.

5.2. Title reading (T-package)

(for details and some further options see paper B TQ of the HYDRA systems manual)

Titles are groups of data, usually punched on cards, which permit to steer certain parts of a program. These data groups are called title items and preceded each by a card carrying

in column	1	an asterix
"	2 - 6	normally blank
"	7 - 10	a Hollerith identifier
"	11 - 20	the number of data words on the following card(s)
"	21 - 80	The FORTRAN format under which the following card(s) are to be read (brackets included).

After the last title item a card with *FINISH in col. 1 - 7 will signal the end of title items.

The initialisation of titles in memory is done by

```
CALL TQINIT (LUN)
```

with LUN logical unit number containing title items. This has to be preceded by memory initialisation (CALL MQINIT).

Reference to title items can be made by

```
CALL QTITLE (LTIT, ID, IFLAG)
```

which returns in LTIT a link to a bank containing as data elements the word-by-word information of the title item with the Hollerith identifier ID, i.e. Q (LTIT + 1) will be the first data word, etc.

IFLAG controls the action of QTITLE in case the title item can not be located: if IFLAG = 0, LTIT will be returned zero and hence the situation must be handled by the calling program. If IFLAG = 1, QTITLE will trap-exit to the R-package (see below) with condition ID 61, and the calling program will not regain control.

Example:

Program

```
     ....
     CALL MQINIT  (EOMEM)
     CALL TQINIT  (1)
     ....
     CALL QTITLE  (LBEAM, 4HBEAM, 0)
     IF  (LBEAM.EQ.0)  GOTO 50
     PB     = Q(LBEAM + 1)
     ELB    = Q(LBEAM + 2)
     PHIB   = Q(LBEAM + 3)
  50 ....
```

On logical unit 1

```
*    BEAM      3 (3F 5.1)
15.3 .0006 3.14
```

5.3. Jumping into and out of processors (J-package)

(For details see paper B JQB of the HYDRA system manual)

For calling (the primary routines of) processors, and for the corresponding return instruction, HYDRA system routines are called which assure the building of a call bank data structure in memory.

Example sequence:

```
+ CDE, Z = Q
     EXTERNAL PROC
     DIMENSION MPROC (4)
     DATA MPROC / 4HPROC, 2, O, 1/
     ....
```

```
CALL JQINIT
```

once at beginning of run, preceded by CALL MQINIT (....)

```
CALL JQBOOK (MPROC)
```

this corresponds vaguely to a CALL MQLIFT; the created call bank will contain 2 reference links (the third word of array MPROC is meaningless) and one data word for communication between calling and called processor. The link to the created call bank will be found in LQDW.

```
IQ (LQDW - 1) = ....
IQ (LQDW - 2) = ....
 Q (LQDW + 1) = ....
```

fills in the call parameters for the processor

```
CALL JQJUMP (PROC)
```

executes the transfer of control into PROC. Note that upon this instruction all information to be used after return must be in bank storage, i.e. ordinary data structures or the call bank. The called processor (PROC) will normally destroy the content of working space. A data saving possibility for larger amounts of data is described in the HYDRA system manual. Our example is completed by giving the usage of call parameters from the call bank in the (called) processor PROC, and the return sequence:

```
     SUBROUTINE  PROC
+ CDE, Z = Q
     +, LNEED, ....
     +, ARG, ....
     +, LAST
CALL MQWORK (ARG, LAST)
     readjusts work space limit
LNEED  = IQ (LQUP - 1)
ARG    =  Q (LQUP + 1)
     LQUP contains now the link to the call bank, LQDW is free for
     calls further down
....
CALL JQBACK
     will return, restoring work space limits and LQDW, LQUP to the
     state prior to the execution of CALL JQJUMP (....).
```

5.4. Reporting and Trapping (The R-package)
(For details and further options see paper B RQ of the HYDRA system manual).

The concept of trapping as an abnormal termination of procedures has been introduced in 2.4.3. above. Let us here look at this mechanism from a different angle.

(a) Assume there are situations in a program whose occurence deserves to be reported to a central reporting routine. We label each of these situations by a unique condition ID, a 3-digit positive integer, and report it by calling the HYDRA system routine RQTELL. At the end of a processing step, or at the end of the program run, we will then be able to print with each condition ID the associated number of occurrences.

(b) Now some of these situations may be too severe for the routine in which they occur to handle them. So we associate to the CALL RQTELL a flag with the meaning: Do not only report the fact that this condition has occurred, but take care to restart the program at the appropriate (higher level) point. In this case, RQTELL will trap, i.e. return not to the point of calling, but to a point previously indicated to it by a CALL RQTRAP.

(c) The setting of traps is under user control. Title items group condition ID-s into trap classes and any condition ID mentioned in a trap class will be trapped regardless whether the RQTELL calling sequence requests this or not. In turn any condition ID given to RQTELL with a trap request will also be trapped: if it is not initialised by appearing as part of a trap class in a title item, and by calling RQTRAP for this class, RQTELL itself will generate a new condition (ID numbers 90, 91, 92) which are always defined by default as belonging to trap class 1, and can hence be intercepted. If they are not, the run will properly abort.

Examples: I want to intercept condition IDs 612, 622 in one place called AA as class 3, every enforced trap by default class 1 in another place BB. I use the T-package to read the title item (see 5.2 above).

```
*      RQTR      5       (5 F 10.0)
  3.     O.      O.     612.     622.
```

Word 1 is the class number, words 2 and 3 are zero to avoid complicating the example. Several title items with the name RQTR can be given. During initialisation, one per run, I execute

```
CALL RQINIT
```

preceded by MQINIT, TQINIT and, if applicable, JQINIT.

In the routine intercepting class 3, I execute

```
....
CALL RQTRAP (3)
IF (IQUEST (1). NE.0) GO TO AA
```

both for initialising the trap class (IQUEST (1) = O) and for intercepting condition IDs 612, 622 (IQUEST (1) = 612 or 622). Note this is one of the uses of the array IQUEST defined by the +CDE, Z = Q card (see 4, above).

Similarly, for intercepting condition IDs requesting a trap, but not mentioned in a title item RQTR, I set the trap for default class 1 by:

```
....
CALL RQTRAP (1)
IF (IQUEST (1). NE.O) GO TO BB.
```

After executing these instructions, the behaviour of some reporting statements will be as follows:

```
CALL RQTELL (115, O)
```

will simply be reported for accounting;

```
CALL RQTELL (612, O) and
CALL RQTELL (622, 1)
```

will both be trapped to statement AA, with IQUEST (1) = 612 or 622;

```
CALL RQTELL (489, 1)
```

will be trapped to statement BB, with IQUEST (1) = 90 and IQUEST (2) = 489.

A list of condition IDs in use by the HYDRA system is given in paper CO11 of the HYDRA system manual. A list of condition IDs in application programs is part of the HYDRA application manual. IDs less than 100 are reserved for system use.

Finally,

```
CALL RQEND
```

will print a summary of the condition IDs that have occurred.

5.5. Utilities

Some HYDRA system routines have been specifically made available to allow handling of data structures, dumping, termination etc. Also some routines of what we call the 'General Section' are frequently used in HYDRA programs. The following examples are an introduction to such utilities. More details and other utilities specifically available to HYDRA programs like histogramming, two-dimensional plotting, the S-package for production accounting, must be looked up in the HYDRA system manual.

(a) Examples of dropping, referring to the data structure in 2.1. above.

```
CALL QDROP (LEV - 2, 0)
```

will remove, i.e.drop and bridge the first VX-bank with its attached TK-banks from the data structure;

```
LVX = IQ(LEV - 2)
CALL QDROP (LVX - 1, 1)
```

will remove all VX-banks, but not the first, and the associated TK-banks. The first call parameter is a K-address (see 3.2. above), the second is a flag to indicate whether the horizontal link should be followed (= 1) or not (= 0) for dropping.

```
LVX = IQ(LEV - 2)
CALL QTOUCH (IQDROP, LVX, 1HS)
```

will drop the first VX-bank and its attached TK-banks without bridging. IQ (LEV - 2) will now point to a dropped bank. The Hollerith S is an option to set the required bit IQDROP in the status word of the

Starting bank, whose L-address (LVX) is given.

```
LVX = IQ(LEV - 2)
CALL QTOUCH (IQDROP, LVX, 'SH.')
```

will drop all VX-banks and associated TK-banks. The Hollerith H is the option to follow the Horizontal link.

```
CALL SBIT1 (Q (LVX), IQDROP)
```

will set the bit IQDROP in the status word of the VX-bank. For a single bank to be dropped, this is the most economic way. IQDROP is the position of the drop bit in the status word and present by the card:

```
+ CDE, Z = Q
```

(b) Examples for printing dynamic memory contents for debugging purposes.

```
CALL DQSNAP ('FIRST', 'LM.')
```

will output Links (system links and working space links)and a Map entry (address, ID) of each bank. 'FIRST' is a text associated to this print.

```
CALL QTOUCH (IQCRIT, LVX, 'S.')
CALL DQSNAP ('SECOND', 'WEMCV.')
```

will dump the entire Working space, links and data, an Extended Map (address, ID and links) of each bank, a full dump of banks with bit IQCRIT set (here the sub-structure LVX), and in Variable format.

Other options for the second call parameter:

F : all live banks, full dump
O : octal format for data

(c) Examples using bit/byte handling routines.

```
CALL SBIT0 (Q (LVX), IQCRIT)
```

sets bit IQCRIT in the status word of bank VX to zero.

```
CALL SBIT1 (Q (LVX), IQDROP)
```

sets bit IQDROP to one.

```
CALL SBIT (N, Q(LVX), IQCRIT)
```

sets the same bit to N (O or 1). Whilst IQCRIT and IQDROP are bit positions in the status word used for system - system and user - system communication, the following instruction tests the user bit 3 (bits 1 to 15 have a user assigned meaning):

```
IF (JBIT (Q (LVX), 3).EQ.O) ....
```

To handle a group of several bits (a byte), the corresponding routines exist:

```
CALL SBYT (3, A, 5, 4)
```

will set a 4-bit-byte starting at position 5 of word A to the value 3; bits 1 to 4, and 9 to maximum (15 in the status word, 32 in any other packing context) will be unchanged.

```
N = JBYT (A, 5, 4)
```

extracts the same byte in an obvious way.