LBL-10812
TR-873

NRCC NATIONAL RESOURCE FOR COMPUTATION IN CHEMISTRY

The XTAL System of Crystallographic Programs:

PROGRAMMER'S MANUAL

February 1980

LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA

Prepared for the U.S. Department of Energy under Contract W-7405-ENG-48 and for the National Science Foundation under Interagency Agreement CHE-7721305

ACKNOWLEDGMENT

This document was prepared in its final form by the authors listed on the title page. They could not have completed their work without the valuable advice of Richard Alden, University of California at San Diego; Arthur Olson, National Resource for Computational Chemistry; George Reeke Jr., Rockefeller University; Steven Sheriff, University of California at Los Angeles; Jurgen Sygush, University of Sherbrooke; Lynn TenEyck, University of Oregon; and Keith Watenpaugh, University of Washington.

The document was used in draft form at a workshop on multiple isomorphous replacement held November 10 through 18 at the NRCC at Berkeley, California.

Support of the NRCC and the University of Maryland Computer Science Center in the preparation and distribution of XTAL Programmer's Manual is gratefully acknowledged. At the University of California, San Diego, this work was supported by Research Grants GM 10928 and RR 00757.

ABSTRACT

This document establishes the basis for collaborative writing of transportable computer programs for x-ray crystallography. The concepts and general-purpose utility subroutines described here can be readily adapted to other scientific calculations. The complete system of crystallographic programs and subroutines is called XTAL and replaces the XRAY (6,7,8) system of programs. The coding language for the XTAL system is RATMAC (5).

The XTAL system of programs contains routines for controlling execution of application programs. In this sense it forms a suboperating system that presents the same computational environment to the user and programmer irrespective of the operating system in use at a particular installation. These control routines replace all FORTRAN I/O code, supply character reading and writing, supply binary file reading and writing, serve as a support library for applications programs, and provide for interprogram communication.

The XTAL system of crystallographic programs is based upon the XRAY program system. Although every attempt is made to test each program of the XTAL system, no warranty, expressed or implied, is made by the authors or their institutions as to the accuracy and functioning of the XTAL system, its subprograms, related program material, or operating instructions. No responsibility is assumed by the authors in connection with the use, attempted use, or applications of these programs.

It would be appreciated if acknowledgment of the use of XTAL be made in published work.

APPENDIX 4

Glossary of System Common Variables

The following list is designed to explain the meaning and use of the system variables used in the XTAL system. The letter preceding the description indicates the variable type. r is real, i is integer, and c is character (in the FORTRAN 77 type definition sense). If a variable is dimensioned, it is followed by a pair of parentheses ().

BFIELD() i Buffer to hold pointers to ending columns when a "FIELD" line has been encounted by AA01. Used by AA02 in translating input line images. See FIELD control card.

BFINFP() r Buffer to hold input floating point data. Each word corresponds to an input field. If the field is a number the buffer contains the number, if the field is void, the buffer word contains the signal VOIDFLG:. If the field is a character string the buffer word contains a packed pair of pointers which show where the character string actually starts and ends +1 in BFINIM. BFINFP is cleared to VOIDFLG: at the start of AA02.

BFINIM() c Buffer to hold the input line image in packed characters.
READ(IOIN1,1,END=2) BFINIM
1 FORMAT(20A4)
is an example of how it may come to be filled. The subroutine AA01 issues the read command as specified by the macro LINEIN:. The subroutine AA02 translates the input line into the floating point buffer BFINFP. See macro LINEIN:.

BFORDR() i Buffer to hold pointers to order in which data is to be placed in BFINFP from fields of input line images. See order control card in AA01 used in AA02.

BFOTFP() r Buffer to hold output floating point numbers to be translated into characters by subroutine AA07.

BFOTLN() c Buffer to hold characters for output line.

BFTITL() c Buffer to hold page title which includes current program, compound ID, page numbers, and date.

1. INTRODUCTION

This document contains the basic information about the structure, language and function of the XTAL System of Crystallographic Programs. It is intended primarily as a guide to programmers developing software for use with, or inclusion in, the XTAL system.

The XTAL system programming language is RATMAC, a high level structured programming language with built-in MACRO features. RATMAC code is transformed into local FORTRAN by the portable preprocessor supplied with the XTAL system. The advantage this approach offers over the use of FORTRAN as a distribution language will become apparent in the sections below. A RATMAC Primer (5) has been written to aid the writing of structured code.

In addition to the programming language there are a number of features fundamental to the XTAL system. The five most important of these are:

(1) All I/O plus certain other essential operations are carried out with XTAL service primitives.
(2) Inter-program communication of crystallographic data is done with the Binary Data File.
(3) Dynamic memory allocation is used.
(4) All crystallographic programs may be executed in either stand-alone or overlay mode.
(5) Programming conventions that specifically avoid the ill-defined areas of FORTRAN have been adopted.

A fundamental premise of the XTAL system is that a number of different individual laboratories can contribute transportable computer codes if clear guidelines exist. Guidelines are essential because of the desirability of a collaborative programming effort. The cost of high-quality computer software is very high. Great care has been exercised in arriving at the conventions spelled out here. They are based on experience gained in the development and adaptation of the XRAY system over a 20-year period. In addition to this hindsight, a forward look at the changes in software and hardware which are taking place in the computer industry has been made. These two considerations collectively have led to this document. XTAL represents a major departure from XRAY in method of coding, and, we hope, will represent greatly improved system with many enhanced features.

It should be stressed that this manual lays down precise but not mandatory rules for XTAL programmers. It attempts to cover the various system procedures, constraints, and programming conventions that should provide the essential framework within which all XTAL programs can coexist, communicate, and be transported from one machine to any other. Thus, as much opportunity as possible is left for

the individual programmer's ingenuity and skill to provide the most efficient and general crystallographic software possible. The idea is to supply subroutines that will aid the programmer in the long haul.

As the XTAL system is expanded and improved, the programmers' documentation will no doubt also be improved. These guidelines should therefore be kept in a loose-leaf binder so that sheets can be added and replaced easily. Constructive comments on all aspects of the XTAL system are encouraged.

2. LANGUAGE RESTRICTION AND RATMAC

The distribution language of the XTAL system is RATFOR (3) which will be processed by the preprocessor RATMAC (4,5). All program authors must refer to the latter document for details on the optimum use of this language.

In simple terms, RATMAC provides a convenient means of structuring and compressing a FORTRAN program. Experience within the computer science community has shown that the facility for structured code leads to an improved algorithmic approach to calculations and a general increase in the efficiency of many programs. More importantly, however, the provision for defining a set of instructions in the form of a single MACRO instruction suitable for insertion into the FORTRAN code at preprocessor time, can provide the ability to introduce code that is specific to a given installation without compromising portability. On both counts, this makes the use of the RATMAC very convenient and efficient for distribution of programs.

2.1 FORTRAN Restrictions

Many of the restrictions placed on FORTRAN for the XRAY76 system have been removed in XTAL by use of the "RATMAC MACRO: instruction." Nevertheless, some FORTRAN features still remain taboo and should be avoided at all costs. The FORTRAN restrictions for XTAL now read:

(1) Character set is restricted to the ASCII set:

0123456789	the integers
ABCDEFGHIJKLMNOPQRSTUVWXYZ	the upper case alphabet
abcdefghijklmnopqrstuvwxyz	the lower case alphabet
+-*/,.=:;#$()[]	plus, minus, star (asterix), slash, comma, period equals, colon, semicolon, hash or sharp sign, dollar sign, left parenthesis, right parenthesis, left square bracket, right square bracket

Additional special characters with significance in RATMAC, but best avoided in XTAL are:

& ampersand logical conjunction
! exclamation point logical negation
¦ bar logical inclusive disjunction
\ backslash logical inclusive disjunction
" double quote string delimiter with macro expansion
> greater
{ left brace statement block delimiter

[left bracket macro protection
< less
} right brace statement block delimiter
] right bracket macro protection
; semicolon statement separator
sharp comment signal
^ caret logical negation
\ backslash delimits non printing values in string

Additional delimiters

@ at sign
% percent
? question mark
~ tilde

(2) Columns 73 to 80 are reserved.

(3) Complex numbers and arithmetic are not permitted.

(4) Logical expressions should not use .EQ. or == relating two or more floating-point numbers (i.e. do not hope for exact equality between floating-point numbers. Use (ABS(Q-W).LT.SMALL))

(5) PAUSE is not used.

(6) For final versions of code, the line input/output instructions, READ, WRITE, PRINT, and PUNCH must not be used, except in macros.

(7) File control instructions, REWIND, BACKSPACE, ENDFILE, etc., must not be used, except in macros.

(8) Carriage-control characters for lineout devices must not be used.

(9) Multiple subscripts are to be avoided in number crunching algorithms. Contiguous single-dimensioned data array QX( ) should be used wherever possible.

(10) Dimension statements containing adjustable subscripts may not be used.

(11) Explicit type statements, INTEGER, REAL, and CHARACTER: should be used rather than DIMENSION. All variables should appear in a type statement.

(12) Statement functions are not used.

(13) Blank common should not be used and named common should only be used in a restricted way described in section 3.7.

2.2 MACRO Instructions

The macro instruction facility of the RATMAC preprocessor enables sets of instructions to be inserted into the FORTRAN code during processing. The use of macros is described in detail by Munn and Stewart (5), and typical examples of the different uses of this very powerful command are well-illustrated by the XTAL nucleus macros described in Appendix 2.

All macro instructions are defined by the use of the mnemonic MACRO: followed by left and right parenthesis enclosing the macro name followed by its replacement string:

```
                                                    ARG
MACRO:(<macro name>:,                                1
      <statement or instruction set>)                2
```

A simple example is:

```
MACRO:(NAME:,PETER)   #
```

During preprocessing every reference to NAME: in the code would be replaced by PETER. Note that the use of a colon (:) as the last character of each macro name is mandatory.

A slightly more complicated macro may be of the form:

```
MACRO:(MAXL:,MAXL=MAXO($1,$2)) #
```

$1 and $2 are mnemonics for argument 1 and argument 2 in the macro as it appears in the RATMAC. For instance, MAXL:(L,LN) in the RATMAC code would appear in the FORTRAN as:

```
MAXL = MAXO(L,LN)
```

For more detailed examples of the use of macros, see reference (5).

2.3 MACRO: Definitions

Macro definitions of the type shown above must, of course, precede the code where they are to be applied. In the XTAL system, all macros pertaining to a given program should appear in one place at the start of statements in that program or subroutine. In this way, there can be absolutely no doubt that the definitions precede the applications and, at the same time, this makes them easy to identify. This is shown for the nucleus macros or in the test program EX01 supplied on the XTAL distribution tape. Another condition of the XTAL system, is that program macros must not be machine specific. If for some reason it is necessary to use a machine-specific macro, this will have to be transferred to the nucleus macros. In this way, machine-specific macros are kept to an absolute minimum and are located in one place for ease of implementation.

2.4 XMACRO: Instructions

Just as the nucleus routines are available to every XTAL program, the nucleus macros are always active for insertion into program code. This is not the case for macros defined by codes for other programs, however. The MACRO table in the preprocessor containing the macro definitions is of limited length. It must therefore contain only the macro definitions that are required for the code being processed.

To remove macro definitions from the macro table, the command XMACRO: is used with the name of the macro to be deleted enclosed between square brackets. For example, the macro given above would be deactivated by the instruction:

```
XMACRO:([MAXL:]) #
```

It is strongly recommended that the XMACRO: instruction be used only in the RATMAC code just before the END image of the appropriate subroutine or program. It is absolutely essential that all program macros be expunged before the END statement of the program or subroutine, lest the macro tables be flooded.

2.5 XTAL Conventions for RATMAC

Certain conventions in the use of RATMAC are necessary for the XTAL system because it is a collaborative package. It is necessary to adopt conventions to make updating and general diagnostic procedures consistent for the collaborating programmers.

The following conventions must be adopted by XTAL authors if their programs are to included in the distribution library. They are established to assure portability:

(1) All macros containing more than one statement must be surrounded by the digraphs $(and $) or { and }. Curly brackets may not be universally available.

(2) An ELSE IF statement should not follow a statement containing a BREAK, NEXT, RETURN, or STOP to avoid the FORTRAN diagnostic, "a part of the program cannot be reached".

(3) Character strings must be defined using the DATASTORE: and DATASTUFF: macros.

```
                                                    ARG
DATASTORE:(<mnemonic of character string>,           1
           <character string>)                       2

DATASTUFF:(<mnemonic of character string as specified
           in a preceding DATASTORE: statement>)
```

See sections 4 and 5 for details. The routines on the distribution tape show examples.

(4) Names assigned to character strings in the DATASTORE:/DATASTUFF: definitions may be a rational combination of the program code and a sequence number. The string name must not in any case exceed five (5) characters. This is because the length of the character string is automatically assigned a name by the DATASTORE: macro that starts with the letter N and ends with the character-string name. In this way, conflicts with variable names in the body of the program are minimized.

(5) The DATASTUFF: macro must follow all type definition, common, and equivalence statements. Many compilers demand all memory reservation statements before any data declaration statements.

(6) All XTAL programs and subroutines must start with the SYSTEMHEADER: macro. This macro causes the generation of local operating system control lines when needed.

(7) All program (entry-point) routines must contain, immediately after the SYSTEMHEADER: macro, the two macros OVERNA: and PROGRAM:. In addition, the macro PROGRETURN: must be inserted just before the END statement. These are the "main" programs of overlay segments. See section 3.3 for details.

(8) Any time the $(or { or $) or } is used for multi-line statements, the brackets should be on their own lines.

(9) Every statement line should end with a # or $# to stop the RATMAC scan as soon as possible. Comments may be placed after the # or $#. All pure comment lines should begin with a #.

3. SYSTEM FEATURES AND SPECIFICATIONS

The XTAL system is designed to promote PORTABILITY and MODULARITY of code, so that COOPERATIVE PROGRAMMING of crystallographic software is possible.

FORTRAN + RESTRICTIONS = PORTABILITY

where

RESTRICTIONS = definition and isolation of FORTRAN inconsistancies and data structure conventions.

MODULARITY requires that each programmer work against a common data base and use standard inter-module communication conventions.

COOPERATIVE PROGRAMMING means that code developed by individuals working in isolation can be incorporated into a set of programs that will achieve a common scientific goal. COOPERATIVE PROGRAMMING requires that the individual programmer submit to arbitraary conventions. The conventions embodied in the XTAL system insure that PORTABILITY, MODULARITY, and COOPERATIVE PROGRAMMING can be achieved with no sacrifice in program efficiency.

In the 12 sections which follow, the XTAL tools, conventions, and restrictions are enumerated in some detail: The sections describe:

(1) XTAL nucleus routines which provide essential primitives
(2) program naming conventions which provide unique names for routines and permit easy assembly of the XTAL system
(3) the storage allocation scheme which provides for efficient utilization of memory
(4) inter-communication protocol which insures proper communication between XTAL routines
(5) a template for a typical program which illustrates use of the communications protocol
(6) SYSTEM common blocks which provide inter-routine communication within a single program
(7) OVERLAY common blocks which provide communication between different XTAL program segments
(8) specifications for integer and floating point words
(9) word packing conventions and utilility macros
(10) bit, byte, word, integer, real, and character manipulation macros
(11) the QXDATA common block which provides storage for all I/O buffers and large arrays
(12) a procedure for dynamic core allocation

3.1 System Nucleus

The kernel of the XTAL system is a set of subroutines which perform basic service functions. These are referred to as the XTAL system "nucleus," and have subroutine names with the two-letter prefix AA, and sequence numbers ranging from 00 to 99.

The nucleus routines fall into five broad classifications:

(a)	Control and Line I/O Subroutines	(AA00-AA19)
(b)	Binary file I/O Subroutines	(AA20-AA39)
(c)	Memory/Error Subroutines	(AA40-AA59)
(d)	Direct access I/O Subroutines	(AA60-AA79)
(e)	Machine-specific macro-called subroutines	(AA80-AA99)

Subroutines currently in these classifications are supplied on magnetic tape. The contents of a tape is shown in Appendix 1. Subroutines of the nucleus are described in brief in Appendix 3.

The programmer uses mnemonic macros to cause actual calls to the nucleus subroutines. However, it should be noted that the "entry point" names of these subroutines correspond to their deck names. The subroutine names have been chosen to eliminate the possibility of using the reserved names of an operating system.

An example is:

```
                                                        ARG
READLINE:(<list of acceptable line identifiers>,         1
          <number in the list>)                          2
```

which a programmer uses to input the next line image. The macro READLINE: results in the correct CALL statement to AA01 being generated in the FORTRAN code.

Because the nucleus routines will, in general, always reside in memory with each program, considerable effort has been made to keep them as small as possible, and yet flexible enough to perform most of the service I/O and memory manipulation functions that the XTAL programmer may wish to use. There is no question that programmers will occasionally meet situations in which a more elegant in-shop software facility will do a better job than a nucleus routine. It may also be possible to generalize the use of such a facility through the use of RATMAC macro statements. This must, however, be done with extreme caution, if the programs are to be part of the transportable library. It should only be done after consultation with the authors at the University of Maryland to ensure that such a facility has an equivalent on other machines.

It is essential for a successful cooperative effort that programmers do not alter the function of the nucleus routines.

3.2 Program Naming Conventions

Each program in the system is cataloged by a two-letter code. There are two reasons for cataloging subroutines in this manner:

(1) To avoid conflict with local machine system reserved entry points, such as EOF, SIN, etc., and

(2) To cause a sort on program mnemonics to place the whole system in proper overlay order.

So that there can be many different programs with non-conflicting names, a labeling scheme has been defined using two character - two number mnemonics. The table in Appendix 1 shows the character pairs (with their functions) that have been reserved so far. The numbers of the subroutines are assigned sequentially in order of the overlays of the given program. For example, the program to keep track of memory and time is called MT, the program which dumps the binary data file is TD, the Beever-Lipson Fourier is FS.

In the descriptions below, the mnemonics XX00, XX01,...XXNN will be used to refer to a general set of related subroutines which carry out a crystallographic calculation.

Consider that XX is the chosen mnemonic for the cataloging of a given calculation. The main overlay is labelled XX00, and the "entry point" which is called from nucleus subroutine AA01, is also XX00. Then each overlay called by XX00 plus any subroutines attached to these overlays are designated XX01, XX02, etc. The call to XX00 is always placed in AA00 of the nucleus.

The first macro used in setting up a subroutine is the SYSTEMHEADER: macro. From the line SYSTEMHEADER: to the line END constitutes a subroutine. For example:

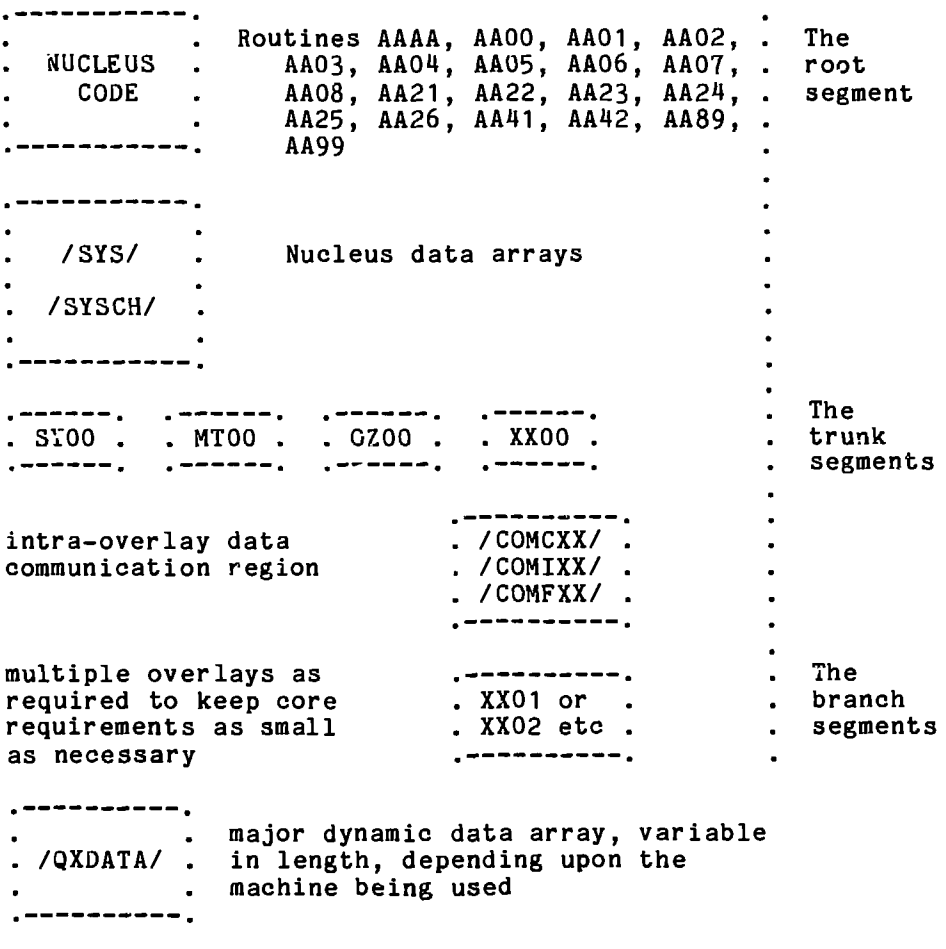
```
SYSTEMHEADER:(<subroutine mnemonic>)  #
. . . . . RATMAC statements
. . . . . of subroutine
END                  #
```

3.3 Storage Utilization

The method for the utilization of storage in the XTAL system has been chosen both to facilitate the use of overlays in a fixed-memory machine and to minimize paging in

a virtual memory machine. The nucleus routines and system common form a "root segment" while each major crystallographic program forms a set of overlays that depend upon the calculation at hand. These overlays consist of "trunk segments" called by subroutine AA00 of the nucleus. The "trunk segment" is given the mnemonic XX00 where XX is the indication of the function of the overlay (e.g. FC, FS, etc.) The 00 overlay may have attached to it three common arrays: COMI: for integer, COMF: for real, and COMC: for character variables which must be passed among the branch overlays. The "trunk overlay" XX00 calls all of the branches XX01, XX02, XX03, etc. which make up the crystallographic task defined for the given overlay. The first routine of any program (XX00) does not itself do any calculations other than managerial tasks that control the sequence of suboverlays.

The following diagram illustrates the overall storage utilization in XTAL; down the page corresponds to increasing addresses in a nonvirtual-memory machine. In a virtual-memory machine with individual programs, the XTAL nucleus insures that only one trunk plus its associated branches would be present in memory at one time.



3.4 Protocol for Program Intercommunication

A program or subroutine is integrated into the XTAL system through the use of the OVERLA:, OVERNA:, PROGRAM:, and PROGRETURN: communication macros together with the naming mnemonics described in section 2.2. The program or subroutine, a general XX00 routine shown as either a trunk overlay or a branch segment in the storage diagram of section 2.3, may not require all four communication macros. However, it is always required that the routine be numbered

in correct sequence to insure that it will be sorted into its proper place in the overlay scheme. A description of these communication macros follows.

3.4.1 OVERLA:

```
                                                          ARG
OVERLA:(<mnemonic of the crystallographic subroutine>,     1
        <overlay number>,                                  2
        <sequence number of the overlay>)                  3
```

This command to overlay is used in subroutine AA00 of the nucleus or XX00 of the crystallographic program and nowhere else.

3.4.2 OVERNA:

```
                                                          ARG
OVERNA:(<mnemonic of the crystallographic subroutine>,     1
        <overlay number>,                                  2
        <sequence number of the overlay>)                  3
```

This macro comes just after the SYSTEMHEADER: macro. In those subroutines which are designed to be overlayed. As pointed out above, the management of the suboverlays is always the business of the root segment, XX00, of the given crystallographic program.

3.4.3 PROGRAM:

```
PROGRAM:(<mnemonic of the crystallographic subroutine>)
```

This macro follows the OVERNA: macro in the subroutine.

3.4.4 PROGRETURN:

This macro is used in lieu of the FORTRAN RETURN statement in a subroutine which has the PROGRAM: macro at the beginning.

The inclusion of a new program would require modification of AA00 to include the name of the program line identifier such as XXXXXX in the A001 DATASTORE: list and the addition to the conditional statements of subroutine AA01 a new line:

```
ELSE IF(KCD.EQ.<entry number>)  $(     #
          OVERLA:(XX00,<overlay number>,0)    #
$)                                #
```

3.5 Template for an XTAL Program

A prototype of a complete crystallographic program follows here showing the use of the macros required to form an overlay scheme. Of course, all calculation-related statements are omitted.

```
SYSTEMHEADER:(XX00)                          #
OVERNA:(XX00,<overlay number>,0)             #
PROGRAM:(XX00)                               #
. . . . . . . . . . . .   the dots represent any
. . . . . . . . . . . .   additional code depending on the
. . . . . . . . . . . .   complexities of control required
OVERLA:(XX01,<overlay number>,1)             #
. . . . . . . . . . . .    assuming overlay 1 has 3 subroutines
. . . . . . . . . . . .     then the next overlay will be XX05
OVERLA:(XX05,<overlay number>,2)             #
. . . . . . . . . . . .
. . . . . . . . . . . .
PROGRETURN:                                  #
END                                          #


SYSTEMHEADER:(XX01)                          #
OVERNA:(XX01,<overlay number>,1)             #
PROGRAM:(XX01)                               #
. . . . . . . . . . . .    common, data statements,
. . . . . . . . . . . .    calculations and calls to
. . . . . . . . . . . .    XX02, XX03, and XX04
PROGRETURN:                                  #
END                                          #


SYSTEMHEADER:(XX02)                          #
SUBROUTINE XX02                              #
. . . . . . . . . . . .     a typical RATMAC
. . . . . . . . . . . .     subroutine
. . . . . . . . . . . .     which serves XX01
RETURN                                       #
END                                          #


. . . . . . . . . . . .    the other subroutines of
. . . . . . . . . . . .    overlay one


SYSTEMHEADER:(XX05)                          #
OVERNA:(XX05,<overlay number>,2)             #
PROGRAM:(XX05)                               #
. . . . . . . . . . . .    this is overlay 2 of XX00
. . . . . . . . . . . .
PROGRETURN:                                  #
END                                          #
. . . . . . . . . . . .
. . . . . . . . . . . .    and so on for the
. . . . . . . . . . . .    subroutines of XX05
```

No overlay scheme is ever more than two deep. No overlay ever calls another overlay at its own level. All calls are top down. AA00 calls XX00; XX00 calls XX01; XX01 calls XX02, XX03, and XX04; XX00 calls XX05; XX01 and XX05 return to XX00; XX00 returns to AA00.

3.6 System Commons /SYS/ and /SYSCH/

All integer, real, and character variables commonly used by the nucleus and other XTAL programs reside in the labelled system commons /SYS/ and /SYSCH/. The length of these common arrays will vary from installation to installation according to the number of characters per word and the lengths of the I/O buffer arrays. The reason for separating type CHARACTER from types REAL and INTEGER, is the fact that FORTRAN 77 demands it.

The macros which generate /SYS/ and /SYSCH/ are shown in detail in the XTAL distribution tape. A detailed description of all of the variables in common is given in Appendix 4.

The macro SYSCOM: placed in a subroutine will cause all of commons /SYS/ and /SYSCH/ to be included in the subroutine as well as the /QXDATA/ common described in a later section.

The length of the common is set at system generation time by the various word-size dependent macros (MXCHWD:, MXCHLN:, etc.). The nucleus macros are summarized in Appendix 2. The /SYS/ and /SYSCH/ variable names (as opposed to the macros) must not be modified in any way by the programmer. These are an intrinsic part of the nucleus programs.

In the overlay mode, /SYS/ and /SYSCH/ are available to all programs simply by the presence of the macro SYSCOM: statement. In the stand-alone mode, however, the contents of /SYS/ and /SYSCH/ would be lost between calculations were it not for the automatic storage and recovery of the variables on device ISAVE: by the main entry-point routine AAAA. This routine writes out the /SYS/ and /SYSCH/ arrays on the completion of each program, and reads it back into the same locations at the start of the next program. Since /SYS/ and /SYSCH/ are preserved on an external file in the stand-alone mode, the programs operate identically in all other respects. This feature is provided for use with virtual-memory operating systems.

When a system is formed for nonoverlay use, that is for stand-alone control, each routine will have to have a version of AA01 tailored to contain one and only one call to the crystallographic programs.

3.7 Overlay Communication Commons /COMI:(XX)/, /COMF:(XX)/, and /COMC:(XX)/.

For many programs, such as a Fourier transform program, variables set up by the first overlay segment of the program are control parameters required by the later overlay segments (parameters such as grid sizes, numbers of layers,

etc.) For these special few quantities, up to three labeled commons may be specified. The one for integers is labeled /COMI:(XX)/ where XX is the two-letter mnemonic of the particular program. Note that COMI: is an XTAL system macro. This means that the actual labeling may be controlled globally for machines with special restrictions on the numbers or kinds of labeled common that may be used. /COMF:(XX)/ is for real variables and /COMC:(XX)/ is for character variables. The programmer should take great care to use only the appropriate variables in each and keep the number of items to a minimum. The variables in common should never be used as running indices in loops. They should be used for communication of limits, total counts, etc. Only local variables should be used for running indices in loops. This practice allows FORTRAN optimizers the best chance to generate efficient code.

3.8 Restrictions on INTEGER and REAL

The XTAL system is designed to accomodate machines with a minimum-integer word size of 16 bits, and a minimum floating-point word size of 32 bits. If there is a problem with the 16 bit integer restriction, then this restriction can be ignored; however, comments indicating that this has been done should appear both in the code and in the documentation.

The maximum permissible integer magnitude is 32,767 for 16 bits and therefore care must be taken that floating-point-to-integer conversions do not exceed this value. Floating-point numbers are expected to range from -1.E+25 to +1.+25, although negative numbers with magnitudes greater than 1.E+20 are treated as special information (see section 4.2 on Line Input).

3.8.1 INT: and IFIX:

Floating-point (real) to fixed-point (integer) conversion macros INT: and IFIX: are available for use in the system. The first is for positive numbers only and the second, which is slower, is for positive or negative numbers. They are used:

<integer>=INT:(<real>), and

<integer>=IFIX:(<real>)

3.9 Word Packing

Packed numbers may only be processed in words of floating-point length (i.e. 32 bits). The macro MXBTWD: is the actual bits in the REAL word of a machine. It was arbitrarily decided to make use of only 32 bits for packing purposes. The packing of numbers is performed through the

macros MOVEBITS:, INTPAK:, and INTUNPAK:. These macros pack each number according to a specified bit position and bit length. In the XTAL convention bit 0 is the right-justified, low-order bit. These macros should be defined with the most efficient bit-manipulation functions available on each machine (e.g. the FLD function on UNIVAC). Packed integer numbers are used extensively in the XTAL system to reduce memory demand and, sometimes, to increase computation speed. For an example, see record 20 of the binary data file in Appendix 5. The macros are described in the next section.

3.9.1 MOVEBITS:

```
MOVEBITS:  Move bits among words                                    ARG

MOVEBITS:(<real or integer word for source of bits>,                  1
          <low order bit position in source word>,                    2
          <real or integer word for destination of bits>,             3
          <low order bit position in destination word>,               4
          <number of bits to be moved>)                               5
```

3.9.2 INTPAK:

```
  INTPAK:  Pack integers into real words                            ARG

  INTPAK:(<integer for source of bits>,                               1
          <real word for destination of bits>,                        2
          <low order bit position in destination word>,               3
          <number of bits to be moved>)                               4
```

Example:

```
INTEGER NFILM,IPT
INTPAK:(NFILM,BUF(IPT+5),20,12)
```

Take 12 bits from "NFILM" beginning with bit 0 and insert them into BUF(IPT+5), beginning at bit 20. This would take a 12-bit film number and pack it into the most significant bits of a buffer (presuming a 32-bit word size).

3.9.3 INTUNPAK:

```
INTUNPAK:  Unpack integers from real words                          ARG

INTUNPAK:(<real word for source of bits>,                           1
          <integer word for destination of bits>,                   2
          <low order bit position in source word>,                  3
          <number of bits to be moved>)                             4
```

Example:

```
INTEGER OLDPAK,IPT
INTUNPAK:(BUF(IPT+1),OLDPAK,0,MXBTWB:)
```

Takes a complete word's worth of bits from a real

variable (BUF) and puts them into an integer variable (OLDPAK). MXBTWD: is a macro defined to be the number of bits per word.

3.10 Character Macros

The whole problem of bits, bytes, words, integers, reals, and/or characters presents a messy problem for transportability of codes. There are in the XTAL system a number of macros supplied to aid in keeping all of this straight. The main rule is do everything possible in floating point (i.e. as reals). Because of the advent of FORTRAN 77, it is necessary to eliminate all equivalence statements which mix types. To this end, a series of macros make possible the operations needed in crystallographic coding, but avoid any equivalencing. They are described in sections 3.10.1 - 3.10.7.

3.10.1 MOVEBYTE:

MOVEBYTE: Move characters from one array to another ARG

```
MOVEBYTE:(<array for source of characters>,                    1
          <index of 1st character in source array>,           2
          <array for destination of characters>               3
          <index of 1st character in destination array>,      4
          <number of characters to be moved>,                 5
          <key>)                                              6
```

KEY = 0 Character-by-character transfer from source to destination.

KEY = 1 Transfer first character of source array to all positions in destination array. This is useful for setting the destination array to blanks.

3.10.2 MOVECHR:

MOVECHR: Move characters in arrays as in FORTRAN 77 ARG

```
MOVECHR:(<array for source of character>,                      1
         <index of 1st character in source array>,            2
         <array for destination of characters>,               3
         <index to 1st character in destination array>)       4
```

This is a machine specific function (see AA89 in its various forms).

3.10.3 MOVECTOR:

MOVECTOR: Move characters to real words

Note that it is MOVE C TO R, not MO VECTOR. ARG

```
MOVECTOR:(<array for source of characters>,                    1
          <index of 1st character in source array>,           2
```

```
          <real array for destination of characters>,     3
          <index to 1st character in destination array,   4
          <number of characters to be moved>,             5
          <key>)                                          6
```

KEY is defined under MOVEBYTE:.

3.10.4 MOVERTOC:

MOVERTOC: Move characters from real array into a character array.

```
                                                          ARG
MOVERTOC:(<array for source of characters>,               1
          <index to 1st character in source array>,       2
          <array for destination of characters>,          3
          <index to 1st character in destination array>,  4
          <number of characters to be moved>,             5
          <key>)                                          6
```

Key is defined under MOVEBYTE:.

3.10.5 MOVEREAL:

MOVEREAL: Move vectors of real words without altering bit patterns (i.e. no normalization).

```
                                                          ARG
MOVEREAL:(<array for source of reals>,                    1
          <index to 1st word in source array>,            2
          <array for destination of reals>,               3
          <index to 1st word in destination array>,       4
          <number of words to be moved>,                  5
          <key>)                                          6
```

KEY = 0 Source and destination are different arrays.

KEY = 1 Source and destination are the same array.

KEY = 2 Clear destination array to value of first word of source array.

3.10.6 MOVERWORD:

MOVERWORD: Move a real word from one location to another without altering bit pattern (i.e. no normalization).

```
                                                          ARG

MOVERWORD:(<word for source of real>,                     1
          <word for destination of real>)                 2
```

3.10.7 CHARACTER: and CHARS:

CHARACTER: and CHARS: Control of Character Arrays

CHARACTER:(<symbol that defines the character array>

CHARS:(<number of characters in a string>)

Because of the conflict between the way that characters are treated in FORTRAN 66 and FORTRAN 77, two special macros have been introduced. Both are used in data declaration statements. CHARACTER: emulates the FORTRAN 77 data type CHARACTER and allows it to be changed to INTEGER on machines using FORTRAN 66. CHARS: is used to make character definitions that are consistent with FORTRAN 77. If an array of 50 characters is to be stored in an array CH, the statement:
CHARACTER:CH(CHARS:(50)) #
is used. On a machine with FORTRAN 77 this is the same as:
CHARACTER CH(50)
On a six character per word machine using FORTRAN 66, it is the same as:
INTEGER CH(9)
Note that there are four extra characters at the end of the array. The MOVEBYTE: and other character handling macros are consistent with this method of handling characters.

3.11 QXMEMORY:, QX DATA ARRAY /QXDATA/

/QXDATA/ is the major common block of the XTAL system. It contains a one-dimensional array which is used for all I/O buffers and other large arrays. Data should be stored in it contiguously starting at the lowest address possible for any given calculation. This will give programs which are optimized with respect to speed and space.

The common statement is part of SYSCOM: and is simply COMMON/QXDATA/QX(1) on machines which allow dynamic core allocation. At load time, on these machines, it is forced to the end of the longest overlay where it is expanded and contracted as necessary. On other machines it is used as:

COMMON/QXDATA/QX(<a number commensurate with available storage>)

In the preparation of general codes to run on a variety of sizes of computing machines with various operating systems, it is very desirable never to have to specify exactly the number of words of memory which are required. To this end, the use of packed one-dimensional arrays for all large data manipulations is desirable. This means that to generate two, three, four, or more dimensional arrays, the program does not rely on statements, such as:

DIMENSION RHO(10,10,10)

Rather, the QXDATA array is used with a pointer system that

maps out the multi-dimensioned array packed as tightly as possible. This assures that whatever size machine is finally available, the nature of the calculation at hand will determine whether the problem will actually run, not an arbitrary choice in the multi-dimensioned variable definition.

There are variables in common /SYS/ which are used by the programmer to manipulate the QX array for the given crystallographic calculation. These are:

(1) QXSTAR - Certain overlay loaders (e.g. CDC) require that the lowest elements of the QX array be reserved for program code. In order that all data references to the QX array may take this into account, it is necessary to define a base pointer above which all data references to the QX array are made. Thus, the first location into which data are stored is QX(QXSTAR+1). The value of QXSTAR is initialized at the start of each calculation by the routine MT00 which sets it equal to the appropriate value in the QXSK array. The QXSK array (in MT00) contains the "skip" indices appropriate to each overlay and operating system. For many machines these indices will be zero. In any case, absolutely no references may be made to the QX array without taking the value of QXSTAR into account. The value of QXSTAR should never be changed within a program. An example involving the use of QXSTAR is given in Section 3.12.

(2) QXWORK is an index which defines the minimum number of QX array elements required by a given program. The value of QXWORK is preset to QXSTAR + QXWK(KCD) by the routine MT00 prior to each program. The array QXWK contains the absolute minimum number of data words required for each program. Elements of the QX array beyond QXWORK are considered to form the work area, which is to be expanded and contracted according to the demands of the problem in hand. The index QXWORK represents the base pointer to the work area, and, in general, memory should never be reduced below this value.

(3) QXREQU is the index passed to the memory allocator routine QXMEMORY: as a request for a new upper limit to the QX array. QXREQU may be set to an index value which is greater than or less than the current upper limit defined by QXAVAL. This dicates whether QXMEMORY: allocates more or less memory. All changes to the length of the QX array are made by setting QXREQU and then calling QXMEMORY:. See Section 3.12 for examples.

(4) QXAVAL is the index returned by the memory allocator routine QXMEMORY: indicating the upper limit to the

QX array currently permitted. Normally this index will be equal to (or for some O/S slightly higher) than the upper limit requested via QXREQU. However, when the memory request defined by QXREQU cannot be met by QXMEMORY:, the value of QXAVAL will be set to the maximum QX index available. It is therefore always necessary to test QXAVAL against QXREQU and take whatever action is appropriate. This is discussed further in Section 3.12.

XTAL programmers are encouraged to store all data in the single-dimensioned floating-point data array QX. The strategy for using this one-dimensional approach is given in Crystallographic Computing, Proceedings of the 1975 Summer School on Crystallographic Computing (1).

In the XTAL system, each program may reserve an area of the QX array between QX(QXSTAR+1) and QX(QXWORK) in which inter-overlay communication (or reserved common) variables reside. On some machines, principally CDC, the value of the pointer QXSTAR may not be 0 because of the way the QX array is loaded in the overlay mode. This is discussed further in section 3.12 under dynamic core allocation. The pointer QXWORK is the last word of the "reserved-common area" of the QX array. Beyond this point, the programmer uses the QX array in the way that best suits the calculation at hand. At any given time, the upper limit of the QX array is defined by the storage allocation pointer QXAVAL. Usually the area immediately following QXWORK is reserved for the binary file buffers (discussed under FILE INPUT/OUTPUT below), but the positioning of the I/O buffers is strictly under the programmer's control.

In the working programs, it is necessary that the programmer calculate how much storage the calculation will require (see section 3.12). This can be accomplished in one of two ways. One is to simply decide on a value, say 20000, that will be needed for the task at hand. The other is to make a calculation based, say , on the number of atoms to be treated times the number of parameters per atom which are to be stored. In either case, once the number is arrived at, it is stored into QXREQU of COMMON /SYS/.

QXREQU = <number of words required in QX array>

Then the QXMEMORY: macro is invoked. Before any calculations are done, the values of QXAVAL must be tested to see if the amount requested is actually available. The programmer again must decide the strategy to employ if QXAVAL is less than QXREQU. If the calculation can not be carried out with any less memory, the following statement would be made:

IF(QXAVAL.LT.QXREQU) IQUIT:(<encoded error message>) #

The IQUIT: macro will be elaborated below.

3.12 Dynamic Core Allocation

The content of this section may seem to be a moot point in virtual-memory machines. The practice of using packed core is, however, very important on these machines as well, since it tends to minimize paging during execution of the codes. Furthermore, all virtual-memory machines have some actual upper limit that can be exceeded by large programs with huge data arrays.

In the XRAY76 system, a procedure was introduced for allocating memory to a program according to the size of the crystal structure and the type of calculation to be performed. This was done in a way that placed the onus on the user to decide how much memory should be allocated to each program. In effect, this meant that memory allocation was usually requested only at the start of each calculation.

In the XTAL system the same basic memory-allocation procedure is used, but now the programmer may use QXMEMORY: in each program to continually request and release memory according to current memory demand. In this way each program retains the smallest memory in which it can operate efficiently and thus reduces the overall computing cost. Under normal circumstances, the XTAL user should not be aware that the "dynamic" nature of the memory allocation procedure is in progress, except from the memory summary of the various maxima put out as each program step is completed (see MT00). The exception is when a priority line-out limit is set to 5. In this case, a message will be printed with every memory request.

All memory allocation requests are made to QXMEMORY: (via the floating-point word QXREQU) for the number of words in the QX array (see 3.11). For example, setting QXREQU = 1000. requests to QXMEMORY: to make available the words from QX(QXSTAR+1) to QX(QXSTAR+1000). This is, however, an overly simple example in a usual programming situation. In the various partitions of the QX array that occur in real programs, there will be regions for I/O buffers, data, matrices, etc.. The programmer must set up and maintain markers for these various variables which are stored in the QX array.

Typically, a request to QXMEMORY: might first be with QXREQU = XMARK, where XMARK = QXWORK + A*STEP. Later, a further increase is made with QXREQU = XMARK + 5.*GROUP. Still later, this may be reduced back to QXREQU = XMARK, and so on. See the test routine EX01 for a working example.

As discussed in 3.11, QXSTAR + 1 is the pointer to the first "useable" word in the QX array. This is necessary because QXSTAR has a nonzero value in some machines (mainly CDC) representing the memory consumed by overlays.

The first request for memory allocation is always made in MT00 before each program is loaded. This provides both the memory for secondary overlays (in the case of CDC), and a minimum memory block necessary for program "start-up." This minimum block of words is in MT00 as QXWK(KCD) and is the boot-strap memory required to initialize the line input and estimate the memory really needed to do this particular independent calculation in addition to the QXSK values which set the skip over any programs.

When the request to QXMEMORY: exceeds the memory available, then QXMEMORY: is designed to return the memory actually available in the variable QXAVAL. This requires that the machine-specific macros MEMMORE: and MEMLESS: either be capable of recognizing what core is accessible at the time of request or use the macro MEMMAX: to return to QXMEMORY: the maximum memory address available or possible. Programmers must always test the value of QXAVAL (not QXREQU) to be certain how much memory may be used. This allows the program to follow an alternative procedure such as give a message and exit, or use a scratch file as virtual memory. It should be noted also that at all times memory requests to QXMEMORY: are tested against the total physical memory limit specified by the macro MEMMAX:.

To provide the user with information on both the largest QX and total memory (nucleus + overlay + QX array) requested and allocated in each program, values are stored in QXRQPG and QXAVPG. The maximum of these values for all programs is stored in QXRQMX and QXAVMX.

It should be noted that if a system MEMSET image is input (see section 4.6) with a nonblank value, all program memory requests are deactivated until another MEMSET image is read containing a blank value. Note also that the request for QX data array on the MEMSET image is for "useable" memory so that QXSTAR is added to QXREQU in READLINE:.

One further /SYS/ variable is available to the programmer for memory manipulation: QXUTMX. This word is the working pointer to the last word of the QX array being utilized at any one time. This is an important variable which, if tested against QXAVAL, signals when to request an increase or decrease in memory. It must be stressed that dynamical core allocation will be most successful when requests to QXMEMORY: are not too frequent (or too infrequent) and when reasonable blocks of words are involved. Repeated requests for small changes in memory, of say five words, will clearly be prohibitive overhead for any calculation to carry. Most machines, in fact, will only allot memory in 'pages'. These pages often are either 512 or 1024 words in length. Applied intelligently, however, this procedure will keep memory demand, and thus coresidence charges, to a minimum without significant increase in CPU time.

4. LINE INPUT CONTROL

In the XTAL system, input images are not processed under format control. A nucleus macro DCODEFLD: replaces the FORTRAN format scanner. This subroutine processes input line images from cards, VDU, teletype or any other line input device. Control is never lost no matter what input is encountered.

4.1 READLINE:, DATASTORE:, DATASTUFF:, Input of Data

In practice, the programmer does not use DCODEFLD: directly, instead the following macro is used:

```
                                                        ARG
READLINE:(<mnemonic of character string defined          1
          by a DATASTORE: statement>,
         <count of number of characters in the string>)  2
```

The macro DATASTUFF: automatically produces the character count for the programmer. An example is the most straightforward way to show how the system is used. Consider the following skeletal program:

```
SYSTEMHEADER:(XX01)
. . . . . .
. . . . . .
DATASTORE:(X001,bATOMbbbSCALEbbMAXHKLbENDbbbb)
     # The lower case b's indicate "must be blanks".
     # There is always one leading blank followed by 6
     # characters, not all of which may be blanks.
     # (i.e. character strings of length seven).
     # This DATASTORE: statement sets up the list of
     # acceptable input lines expected by the program XX01
     # at some point.
. . . . . .
. . . . . .
DATASTUFF:(X001)   # this actually causes the data
                   # to be placed in DATA statements
. . . . . .
. . . . . .
. . . . . .
     # at the point where the logic dictates the actual
     # input to take place, the following statement appears...
READLINE:(X001,NX001)    # the integer variable NX001 is
                         # generated by the DATASTORE: macro and
                         # the actual count of characters in X001
                         # stored in NX001 by the DATASTUFF:
                         # macro.
     # In the COMMON/SYS/ there is an integer variable KCD
     # which is set by subroutine AA01 as a signal indicating
     # which of the 4-character line mnemonics have been
     # encountered.  If none, an error exit has occurred.
     # In this example, KCD=1 for ATOM, 2 for SCALE,
     # 3 for MAXHKL, and 4 for END.
```

```
. . . . . .
. . . . . .
IF (KCD.EQ.1)  #
$(  #
. . . . . # treat atom input
$)  #
ELSE IF (KCD.EQ.2)  #
$(  #
. . . . . # treat SCALE input
$)  #
. . . . . .
and so on
```

Once the READLINE: has been invoked, the whole input line is automatically translated by the DCODEFLD: macro.

This macro leaves all of the fields of the input line standing in two input buffers. The first BFINIM (BuFfer of INput IMage) is of type CHARACTER:, and is in COMMON/SYSCOM/. The second BFINFP (BuFfer of INput Floating Point), is of type REAL and is in COMMON/SYS/.

The ramifications of DCODEFLD: will now be explained.

DCODEFLD: scans an input line in two different ways. The first and most important way is as a free-format translator. In this case, each line is decoded, field-by-field, into an input buffer consisting of a collection of floating-point words. The concept of a field is delineated in the next section. The second way of scanning, is by input line columns that have been specified by previous input parameters. These parameters are entered in the input stream on a FIELD line and serve to define the delimiter of each field. Input lines like the FIELD line are detailed in section 4.5.

4.2 CHRLIMOUT:, The Concept of a Field

A field is a string of contiguous characters in the input line image. A field is scanned from left to right. A blank, comma, or equals sign is treated as the signal that a field has been spanned. Definite input columns may be set to define the end of fields for lines that have no intervening blanks, commas, or equal signs. In either case, each field is tested for the kinds of characters present and, field-by-field, a floating-point number is stored in an input buffer within DCODEFLD:. The size of this buffer is defined by the macro MXFDIM:

MXFDIM:(<integer defining maximum number of fields per image>)

Three distinct classes of floating-point numbers are stored. The first class is the actual floating-point representation of a real number. The second class is a signal number

indicating a void field was encountered. The third class is a floating-point number containing a pair of packed pointers. These pointers point to the beginning and end of character-string fields in the input image. The macro CHRLIMOUT:

```
                                                          ARG
CHRLIMOUT:(<integer argument which serves as field index>,1
           <integer returned which points to the first    2
            character in the field>,
           <integer returned which points to character    3
            just beyond field>)
```

is used to fetch the beginning and ending + 1 character pointers. These pointers are used in conjuction with MOVECHR: to move the character strings out of the input image buffer BFINIM.

4.3 Interpretation of Input Line Images

Input lines are read starting in the column following the image name which has been screened by macro READLINE:. Each input image is decoded either as a floating-point real number, as a character string, or as a special instruction signal. The FORTRAN type INTEGER is subsumed into the type REAL. When integers are required in subroutines, the macros IFIX: or INT: are used to move the reals into local integer variables. In general, in the XTAL system, integers are used only for counting purposes. The use of equivalence between integer and real is assiduously avoided because of the possibility of different word lengths.

4.3.1 REAL, CHARACTER, and Special Characters

Fields in the input stream which begin with characters 0123456789.+- are by convention, the signal to DCODEFLD: that the string shall be decoded into the input buffer as a floating-point number. The first blank or comma encountered defines the end of the field.

Fields which begin with characters not including 0123456789.+-$*: are, by convention the signal to DCODEFLD: that the string shall be treated as a character string. In this case, as in the case of numbers, the first blank or comma encountered defines the end of the field.

The word in the floating-point buffer BFINFP is set to a packed number defining the beginning and ending of the character string in the BFINIM array. This is packed as XXYY. Packing is carried out by use of the macro CHRLIMIN:. The word is then scaled by -1.E+20 to make it identifiable from actual floating-point numbers. XX is the column number of the first character in the string and YY is the column number of the blank, comma, or equal delimiter character

that is one greater than the column of the last character (see 4.3.2). Note that (YY-XX) is the length of the character string. The macro CHRLIMIN: is responsible for packing up XX and YY and the macro CHRLIMOUT: does the unpacking of XX and YY. Note that the use of packed pointers places an additional upper bound on the magnitude of the negative numbers which may be read in via macro DCODEFLD:.

4.3.2 String Delimiter Character Equal (=)

Sometimes it is necessary to input fields which have embedded blanks or that start with the characters 0123456789.+-. The = is reserved to delimit such input character strings. Thus =23-APRIL 1974:JS= will be stored as a 16 character string and the corresponding input buffer set to the appropriate large negative number which points to the first and last character plus one in the input image. The equals signs are stripped on input; they are not part of the input string. An equal sign is therefore a reserved character which cannot be input.

4.3.3 Field Delimiter Characters Blank ( ) and Comma(,)

The general field delimiter is the blank or the comma. Either of these characters will terminate an alphanumeric string except when this string is bounded by equals (=) characters (see above). Strings of delimiter characters are interpreted differently. A blank character is treated as a field delimiter only if it is immediately preceded by character other than blank, comma, and equals. A comma is always a field delimiter independent of what character it precedes (except within equals). A series of N commas may be therefore used to delimit N fields and can be used instead of the skip parameter (see 4.3.4). The combination blank-comma after an input field is equivalent to comma-comma.

4.3.4 Field Skip Character ($)

A character string starting with a dollar ($) followed by a number, is interpreted as a field skip signal. The string $NN in the input image will skip the current field and NN-1 additional fields. That is, the field position pointer to words in the floating point buffer BFINFP is incremented by NN. The field skip string is not stored in the input buffer, BFINFP.

4.3.5 Field Position Character (*)

The character string starting with an asterisk (*) followed by a number is interpreted as a field position signal. The string *NN will position the field position

pointer at NN so that data immediately following will be placed in this field of the floating-point buffer BFINFP. For example, *10 places the next character string encountered in field 10 of BFINFP. Note that field position numbers do not have to be used in ascending order. Field position strings are not stored in BFINFP. Further note that once set, subsequent fields are stored in NN+1, NN+2, etc. unless other *NN or $MM directives are given.

4.3.6 Image Delimiter Character (:)

The character colon (:) is used to denote the end of field scan (except when enclosed in equals signs). This allows additional information which is not processed to be placed on the input image. For example, the image containing:

```
ATOM C12 1 .5 .6 .9 3.5 : NEW SITE AT -X,-Y,Z
```

is only scanned through the field containing 3.5.

4.3.7 Blank (or Void) Fields

All words in the input floating-point buffer BFINFP are initially set to a large negative number defined by the macro VOIDFLG: as -4.E+20. All fields input on BFINFP which contain nonblank information will overwrite the appropriate field word in BFINFP. In this way, fields containing 0.0 are distinguished from blank fields. The value of VOIDFLG: also represents a boundary between the pointers to packed character strings (of the form -XXYY.E+20) discussed earlier, and floating-point numerical data. The magnitude of an input negative number must not exceed 1.E+20. This is an important feature, since it allows the programmer to be able to tell the difference between void fields and zeroed fields, or even to know whether numbers or character strings have been encountered in a given string.

4.3.8 REAL Strings, Numbers

As noted above, any field, not bounded by equals signs, which starts with one of the characters 0123456789.+- is a numerical field. Furthermore, once the scan begins, no other characters but these may be placed in the field. No more that one (.) is allowed and no more than two signs. The first sign relates to the number itself, the second defines the start of the exponent field, if any. Consider for example the following fields:

```
3  3.0  30.-1  +3  +3.  +.3+1  +30.-1  30-1  .03+2
```

All of these numbers are syntactically correct representations of the positive number three. Moreover,

since the type integer is subsumed into real, it does not matter where in the field the 3 appears. Trailing blanks are not treated as zeros. The range of numbers from -1.+24 to -1.+20 are used as a void field signal and for character storing purposes and are treated as out-of-bounds. Negative numbers greater than -1.+20 are stored as written. If this limit is exceeded, an error message results and the largest magnitude that the particular machine will hold is stored in the floating-point buffer.

4.4 Line Input Devices

Line input information is read by the macro READLINE: via the macro LINEIN:. Only the device number which resides in IOIN1: is actually used to access data. The other variable IOIN2: is for special input (eg input of HKL data from an alternate character file). In these cases IOIN2: is substituted for IOIN1: in the requesting program and READLINE: continues to input images on this device until an end-of-file is detected. READLINE: then automatically reverses the device numbers and inputs images from IOIN1: again.

LINEIN: is a very machine-specific macro and requires reference to Appendix 2 for details.

4.5 Formatted Input - Use of the FIELD Command

If the use of free-format is unacceptable for any reason, one may use a 'FIELD' image in the input stream. This event signals the image processing macro, DCODEFLD:, that specified columns will now serve to terminate fields instead of blank, comma, or equal. The 'FIELD' image is read by macro READLINE: and numbers which are the ending columns of the fields are stored for use by DCODEFLD:. The image:

```
FIELD 10 20 37 45
```

implies that the first field is in columns 1 to 10, the second in 11-20, the third 21-37, the fourth to be 38-45, and the last to be the rest of the input line. A FIELD image containing following blanks will signal the macro DCODEFLD: to revert to free-format input as described above.

4.6 Ordering Input Fields - Use of the ORDER Command

In addition to the FIELD command, there is also an ORDER command. This is initiated by entering an 'ORDER' image in the input stream. This permits one to direct the interchanging of input fields. Thus ORDER 5 6 1 3 2 will result in the fields on input images following the 'ORDER' image to be processed as reordered data. The data in the first field on the input image is directed to word 5 of

BFINFP, data in field 2 to 6 , 3 to 1, 4 to 3, and 5 to 2. The rest go as usual. When this command is given incorrectly, grievous harm may ensue. The normal order may be restored by an input line ORDER followed by blanks.

4.7 System Input Images

The nucleus routine AA01 which is invoked by the macro READLINE: screens for special input lines. These lines supply the control parameters of the XTAL system and give the user the chance to direct input-output and other system functions.

The eight controls are:

(1)	TITLE	page heading
(2)	REMARK	listing identification
(3)	FILES	input-output file control
(4)	MEMSET	memory allocation control
(5)	SETID	"stranger" data input control
(6)	FIELD	input data format control
(7)	ORDER	input data order control
(8)	FINISH	end of run signal

These lines are read free-format as described in section 4. A description of the data read in each input line follows.

4.7.1 TITLE (Optional)

This image contains a character string to be put out at the top of each printed page. It is stored in array BFTITL of /SYSCH/. when binary data files are being created, the current TITLE image is stored in the BDF as a packet of logical record 2.

Field	Contents	Default
1	Character string, starting in column 7.	Blank

4.7.2 REMARK (Optional)

This image contains a character string to be put out directly on line output units 1 and/or 2, and is used to force "once-only" remarks.

Field	Contents	Default
1	Character string, starting in column 8.	Blank

4.7.3 FILES (Optional)

This image permits the user to specify device numbers for input/output units, and to change the priority limits and line-length for line output units 1 and 2 (IOOT1: and IOOT2:). Unspecified fields remain at the previously specified value (i.e., Value Remains Unchanged, VRU).

Field	Contents	Default
1	Device number for FILE A	VRU
2	Device number for FILE B	VRU
3	Device number for FILE C	VRU
4	Device number for FILE D	VRU
5	Device number for FILE E	VRU
6	Device number for FILE F	VRU
7	Device number for FILE G	VRU
8	Device number for FILE H	VRU
9	Device number for line input unit 1	VRU
10	Device number for line input unit 2	VRU
11	Device number for line output unit 1	VRU
12	Device number for line output unit 2	VRU
13	Device number for line ouptut unit 3	VRU
14	Priority limit for line output unit 1	VRU
15	Priority limit for line output unit 2	VRU
16	Max. length (characters) for line output units 1 and 2	VRU

4.7.4 MEMSET (Optional)

This image is to specify either the amount of usable QX array to be made available (i.e. beyond QXSTAR), or to request the total number of words required for nucleus, overlays, and QX array. The MEMSET request remains in effect until a blank MEMSET image is encountered and overrides all automatic memory requests.

Field	Contents	Default
1	Memory request for a definitive number of floating-point words. Positive value specifies request	Blank resets MEMSET to

	for usable QX array only. Negative value specifies request for total memory size.	"program control"

4.7.5 SETID (Optional)

This image permits the user to force the image reading macro READLINE: to treat images that follow as having a specific name. This command is used when reading stranger images. A blank SETID image switches off this condition.

Field	Contents	Default
1	One to six character mnemonic to be assumed as the ID name of all non-system images that follow this image.	Blank resets SETID Condition.

4.7.6 FIELD (Optional)

This image is used to specify a fixed-input format for images that follow. The character (or column) values input define the low-order (right-justified) character position. The high-order (left-justified) position is the previous field value plus one. Note that the left-justified position of field 1 is 1, and care must be taken if an ID name is present (see ORDER image below). A blank FIELD image switches off the fixed format condition. Cautionary note: all fields expected as input must be specified on the FIELD image.

Field	Contents	Default
1	Character postion (right-justified)	Error
2	As above for field 2.	Error
3	As above for field 3.	Error
.	. . . . .	
N	As above for field N. (N input data are expected).	Error

4.7.7 ORDER (Optional)

This image specifies the order in which fields, encountered on following images, will be interpreted. This enables fields in either fixed-format (using FIELD) mode, or the usual free-format, to be ordered according to the expectations of the program into which they are being read. This facility is particularly useful for inputting stranger images for which certain data must be reordered or ignored.

A blank ORDER image switches off this condition.

Field	Contents	Default
1	Word number of the input F.P. buffer that the first field encountered is to be placed.	1
2	ditto for second field encountered	2
.	. . . .	.
.	. . . .	.
N	ditto for the Nth field encountered	N

4.7.8 FINISH (Mandatory for overlay mode)

This image signifies the end of all calculations in the overlay operational mode, and it ensures a correct exit from the base overlay. The FINISH image has no additional fields.

4.8 COMPCHAR: Checking for a Character-String Match

The macro COMPCHAR: is used in programs to test for matches between character strings. The handling of character strings is described in section 2.4. COMPCHAR: utilizes subroutine AA06 which depends upon the machine-specific macro COMPCHR:. COMPCHAR: is invoked by:

	ARG
COMPCHAR:(<character array to be searched for location of defined strings>,	1
<index to first character in array to be searched>,	2
<character array containing the table of defined strings>,	3
<index to first character in defined strings table>,	4
<length of every defined string>,	5
<increment of defined strings table index between successive searches>,	6
<total number of characters in the defined strings array>,	7
<key>)	8

key = 0 means no match was found

key = N where N is the index showing which defined strings were found in the searched array

This is a very powerful string comparing routine. One example of its use may be taken from AA02, the subroutine of READLINE: which does the free format decode of input lines.

In that routine a string search is made by:

```
DATASTORE:(A021, 0123456789.-+$,*$:=$$ )
. . . . . .
DATASTUFF:(A021)
```

The digraphs $, $: and $$ are used to override the usual RATMAC interpretation of comma, colon, and dollar signs (2).

In the loop over all of the input columns, the macro is invoked by:

```
COMPCHAR:(BFINIM,N,A021,1,1,1,19,KEY)
```

where BFINIM is the character array in COMMON /SYSCH/ which holds the input line image. N is the index to the column being investigated, A021 is the character array of allowed characters, the first 1 points to the first byte of A021, the second 1 indicates that 1 byte strings are to be compared, the third 1 gives the step to be taken on each successive compare to bytes in A021, the 19 gives the number of characters in A021. If the Nth character of BFINIM is a blank, KEY will be given the value 1; if it is a 3, KEY will be given the value 5, if $ 19, etc.

NOTE that 19 could have been designated by NA021 from DATASTUFF:.

The character strings are numbered starting from 1 at the left end of array A021.

5. LINE OUTPUT CONTROL

As with input, the XTAL system does not use the FORTRAN line output control subroutines. The whole process of handling line output in XTAL is much more powerful than FORTRAN formatted output. Furthermore, it requires less computer storage and there are no fatal errors associated with its use.

The two FORTRAN procedures associated with FORMAT and WRITE statements have been replaced with the three macro functions:

(1) NCODEFLD: translates real numbers into output character strings and takes care of I, E, and F format conversions.

(2) MOVEBYTE: moves character strings and takes care of A and H format conversions.

(3) WRITLINE: causes the actual writing of the line including the possibility of extra blank lines, page titles, subtitles, and the lines encoded by 1 and 2 above.

At first, the macros described here will seem strange for those familiar with the use of standard FORTRAN statements. However, we feel the effort required to learn the new technique will be repaid in clean output. During check out, it is sometimes expedient to use PRINT or WRITE statements for dumping purposes. These can be removed when the program is ready to be used in production runs. The macro DEBUG: described in the RATMAC Primer TR-804 is an elegant way of doing this.

The concept of output fields is identical to that already discussed for input fields, except that now data is encoded from a floating-point output buffer (usually BFOTFP in COMMON /SYS/) to a character output buffer (usually BFOTLN, in COMMON /SYSCH/). The size of the floating-point output buffer is set by the macro MXFDLN: (usually to 50) and the maximum length of the character buffer is set by the macro MXCHLN:. It should be noted, however, that the length of the output line is an input parameter, LINCHR (field 16 on the FILES image), and may be varied to suit the output unit or calculation.

5.1 NCODEFLD: Formatting

All numerical data is translated from floating-point numbers to character strings using the nucleus routine NCODEFLD:. Because NCODEFLD: only encodes reals, integer numbers must be made real before invoking it.

```
                                                              ARG
NCODEFLD:(<array of reals to be encoded>,                      1
          <index to first element of array to be encoded>,2
          <array of characters into which the encoded          3
           numbers are to be placed>,
          <array of reals which define the format control      4
           words to be used in the encoding process>,
          <number of items to be encoded>)                     5
```

The array of reals which define the format control words consists of packed words of the form CCCLLDT. They are used to specify the output format of each number.

CCC is the right-justified column position in the output character array.

LL is the total width of the numeric character string (including signs, decimal points, and exponent).

D is the number of digits after the decimal point in E and F type formats, or the number of forced digits in I type format.

T is the format type.

NCODEFLD: treats nine types of output format types:

T = 1 specifies E-format of the general form S.YYYYSZZ where S is the sign character (-, +, or blank), YYYY are the most significant digits after the decimal point (in this case D = 4), and ZZ is a two digit integer exponent. The absolute minimum LL of an E-format is 4. For LL less than 4, an overflow is indicated by a field (LL long) of asterisks.

T = 2 specifies F-format of the general form SXXX.YY where S is the sign character (- or blank), XXX is the component of the number greater than one. The present length of YY, the fractional part, is defined by D in the format word. The value of LL must allow for a minimum width of D + 1 for positive or D + 2 for negative numbers An overflow of LL automatically changes T to 1 and rules for the E-format then apply.

T = 3 specifies I-format (base 10) of the general form SXXXX, where S is the sign character (- or blank) and XXXX is the rounded-off integer value. The value of LL must allow for the width of the maximum integer if positive and the maximum integer + 1 column for the sign if negative. otherwise NCODEFLD: automatically changes T to 1 and rules for E-format then apply. The value of D in the format specifies the minimum number of digits to be output. This

may be used either to output a zero integer as blank (when D = 0), or as 0 (when D = 1), or force leading zeros. For example, the integer nine will be output as 09 if D = 2.

T = 4 specifies I-format (base 2) of the form SXXXXX, where X is either the digit 0 or 1. All rules are identical to T = 3.

T = 5 specifies I-format (base 8) of the form SXXXXX, where X is a digit = 0,1,2,...,7. Useful for memory addresses (e.g., in CDC).

T = 6 specifies I-format (base 36) of the form SXXXXX where X is a digit - 0,1,2,...,Y,Z. Useful for population plots.

T = 7 specifies unpacked binary digits of the general form XXXXX. A maximum LL value of 32 must be observed, since use of greater than 32-bit words is not permitted in the XTAL system. (0 or 1) are unpacked from the floating-point word. Unlike type 4, in which the actual integer or real value of the number is output in base 2, type 7 causes the actual bit pattern starting from bit-position 0 (right-justified) to bit (LL-1) to be output. Only LL bits are unpacked.

T = 8 specifies unpacked hexadecimal digits of the general form XXXXX. The values of X (0,1,2,...,F) are unpacked as 4 bit bytes from bit-position 0 (right-justified) to bit-position (LL-4). Only LL-4 bit bytes are unpacked.

It should be noted that because the macro NCODEFLD: is capable of encoding these words as either binary, octal, or hexadecimal character strings for output by WRITLINE:, the programmer may display packed information in an easily-readable form during program development.

5.1.1 Formulation of NCODEFLD: Format Arrays

The method of preparing the array of reals to be used as the format control words can be carried out by either forming the control words directly or by invoking the FMT: macro. In either case, a REAL array is specified and then followed by a DATA statement which loads the format control words into the array. For example, the array of format control words corresponding to the conventional FORTRAN FORMAT statement:

```
100 FORMAT(10X,I5,2F12.6,E15.7)
```

would be coded in XTAL as:

```
REAL IFMT(4)   #
```

```
DATA IFMT/150513.,271262.,391262.,541571./  #
```

or alternatively, if the FMT: macro is used:

```
DATA IFMT/FMT:(15,I5.1),FMT:(27,F12.6),      #
          FMT:(39,F12.6),FMT:(54,E15.7)/     #
```

The one after the decimal point in the I5.1 indicates the minimum number of integer figures to print. This provides for the printing of leading zeros.

Later in the program, where one would write in FORTRAN:

```
WRITE(6,100)(BFOTFP(J),J=1,4) #
```

the following two statements would appear:

```
NCODEFLD:(BFOTFP,1,BFOTLN,IFMT,4) #

WRITLINE:(0,0,0,1,3) #
```

The use of FORMAT words has been simplified for the most common format types, T = 1, 2, or 3, by making available the macro FMT:(<CCC>,<T><LL>.<D>)

NOTE WELL: the macro FMT: is a complicated one, so it has high overheads during preprocessing.

5.1.2 DATASTORE:, DATASTUFF:, and MOVEBYTE:

The use of the DATASTORE: and DATASTUFF: macros permits setting up character strings for output. The macro MOVEBYTE: may then be invoked to move these strings to the line output buffer in /SYSCH/, BFOTLN. The macro is invoked by:

```
                                                          ARG
MOVEBYTE:(<array which is the source of character(s)>,     1
          <integer which indexes character in source       2
           array>,
          <array which is the destination for              3
           character(s)>,
          <integer which indexes character in destination  4
           array>,
          <integer number of characters to go into         5
           destination array>,
          <key>)                                           6
```

key = 0 means move character for character from source to destination

key = 1 means move only the first character of the source to every character of the destination array

If the destination array is BFOTLN and an array set up by DATASTORE: is specified as the source array, then MOVEBYTE: will load BFOTLN for printing. An example follows:

```
DATASTORE:(XX001, NOW IS THE TIME FOR)    #
DATASTUFF:(XX001)     #

MOVEBYTE:(XX001,1,BFOTLN,1,NX001,0)       #
WRITLINE:(0,0,0,1,3)  #
```

5.1.3 NCODEFLD: Placing Data in DATASTORE: Arrays

The XTAL method of carrying out the FORTRAN statements:

```
100  FORMAT(5HOX = ,F10.5)
     PRINT 100, X
```

would be:

```
DATASTORE:(XX002, X =              )   #
```

The blanks will later be overstored by use of NCODEFLD:

```
DATASTUFF:(XX002)   #
NCODEFLD:(X,1,XX002,151052.,1)    #
WRITLINE:(1,XX002,NX002,3,3)      #
```

The details of the arguments of WRITLINE: are given below. The purpose here is to show the use of NCODEFLD: in placing coded numbers into an array to be used for output.

The total possibilities are very great and each programmer will develop various ways of taking advantage of the separation of these functions into the various macro procedures.

5.2 WRITLINE: Output of Character Strings

This macro is the standard line output function of XTAL. It provides pagination and titling of all output. The output may be flexibly controlled by a key described below. The output lines are taken from one explicit and two implicit sources:

(1) An explicit character array specified in the calling sequence. These may be messages or column headings.

(2) The character array BFTITL which contains the image of the current page title.

(3) The character array BFOTLN which contains the image of the current output line.

For simple line output with no headings, the characters may be placed in either the explicit array or in BFOTLN. The choice is a matter of convience to the programmer. When headings are involved, the heading is placed in the explicit array and the numerical output is placed in BFOTLN. There is a control which may be used to assure that two lines are not

split between two pages. Every page will have a title which is a distinct line that appears above the heading. Every line that is printed has an associated priority which allows the user to control the extent of output. All lines may be directed to two different output devices. The priorities and printers are under control user at execution time. This is important for control of output in an interactive computing environment.

Character string information may be output directly by WRITLINE:

```
                                                          ARG
WRITLINE:(<number of blank lines before actual print>,     1
          <explicit character array to be printed>,        2
          <number of characers in the explicit array>,     3
          <key>,                                           4
          <printing priority for the line>)                5
```

<number of blank lines before actual print> allows for single, double, or greater spacing. It simulates FORMAT(1H0) etc. The character array to be printed is most often a string defined by the use of the DATASTORE: macro. If required, numerical data can also be inserted into this array using NCODEFLD: as is shown in section 5.1.3. Any other character string may be written out; for instance, the input image in buffer BFINIM can be directly using WRITLINE:. In these cases, the use of an auxiliary buffer (i.e. other than BFOTLN) for output is treated as a header line and the appropriate key must be set as argument <key> in WRITLINE:. These keys are discussed below.

Character strings may be output by moving them into the standard output buffer BFOTLN with the character-mover routine, MOVEBYTE:, or by the numeric encoder, NCODEFLD:. In this way, alphabetic and numeric data may be arranged according to the output format desired.

The use of BFOTLN as the output buffer is implicit and under the control of <key>. The use of <explicit character array to be printed> simultaneously with BFOTLN is a powerful means of creating headings.

<number of characters in the explicit array> specifies the length of the array. If this integer is zero, blanks will be put out depending upon the value of <key>.

5.2.1 WRITLINE: Keys

<key> gives the programmer control over the way WRITLINE: puts out the special character array and the line output buffer, BFOTLN, of COMMON/SYSCH/. The use of this key in conjunction with the COMMON/SYS/ variables LINRM and LINCT provides a powerful tool for the programmer. Typically <explicit character array to be printed> will contain a subtitle, a column heading, or text. WRITLINE: counts lines as they are printed (LINCT) and, if the number of lines

remaining on a page is less than LINRM, a new page will be ejected before the <explicit character array> is printed.

<key>=1 BFOTLN will always be output, but may be preceded by the <explicit character array>.
The <explicit character array> will be put out on first call and at the top of every subsequent page. This feature is designed to make subheadings for output lists. Setting LINRM to the number of lines in the array will assure that the subheading is not split between two pages.

<key>=2 BFOTLN will be output, and the <explicit character array> will be written on every call. Use of LINRM assures that both parts appear on the same page.

<key>=3 BFOTLN is not output. This option used for multiple line heading or printing messages (text). Blank lines may be generated by setting <the number of characters in the explicit array> to zero and <number of blank lines before actual print> appropriately.

5.2.2 WRITLINE: Priorities and Devices

<printing priority for the line (PR)>

The integer argument PR is used to signal WRITLINE: with respect to the importance of the output. The following table describes the priority levels available:

PR=1 for essential messages and data

PR=2 for abbreviated messages and data appropriate in interactive modes of operation.

PR=3 for standard messages and data usually on a line printer.

PR=4 for expanded data and messages, Fourier maps, structure factor lists, etc.

PR=5 for diagnostic data and messages, dumps, error traces. Also echoes input images from READLINE:.

Reference to the macro call shown at the beginning of this section shows that it has five explicit arguments. In addition, there are seven implicit arguments, BFOTLN (BuFfer for OuTput LiNe), IOOT1:, IOOT2:, IOT1P:, IOT2P:, LINRM, and LINCT in SYSCOM:. These implicit arguments are the standard output line buffer, (BFOTLN), the standard line output file priority, (IOOT2:), the current standard line output file priority, (IOTP1:), the alternate file priority, (IOT2P:), the number of lines which must be available on a page if printing is to occur before page restoration, (LINRM), and the current value of the line count.

5.2.3 WRITLINE: Output Units

WRITLINE: will write on two different output units simultaneously. This function is controlled by the use of the FILES control line during execution (see section 4.6). This control is dependent upon the unit designations IOOT1: and IOOT2:. If these units are assigned the same "logical unit number" then there is just one output stream, IOOT1:, and printing will be at the priority specified by PR for each WRITLINE: call.

On the other hand, if two different units are specified for IOOT1: and IOOT2: each will be written only if the output corresponding to priority variables IOT1P: and IOT2P: are less than or equal to the value of PR. This function allows a user at a VDU or teletype to "spool" low-priority output to a file while observing high-priority output at the terminal.

5.3 Examples of Use of WRITLINE:

There are four common ways that a programmer will want to use WRITLINE:; first, to write a message, second to write encoded data, third to write a message containing encoded data, and fourth to write headed data.

5.3.1 WRITLINE: Used to Produce a Message

```
DATASTORE:(X01, NOW IS THE TIME)   #
. . . . . .
DATASTUFF:(X01)   #
. . . . . .
WRITLINE:(0,X01,NX01,3,3)   #
```

5.3.2 WRITLINE: Used to Write Encoded Data

In this example, the floating-point variable X is written out as 1X,F12.4.

```
NCODEFLD:(X,1,BFOTLN,131242.,1)
WRITLINE:(0,0,0,1,3)
```

5.3.3 WRITLINE: Used to Write a Message with Data

In this exapmle, the value of the floating point variable,X, is written out as F12.4 at the end of a defined message "VALUE OF X IS".

```
DATASTORE:(X03, VALUE OF X IS              )   #
. . . . . .
DATASTUFF:(X03)   #
. . . . . .
NCODEFLD:(X,1,X03,271242.,1)   #
WRITLINE:(0,X03,NX03,3,3)   #
```

5.3.4 WRITLINE: Used to Write Headed Data

Consider a program in which it is desired to list h, k, l, sin theta over lambda, and intensity in a loop. The output is to be headed on the first write and at the top of every page. The skeletal outline of this program using the facilities described above could be as follows:

```
SYSTEMHEADER:(XX00)                    #
. . . . . .    preliminary program material
. . . . . .
REAL FMT(5)                            #
DATASTORE:(X001,   H   K   L  SIN(T)/L   INTENSITY)   #
. . . . . .
. . . . . .    all other variable declarations
. . . . . .
DATASTUFF:(X001)                       #
DATA FMT/40413.,80413.,120413.,220852.,341131./    #
. . . . . .
. . . . . .    preliminary program
. . . . . .
LINRM = 3      # set up to head output within the loop
               # head columns and start printing
               # unless there are less than three
               # lines left on the current page
. . . . . .
FOR (J=1; J.LE.NREF; J=J+1)
$(  # loop over reflections
. . . . . .
. . . . . .
BFOTFP(1) = FLOAT(H)                   #
BFOTFP(2) = FLOAT(K)                   #
BFOTFP(3) = FLOAT(L)                   #
BFOTFP(4) = STOL                       #
BFOTFP(5) = TENSIT                     #
NCODEFLD:(BFOTFP,1,BFOTLN,FMT,5)       #
WRITLINE:(0,X001,NX001,1,5)            #
. . . . . .
. . . . . .
$)                                     #
. . . . . .                            #
END                                    #
```

6. BINARY FILE INPUT/OUTPUT CONTROL

One of the central features of the XTAL system is the use of binary data files for communication between major crystallographic programs. In this section, the routines which are used for the reading, writing, and copying of these files are presented. The structure and contents of a binary data file are set out separately in Appendix 5.

We now summarize the macros which support binary I/O activity; the conventions to be used for various kinds of binary files; the primitive macros which support the frequently used macros; and binary input output conventions.

Some sketchy examples are given here, but the main example is to be found in the EX00, EX01, and EX02 codes on the XTAL distribution tape.

Four types of binary files are used in the XTAL system. Three of these types are sequential in nature and the fourth is a random-access file. Two of the three sequential files are data files, and the third is a scratch file. The random-access file is a scratch file.

6.1 Binary Data Files

Two of the sequential file types are called binary data files..The first of these is T*H*E B*I*N*A*R*Y D*A*T*A F*I*L*E (BDF) described in Appendix 5. The second type is structured in a similar vein and will be referred to as an auxiliary BDF. The concept of a logical record and a packet are spelled out in detail in section 6.3.

The structure of these types of binary files is carefully prescribed and documented in order to allow communication of non-transitory data from one program to another (e.g., for reflection information from the diffractometer tape to the reflection processing routines). On these files logical records of type 1 and type ENDRECORD: are reserved for special uses. Record 1 is a label record with the last name of the writing program and the date written into it. The last record defined by the macro ENDRECORD: is the end-of-file record and serves to signal macros WRITEPKT: (the packet writer) and READWPKT: (the packet reader-copier) that files are to be terminated and pointers reset to the beginning of the file.

The use of these routines may be learned by studying the code EX00 supplied on the XTAL distribution tape.

6.2 Scratch Binary Files

From the point of view of the programmmer adding working programs to the XTAL system, only the six MACROS listed above are usually utilized. In the following section is given the background material to the primitives which support BDF activity. These primitives are used when large scratch files are needed to supplement immediate access memory.

6.2.1 Binary Sequential Files

These files are written by the use of the macros:

BINSEQOPEN:(<integer which designates file>)

BINSEQREW:(<integer which designates file>)

```
                                                          ARG
BINSEQWRIT:(<integer which designates file>,               1
            <QX array area to be put out in FORTRAN as:    2
             (QX(I),I=<starting index>,<ending index>)>,
            <buffer length in real words>,                 3
            <relative word address on mass storage>,       4
            <QX(<starting index>)>,                        5
            <integer error flag; read OK if zero>)         6

                                                          ARG
BINSEQREAD:(<integer which designates file>,               1
            <QX array area to be input in FORTRAN as:      2
             (QX(1),I=<starting index>,<ending index>)>,
            <buffer length in real words>,                 3
            <rel word address on mass storage>,            4
            <QX(<starting index>)>                         5
            <integer error flag; write OK if zero>)        6
```

BINSEQEOF:(<integer which designates file>)

The use of these macros is illustrated in AA21, AA22, AA25, and AA26. Further comments are given in Appendix 2. It is intended that these macros be used only for data storage and retrieval of a transitory nature. This type of scratch file is used for extending memory during a calculation and is not to be used for inter-program communication because of the uncontrolled file structure and buffer lengths.

There are very rigid restrictions on the use of these reads and writes. The length of the buffer must be limited to a single value specified by BINSEQBUF: since the macro primitives expect only fixed length records to be written. Careful study of the techniques involved must be made by the programmer.

6.2.2 Random-access Files

These files are characterized by directory tables that point to specific word strings on mass storage which are to be written and read randomly. The directory tables are kept in the program which uses this type of file.

As of February 1980, these techniques are ill-defined and will require more study to specify fully.

6.3 Sequential Binary File Input/Output Control

The structure of the XTAL binary data file is given in Appendix 5. Sequential files which do not have strict BDF format will nevertheless be structured much like a BDF. These, for example, will be files with raw diffractometer data or data of the output of a Fourier Transform to be input to a search routine. That is, the basis for management of this type of file is through the concept of "logical records" and packets".

6.3.1 Logical Records

A logical record is a set of data which the programmer defines as belonging together for calculational purposes.

6.3.2 Packets

A packet is a subset of a logical record which repeats a pattern of the same quantities attached to different indices. In crystallography, several examples of the use of logical records and packets spring to mind. One would be a logical record containing atomic parameters. In this case, each packet contains the data for a single atom. The packet may contain x, y, z, population parameter, U and an atom designator. The size of the packet is six; with anisotropic temperature factors, it is eleven.

6.3.3 Binary Control Macros

The macros WRITEPKT:, READWPKT:, POINTPKT:, COPYFILE:,BUFGET:, and BUFPUT: support activities of reading and writing logical records and files.

WRITEPKT: The packet writer	ARG
WRITEPKT:(<integer which designates output file>,	1
<integer which designates logical record>,	2
<integer which specifies packet size>,	3
<integer returned as index into QX array where output packet is to be stored>)	4
READWPKT: The packet reader-copier	ARG
READWPKT:(<integer which designates input file>,	1

```
          <integer which designates logical record>,      2
          <integer returned which specifies packet size>, 3
          <integer returned as index into QX array where  4
           input-output packet is stored; zero if logical
           record drained>,
          <integer which designates output file; zero     5
           implies read only function; non-zero implies
           read-write>)

POINTPKT:  The packet address builder                     ARG

POINTPKT:(<integer which designates input file>,          1
          <integer which designates logical record>,      2
          <integer returned which specifies packet size>, 3
          <integer returned which as index into QX        4
           array where input-output packet is stored>,
          <integer which designates output file as in     5
           READWPKT:>,
          <integer which designates number of items       6
           sought>,
          <integer array with numbers of file indices     7
           from Appendix 5 stored>,
          <integer array for return of relative pointers  8
           to sought quantities>)

COPYFILE:  The logical record copier                      ARG

COPYFILE:(<integer which designates input file>,          1
          <integer which designates output file>,         2
          <integer which designates first logical record  3
           to be copied>,
          <integer which designates last logical record   4
           to be copied>)

  BUFGET:  The local REAL variable from QX buffer mover   ARG

  BUFGET:(<integer index into QX array>,                  1
          <real variable to be stored locally>)           2

  BUFPUT:  The local real variable to QX buffer mover     ARG

  BUFPUT:(<real local variable to be placed in QX         1
           buffer>,
          <integer index into QX array>)                  2
```

These last two macros can have wider use for moving variables into and out of the QX array.

```
ENDRECORD:  The BDF end-of-file signal is described under
            point 6 below.
```

From a programmer's point of view, nine activities must be controlled:

(1) The choice of files to be written (created), read and/or copied. These files are designated by

integers 1, 2, 3, ..., MXIOUN: which correspond to FILE A, B, C, ...

(2) The buffer region in the QX data array which will be set aside as a buffer region for the specified file. The value of the location just before the start of the buffer in the QX array must be stored in the COMMON/SYS/ integer array, IOMARK(N), where N is the index of the file. See the dynamical storage section for QX array protocols. The length of the buffer is always BINSEQBUF: and this amount of space must be utilized in a "working" program.

(3) The actual binary device number (often called a FORTRAN logical unit number) is stored in IOUNIT(N) of /SYS/. The programmer may wish to interchange these for switching units after copying a file. See READWPKT: for an example of this procedure. NOTE: When READWPKT: is used in the read-copy mode, it always interchanges files at the end of the copy.

(4) FILE A, IOUNIT(1) and FILE B, IOUNIT(2) are usual units for the binary data file.

(5) Files are created by the use of the binary file writer, WRITEPKT:. An example of the use of this subroutine is:

```
WRITEPKT:(N,LREC,PACK,IP) #
```

where all of the arguments are of type INTEGER, N is the output file designator, LREC is the logical record designator, PACK is the size of the packets of LREC (never zero), and IP is a pointer which gives the location in the QX array where the packet is to be stored. That is:

```
QX(IP+1)=<First floating-point word of packet> #

QX(IP+2)=<Second floating-point word of packet> #

 . . . .     .     .     .     .     .     .

 . . . .     .     .     .     .     .     .

QX(IP+PACK)=<Last floating-point word of packet> #
```

The programmer should note well that pointers into the QX array are always one less than the actual QX element of the first item. This is done for consistancy and to agree with the BDF conventions described below.

The logic of WRITEPKT: is such that no actual writing takes place until the physical buffer is exhausted. With each call, IP ranges back and forth

over the region of the QX array specified by IOMARK(N) to IOMARK(N) + BINSEQBUF: - 1. Logical records should be written in the order 2 to ENDRECORD: -1. Not every record need be on the file if the data are not required for subsequent calculations. An ENDRECORD: must always be written to the file. In the calling sequence of WRITEPKT: the value of the variable PACK is supplied to the program to set the size of packets written to the file. This quantity must remain constant for all packets of a given logical record.

(6) Logical records 1 and ENDRECORD: are special. The first must be formed with great discretion when WRITEPKT: and the reader-copier READWPKT: are used. At the time a BDF record is created, 50 packets of length the-number-of-words-to-hold-16-characters must be written to logical record 1, filled with blanks. The EX01 program shows this procedure. The ENDRECORD: must always be called when WRITEPKT: is used or when READWPKT: is used in the read-copy mode. This signals that the file activity is completed. When READWPKT: is used in the read-only mode, IOLRPT(N), which indicates the logical record type being handled, may be set to zero to signal that the use of file N is over, thus avoiding reading all the way to ENDRECORD:.

(7) READWPKT: is used to read, or read and copy files. An example of the usual way this subroutine is used might be:

```
READWPKT:(N,LREC,PACK,IP,NO) #
```

where N is the input file designator, LREC is the logical record sought, PACK is the size of the packet found on the file for the specified logical record, IP is the pointer to the QX array where the packet is to be found, and NO is the output file designator. All of the arguments are of type INTEGER. If NO is zero, the subroutine is in the read-only mode. This is a single-buffer subroutine which uses the buffer array specified by IOMARK(N) and means that in the read-copy mode, files may not be expanded or contracted. That is, PACK is fixed and not under programmer control. The pointer IP is returned as zero when the logical record is drained. In READWPKT: PACK is read out of the file N by the program. Writing, of course, is to file NO from the buffer of file N.

(8) When it is necessary to simultaneously read a file with one set of packet sizes, and write a file with a different set of packet sizes, WRITEPKT: and READWPKT: must be used together. Furthermore, two IOMARK( ) values must be set. Then pairs of calls

are made to WRITEPKT: and READWPKT: with appropriate copying from the input buffer to the output buffer. To facilitate the transfer of logical records for which PACK remains constant, the macro COPYFILE: may be used. For this program, an example is:

```
COPYFILE:(N,NO,LREC,LRECE) #
```

where N is the input file, NO is the output file, LREC is the first logical record to be copied and LRECE is the last logical record to be copied during the call. All of the arguments are of the type INTEGER.

(9) Macro POINTPKT:, the packet address builder, is used to locate specified quantities in the directory-type logical records of the binary data file. For directory-type logical records (see Appendix 5 for specifications) the first packet of each logical record contains identification numbers which uniquely define the contents of all subsequent packets in this record. This packet is referred to as the directory.

POINTPKT: is used to locate these identifications numbers and return their address (i.e. sequential word-position) within the packet. A typical calling sequence of POINTPKT: is

```
POINTPKT:(N,LREC,PACK,IP,NO,MAXWNT,IWANT,RELPT) #
```

where the first five integer arguments are those of READWPKT: And are passed through to that macro unaltered. MAXWNT is a count of the number of quantities wanted from the LREC specified while IWANT is an array which contains numbers specified as the "identification number" in the binary data file. This subroutine then searches the first packet for matches between the numbers in the IWANT array and the numbers in the first packet. When a match is found the RELPT integer array is set to a proper "offset" value to be applied to subsequent values of IP to fetch or store the desired quantities from the buffer. That is, IP + RELPT(N+1), will be the pointer to the item indicated by IWANT(N). The macros BUFPUT: and BUFGET: are used for this operation. The first element RELPT(1) serves as a signal. When it is zero, all sought quantities are in the file. When it is 1, one or more are missing. The missing ones have RELPT values of zero. The usual way of setting up a packet picking operation is to place a statement in the program which declares the type of these variables, such as:

```
     INTEGER IWANT(4),RELPT(5),RIP1,RIP2,RIP3,RIP4
     EQUIVALENCE(RELPT(2),RIP1              EQUIVALENCE(RELPT
                (RELPT(4),RIP3),RELPT(5),RIP4)
```

This would then be followed by a DATA statement:

```
     DATA IWANT,RELPT/1,7,11,21,5*0/   #
```

This implies that four quantities are wanted from the LREC specified. That is, those with identification numbers 1, 7, 11, and 21 are required. The process is started by a call:

```
     POINTPKT:(1,14,PACK,IP,2,4,IWANT,RELPT) #
```

which reads the table of contents packet and sets up the relative packet pointers.

After the call to POINTPKT:, it would be usual to have the sequence of instructions:

```
     IF(RELPT(1).NE.0) IQUIT:(XXYYZZ.)  #

     REPEAT               # start of repeat loop

     $(                   #

     READWPKT:(1,14,PACK,IP,2)   #

     IF (IP.LE.0)  BREAK         #

     BUFGET:(IP+RIP1,HKL)        #

     BUFGET:(IP+RIP2,PARAM2)     #

     BUFGET:(IP+RIP3,PARAM3)     #

         . . . . .               # calculations

         . . . . .

     BUFPUT:(PARAM4,IP+RIP4)     #

         . . . . .

     $)   # end of repeat loop
```

and if the calculation is completed:

```
COPYFILE:(1,2,15,ENDRECORD:) # files being
                         # expanded or contracted
```

or

```
READWPKT:(1,ENDRECORD:,PACK,IP,2) # files simply
                                  # being copied
```

would finish the copying of the files and close them out. See the example programs EX00, EX01, and EX02 for a more detailed use.

When in the single-buffer and read-write mode, READWPKT: will always interchange files 1 and 2 when the ENDRECORD: has been processed. If having the files interchanged is not desirable, the following sequence of instructions would restore them to their original values.

```
J = IOUNIT(1)          #

IOUNIT(1) = IOUNIT(2)  #

IOUNIT(2) = J          #
```

In summary, it is recommended that examples shown in the routines already coded be studied as illustrations of how the four macros are used to create, read, copy, and fetch specific quantities from sequential binary files. It is hoped that this process will be compact, fairly efficient, and convenient in the XTAL system. The routines EX00, EX01, and EX02 is the XTAL distribution tape show an example of the use of the macros described above.

Note that there are, as in the case of line input-output, few faults which will cause these I/O programs to stop. Many times the programmer will put checks which are really testing the validity of the code itself or of the machine or operating system. These checks are then made millions of times to no purpose. The programmer may need to add additional dumps during check out, but these should be removed from the checked production code. They take up unnecessary space and time during production execution. See RATMAC Primer (5) for the formation of a DEBUG: macro.

Please keep in mind that the purpose of this method of I/O is to give the programmer control over buffer regions, to eliminate large FORTRAN I/O libraries; to match buffer size to local mass storage specifications; and to make possible the readings of small packets of data in the logic of the program while actually minimizing physical I/O activity.

7. PROGRAM DOCUMENTATION

Program documentation is one of the most important components of software development. With an increasing number of structure analysts, who have little or no formal crystallographic training, using the XTAL system, programmers will be expected to put a particular effort into producing a clear, concise write-up. It is worth remembering that in commercial software organizations as much time goes into the production of supporting documentation as into the software development itself.

7.1 XTAL Handbook

To ensure clarity of presentation, each program will appear as a single chapter in the new XTAL handbook. A chapter will contain all relevant information to a given program and it will be the program author's responsibility to ensure the best possible description. In fact, the program write-up should be treated as a chapter in a book would be treated. It will represent a publication for the author which can be suitably referenced by users.

The standard layout required for each program chapter of the XTAL handbook is detailed below. Chapters will appear in the handbook alphabetically according to the calling name of the program (e.g., ABSORB, etc.), with the exception of the introduction to the use of the XTAL system and a description of the SYSTEM images (see above) which will form Chapter 1. RATMAC listings of the programs will not appear in the handbook, these will be distributed on magnetic tape. The appendices will also contain implementation instructions and listings of the input and output of two or three test decks. These test decks will be distributed with the XTAL programs and RATMAC preprocessor. An appendix will also describe in detail the method of communicating errors and fixes to program authors (see UPDATE PROCEDURE below).

7.1.1 Layout for Handbook Chapter

At the beginning of each chapter the following information should be supplied: a) program name - one to six letters; b) a brief title telling the function of the program; c) the author(s) name(s); d) the author(s) address(es); and e) the author(s) telephone and Telex numbers. The chapter should then be arranged according to the following format.

SYNOPSIS - Give a brief description of the type and scope of the calculations the program performs, highlighting the important and distinctive features.

INTRODUCTION or MATHEMATICAL BACKGROUND - Introduce the calculation with relevant references to previous publications and programs. References should appear in the text (in brackets). All important mathematical expressions used in the program must be carefully defined. All terms must be described carefully, particularly those for which the user must supply parameters or understand the output data.

OPERATIONAL PROCEDURE or ALGORITHM - Describe briefly how the program is partitioned and the major steps in the calculation procedure. A simple flow chart may be used but it must be supplied in a form suitable for direct offset printing. Reference the subroutine names wherever possible to assist the user in the event of an error.

PROGRAM AND HARDWARE RESTRICTIONS - Carefully list limits associated with input parameters, data size, and memory. Also if there are areas which may be machine-specific, mention them.

MEMORY AND FILE UTILIZATION - Give typical examples of the amount of direct-access memory required to execute a range of problems. If possible, give a parameter-to-memory-size equation. Specify which files will be input, output, and scratch. If the BDF is updated, indicate which data are inserted, modified or removed.

INPUT IMAGE FIELD DEFINITIONS - All input image types must be described concisely. The description should contain a capitalized heading; an indication of whether the line is optional; a brief description highlighting any particular requirements; and the actual field definitions listed as Fields, Contents, and Defaults. A typical example of such a layout is shown above for the SYSTEM images (sect. 2.6).

LINE OUTPUT CONTROL - The lines put out by a program are conditional on the priority limits OT1PR and OT2PR of the output devices IOOT1 and IOOT2. The priority value associated with each line of output (in the call WRITLINE:) must be listed to ensure that the output information needed is received.

7.2 Program Comments and Sequence Codes

As discussed in the preprocessor section above, each line of program code should contain a comment, starting with a crosshatch # in column 40 (if feasible) and ending on or before column 72. These comments should describe as succinctly as possible the function of each line or group of lines in a way that will be clear to someone other than the author. One method of achieving simple and more-readable comments is to add them after the program is debugged. At this stage, the author has a better overview of the program and looks at the code more as a user would. In any case,

authors need not be reminded that the success and longevity of a program are often dependent on its readability. This enables errors to be found more easily and allows the inevitable modifications to be made without interfering with existing code.

8. UPDATE PROCEDURE

With the XRAY system, users were encouraged to send information on program errors and fixes to the authors at the University of Maryland for eventual distribution to users as updates. This proved to be an extremely difficult task both because of the sheer volume of updates for all programs, and because it was virtually impossible to assess the validity of all corrections and enhancements on programs that did not originate locally. A more synergistic approach to updates is to have authors check corrections and faults on their own programs and then send these on to the current designated clearinghouse for distribution.

Accordingly, for the XTAL system, users are required to forward all errors and fixes to the author identified for that purpose in the XTAL handbook. It will now be the author's responsibility to test these modifications, and periodically send these to the clearinghouse with suitable documentation. One of the most difficult aspects of updates in the past was that users, and the authors themselves, have failed to comment updates to say what they fix, or what new feature they provide. It must be recognized that some installations, for reasons that are not entirely clear, are reluctant to implement any but the absolutely essential updates. Certainly many installations wish to stabilize the program system they are using and have little interest in additional enhancements, particularly when they are uncommented (as they usually are).

The method of handling updates will be to distribute them in a form which can be treated by a special editor routine. This routine will expect updates in a form described below. It will scan the named symbolic decks and carry out insertions and deletions in the RATMAC code. These insertions and deletions will be marked in the output code and will be made subject to priorities and machine designations described.

8.1 The Update Code *U

The update processor allows a series of commands for the insertion and deletion of lines of code. These commands are:

*U,<priority>,<machine>

*N,<deck name>
designates XTAL deck as given in the SYSTEMHEADER:(<deck name>)

*I,<line number>,<priority>,<machine>
insert following RATMAC code after specified line

conditional on <priority> and <machine> matching *U specifications.

*R,<beginning line number>,<ending line number>, <priority>,<machine>
insert the following RATMAC code in place of the deleted lines conditional on <priority> and <machine> matching *U specifications.

*D,<beginning line number>,<ending line number>, <priority>,<machine>
delete the specified lines

** end of an insert line

*E end of updating procedure

After *U any number of *N, *I, *D, *R, **, may be used. They must, however, be presented in the same order as the <deck name> order on the XTAL system file.

The *U processor has two input and one output stream. They are: the XTAL system edit deck, and the updated XTAL decks. Only the "decks" of the updated symbolics are copied to the output stream.

8.2 Priority Codes

<priority>

1 Designates an essential update that corrects a code fault causing a calculation error (eg misspelled variable name)

2 Designates an important update that corrects code causing an implementation fault on a specific machine (eg a macro provided for IBM is incorrect).

3 Designates an enhancement that improves some feature of the program, or expands its capability (eg additions to the Fourier program for a squared Patterson calculation).

8.3 Machine Specificity Codes

<machine>

0 Implies update for all machines

1 Implies update for IBM machines

2 Implies update for CDC machines

3 Implies update for DEC machines

4 Implies update for Data General machines

5 Implies update for Honeywell machines

6 Implies update for ICL machines

7 Implies update for Univac machines

8.4 Implications of the Use of *U

The update code assumes considerable importance in the operation of the XTAL system, because it provides (a) for a machine-independent method of communicating updates (rather than ... UNIVAC -NN,MM or CDC *DELETE XXNN,XXMM); (b) for a concise method of distributing the update priority, machine specificity, and function, and (c) for this information to be imbedded in the program listing.

To illustrate this, here is a simple example. If there were a requirement in the subroutine XY05 to delete statements 57 to 59 and insert 2 updates lines in their place; and to insert another 2 update lines after statement 71, the update information would appear as follows:

```
*U,1,0
*N,XY05
*R,57,59,1,0
A = QX(I+1)     # EXTRACT A FROM QX ARRAY
B = QX(I+2)     # EXTRACT B FROM QX ARRAY
**
*I,71,1,0
QX(J+7) = A     # DEPOSIT A IN NEW POSITION
QX(J+8) = B     # DEPOSIT B IN NEW POSITION
**
*E
```

In this example, the <priority> of the update code indicates the update is important and applicable to all machines.

The *U editor will transfer information on the edit to the output symbolic deck. Deleted lines will be given a # in column one, moved over, marked, and left in the output. New lines which are inserted will be marked as well. The process will use columns 73 to 80 of the edited deck for the purpose of marking the output. This editor will thus leave the

information on changes from the base system in the edited symbolic decks.

Finally, program authors will be required to use considerable judgement and tact on whether updates they receive are worthy of the effort of implementation. The rule of thumb should be that updates should be kept to an absolute minimum and fall into the classes designated as essential, important, or a desirable enhancement. Very few users wish to change 10 lines of code to improve a page layout!

APPENDIX 1

The Distribution and Programs of XTAL

What follows is a description of the XTAL system distribution tape taken directly from the first file of that tape. The original Guidelines for Authors (2) had listings of the nucleus programs and some examples of their use.

THIS TAPE CONTAINS 11 FILES INCLUDING THE FIRST WHICH IS THE TABLE OF CONTENTS FILE

FILE 2	RATMAC IN FORTRAN USING ONE CHARACTER PER WORD ARRAYS
FILE 3	RATMAC IN RATMAC
FILE 4	RATMAC IN RATMAC MACHINE INDEPENDENT ROUTINES
FILE 5	RATMAC IN RATMAC MACHINE DEPENDENT ROUTINES
FILE 6	RATMAC TEST DECK
FILE 7	OUTPUT OF RATMAC TEST DECK. FIRST THE PRINTED OUTPUT, THEN THE LISTING OF THE FILE PRODUCED BY THE PREPROCESSOR.
FILE 8	XTAL MACROS
FILE 9	XTAL SYSTEM IN RATMAC
FILE 10	XTAL TEST DECK
FILE 11	XTAL SYSTEM SPECIAL SUBROUTINES FOR SPECIFIC MACHINES

THERE WILL BE TWO MAGNETIC TAPE FORMATS FOR EXCHANGING PROGRAMS AND DATA BETWEEN DIFFERENT INSTALLATIONS: (1) ANSI LABELED TAPE(REF.ANSI X3.27-1977, LEVEL 3) FOR USERS WITH COMPATIBLE COMPUTERS AND (2) AN UNLABELED TAPE FOR USERS WITH DISSIMILAR MACHINES. THESE TAPES MAY BE OF ANY TRACK AND BIT DENSITY AGREED UPON BY THE CORRESPONDING PARTIES; FOR THE PRESENT, DEFAULT SHALL BE 800 BPI AND 9 TRACK. THE FOLLOWING DESCRIBES THE UNLABELED TAPE.

```
:= MEANS 'IS DEFINED TO BE'
CHARACTER := USASCII/7
RECORD := 80 CHARACTERS
BLOCK := 45 RECORDS, ZERO FILLED WHEN NECESSARY
FILE := <ANY NUMBER OF BLOCKS> END-OF-FILE MARK
FIRST FILE := RECORDS CONTAINING THE TABLE OF CONTENTS
              OF THE FOLLOWING FILES
TAPE := <FIRST FILE><ANY NUMBER OF FILES> END-OF-FILE
        MARK
```

THE EIGHTH BIT OF EACH CHARACTER, THE MOST SIGNIFICANT BIT, SHALL BE ZERO. THE TAPE IS BLOCKED WITH 45 RECORDS (3600 CHARACTERS) IN EACH AND EVERY BLOCK BECAUSE THIS LENGTH IS COMPATIBLE WITH ALL KNOWN MACHINE REGISTER SIZES. ALL INCOMPLETE BLOCKS SHALL BE ZERO FILLED IN ORDER TO KEEP BLOCK SIZES CONSTANT. THE TABLE OF CONTENTS IN THE FIRST FILE CONSISTS OF THE FILE NAME WITH NO IMBEDDED BLANKS, FOLLOWED BY A CHARACTER BLANK AND IF DESIRED, BY A

DESCRIPTIVE COMMENT. WHEN ALL FILES ARE SO DESCRIBED, A RECORD OF ALL BLANKS SHALL BE PRESENT. NEXT MAY FOLLOW ANY NUMBER OF RECORDS OF DESCRIPTIVE TEXT AND MESSAGES EACH SUBSEQUENT FILE, DESCRIBED IN FILE 1, SHALL CONSIST OF FILLED BLOCKS OF RECORDS CONTAINING DATA IN CHARACTER FORM. FROM THE DEFINITIONS IT IS APPARENT THAT THE END OF TAPE IS SIGNALED BY TWO SEQUENTIAL END-OF-FILE MARKS. THE ANSI CHARACTER TABLE IS AS FOLLOWS:

Character set information

o

lower case alphabetics

o

abcdefghijklmnopqrstuvwxyz

o

UPPER CASE ALPHABETICS

o

ABCDEFGHIJKLMNOPQRSTUVWXYZ

o

digits

o

1234567890

o

FORTRAN 77 Special Characters

o

= +-*/(),.$':

o

The above set contains

o

equals,blank,plus,minus,asterisk,slash,left paren,
right paren,comma,period,currency sign,apostrophe,colon

o

Additional special characters with significance in
RATMAC
& ampersand logical conjunction
! exclamation point logical negation
¦ bar logical inclusive disjunction
\ backslash logical inclusive disjunction
" double quote string delimiter with macro expansion
> greater
{ left brace statement block delimiter
[left bracket macro protection
< less
} right brace statement block delimiter
] right bracket macro protection
; semicolon statement separator
sharp comment signal
^ caret logical negation
\ backslash delimits non printing values in string

o

Additional delimiters

o

@ at sign
% percent
? question mark
~ tilde

The following partial list shows the proposed programs for the XTAL system. Those with assigned overlay numbers are in the process of being coded and checked out.

User Calling Mnemonic	Program Filing Mnemonic	Overlay Number (Base 10)	Program Function
	AA	0	Nucleus
ABSORB	AB	29	Absorption corrections by Gaussian quadrature
ABSCOR	AC	10	Absorption corrections by Tompa method
BONDLA	BN	6	Bond lengths and angles-connected set
BONDAT	BT	11	Generation of bonded atoms by geometric consideration
CAMEL	CM	12	Calculate absorption correction from diffractometer measurements
CONTRS	CN	13	Contour sections of Fourier maps
CRITIQ	CQ	14	
CRYLSQ	CR	15	Full matrix least-squares
DATADD	DA	16	
DATGEN	DG	17	Generation of test data sets
DATIN	DI	18	
DATRDN	DR	3	Data reduction to apply symmetry operations, etc.
DIFSYN	DS	19	Diffential difference synthesis
EVAL	EV	20	Evaluation of E(h,k,l) from intensity data
EXAMPL	EX	60	Example of use of binary data file generation subroutines. Test of AA22,23,24,25,and 26

FC	FC	5	Structure factor calculation generate over atoms
FFT	FF	21	Fast Fourier transform
FOUREF	FR	22	Automated Fourier refinement
FOURR	FS	8	Fourier transform Beever-Lipson method
GENTAN	GT	23	Tangent formula generation for direct methods
LOADAT	LA	4	Load atom parameters
LATCON	LC	24	Calculate least-squares lattice constants from two theta data
LISTFC	LF	25	List reflection data in compact form for archiving or publication
CROP	LP	26	List atom parameters in compact form for archiving or publication
LSQPL	LS	27	Least-squares planes and lines calculation
MULDMP	MD	28	Dump information for Multan
MIR	MI	9	Multiple Isomorphous replacement phase refinement program
	MT	62	Core management routine
MULIST	MU	43	
	OZ	61	Diagnostic message generator
POWGEN	PG	30	Generate the powder pattern expected from single crystal data
PEKPIK	PP	31	Search Fourier maps for peak coordinates
PROMPT	PR	32	Prompting program for screening input data for all these programs
PHISIN	PS	33	Direct methods starting phase set evaluation

PLOT	PT	34	Interface to line plotters and line printers plotter simulation
RIGBOD	RB	35	Ridged group transformations and preparation for least-squares refinement
HKLIN	RI	2	Input reflection data (intensities, etc.) into a binary data file
RSCAN	RS	36	Scan over reflection subsets to produce various reliability indicies (R values)
SCALE1	SC	37	
SLANT	SL	38	Interpolate an arbitrary plane from a Fourier map
SINVAR	SN	39	Search for structure invariants for direct methods
SORTRF	SR	40	
STARTX	SX	1	Produce an ab-initio binary data file with unit cell, symmetry, and related data
	SY	63	System initialization routines - sets of system common
DUMCOP	TD	7	Dump or copy BDF
WTANAL	WA	41	Analyze F distributions to determine weights for least-squares
WTLSQ	WL	42	Calculate analytical weights for least-squares

APPENDIX 2

XTAL Nucleus Macros

See Section 2.2 and RATMAC Primer (5) for the definition and discussion of macros. Macro auguments appear as $1, $2, $3, . . for the first, second, . . . etc. arguments to the MACRO. The macros themselves are to be found as the eighth file of the distribution tape.

AND: Logical conjunction

ARCCOS: Arccosine, arcsine and arctangent are not standard. [ACOS($1)] UNIVAC [ARCOS($1)] IBM CDC PDP

ARCSIN: [ASIN($1)] UNIVAC [ARSIN($1)] IBM CDC PDP

ARCTAN2: Returns angle from zero to two pi in radians. $1 is sine of angle and $2 is cosine of angle. [ATAN2($1,$2)] UNIVAC,CDC

BINSEQBKSP: Causes sequential input-output binary records to be backspaced. The function is a dummy when direct access to disc addresses are available. The FORTRAN statement is [BACKSPACE $1]

BINSEQCLOSE: Closes out a binary file. Some machines will have an operating system that demands that this be done.

BINSEQEOF: Marks end of file on sequential binary files. When on mass storage such as disc or drum, this macro should release any extra tracks back to the local operating system. The FORTRAN statement is [ENDFILE $1]

BINSEQOPEN: Calls to open a file. Some machines will have an operating system that demands this be done.

BINSEQREW: Rewinds the sequential binary file. The FORTRAN statement is [REWIND $1]

BINSEQBUF: Sets the size of the sequential binary file buffers. This number should be fine-tuned to the local mass storage device and/or local FORTRAN operating system buffers.

BINSEQREAD: Best accomplished by using local operating system facilities. Failing that, the FORTRAN statement [READ($1)$2] as the macro will give the correct effect. Careful tuning of BINSEQBUF: is essential for optimum efficiency.

DATASTRING:, and DATASTUFF: are all intimately interrelated with this concept. A character code which does not use a number of bits which gives an integral number of characters per word will not work. There are ways to make it work but, as written in XTAL the more general process will not work. **the programmers caveat is well and truly given**. On most machines INTEGER or INTEGER*4 will work very well. A few may require REAL, but since all character strings are moved by use of one of the three macros MOVEBYTE:, MOVECTOR:, or MOVERTOC:, the whole mess can be kept consistant with the rules of FORTRAN 77.

CHARDEF: Defines the number of characters in a word. Used on some machines in AA05 to move characters.

CHARS: Calculates the number of words to hold 'N' characters. Used in statements such as:
CHARACTER:CH(CHARS:(6))
which through the macros allows the correct number of words for 6 characters. Thus, it mimics the FORTRAN 77 conventions.

CHRLIMIN: Sets a floating-point word in the input buffer BFINFP, an array in COMMON /SYS/, so that it contains two pointers to the characters in the input line image. These pointers are packed as -XXYY.E+20 where XX is the starting column and YY is one beyond the last column of a character field. A detailed description of line image input is given in section 4.

CHRLIMOUT: Translates back the integers stored by CHRLIMIN. $1 is the member in the BFINFP array, $2 is the starting column and $3 is the last column plus one in the BFINIM array in COMMON /SYS/.

COMC:, COMF:, COMI: Specify communication commons of major links. One for integer, one for real, and one for character variables. See section 3.7.

COMPCHAR: Finds matches in strings of characters in different arrays. Causes a call to subroutine AA06 which has as arguments the character strings to be compared and parameters delimiting the search. An integer index is returned which either indicates the failure to find a match or acts as a pointer to the match. See section 4.7 for details.

COMPCHR: Hooks up to the machine specific method of comparing character strings. Used by AA06 particularly.

CONVCHR: Converts the number of characters to the number of I/O items.

COPYFILE: Invokes subroutine AA24 which copies logical records in sequential files. See section 6.3.

CPI: The total number of characters in an input line image. A function of MXWDIM:, MXCHWD:, and CUT:.

CPL: Current characters per output line image. A function of MXCHWD:, MXCHLN:, and CUT:.

CP6: The total number of words in a six character string.

CUT: Flag to signal whether FORTRAN 66 or FORTRAN 77 rules apply in a given machine.

DATASTORE: Plays a central role in the storing of character strings on various machines using FORTRAN 66. The RATMAC

```
DATASTORE:(A001, NOW IS THE TIME...)
```

causes the string:

```
NOW IS THE TIME...
```

to be counted and broken up into appropriate word long strings. The result on a 6 character per word machine is as follows....

```
INTEGER A001(4)   integer for CHARACTER: on this
                  machine
INTEGER NA001     this variable is generated to
                  give automatic count of
                  characters in the string.
```

In a ten character per word machine....

```
INTEGER A001(2)
INTEGER NA001
```

would be generated by DATASTORE:. The characters themselves are put aside in a macro called A001: for later use by the DATASTUFF: macro. This division of labor is necessary because FORTRAN77 will not allow any data statements before all TYPE, COMMON, EQUIVALENCE, and DIMENSION statements.

DATASTRING: Peels characters out of macros created by DATASTORE: and builds the correct size character strings for the machine at hand. It is a recursive macro and is used by the macro DATASTUFF:.

DATASTUFF: Produces the data statements, in conjunction with DATASTRING:, required to store the strings

supplied by DATASTORE:. From the example started under DATASTORE:, the output will be on a 6 character per word machine...

```
DATA A001(1)/6H NOW I/
DATA A001(2)/6HS THE /
DATA A001(3)/6HTIME../
DATA A001(4)/1H./
DATA NA001/19/
```

on a ten character per word machine....

```
DATA A001(1)/10H NOW IS TH/
DATA A001(2)/9HE TIME.../
DATA NA001/19/
```

All of this effort is obviated under FORTRAN 77 or on machines where byte addressing is possible.

DCODEFLD: Decodes the fields of an input line. Used, for example, near the end of AA01, BN03, and SX03.

DUMP: Sets up connection to local operating system memory dump.

ENDRECORD: Serves as a signal logical record number representing the end of a sequential file.

FILESTATUS: Provides the function for testing if a file has been opened or written on. $1 is the file unit to be tested and $2 is the status code. The status code =0 if the is active (ie. open or written on) and =1 if closed. On UNIVAC:

```
[ $( $2=0; IF(UNIT($1).EQ.0)$2=1 $) ]
```

GETDATE: Gets date from local system. $1 is the day in floating-point, $2 is the month and $3 is the year (each as two digits).

IFIX: Rounds positive and negative floating-point numbers to integers. Invokes the sign function. Used to avoid truncation. $1 is the floating point argument.

INCLUDE: Provides a mechanism for simulating non-standard FORTRAN INCLUDE statements. RATMAC will also recognize INCLUDE in certain implementations. The INCLUDE statement is used to duplicate one copy of a set of COMMON statements in many different subroutines.

INT: Rounds positive floating point numbers to integers. Faster than IFIX:.

INTPAK: Places integers into packed reals; used for example to store h, k, l, and sign codes into reflection records. Can be used interchangeably with MOVEBITS: (see section 3.9).

INTUNPAK: Retrieves packed integers from real words. Reverse of INTPAK:. (see section 3.9).

IODEVNUMA: Minimum device number for binary files. IOUNIT(1) (ie. FILEA) will be assigned this value. The rest are set sequentially from this point (see MXDVNM:).

The next eight macros are all concerned with input-output file handling.

IOCHR: Designates variable in COMMON/SYS/ which gives current maximum width of printed output lines. Needed by AA08 as information supplied by a FILES input line. This macro and the seven which follow are used to avoid equivalence statements in COMMON/SYS/.

IOIN1: and IOIN2: Refer to the standard and special line input files.

IOOT1: and IOOT2: Refer to the standard and special line output files.

IOOT3: Refers to the line output file. May be thought of as a "card punch".

IOT1P: and IOT2P: Store the current values of line printer's IOOT1: and IOOT2: priorities (see section 5).

ISAVE: Defines I/O unit for saving common in non-overlay mode of operation. See PROGRAMODE: and subroutine AAAA.

IQUIT: Creates error message exit. $1 is a floating point number of the form XXYYZZ. XX is the KCD (overlay) number of the program, YY is the subroutine number, and ZZ is an arbitrarily assigned sequential number. (e.g. $1 value in AA03 might be 000302.). In the IQUIT mode, control is passed to AA42 and then to the overlay OZ00 to print the IQUIT message. The idea is that as programs are developed, the author will keep a running log of the meanings of the various fatal errors in the program which can be published as a user guide, then later coded into OZ00 and its subroutines. If the argument is minus, then AA42 simply prints the minimal message
FAILURE IN PROGRAM XX. SUBROUTINE YY. REASON ZZ.
and returns to the calling program.

LINEFMT: Output line format used in AA08; e.g. for CDC:
```
1 FORMAT(1X,13A10,A2)
```

LINEIN: Input of card image. There are four arguments. $1 is the input device number, $2 is the input buffer array name, $3 is the return length of the input image (to last blank), and $4 is the EOF code (=0 if no EOF; =1 if EOF encountered). This macro may be set up to use standard FORTRAN I/O by...

```
[$(READ($1,1,END=2)$2;GOTO3;2 CONTINUE;$4=1;
3 CONTINUE
1 FORMAT(INCR(ARITH:(ARITH:(MXCHIM:,-,1),
/,MXCHWD:)) A MXCHPW:) $)]
```

For example, this produces:

```
1 FORMAT(8A10) FOR 60 BITS/WORD;6 BITS/CHAR
1 FORMAT(20A4) FOR 32 BITS/WORD;8 BITS/CHAR
```

For efficiency a call to a machine language subroutine which moves the next input line into the buffer BFINIM in contiguous characters can save time and especially space on many machines.

On UNIVAC

```
[CALLAA80($1,$2,$3);$3=$3*MXCHWD:]
```

does the job. $2 is BFINIM and $3 is CDCHR. $1 is a dummy argument on UNIVAC since the @ADD command of the operating system permits changing streams under executive control.

LINEOUT: Output line to devices IOOT1: and/or IOOT2:, conditional on the value of IOT1P: and/or IOT2P:, respectively. There are three arguments: $1 is the priority rank of the requested output line and this is tested against IOT1P: and IOT2P: to determine if the line is to be output on IOOT1: and IOOT2:, respectively. (ie. output only occurs on IOOT1: if $1.LE.IOT1P:, etc.) $2 is the output buffer array and $3 is the number of words to be output. It is used by subroutine AA08. The FORTRAN statements for this macro would be ...

```
IF($1.LE.IOT1P:)WRITE(IOOT1:,1)($2(KIK),KIK=1,$3)
IF($1.LE.IOT2P:)WRITE(IOOT2:,1)($2(KIK),KIK=1,$3)
```

LINEPCH: Same as LINEOUT: but is applied to file IOOT3: for punch card images.

LINEPRIOR1: Contains the default priority limit for the line output device 1 (IOOT1:). This is deposited in IOT1P: by SY00 during system initialization.

LINEPRIOR2: Contains the default priority limit for the line output device 2 (IOOT2:). This is deposited in IOT2P: by SY00 during system initialization.

MASTERSPACE: Defines the character used by local operating systems as a control character. Usually it will

initiate action by the monitor. See also macro SYSTEMHEADER: (will not be needed on many machines).

MEMDUMP: Forces memory dump on those machines and operating systems where the procedure would be useful.

MEMLESS: Requests operating system to reduce the size of immediate access store available for XYDATA array. $1 is highest address to be available after MEMLESS: has acted. Care must be exercised since $1 is both read and written (see AA41).

MEMLOC: Finds actual memory address. The FORTRAN routine on many machines is LOC(X); where the LOC function returns the memory address of the variable X.

MEMMAX: Defines the memory in words that can be used for nucleus plus the overlay plus the QXDATA array.

MEMMORE: Fetches more memory space during execution; used by subroutine AA41. Each operating system will have different rules governing the use of MEMMORE: and MEMLESS:. If the given computer or operating system does not allow dynamic allocation simply set the QX array in COMMON/QXDATA/ to a large value, set MEMMAX: equal to the same value and make the MEMMORE: and MEMLESS: macros null.

MNFPNBR: Minimum floating-point number of the given machine. Floating-point numbers less than this value constitute floating point underflow (magnitude).

MOVEBITS: Moves bits from one word to another by
MOVEBITS:(FROM,N,TO,M,NB)
where FROM is an input register, N is the starting low order bit position - bits numbered from the right of the register.

```
33222222222211111111110000000000  Read down
10987654321098765432109876543210
```

values of N and M therefore run from 0 to 31 inclusive and represent the power of 2 which is the string of bits starting position. NB is the number of bits to move from FROM to TO. M is the starting bit position in TO. For many machines, a machine language subroutine will be needed (see section 3.9).

MOVEBYTE: Aids in moving characters from one array to another. Simulates or makes possible the use of type CHARACTER of FORTRAN 77 (see section 3.9).

MOVECHR: Moves characters from one position in a packed floating point word (i.e. a word of the same number of bits as the given machines F.P. registers) to another position in another, or the same word. The arguments are FROM,N,TO,M, as in MOVEBITS:; here NB is understood to be MXBTCH: (see section 3.9).

MOVECTOR: Moves character strings to storage in REAL words. This is necessary for loading I/O buffers and bypassing restrictions of FORTRAN 77 (see section 3.9).

MOVEIN: Moves characters into REAL words.

MOVEOUT: Fetches packed characters out of REAL words.

MOVEREAL: Moves REAL word arrays from one array to another.

MOVERTOC: Moves character strings packed in REAL words into arrays of type CHARACTER:.

MOVERWORD: Moves REAL words which may be packed. The purpose of doing this is to avoid normalization which occurs on some machines when A = B is used to move words from one register to another.

MXBTCH: Number of bits used to represent a character.

MXBTWD: Number of bits used to represent a real (floating point) register. (eg. CDC=60, IBM=32, UNIVAC=36 ...) Note that no bit packing is done in XTAL beyond 32 bits.

MXCHIM: Number of characters in an input line image.

MXCHLN: Number of characters in an output line image.

MXCHWD: Number of characters which may be packed into a floating point register.

MXDVNM: Maximum number which may be used as a device number (often called FORTRAN logical unit number). See IODEVNUMA:.

MXFDIM: Maximum number of input fields to be allowed on an input line image. This serves to limit the size of array BFINFP used by the decode subroutine AA02. AA02 translates each field of the input line into the BFINFP array.

MXFDLN: Maximum number of fields for translation to an output line by the encode subroutine AA07. AA07 translates from floating point to characters for printing.

MXIOUN: Maximum number of binary storage devices, (eg. tapes, discs).

MXLNPG: Maximum number of lines per page of printed output.

MXSGFP: Maximum number of significant digits in a floating point mantissa.

MXWDIM: Maximum number of words in input line image buffer BFINIM(). This macro uses MXCHIM: and MXCHWD:.

MXWDLN: Maximum number of words in an output line buffer BFOTIM(). This macro uses MXCHLN: and MXCHWD:.

MXWD6C: Number of words (F.P. registers) needed to hold six characters, packed together.

NCODEFLD: Encodes floating-point arguments into character strings. A function corresponding to FORMAT operations in FORTRAN. Causes call to AA07, see section 5.

NEXTPAGE: Moves output line device to top of next page. The steps needed to use usual FORTRAN I/O would be...

```
     [KIK=IOOT1:WRITE(KIK,77);77 FORMAT(1H1)]
or   [PRINT 77;77 FORMAT(1H1)]
```

or their equivalent. See macro LINECUT: and subroutine AA08. This function applies only to device IOOT1: and not to device IOOT2: as set forth in the AA08 description.

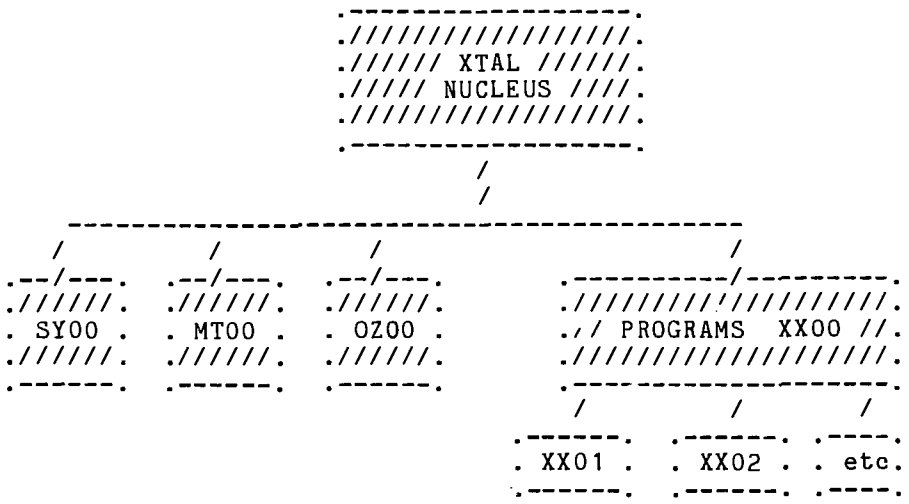
NO: Integer 0.

NOT: Logical negation.

OCT: Converts decimal to octal numbers; CDC-specific. Needed to resolve the overlay identifications of CDC.

OVERLA: Indicates the use of an overlay. On most machines the macro MACRO:(OVERLA:,CALL $1) is used and appropriate directives are given to the segment loader (mapper, link editor, or whatever it may be called). On some machines this may be insufficient. In this case the other arguments are set out in CDC specific terms. Arguments $2 point to the primary

overlay, $3 to the suboverlay within the primary one. All calls to primary overlays are in AAAA, AA01, AA02, and AA42. All secondary overlays are called from XX00, where XX is FC, SX, etc. The structure of XTAL is such that the nucleus is the root segment, the primary overlays are the first level of branching, and the secondary overlays are managed by each of the first level overlays. Roughly as indicated below....



OVERNA: Defines subroutines which are overlay segments. For most machines it is null. When used the arguments are: $1 is the subroutine name, $2 is the principal overlay number, while $3 is the secondary overlay number, a la CDC. Good luck to others.

PAGESET: Suppresses or overrides the automatic pagination that occurs on most lineprinter type devices. This enables pagination to be completely under the control of the nucleus AA08 and makes possible continuous line output for fouriers and plotting simulation.

POINTPKT: Gets packet pointers for a given logical record. Closely related to READPKT: (see section 6, especially 6.3).

PROGRAM: Substitutes "PROGRAM" for "SUBROUTINE" (CDC specific). On most machines it will simply be:

```
MACRO:(PROGRAM:,SUBROUTINE $1)
```

(see section 2.1 and PROGRETURN:).

PROGRAMODE: Signals subroutine AAAA which mode the system is operating in. The value 1 indicate that all major programs are separate. This has been done to facilitate implementation on virtual memory machines, or machines which do not support a segment loader or overlay structure. The value 2 means that the system is in a conventional overlay status. The difference from the users view is that in mode 1 each function must be executed separately by the local operating system "EXEC" command, (whatever it may be). While in mode 2, only one "EXEC" command is used and the XTAL data cards themselves cause the loading of the requested overlays. The logic is in progam AAAA, which tests this flag.

PROGRETURN: Allows the elimination of the RETURN statement in subroutines which are programs in the CDC sense (if any or none depending on your point of view); CDC specific. It must be used in all subroutines which use PROGRAM: and PROGRMAIN:. On most machines it will simply be:

```
MACRO:(PROGRETURN:,RETURN)
```

PROGRMAIN: Distinguishes a main program from other subroutines; CDC specific. On most machines it will simply be:

```
MACRO:(PROGRMAIN:,SUBROUTINE $1)
```

PUNCHFMT: FORTRAN FORMAT statement for 'punched' output.

QXBASE: Sets base number of words in QX data array for given machine and operating system. On dynamic memory machines, it is set to 1 and modified in MT00. On virtual memory machines, it is set to a large number consistent with memory available. See SYSCOM:.

QXMEMORY: Invokes call to subroutine AA41 which is used for memory management. See sections 2.5 and 2.6.

READLINE: Invokes call to subroutine AA01, the line input routine (see section 4).

READWPKT: Invokes call to subroutine AA22 and its subroutines for reading and copying packets of sequential binary data files. See section 6.

REALNOT: Performs a "NOT" operation on a real variable.

SMALL: Ten times the value of MNFPNBR:.

STARTUP: Takes care of any one per execution operation required by a given operating system. Invoked in subroutine SY00.

SYSTEMHEADER: Generates appropriate local operating system control lines. As RATMAC translates the symbolic program certain operating systems will demand prompting. This macro is provided for that purpose. One example is the UNIVAC EXEC VIII which needs a @FOR or @FTN line before each subroutine. The @ must go in column 1. Thus...

```
[$PMASTERSPACE: FTN,ISA$B$B$1,Y,$1]
SYSTEMHEADER:(AAAA)
```

gives: @FTN,ISA AAAA,Y.AAAA

to prompt EXEC VIII. another example is for CDC where:

```
[$PMASTERSPACE:DECK,$1]
```

would be appropriate, giving:

```
*DECK,AAAA
```

The macro MASTERSPACE: must be set to @ in one case and to * in the other. Many operating systems do not require this type of prompt, in which case SYSTEMHEADER: should be made null. Note the use of the $P and $B digraphs to keep the line from hitting 6 spaces right and to keep the columns which must be blank blank.

TIMCONFAC: Conversion factor from local operating system time units to minutes. Used by MT00.

TIMECLOCK: Gets the current wall clock time for subroutine MT00. $1 is the time as a packed character string of the form HH:MM:SS or HHMMSS or HH.MM.SS. $2 is the number of characters in the string.

TIMEPROG: Gets out the charge time in $1 and the CPU time in $2 as floating point.

UNITCD: Device number corresponding to usual line input. (e.g. card reader, teletype, vdu, etc.) Was unit 5 in FORTRAN for years and years. Used to set variable IOIN1.

UNITLP: Device number for printed output. Serves to initialize COMMON/SYS/ variables IOOT1: and IOOT2:.

UNITPCH: Device number for punched output, IOOT3:.

VOIDFLG: Signals a void input field; it is a real number (floating register) value. See AA02 and macros CHRLIMIN: and CHRLIMOUT:. This value is stored in COMMON/SYS/ real variable BFINFP(), an array, to signal when neither numbers nor character strings have been encountered in input line images. It allows programs to detect void input fields.

WRITEPKT: Invokes call to subroutine AA21 for writing packets into binary data files. See section 6.

WRITLINE: Invokes call to subroutine AA08 for writing output on line printers. See section 5.

YES: Integer 1.

APPENDIX 3

XTAL Nucleus Routines

Nucleus routines all have deck names and entry points in the AANN series, where the NN are grouped.

NN	Purpose
00 - 19	Character reading-writing, general system management
20 - 39	Binary sequential file operations
40 - 59	Memory allocation, date-time, error processing
60 - 79	Direct access file operations
80 - 99	Machine language codes which supersede FORTRAN I/O. These are always called from macros and may be replaced by FORTRAN statements.

AAAA Program AAAA
Main Entry Point of XTAL System
The precise function of this routine is determined by the mode of executing the XTAL system. There are essentially two distinct modes of operation - mode1 for stand-alone or separate program execution and mode2 for overlaid program execution. In both cases the program AAAA is the entry point for initiating the system and for mode1 it is also the exit (stop) routine. The program mode is set by the macro PROGRAMODE: and will vary from site to site according the local operating conditions and loaders available.

Mode1 *** Stand-alone Operation ***
In this mode each program is loaded and executed by a separate JCL command with nucleus routines (AAAA to AA99) treated as a program library (or the equivalent). The entry point for all programs is always AAAA. This because AAAA is responsible for the maintenance of the system COMMON so that it is retained between chained calculations. On the successful completion of each program AAAA stores the COMMON onto scratch ISAVE: (in this mode ISAVE: is always reserved for this purpose) and automatically reloads it at the commencement of the next program. For the first program in a chain AAAA initiates COMMON via the routine SY00.

Mode2 *** Overlay Operation ***
In this mode AAAA is the single entry point to

the base overlay (ie. The (0,0) overlay in CDC notation). In this mode it is also responsible for initiating SYSCOM via SY00 and continually calling AA01 for the next program initialization. Exit in this mode is initiated by a "FINISH" image in the input stream and is via OZ00.

AA00 Subroutine AA00
Controls Sequence of Operations
As with the program AAAA the function of AA00 varies according to method of system implementation. The main purpose of AA00 is to sequence the program calls to each separate calculation. DATASTORE:(A001) in AA00 contains the input image names of each program entry-point in the XTAL system and this is supplied to the image-sifting routine AA01 which identifies if a call to a particular program is requested on input. This is communicated to AA00 via the key-card signal (KCD) in COMMON/SYS/ and used by AA00 to call the appropriate program. The mechanism for doing this in AA00 will vary according to the machine and loader. For stand-alone operation or IBM-type overlay calls each program request will appear as a separate call of the type:
CALL SUBRX
For CDC-type overlays there will be only one statement for all program requests of the type:
CALL OVERLAY(4HXTAL,KCD,0)
On completion of each calculation AA00 returns to AAAA. See section 2.

AA01 Subroutine AA01, READLINE:

Reads Input Line	ARG
(<list of seven character arguments, left justified>,	1
<total number of characters in the list>)	2

Enters all input images using the macro LINEIN: and tests if the image name that starts in column 1 is in the list array (supplied as an argument above of the calling program). NLIST is the length in characters of list. Images identified as system images (ie. SETID, ORDER, MEMSET, FIELD, FILES, TITLE, REMARK, or FINISH) are processed and necessary data removed using the image decoder AA02. Images identified from the list array reside in the input buffer BFINIM and the order number is returned to the routine calling AA01 via the word "KCD" in common. If the image is not found in the supplied list, or the system list, the routine enters an error mode and invokes IQUIT:.

AA02 Subroutine AA02, DCODEFLD:
Decodes Input Image
Translates (or decodes) the characters into either

floating-point numbers or pointers to character strings - whichever is appropriate to the data encountered (since all images entered into the XTAL system with the macro LINEIN: (in AA01) are in character (A) format). Translated floating-point numbers are deposited in the buffer BFINFP as fields in an order determined either on the input image or according to the "ORDER" and FIELD" images. This is described in more detail in section 4.

AA03 Subroutine AA03, MOVEREAL:

Moves Real Array	ARG
(<source REAL array>,	1
<starting word in source array>,	2
<sink REAL array>,	3
<starting word in sink array>,	4
<number of words to be moved to sink array>,	5
<key to direct action of subroutine>)	6

Moves REAL words from a source array to a sink array.

*** KEY = 0 when source and sink are different arrays
*** KEY = 1 when source and sink are the same array
*** KEY = 2 when it is desired to clear the sink array to the value of the first word of the source array.

AA04 Subroutine AA04, MOVEBYTE:

Moves Character Array	ARG
(<source character array>,	1
<starting character in source array>,	2
<sink character array>,	3
<number of characters to be moved to the sink array>,	4
<key to direct action of subroutine, same as AA03>)	5

Moves a character string of specified length from a source array to a sink array. The first character of the string is located in the specified byte of the source array (where byte 1 is the left-justified byte) and this is moved into specified byte position sink array. This routine depends on the macro MOVECHR:. The source and sink arrays must be type CHARACTER:.

AA05 Subroutine AA05, MOVECTOR: and MOVERTOC:

Moves Character Arrays to Real and Vice Versa	ARG
(<real array>,	1
<number of first character in real array, "left justified"=1>,	2
<character array>,	3
<number of first character in character array>,	4
<number of character to be moved>,	5
<key to establish function to be carried out	6

during call>,
<key 1 to control action>) 7
*** KEY = 0 move characters into packed real array
*** KEY = 1 unpack characters from real array and place in character array

Note well that unlike AA03 and AA04, the source and sink arrays are dependent upon the key setting. Use of the macros MOVECTOR: and MOVERTOC: take care of the necessary switching and avoid any mixed mode operations at a higher level.

*** KEY1 = 0 move on a character for character basis
*** KEY1 = 1 move only the first character of the source array to all characters of the sink array.

AA06 Subroutine AA06, COMPCHAR:
Compares Character Strings ARG
(<character array to serve as source for string sought>, 1
<index to first character in source array>, 2
<character array to serve as a table of strings sought during the compare>, 3
<index to first character in strings sought array>, 4
<length of string to be compared>, 5
<increment of index to be used during search for match>, 6
<total number of characters in the strings sought array>, 7
<key>) 8

The key is an integer which is returned as zero if no match is found or a number which points to the string matched.
Details of the use of this subroutine are given in section 4.7. This is not an easy routine to learn to use, but once learned, it is very powerful for string comparisons.

AA07 Subroutine AA07, NCODEFLD:
Encodes REAL to CHARACTER ARG
(<source floating-point (REAL) array to be encoded>, 1
<first word of source array to be encoded>, 2
<character array to act as sink for encoded numbers>, 3
<real array which contains FORMAT information, one word per item to be encoded>, 4
<number of items to be encoded>) 5

Encodes floating-point numbers from the source array into character strings in the sink array according to coded-format information in the FORMAT array.
NOTE -- It is usual practice to use the buffer

BFOTLN as the output real source array because this enables both a header line and the numeric line to be output with one call to AA08, WRITLINE:. Section 5.1 is devoted to the use of this subroutine.

AA08 SubroutINE AA08, WRITLINE:

Writes Printer Lines	ARG
(<number of blank lines to write before printing>,	1
<character array for output on line device>,	2
<number of characters in character array>,	3
<control key>,	4
<PR, priority of the printed line>)	5

Outputs buffer BFOTLN and/or the character array (of specific length) on the devices IOOT1: and IOOT2:. When the string array is to be output (depending on the key value) it will always precede the outputting of array BFOTLN (if this is to be output). See section 5.2 for details.

AA21 Subroutine AA21, WRITEPKT:

Writes Packets for Binary I/O	ARG
(<integer index to output binary file>,	1
<integer index to logical record>,	2
<integer value of packet size of logical record>,	3
<integer index to storage of packet in QX buffer array>)	4

Writes sequential binary files to a mass storage device. Familiarity with the structure of binary data files will help one understand this subroutine. Its purpose is to create binary data files or sequential scratch files of a similar structure. In using this subroutine the logical record to be written and the size of the packets in the logical record must be specified. The integer index to the output binary file may be 1, 2, 3, ... MXIOUN:, corresponding to files a, b, c,..... The integer index to the storage of the packet is a pointer set by the subroutine AA21 to point to the place in the QX array where the packet is to be stored. The logic of use is slightly different than a normal WRITE(IOUNIT) list.... In that one must call AA21 and then move data to the output buffer based on the value of the pointer returned. The actual write does not occur until either the buffer becomes full or ENDRECORD: is specified as the LOGREC. to be written. The typical pattern of use may be seen in section 6.

AA22 Subroutine AA22, READWPKT:

Reads and Copies Packets for Binary I/O	ARG
(<integer index to input binary file>,	1
<integer index to logical record sought>,	2
<integer value of packet size of a logical record>,	3

<integer index to point to a packet in QX buffer array>, 4
<integer index to output binary file; zero implies read only>) 5

Reads, or reads and writes, sequential binary files. Not to be used for writing if the packet size is to be changed. It uses just one packed buffer so the file size is fixed during reading, and reading and writing. For situations where the file is to be expanded, two buffers must be used and AA22, WRITEPKT:, and AA24, COPYFILE:, utilized. See section 6 for details.

AA23 Subroutine AA23, POINTPKT:
Decodes Packet Directory ARG
((<integer index to input binary file>, 1
<integer index to logical record sought>, 2
<integer value of packet size of logical record>,3
<integer index to storage of packet in QX buffer array>, 4
<integer count of number of quantities sought>, 5
<integer array of indices from binary data file specification in Appendix 5>, 6
<integer array to receive relative index pointers within packets for this file>) 7

Fetches relative pointers from packet1 of the "directory" type records in the binary data file. The first packet of a directory type record contains numbers, defined in the BDF description of Appendix 5, which identify the items stored in each of the following packets of the logical record. The first arguments are READWPKT:, arguments of AA22, reflection. See section 6.

AA24 Subroutine AA24, COPYFILE:
Copies File for Binary I/O ARG
((<integer index to input binary file>, 1
<integer index to output binary file>, 2
<integer index to first logical record to be copied>, 3
<integer index to last logical record to be copied>) 4

Copies the first designated logical record to the last designated logical record; used in conjunction with two buffers. See section 6.

AA25 Subroutine AA25
Reads Subroutine AA22
(<integer index to input binary file>)

AA26 Subroutine AA26
Writes Subroutine AA22

AA41 Subroutine AA41, QXMEMORY:
Allocates Dynamic Memory
Allocates memory via the machine specific macros MEMMORE: and MEMLESS:. The memory request is delivered to AA41 in the SYSCOM word QXREQU, which is the maximum value of the QX array required. This is converted into an absolute address before invoking MEMMORE: or MEMLESS:. The amount of core available is returned and converted back in terms of a QX array value. This is stored in the value QXAVAL.

AA42 Subroutine AA42, IQUIT:
Treats Error Signal
(<Encrypted Error Message>)
Designates where an irreconcilable error is encountered in the execution of any XTAL program, and what type of error it was. IQUIT: incorporates a call to AA42 which in turn is responsible for calling the time/core reporter (MT00) and the error message outputter (OZ00).
Extracts the encrypted message from a real value XXYYZZ., where XX is the overlay number, YY is the subroutine within the overlay, and ZZ is the message number assigned by the programmer. This is printed out as:
FAILURE IN PROGRAM XX. SUBROUTINE YY. REASON ZZ.
Programmers must take care to document the reasons as they program.

A-3.1 Machine Specific Routines

AA79 thru AA99
```
* These routines are all called by macros and all
* may be replaced by using usual FORTRAN input output
* statements, by calls to local operating system
* service subroutines, or by simply not implementing
* the feature they provide.
* All calls to these routines are through macros and
* only through macros.
```

AA79 Subroutine AA79 (<UNIT>, <image buffer>, <words-to-punch>)

	ARG
<UNIT>,	1
<image buffer>,	2
<words-to-punch>)	3

May be thought of as:
PUNCH 1,(ARG2(I),I=1,ARG3)
Causes a line image to be punched; uses the local operating system facilitites. When supplied, it is called from the macro LINEPCH:. On many machines this will be quicker and take far less memory than the FORTRAN library routines.

AA80 Subroutine AA80 ARG
(<UNIT>, 1
<image-buffer>, 2
<words-read-in>) 3

For CDC, it may be thought of as
READ(INPUT-DEVICE,1)(ARG2(I),I=1,8)
1 FORMAT(8A10)
Uses the local operating system and avoids the FORTRAN library. On systems which allow concatenation of input streams <UNIT> may be ignored and the system facility used. For example @ADD on UNIVAC.

AA82 Subroutine AA82, ID
Fetches date in some encoded form from the local operating system. Macro GETDATE: decodes it into standard XTAL form.

AA84 Subroutine AA84 ARG
(<UNIT>, 1
<image-to-print>, 2
<words-to-print>) 3

Prints output, using local operating system routines. The FORTRAN equivalent on CDC is:
WRITE(<UNIT>,2)(ARG2(I),I=1,<words-to-print>)
2 FORMAT(13A10)
See macro LINEOUT: for details.

AA85 Subroutine AA85
(<UNIT>)
Attaches unit to job; a dummy on many operating systems. See macro BINSEQOPEN:.

AA86 Subroutine AA86
(<UNIT>)
Ends off files in an orderly way. Uses system facilities to carry out ENDFILE UNIT. See macro BINSEQEOF:. However, care must be exercised that this function and AA85 are compatible with one another. The logic in subroutines AA22 and AA21 should be examined. Calls to AA85 and AA86 are in pairs for each unit.

AA87 Subroutine AA87 ARG
(<UNIT>, 1
<number-of-words>, 2
<file-address>, 3
<memory-address>) 4

Writes binary files on unit; used in conjunction with macro BINSEQWRIT:. The file address is the relative address on the mass storage device,starting at zero,for the first buffer full the memory address points to the start of the buffer.

AA88 Subroutine AA88
Reads from mass storage device; otherwise, just like AA87.

AA90 Subroutine AA90
(<UNIT>)

Causes FORTRAN equivalent of:
WRITE(<UNIT>,1)
1 FORMAT(1H1)
but uses local operating system facilities instead. Used with macro NEXTPAGE:.

AA91 Subroutine AA91
Sets page size parameters; UNIVAC specific. See macro PAGESET:.

AA92 Subroutine AA92
(<core-address>)
Requests core to given core address. See macro MEMMORE:.

AA93 Subroutine AA93
(<core-address>)
Releases core back to operating system. See macro MEMLESS:.

A-3.2 System Program Overlays

MT00 Program MT00
Reports Time and Sets Dynamic Core
Reports on LINEOUT device the status of the execution and memory times, and the various memory status words. Memory information is output in decimal and octal words. The abbreviation "MEMMAX" refers to the total number of words from the first word in AAAA to the last word assigned in the QX data array. "DATAMX" refers to the number of words for data in the QX array alone. (NOTE: for CDC operations, the number of words in the QX array overlaid by the (n,m) routines has been removed from DATAMX). The use of the word "RQUEST" in the AA40 output is the abbreviation for "requested by the system or user", while "ALLOTD" means "actually allocated to the program by the operating system via AA41". The previous program name indicates the memory allocation data for the previous calculation. And "SO FAR" means "during the total execution up to this stage".

In addition to reporting on the time and core consumed by the existing program, MT00 is also responsible for requesting the memory required to load in the next program. This is done by accessing two arrays, QXSK and QXWK, with the program pointer

KCD. The first array, QXSK, contains the "SKIP" memory required on CDC-type machines where the QX working array is not automatically loaded beyond the longest overlay. For these machines, it is necessary to define the start of the useable QX array at QX(QXSTAR), where QXSTAR = QXSK(KCD). The values of QXSK must be the value in words of the longest (n,m) overlay (including the (n,o) overlay). For most other machines all values of QXSK will be zero. The QXWK array contains the minimum amount of useable memory required in the QX array to initiate the program. This is referred to as the "STARTUP MEMORY", and is in general the absolute minimum required for the calculation. A request to the memory allocation routine AA41 is made for QXREQU = QXWK(KCD). Requests for additional memory are made within the program itself on a dynamical basis.

OZ00 Program OZ00
Terminates Run
Services an error call to subroutine AA42 (via the macro IQUIT:) or services a finish and exit call from AA01 (on receipt of a finish system image). In both cases the variable IQUIT in the system common /SYSCOM/ contains a code which specifies why OZ00 was called. If IQUIT = 000000. this signals a normal finish exit is called from AA01. All other values of IQUIT indicate an error has occured and is in the form of a code XXYYZZ. Where XX is the overlay number, YY is the subroutine number, and ZZ is the error sequence number in that subroutine. OZ00 outputs the error code to indicate why and where the error is. OZ00 also outputs a memory dump provided one of the priority limits is set to a value of 5.

SY00 Program SY00
Initialize variables and Prints Logo
Initializes all variables in the system COMMON /SYSCOM/ and prints the XTAL system sign on heading with an output priority of 3.

APPENDIX 4

Glossary of System Common Variables

The following list is designed to explain the meaning and use of the system variables used in the XTAL system. The letter preceding the description indicates the variable type. r is real, i is integer, and c is character (in the FORTRAN 77 type definition sense). If a variable is dimensioned, it is followed by a pair of parentheses ().

BFIELD() i Buffer to hold pointers to ending columns when a "FIELD" line has been encounted by AA01. Used by AA02 in translating input line images. See FIELD control card.

BFINFP() r Buffer to hold input floating point data. Each word corresponds to an input field. If the field is a number the buffer contains the number, if the field is void, the buffer word contains the signal VOIDFLG:. If the field is a character string the buffer word contains a packed pair of pointers which show where the character string actually starts and ends +1 in BFINIM. BFINFP is cleared to VOIDFLG: at the start of AA02.

BFINIM() c Buffer to hold the input line image in packed characters.
READ(IOIN1,1,END=2) BFINIM
1 FORMAT(20A4)
is an example of how it may come to be filled. The subroutine AA01 issues the read command as specified by the macro LINEIN:. The subroutine AA02 translates the input line into the floating point buffer BFINFP. See macro LINEIN:.

BFORDR() i Buffer to hold pointers to order in which data is to be placed in BFINFP from fields of input line images. See order control card in AA01 used in AA02.

BFOTFP() r Buffer to hold output floating point numbers to be translated into characters by subroutine AA07.

BFOTLN() c Buffer to hold characters for output line.

BFTITL() c Buffer to hold page title which includes current program, compound ID, page numbers, and date.

BLNKWD	c	Word filled with character blank.
CDCHR	i	Count of the number of characters in the input line image - set by LINEIN:
CDCOL	i	Pointer to the first blank character in the input line image - communicated between AA01 and AA02.
CDEOF	i	Signal for end of file on line input device - set by LINEIN: to 1 if EOF encountered; otherwise zero.
ELAPST	r	Last snapshot value of the CPU clock. Is used to calculate the CPU elapsed time for each calculation.
FIELDF	i	Flag to signal that input line images are to be decoded as fixed fields (1), rather than in free format (0).
IDPRPG()	c	Array containing name of previous program.
IDSETI()	c	Image of the set of characters which the "SETID" input line has brought in. See AA01 "SETID" command.
IOCHR:	i	Current printer output line width as a number of characters. Used to control output lines so that various widths of printers can be accomodated automatically. Set by field 16 of a "FILES" line image. Initialized in SY00, used by AA08.
IOIN1:	i	Special file designation for the input line device. The value may be changed by a "FILES" line image.
IOIN2:	i	Monitor or executive default value for IOIN1.
IOLRHD()	i	Pointer to header word of logical record. The value of the pointer shows the relative location of the first word of the current logical record as it resides in the binary output buffer in the XYDATA array (QX array).
IOLRPT()	i	Type number of logical record currently being processed.
IOMARK()	i	Basepoint of input output buffer in the XYDATA (QX) array. This corresponds to the appropriate QXMARK pointer set when the I/O buffer is set up in AA02.
IOPKPT()	i	Pointer set by AA21 and AA22 to show location of the next packet. This is the pointer used by

programs to fetch and store packets from the input output buffers. Each call to AA21 and AA22 changes the value back and forth through the QX array as specified by the corresponding IOMARK().

IOPKSZ() i Used by AA21 to set the packet size for an output logical record. Set by calling program to inform AA02 of proper action.

IOOT1: i Primary line output device controlled by macro LINEOUT: according to the priority limit IOT1P:.

IOOT2: i Second line output device controlled by macro LINEOUT: according to the priority limit IOT2P:.

IOOT3: i Punch or special output card image file. The value may be set by use of a "FILES" line image.

IORWFL() i Flag for signaling whether AA22 is reading only FILEIN or both reading FILEIN and writing FILEOUT from the same buffer. Set by calling program.

IOT1P: i Priority limit of line output device 1. Initially set to the macro value LINEPRIOR1: but may be specified as field 14 of an input FILES image.

IOT2P: i Priority limit of line output device 2. Initially set to the macro value LINEPRIOR2: but may be specified as field 15 of an input FILES image.

IOUNIT() i Actual values of the units. These are what have been called "logical unit numbers" or device number for binary files. They are in fact pointers to mass storage files or devices. These numbers may be described in FORTRAN:
IUNIT = IOUNIT(FILEIN)
READ(IUNIT) BUFFER

IOPRCT i Count of records from the beginning of a file. The value of this integer starts at zero.

IQUIT r Error flag set by the macro IQUIT: of the form XXYYZZ. XX is the program overlay number (KCD), YY is the two digit sequence number assigned to the subroutine and ZZ is the error number. (e.g. AA08 would be 000801.)

KCD	i	Variable used to communicate members of input images found by subroutine AA01. A list of expected line identifiers is supplied to AA01 by, for example: DATASTORE:(X011, ATOM B BIJ U END) placed in program X. A call from program X by: READLINE:(X011,NX011) will cause KCD to be set to 1 if an atom line comes into BFINFP,...., or 5 if an END line image comes into BFINFP. Since AA00 uses this same call, the value of KCD will point to the correct overlay when AA01 is called by AA00 with the ordered list of program names.
LINCT	i	Line count remaining for current page of line output device IOOT1:. Causes pagination when zero and is reset to macro value MXLNPG:.
LINRM	i	Number of lines which must be available on a page for a subheading if the page restore is to be suppressed. Used to communicate with AA08 for listing column headings (subtitles).
MAXCOM	i	Current length of system COMMON /SYSCOM/ in real words.
MAXMEM	r	Value of the maximum memory possible; used in AA41 and SY00.
MEMSTF	i	Signal to show that a MEMSET line has been encountered. Used in AA01, AA41, and SY00 for memory allocation control.
MPAGE	i	Count of pages written by current program.
NCARD	i	Count of "cards" punched during run.
NIMAG	i	Number of images read from the input device since the start of the run.
NPAGE	i	Count of total pages written on line output device during current run.
OTPRMX	i	Value of the maximum printer priority at current time.
QXAVAL	r	Total number of words currently available for use in the QX data array.
QXAVPG	r	Maximum QXAVAL for current program.
QXAVMX	r	Maximum QXAVPG for all programs so far.
QXREQU	r	Number of words of memory requested for use in the QX data array.

QXRQPG	r	Maximum QXREQU for the current program.
QXRQMX	r	Maximum QXRQPG for all programs so far.
QXSTAR	r	Number of words of the QX data array not accessible for data storage due to overlap of program overlays. This value is set by MT00 and for most machines is zero.
QXWORK	r	The pointer of the QX data array which defines the last word of the "simulated program common area". Beyond QXWORK is the normal working area of the QX array.
QXUTMX	r	Maximum QX array pointer currently in use.
SETIDF	i	Flag to show that a SETID image has been encountered and input lines are being processed as if the ID set were on the lines. 0/1 for no/yes.
SYNTXF	i	Flag set by calling programs to suppress syntax checking in AA02 the input line decoding program. Certain types of special input lines such as "SYMTRY" violate the usual free format rules of AA02. Setting SYNTXF to 1 before such reads and back to zero afterwards will prevent diagnostics being printed.

APPENDIX 5

The BDF for XTAL

The XTAL binary data file (BDF) is divided into separate elements called "logical records". Each logical record contains specific crystallographic information that may be referred to by the logical record type numbers 1 to 25. The length of each record (in words) will vary according to type of information it contains, the type of structure being processed, and the current state of the analysis.

For convenience of access, each logical record type is subdivided into packets of words which form the particular logical subset of the data contained therein.

Packet Types

The different types of information stored in the binary data file necessitate three different logical record constructions. Records containing character information are distinct from those containing numerical data. The numerical records are also of two distinct types. One contains information that is of fixed length and is located in specific words of the record, while the other numerical record contains data which may vary greatly from structure to structure. These logical records are now summarized:

Character records contain only packed characters. The packet size of character records varies according to the number of characters per packet and the length of the floating point register. Logical records 1, 2, 7, and 10 are of this type.

Specific Information records contain numerical information data which have fixed location in a specific packet. Each word of data is accessed by adding the appropriate sequence number to the packet pointer provided by the file handling routines AA21 and AA22. Logical records 3 and 5 are of this type.

Directory driven records contain numerical data which can vary according to the size and nature of the structure. The first packet is used as a directory to the data contained in all subsequent packets. In this way the packets need only be as large as the available data or the calculation requires. This is achieved by assigning identification numbers to each unique data type and inserting these numbers into the first packet in the identical order that the actual data appears in subsequent packets. A pointer to the word containing any given data type is provided by the nucleus file handling routine AA23.

A-5.1 Structure and Contents of the Logical Records

LR 1 File History (character)

Log rec	Packet size	Sequence number	
1	number of words to hold 16 characters	Packet 1	Contains the compound ID
		Packet 2	This and subsequent packets contain program ID's and date as hhddmm at the time of updating. After 50 updates, the list is reset and starts over again.

LR 2 Label Information (character)

Log rec	Packet size	Sequence number	
2	number of words to hold 80 characters	Packet 1	Contains date and time of creation of BDF.
		Packet 2	Contains title in force at time of creation.
		Packet 3	This and subsequent packets contain images of any label information supplied.

LR 3 (spare)

Reserved for possible use as BDF status keys to enable lookahead capability in sequential mode.

LR 4 Cell Constants (specific information)

Log rec	Packet size	Sequence number	
4	9		
		Packet 1	
		IP+1	a cell dimension in Angstroms
		IP+2	b
		IP+3	c
		IP+4	cos(alpha)
		IP+5	cos(beta)
		IP+6	cos(gamma)
		IP+7	alpha in cycles (2pi = 1.0000)
		IP+8	beta
		IP+9	gamma
		Packet 2	
		IP+1 - IP+9	Estimated standard deviations of the quantities of Packet 1.

Packet 3	
IP+1 - IP+9	Reciprocal cell constants in same order as Packet 1.
Packet 4	
IP+1 - IP+9	Transformation matrix from fractional coordinates to orthogonal Angstrom coordinates.
Packet 5	
IP+1 - IP+9	Transformation matrix from Miller indices to orthogonal pseudo Miller indices.
Packet 6	
	Miscellaneous cell information
IP+1	cell volume
IP+2	observed crystal density

LR 5 Symmetry Information (specific information)

Log rec	Packet size	Sequence number	
5	12		
		Packet 1	Contains miscellaneous information
		IP+1	Code to indicate lattice type as.....
			lattice type P I R F A B C
			acentric cell 1. 2. 3. 4. 5. 6. 7.
			centric cell 8. 9. 10. 11. 12. 13. 14.
		IP+2	Centric/acentric indicator 0/1
		IP+3	Number of symops
		IP+4	Number of distinct rotation matrices and translation vectors exclusive of lattice translations and center, if any.
		IP+5	Number of rotation matrices of identical pattern of zeros
		IP+6	Cell multiplicity factor to place a and b parts of the structure factor on the scale of int.tab. vol 1. This factor accounts for lattice type.
		Packet 2	Contains the rotation matrices and translation vectors for first equivalent position..
		IP+1	r(1,1)
		IP+2	r(2,1)
		IP+3	r(3,1)
		IP+4	r(1,2)
		IP+5	r(2,2)
		IP+6	r(3,2)
		IP+7	r(1,3)
		IP+8	r(2,3)
		IP+9	r(3,3)
		IP+10	t(1)
		IP+11	t(2)
		IP+12	t(3)

Packet 3 to n+1 for the remaining n equivalent positions. The maximum value of n is 24. Matrices involving an inversion center or non-primitive translations are excluded.

LR 6 (spare)

LR 7 Scattering Factor Names (character)

Log rec	Packet size	
7	number of words to hold six characters	Names of scattering factors contained in LR 8. Each packet contains the characters supplied as a scattering factor type. One packet for each different scattering factor type.

LR 8 Atom-type Parameters (directory)

Log rec	Packet size	Ident. number	Directory in packet 1, first atom-type in packet 2
8	varies	1	number of atoms of this type per unit cell
		2	atomic weight
		3	atomic number
		4	number of electrons in neutral atoms or ions
		5	atomic bond radius in Angstroms
		6	atomic contact radius in Angstroms
		7	
		8	
		9	effective spin quantum number
		10	neutron scattering length in cm*10**-12
		21	real part of dispersion scatt. factor for data-set 1
		22	real part of dispersion scatt. factor for data-set 2
		..	. . . . .
		61	imag part of dispersion scatt. factor for data-set 1
		62	imag part of dispersion scatt. factor for data-set 2
		..	. . . . .
		100	atomic scattering factor at s = 0.00
		101	atomic scattering factor at s = 0.01
		102	atomic scattering factor at s = 0.02
		...	. . . . .
		1nm	atomic scattering factor at s = 0.nm
		...	. . . . .

	299	atomic scattering factor at s= 1.99

Note (1) No scattering factor table is required if interpolated values have been stored with each hkl in the reflection record 20.

Note (2) Scattering factors at all s-intervals of 0.01 need *not* be present for interpolation.

Note (3) Additional scattering factors for a given atom type are stored 300-499, 500-699,700-899,...

LR 9 (spare)

LR 10 Data Set Definitions (character)

Log rec	Packet size	
10	words/ 12 characters	Strings of 12 characters used to describe data sets. Order of strings corresponds to data-set number. Data sets may be defined as isomorphs, graphs of partial structures, or residues.

LR 11 Experimental Parameters (directory)

Log rec	Packet size	Ident. number	Directory in packet 1, one parameter set per packet
11	varies	1	data-set key (1 designates data-set1, etc.)
		2	wavelength (weighted mean) in Angstroms
		3	wavelength line1 in Angstroms
		4	wavelength line2 in Angstroms
		5	wavelength line3 in Angstroms
		6	relative weight wl line1
		7	relative weight wl line2
		8	relative weight wl line3
		9	measured density
		10	linear absorption coefficient in 1/cm
		11	temperature of measurement in degrees celsius
		12	sorting order of hkl
		13	a cell dimension in Angstroms
		14	b
		15	c
		16	cos(alpha)
		17	cos(beta)
		18	cos(gamma)
		19	alpha in cycles (2pi = 1.0000)
		20	beta
		21	gamma

31-39	diffractometer orientation matrix r11,r21, ,...,r33
100	number of scale groups for this data set
101	frel scale factor for scale-group 1
102	frel scale factor for scale-group 2
...	. . . . .
100+n	frel scale factor for scale-group n (maximum allowed 64)

LR 12 Data Set Information (directory)

Log rec	Packet size	Ident. number	Directory in packet 1, one parameter set per packet
12	varies	1	data-set key (1 designates data-set1, etc.)
		2	overall temperature factor UOV in Angstroms squared
		10	packed word of =eval= fragment types
		11	maximum /h/
		12	maximum /k/
		13	maximum /l/
		14	minimum sin theta/lambda
		15	maximum sin theta/lambda
all initialized to IDFLG:		30	scale, data set to parent
		31	temp. factor, delta B, relative to parent
		32	closure error
		33	closure error, anomalous
		101	extinction type (0=none, 1=iso 1, 2=iso 2, 3=gen iso 1,2, and prime, 4=aniso 1, 5=aniso 2, 6=gen aniso)
		102	distribution (0=Gaussian, 1=Lorentzian)
		103	isotropic type1 parameter
		104	isotropic type2 parameter
		105-110	anisotropic type1 parameters
		111-116	anisotropic type2 parameters

LR 13 (spare)

LR 14 (spare)

LR 15 Atomic Identification (character)

Log rec	Packet size	
15	number of words to hold eight charac-	Each packet contains the string of characters which constitute an atom identification (6 characters) plus a 2 character dataset pointer (data number in ASCII). Their relative position in

ters	the packets is linked to the following record 16 which contains the atom parameters.

LR 16 Atom Parameters (directory)

Log rec	Packet size	Ident. number	Directory in packet 1, first atom data in packet 2
16	varies	1	x parameter in fractions of unit cell
		2	y parameter in fractions of unit cell
		3	z parameter in fractions of unit cell
		4	individual isotropic t.f. as B
		5	individual anisotropic t.f. stored as betas beta11
		6	beta 22
		7	beta 33
		8	beta 12
		9	beta 13
		10	beta 23
		11	population parameter
		12	anomalous population parameter
		13	neutron scattering factor
		21	atom multiplicity for atoms in special positions
		22	xray scattering factor pointer as a packet sequence number of LR 8.
		23	temperature factor type (0=overall;1=iso; 2=aniso)
		24	atom-group key for group refinements
		25	model-refinement key for refining different models

LR 17 Std Dev in Atom Parameters (directory)

Log rec	Packet size	Ident. number	Directory in packet 1, first atom s.d. in packet 2
17	varies	1	sigma x
		2	sigma y
		3	sigma z
		4	sigma b
		5	sigma beta 11
		6	sigma beta 22
		7	sigma beta 33
		8	sigma beta 12
		9	sigma beta 13
		10	sigma beta 23
		11	sigma of population parameter
		12	sigma of anomolous population parameter
		13	sigma of neutron scattering factor

LR 18 Refinement Constraints (directory)

Log Packet Ident. Directory in packet 1, first constraint in
rec size number packet 2
18 varies

Note: The general form of the constraint equation is...
p(s)*f(s)=q+p(r1)*f(r1)+p(r2)*f(r2)+...+p(rn)*f(rn)

1	packet sequence number of subject atom in logical record type 11
2	parameter identification number of subject atom
3	multiplication factor of subject parameter
4	constant Q in constraint equation
5	constraint classification key
6	site multiplicity of subject atom
11	packet sequence number of the reference atom 1
12	parameter identification number of reference atom 1
13	multiplication factor for parameter of atom 1
21	packet sequence number of reference atom 2
22	parameter identification number of reference atom 2
23	multiplication factor for parameter of atom 2
..	. . . . .
n1-n3	packet, parameter, and mult. factor for atom n

LR 19 (spare)

LR 20 Reflection Information (directory)

Log Packet Ident. Directory in packet 1, first reflection
rec size number in packet 2
20 varies

* numbers 1- 999 identify crystal-specific data
* numbers 1000-1999 identify data-set 1 information
* numbers 2000-2999 identify data-set 2 information
* . . . . . .
* numbers n000-n999 identify data-set n information

Crystal specific

1 Miller indices packed word, with bit pattern

29-21	20-12	11-3	2-0
/h/	/k/	/l/	sign code (see belowa)

2 sin(theta)/lambda

3 reflection multiplicity and reinforcement factor
9-5 epsilon
4-0 hkl mult.

4-15 equivalent indices packed table (up to 12 words). The table appears in sets of *two* words.

*** word1 describes index magnitudes
29-21 /h/
20-12 /k/
11-3 /l/
2-0 no. of sign/phase codes in word2

*** word2 describes the index signs and phase shifts

23-21 phase code4 sign code4
20-18 phase code4 sign code3
17-15 phase code3 sign code3
14-12 phase code2 sign code2
11-9 phase code2 sign code1
8-6 phase code1 sign code1
5-3 phase code0 sign code0
2-0 phase code0 sign code0

code	sign hkl	phase-shift degrees	cycles
0	+++	0	0.00000
1	++-	60	0.16667
2	+-+	90	0.25000
3	+--	120	0.33333
4	-++	180	0.50000
5	-+-	240	0.66667
6	--+	270	0.75000
7	---	300	0.83333

501 interpolated scattering factor for atom type1
502 interpolated scattering factor for atom type2
... .
510 interpolated scattering factor for atom type10

The 700 numbers are used to store estimated phase sets for the "native" structure or "parent" substance

700 Figure of merit; weight of the 'best' Fourier coefficient
701 cos alpha for the 'best' Fourier coef.
702 sin alpha for the 'best' Fourier coef.
703 cos alpha most probable
704 sin alpha most probable
705-709 alternate phase set 2

795-799 alternate phase set 20

*

```
                * n000-n199 identify measurement parameters
                * n200-n299 identify reduction parameters
     for all    * n300-n499 identify reduced structure
                            factor data
   data sets    * n500-n599 identify Hendrickson coefficient
                            data
        n       * n600-n699 identify normalized s.f. data
                * n700-n799 identify structure factor
                            phase data
                * n800-n899 identify refined structure
                            factor data
                * n900-n999 identify refinement parameters
                *
```

data-set1

1000	total gross counts
1001	total background counts
1002	ratio of scan to background time
1003	net counts
1004	sigma(net counts)
1005	phi diffractometer angle in cycles
1006	chi or kappa
1007	omg
1008	2th
1009	2th scan range
1010	omg scan range
1200	absorption weighted mean pathlength tbar
1201	absorption correction factor to irel
1202	extinction correction factor to irel
1203	thermal diffuse scatt. correction factor to irel
1204	1/lp factor
1205	irel scale factor to scale counts to irel
1300	relative intensity (irel)
1301	sigma(irel)
1302	relative f squared (f2rel)
1303	sigma(f2rel)
1304	relative /f/ (frel)
1305	sigma(frel)
1306	relative /f/ friedel related -h,-k,-l (frel*)
1307	sigma(frel*)
1308	rcode reflection status key (user designated)
1309	scale group number
1501-1504	A,B,C,D Hendrickson coefficients Phase probability distribution (isomorphous)
1505-1508	A,B,C,D Hendrickson coefficients Phase probability distribution (anamolous)
1600	normalized structure factor 1; assuming random atoms
1601	normalized structure factor 2; with fragment information

1602	expectation value for f**2; assuming random atoms
1603	expectation value for f**2; with fragment information
1604-1630	group s.f. in sequence designated by LR 17 (ID 10)
1631	weight of s.f. phase estimate 1 with id 1701
1632	weight of s.f. phase estimate 2 with id 1702
....	. . . . .
1694	weight of s.f. phase estimate 64 with id 1764.
1700	current structure factor phase estimate (in cycles)
1701	structure factor phase estimate 1 (in cycles)
1702	structure factor phase estimate 2 (in cycles)
....	. . . . .
1764	structure factor phase estimate 64 (in cycles)
1800	calculated /f/ (fcal)
1801	A sum normal S.F. only (=/f/cos(phase))
1802	B sum normal S.F. only (=/f/sin(phase))
1803	A dispersion contribution only
1804	B dispersion contribution only
1805	A total excluding extinction correction
1806	B total excluding extinction correction
1807	translation function coefficient
1810-1817	partial structure factor values in order 1800-1807
1900	least squares weight last used
1901	least squares weight1
1902	least squares weight2
1903	least squares weight3

LR 21 (spare)

LR 22 (spare)

LR 23 (spare)

LR 24 (spare)

LR 25 END-OF-FILE Record (specific)

Log rec	Packet size	Sequence number	Description of contents
25	0		This record serves as EOF signal to nucleus

A-5.2 Physical Structure of the Binary Data File

The physical structure of the BDF on the output or input device is not of particular importance to the XTAL user or programmer. This is because the XTAL nucleus routines handle all the bookkeeping operations and return data in terms of logical records and packets. However, for those who wish to write their own BDF drivers, a brief description of the BDF structure follows. The length of all logical records is determined solely by the crystal and amount of information it contains. Storage requirements in direct-access memory force certain physical constraints on the maximum number of words that can be output or input to or from an I/O device at one time. The memory reserved for this transfer is referred to as the I/O buffer, and in the XTAL system these buffers are located in the data array QX( ). The length of these buffers is specified by the macro (BINSEQBUF:) when XTAL is implemented. This value will depend on the core available, and other hardware constraints, such as the disc track length. Once the buffer length has been set for a given installation, it must not be changed. To optimize the transfer of the binary data file to and from the fixed length I/O buffers, it is necessary to both pack and position logical records according to length. This operation, in turn, requires that three additonal words at the front of each logical record or buffer are used for bookkeeping purposes. These three floating point words are referred to as lead words and set in the following way:

lead word 1 is the length in floating point words, including the three lead words, of the part or all of a given logical record in this buffer. The end of a buffer is signaled when the first word following a record has the value of +1. or -1. The +1. signals that the preceding logical record does *not* continue into the next buffer. The -1. signals that the preceding logical record is incomplete and continues into the next buffer.

lead word 2 is the logical record type number (1 to ENDRECORD:). This number is negative when the last part of a logical record is in the current buffer. It is positive when more of the logical record follows in the next buffer.

lead word 3 is the packet size in floating point words for the given logical record.

REFERENCES

1. Ahmed, F.R., ed., Crystallographic Computing, Proceedings of the 1975 Summer School on Crystallographic Computing, Munksgaard, Copenhagen, (1970).

2. Hall, S.R., and Stewart, J.M., TR-700, The XTAL System of Crystallographic Programs: Guidelines for Authors, Computer Science Center, Univeristy of Maryland, College Park, Maryland, (1978).

3. Kernighan and Plauger, Software Tools, Addison-Wesley Reading, Massachusetts, (1976).

4. Munn, R.J., and Stewart, J.M., TR-675, RATMAC: Kernighan and Plauger's Structured FORTRAN Programming Language, Computer Science Center, University of Maryland, College Park, Maryland, (1978).

5. Munn, R.J., and Stewart, J.M., TR-804, RATMAC Primer, Computer Science Center, University of Maryland, College Park, Maryland, (1979).

6. Stewart, J.M., et al., Technical Report 67-58, X-Ray 67 Program System for X-Ray Crystallography, Computer Science Center, University of Maryland, College Park, Maryland, (1967).

7. Stewart, J.M., et al., Technical Report TR-192, The X-ray System of Crystallographic Programs, Computer Science Center, University of Maryland, College Park, Maryland, (1972).

8. Stewart, J.M., et al., TR-446, The X-Ray System of Crystallographic Programs, Computer Science Center, University of Maryland, College Park, Maryland, (1976).