IML - MIR
(INTERMEDIATE MUSICAL LANGUAGE-MUSIC INFORMATION RETRIEVAL)

USERS' MANUAL

T. HALL
P. H. PATRICK
J. SELLECK

JUNE 1972

The following is intended as a guide for anyone interested in importing the I.M.L.-M.I.R. system. But it would be best (perhaps even necessary) for anyone who desires to import it to spend a few days at the Princeton University Computer Center.

Data Available

Cards:

20 masses of Josquin. (Fully corrected (?))
Fragmenta Missarum and some 60 Motets. (Uncorrected)

Tapes:

I.M.L. data tape. (Contains card images of 20 masses)
M.I.R. data tapes:-
(i) 20 masses each as a separate file
(ii) 20 masses combined into one file.

I.M.L.-M.I.R. system programs

(Time and core estimates are those required by the IBM 360/91 computer at Princeton. (8 bit byte, 4 byte word.))

(i) I.M.L. conversion program; approx 3500 cards, in Fortran except for Hale Trotter's assembler I/O routines. Requires 180K bytes of core, average running time for conversion of a mass 0.5 to 1.0 minutes depending on the size of the mass.

(ii) M.I.R. subroutines (to which the user appends his own specific program); approx. 1500 cards, in Fortran except for Hale Trotter's assembler I/O routines. Core size (minimum of 120K bytes) and running time dependent on the complexity of the user program.

(iii) Miscellaneous updating programs.

At present, future development of the I.M.L. code and conversion program is doubtful. The program is bulky and cumbersome and it might be better to do any future coding in a more widely used language (Ford-Columbia ?) and then write a new conversion program. (The coding, proofreading, and correcting of data constitute a long tedious process full of possibilities for minor errors, which when undetected cause

much programming time to be wasted. These difficulties should be reduced by improved coding techniques and eventually eliminated by optical-scanning devices.)

The M.I.R. subroutines, however, can be easily grafted onto any system; they are independent of the intermediary code, the only restriction is that the converted data be packed onto the tape (or disk) in a specific format. It is on the M.I.R. subroutines that much development can and should be done. True, at the moment we are limited to the music of Josquin, but our greatest need at present is the development of our music-theoretical knowledge and abilities to help us know what questions to ask and how to interpret the results. These techniques can be developed while we await more data.

September 1972

I.M.L. - M.I.R
(Intermediate Musical Language - Music Information Retrieval)

Users' Manual

T. Hall
P. H. Patrick
J. Selleck

June 1972

Foreword

A number of the features of this system would be puzzling to anyone who did not know something of its history.

The original idea for the whole system was Michael Kassler's, but many people have worked at it over the years. The roles of the undersigned in the early stages were mainly confined to encouraging a number of enthusiastic students in their efforts to develop such a system - chiefly Michael Kassler, Tobias Robison, and Hubert Howe. We chose the body of music to which it was first to be applied:- the Masses of Josquin Desprez, to which we later decided to add the Motets*. Neither of us had any knowledge of computers or programming at the outset, and we have up to the present considered our function to be mainly that of making clear the kind of question we should like the system to be able to answer, leaving it to the students, and later to John Selleck, to devise the means by which the answers were to be arrived at. But in the long run, to get the maximum help from the system, the scholar should doubtless be able to write his own programs.

The I.M.L. System

It was clear at the outset that creating a code into which musical notation could be translated for feeding into the computer would be a shorter task than working out the information retrieval system. Hubert Howe and Alexander M. Jones created this code, and described it in I.M.L., an Intermediary Musical Language (1964, mimeographed; out of print). It had two basic premises:

(1) that it need never deal with more than one note at the same time on any single staff; and
(2) That it should be easy as possible to learn, for anyone who could read music in the treble clef.

The first of these premises embodies a limitation which we

---

* At the time of writing we have Masses 1 - 18 of the Smijers Edition fully corrected (?) and converted onto tape, and Masses 19 - 20 and most of the Motets coded but only partially proofread and corrected.

were quite willing to accept. It obviously simplified coding, and it seemed sensible to us to see what help the computer could give us with a considerable but limited body of music, which has a certain homogeneity of notation and concerning which each of us had many questions to ask. The second premise, or at any rate some of the conclusions that were drawn from it, now seems less well advised, and is responsible for some of the faults of I.M.L. In practice, the coding has always been done by music students, so such requirements as that it should be possible to code everything as if it were in treble clef have turned out to be unnecessary, and other features of the language have proved to have serious disadvantages. But before we learned these lessons we set students to work coding the Josquin works, and we had some 1200 pages of music coded on punch-cards long before we had a working system for obtaining the information that we wanted to extract from them.

Not all the key-punchers understood the Jones-Howe instructions in exactly the same sense, and those instructions, naturally enough, did not cover every contingency that was to arise in the coding. To meet unforeseen situations, ad hoc decisions were made by individual key-punchers, the results of which later had to be grafted into the system. Some key-punchers coded the time-signature ¢ as C/; some as C-. Some were consistent in where they left spaces (which the computer considered as blank characters); others interpreted the instructions more freely in this respect. So when the retrieval system was worked out - partially by Tobias Robison, for the I.B.M. 7094 computer, and described by him as he envisioned it in Tobias D. Robison, "IML-MIR: A Data-Processing System for the Analysis of Music", in Harald Heckmann (Ed.), Elektronische Datenverarbeitung in der Musikwissenschaft, Regensburg, 1967, pp. 103-135; rewritten and completed in Fortran IV by John Selleck, with revisions by P. Howard Patrick and Thomas Hall, in the form herein described - provision was made in it for interpreting the I.M.L. coding as if many of these decisions could be made at the option of the key-puncher, as indeed in the current version of I.M.L., herein described, many can be. That is - rather than try to make all the data uniform in these respects, M.I.R. was constructed so as to interpret the data correctly whichever option had been used. But for anyone starting afresh to keypunch music in I.M.L., we strongly recommend, to avoid confusion in proofreading of the coded data, that wherever this manual offers optional ways of coding a single way be

adopted.

No one would assert that if we were beginning the project today, with what we have learned in the meantime, we would use a system as cumbersome and with as many likely occasions for error as I.M.L. The system is here presented in the form into which it has evolved. It works, and it is presented for what it is worth. We expect that it will continue to evolve, and we hope that its inconsistencies and difficulties will be reduced.

The M.I.R. System*

The M.I.R. system performs the basic retrieval operations with considerable agility. However, the dozen or so retrieval routines available to the user are only a skeleton. The M.I.R. system only retrieves information; what information should be retrieved, and how the results should be interpreted are the major problems which face the user. It consists of a few basic retrieval routines plus a few more that were worked out as parts of programs designed to answer specific questions but that may be useful for other purposes as well.

The reader of this manual must know Fortran IV; its terms and procedures are not explained here. Fortran IV was chosen as the most widely used computer language, and the one least dependent on a particular type of computer. Every term or expression here typed all in capital letters is explained either in this manual or in a Fortran manual or textbook. But of course every computer installation has its own conventions, with which the user will have to familiarize himself in order to apply the I.M.L.-M.I.R. system.

Lewis Lockwood

Arthur Mendel

---

* It should be pointed out that the M.I.R. system is not dependent on I.M.L. coded music. With some programming expertise any form of coded music can be used as a data source.

Table of Contents

Chapter 1

Intermediate Musical Language

The Intermediate Musical Language(I.M.L.), designed originally by Jones and Howe in 1964, has undergone numerous corrections and additions since its first formulation for the 7094 system. Most of the changes, however, were not caused by the change of computers, but arose from working with the I.M.L. - M.I.R. (Music Information Retrieval) system in general. The language is not as simple as conceived originally and care is taken here to state explicitly all the rules which must be followed without deviation to code music in an I.M.L. version which will be correctly interpreted by computer programs. For purposes of reference, some statements will be repeated many times wherever they are applicable. The present description concerns the language as it now (1972) exists for the 360 computer programs.

As mentioned in a previous writing by Jones and Howe, I.M.L. is not a language for handling simultaneities on a single staff. It can consider only a set of monodic voices, and is particularly suitable for vocal polyphony of the Middle Ages and Renaissance. It was designed, in fact, with only this music in mind. With some effort it can be made to function in a limited way for other music.

Section and Lyne

In this manual, the word section means one of the five main divisions of the mass (Kyrie, Gloria, Credo, Sanctus, Agnus) or one of the parts of a motet (Prima Pars, Secunda Pars, etc.), while sub-section means any portion of a section that is separated from another portion of the same section by a double bar, (e.g., Christe, Qui sedes, Confiteor, In nomine, Agnus 2).

The term lyne is more or less synonymous with "part" or "voice", but each lyne is identified by order-number rather than by name. The order is from the highest to the lowest in a system of music, the terms "highest" and "lowest" referring only to the position in the graphic representation on the page, not necessarily to relative pitch. Lyne 1 in one section of a composition may be the SUPERIUS, while in another section lyne 1 may be the ALTUS, each being simply the top part of the music being encoded.

Musical notation when encoded in I.M.L. consists of a string of symbolic data typed on consecutive punch-cards. There are specialized symbols and symbol-groups of two kinds: program-analysable and non-program-analysable.

Non-program-analysable comments are included merely for the reader's convenience to explain some aspect of the coding. They must be enclosed in single quotation-marks.

Program-analysable comments are of two classes: verbal symbols and non-verbal symbols.

a. Verbal Symbols

All such elements must, like non-program-analysable comments, be enclosed in single quotation-marks. Immediately following the opening quotation-mark comes an equal-sign, and immediately following that a specific keyword, which identifies the type of information contained. A single space (blank character) must occur after the keyword, and usually no spaces can occur in the information part of the comment. Exceptions to this will be noted. In coding for the computer, spaces must be thought of as blank characters, since they are sometimes required, sometimes prohibited, and sometimes optional. To differentiate an optional space from a required space, the latter is always represented in this manual by the letter b. In actual coding, a space (blank character) is used; no lower-case letters are used in I.M.L.

In the keypunching of the Masses and Motets of Josquin des Prez, a convention of punching only in the first 70 card columns was maintained for a while; then some were punched up to column 72. Either course may be taken now, but the former is preferable; if columns 71 and 72 are keypunched they must not be used for a significant blank character in a program-analysable comment. Column 72 on one card is directly succeeded by column 1 on the next card, and except as regards the prohibition just mentioned they may be thought of as members of a continuous series. Further important exceptions are described in b.1 and b.13 below. It is strongly recommended that a new card begin only where blank characters are optional.

Except where otherwise indicated, keywords refer to everything that follows (i.e., their application holds over from one sub-section to the next unless they are replaced by

other keywords). In the reformatted printout produced by the conversion process (see Chapter II), each keyword appears on a separate line. It is a good idea to punch them this way also, i.e., one comment per card.

KEYWORDS

1. TITLE:

   This is the keyword which precedes a single space and the title of the composition in question (which may include spaces)* terminated by a ' character; e.g.:

   '=TITLEbMISSA GAUDEAMUS'

   The actual title (after the keyword) can be at most 44 characters long.

2. COMPOSER (32 characters), AUTHOR (32 characters), PUBLISHER (24 characters), PLACE (16 characters), EDITOR (24 characters), and KEYPUNCHER (28 characters), are other keywords for comments similar to TITLE and SUBTITLE. Spaces may occur in the informational part of any of these comments. **

3. SUBTITLE:

   This keyword indicates that the name of a sub-section of a composition will follow (after an intervening space), e.g.:

   '=SUBTITLEbKYRIE 1'

   the actual subtitle (after the keyword) can be only 24 characters long. Spaces may occur in the informational part of this comment.

4. LYNE:

   The names of all the lynes in the sub-section to be encoded must be listed after the keyword LYNE plus a single space, as in the following example:

   '=LYNEb1-VIOLINb2-VIOLAb3-CELLO'

---

*A complete statement of where space is required, where not permitted, and where optional appears at the end of this chapter.

** Provision has been made for additional information if there is reason to include it; see APPENDIX 3A, p . (Page-numbers have been left blank for the user to fill in, since whenever any revision of this manual is made the page-numbers may change.)

As regards format, the above example must be followed; the instrument- (or voice-) names may of course vary. No spaces are allowed in the lyne-name, which can be a maximum of 12 characters. (If a space were key-punched, as for example in QUINTA VOX the second word would be omitted in the conversion.) Between the end of a lyne-name and the number of the next lyne, at least one space must occur. There must be a hyphen between the lyne-number and the lyne-name.

5. TUCK:

This keyword occurs alone in the comment:

'=TUCK'

to indicate that the encoding of the staff-positions of the notes will be done using the diatonic letter-names indicated by the actual clef in the music. Otherwise, the coding of all lynes is done as if all clefs were treble-clefs (see b.2 below, p).

6. UNTUCK:

This keyword alone in the comment:

'=UNTUCK'

cancels the effect of the TUCK comment. After it, the notes must be keypunched as if they were in the treble-clef.

Keywords 7-11 differ from the preceding in that each governs only the particular lyne with which it is associated.

7. CLEF:

This is the keyword for a comment referring to the clef of a particular lyne being coded. After an intervening space, the letter G for treble clef; Cx for all C-clefs, where x stands for the number of the line of the staff (counting from the lowest up) which is middle-C in the particular clef (e.g. soprano clef would be coded '=CLEFbC1'); F or F4 for bass clef; F3 for baritone clef; F5 for contra-bass clef; UG for

treble clef with 8ve-higher meaning; LG for treble clef with 8ve-lower meaning; and G1 for French-violin-clef -- are the only possible indications. No spaces are allowed except one after the keyword.

8. TIME:

This is the keyword for a comment referring to the time-signature of the lyne being coded. After a single intervening space, the time-signature, if in "fractional" form, is coded as the "numerator" followed by a slash (/) followed by the "denominator", e.g.:

'=TIMEb4/4'

if the time-signature is in some other form, i.e. as a single numeral, a letter, or a combination of letter(s) and numeral(s), it must be enclosed within parentheses. Without a RETIME comment (see 9 below), such time-signatures will have no meaning for the conversion program with the exception of C, C-, or C/ (note the two equally permissible ways of representing ¢, some key-punchers having used one way and some the other), all of which are interpreted as being equal to 2/1, unless specified differently by the RETIME comment. No spaces are allowed except after the keyword.

9. RETIME:

This keyword followed by a space indicates information is to be given as to the value of some following time-signature, that time-signature being one enclosed within parentheses, as mentioned above. The comment has a form like the following example:

'=RETIMEb(C-) = 1/1'

This will cause C- (but not C/) to be interpreted as meaning 1/1 instead of 2/1, or any other previous value for this particular time-signature. Besides a single space after the keyword, which must occur, any number of blank characters on either side of the second = symbol may or may not be included. (Note that RETIME is not actually a specific-program-analysable comment, but rather a

general-program-analysable comment. Its effect is not restricted to a particular lyne, but defines the meaning of a time-signature which can appear in any lyne. It was included here since its description is better understood after reading the specifications for the TIME comment.)

10. REDUCE:

In the process of encoding I.M.L., one may run across cases of two different meters in different places where the measures in both cases are meant to be the same length. This may occur as two different meters in different lynes at the same time, or between successive passages in a composition. In the former case always, and in the latter case at the discretion of the keypuncher, or in accordance with his instructions, a REDUCE comment is used to indicate the 'true' value of the durations of the notes in one meter in relation to those in other meters. E.g.:

'=REDUCEb1-3/2'

would be the indication signifying that in lyne 1 the durational values of all the notes are to be interpreted as two-thirds of their written value.

The comment:

'=REDUCEb1-2/3'

nullifies the effect of the first comment. No spaces are allowed in REDUCE comments except the required one.

11. KEY:

This keyword indicates that information about 'key-signature', or the permanent accidentals for a lyne of music will follow. Two methods are employed in indicating the 'key', or what is to be considered the 'key', of a given lyne of music from some point on in a composition.

The form:

'=KEYbxyyyyy'

is used where x is the tonic of the major key

which in tonal music would be indicated by the 'key-signature'. The y's stand for FLAT or SHARP, but since the computer will disregard all except the first letter of each of these words, abbreviation to F or S is optional. E-g., C, EFLAT, CSHARP, BF, FS would all be valid key-signature representations.**

The "key-signature" may also be represented by a list of the actual sharps and/or flats involved. E.g.:

'=KEYb(AS,BT,GS)'

where the (hypothetical) signature consists of sharps on A and G and a flat on B.** They must be enclosed in parentheses and separated by commas.

No spaces are allowed in either form of the comment. The absence of any initial KEY comment for a lyne of music means that "C-major" will be assumed, i.e. no accidentals. A "key-signature" of C-major must be stated, however, if it is to replace a previous indication in the lyne which was not C-major.

** It is only in the representation of 'key-signatures' that 'flat' is represented by F; everywhere else it is represented by T.

b. Notational features indicated by non-verbal symbols

1. Lyne- and measure- numbers:

Always the first indication within any representation of a staff, these are indicated as numbers enclosed within dollar-signs in the following manner:

mn$

where m is a number (maximum 2 digits) assigned to the lyne (see a.4 above) and n is the measure-number (maximum 3 digits). The first measure in the top part of a score would be indicated as $1$1$. The numbering of the measures in any subsequent sub-section may continue the series of the previous sub-section or may begin with 1, at the key-puncher's option.

Spaces are allowed between the $ symbols and the numbers they enclose, but not between the digits (e.g. $3$1 0 2 $ is incorrect, but $ 1 $ 345 $ is allowed). The strings of symbols for this indication must appear on a single card, i.e., they must not be divided into two parts, each part being on a different card. It is a good idea to keypunch this feature alone at the beginning of a new card. This will facilitate location of the particular staff on a printout of the cards.

Whenever a comment referring to clef, time-signature or "key-signature" occurs in the course of a lyne, the measure in which it occurs must be treated as if it were at the beginning of a new system, even if this is not the case in the printed music. The lyne must be ended at this point with an end-of-staff barline (see b.11 p), and a new mn$ indication followed by the comment indicating the change must precede the code for the remaining notes in the staff. I.e., no staff encoded in I.M.L. can contain internal changes of clef, "key-signature", or time-signature. And if such a change occurs in the score, arbitrary division-points must be made in the I. M. L. coding. All lynes of a sub-section may be broken at the same point, but this is not necessary. That the notes in any one

lyne be in order is all that is required.

2. Staff-position:

Staff-position in I.M.L. is indicated by the letter-names of the notes:

A B C D E F and G

For those not thoroughly familiar with other clefs, in order to facilitate keypunching, all staves are coded as if they were written in the treble clef. (But see a.5 and a.6 above concerning TUCK and UNTUCK conventions.) A rest is indicated by the letter R (see b.10 below).

3. Accidentals:

To indicate an accidental, one of the following characters is used:

N for natural
S for sharp
SS for double-sharp
T for flat
TT for double-flat
NS for natural and sharp
NT for natural and flat

In the 16th-century music for which this language was originally designed, the situation is often encountered in which an editor has suggested that certain notes receive accidentals which are not indicated in the original sources of the work. These are usually placed above the note which they govern. Such accidentals are indicated in I.M.L. by enclosing the characters N for natural, S for sharp, or T for flat (only these indications are possible) within parentheses. The specification of an accidental must immediately follow the symbol for the staff position of the note in question.

4. Durations:

Durations are indicated by numbers which are shown in the following table of equivalents*:

---

*In the following discussion, the word note is used to include rests. See 10 below concerning the durations of

Double-breve 111
Breve 11
Whole-note 1
Half-note 2
Quarter-note 4
Eighth-note 8
16Th-note 16
32Nd-note 32
64Th-note 64
128Th-note 128
Grace-note 0

5. Register

The indication of the register to which a note belongs is governed by the following set of conventions:

(a) The normal register for the treble-clef, or any clef whose notes are being interpreted in UNTUCK mode, is defined as extending from the first line to the fourth line of the staff (first line being the bottom line).

(b) The indication U is placed immediately before the staff position of a note in the register one octave above the normal register, or above the register of the immediately preceding note. UU is coded for two octaves above the previous register; UUU, for three octaves, etc. L is placed similarly if the register of the current note is one octave lower than the previous register; LL if two octaves, etc.

(c) The register of each note is governed by that of the preceding note, unless there are indications (some L's or U's) to the contrary.

---

rests. Dots, which indicate that the duration of a note is to be extended by half the value indicated by the immediately preceding symbol (i.e., any number in the right-hand column above, or dot), are represented in I.M.L. by periods which follow one of the numbers above. The indication of the duration of the note must follow that of its staff-position and accidentals (if any). More than one dot for a note would be represented by as many periods.

Once the indication U appears before a note, that register now governs all successive notes in the same lyne until an end of staff is reached, or until it is cancelled by an L, as it must be when a change to the next lower register occurs. That is, except at the beginning of a staff, where the normal register is assumed, the indications U and L are relative to the register of what has immediately preceded. So if, for example, a lyne rises one note above the normal register (indicated by U before the letter-name) and then returns within the normal register, this must be indicated by an L.

(d) At the beginning of any staff (real or encoded as such in I.M.L. because of a change of clef or signature) the normal register of whatever clef is in force for the lyne is assumed, irrespective of the register of the immediately preceding note.

Normal registers for clefs which are read according to the TUCK convention are as follows:

(i) G-clefs -- first line to fourth line

(ii) C-clefs -- whatever line represents middle-C to the line representing the first B above. (This has the effect of putting most of the notes in the tenor clef below the normal register.)

(iii) F-clefs -- from the space representing the C below middle-C to the space representing the B just below middle-C.

6. Text-syllables:

These are coded after the duration indication, and are represented in I.M.L. by any string of alphameric characters except commas, enclosed by commas. The comma itself, when it occurs in the text, is replaced by the + symbol. Italics are indicated by the number 8 placed before each syllable, except when syllable are connected by hyphens in which case only the first syllable need be preceded by the character 8. The 8 may be followed by a space, but need not be.

7. Triplets:

Triplets are indicated in I.M.L. by a + symbol followed by a 3 in parentheses. The triplet group extends from the note immediately after the +(3) to the note just before a solitary + symbol. (Remember that the word note includes the meaning "rest".) The +(3) comes at the beginning of all the information about the first note with no spaces intervening; the + indication comes just after the last note with no spaces intervening. A note-symbol-group* containing the + indicating the beginning or the end of a triplet-group must not include any blank characters, and must be separated from adjacent note-symbol-groups by at least one space.

This feature of I.M.L. is translated by the conversion program into a proportional reduction in the durational values of the notes concerned. I.M.L. contains provisions for other groups of notes which subdivide metric values by a number different from what would be normal, as well as for nested groups, but since they are not applicable to music around 1500 they are not described here. (See Appendix 1B)

8. Ties:

The symbol which is used to indicate the beginning or the end of a tie is *. An asterisk representing a tie is always the very first or the very last indication in the representation of the characteristics of a particular note, depending on whether it stands for the beginning of a tie or for the end. If a triplet-indication occurs on the same note as that for a tie, the information for the tie precedes that for the triplet at the beginning of the note, and follows it at the end.

There must be no intervening spaces in the note-symbol-group containing asterisks, and there must be at least one space separating a note-symbol-group containing the tie indication from the groups on either side. This space is the

---

* I.e., the group of all symbols pertaining to a single note, together with its text, if any, and any indicators of a fermata, ligature, tie, triplet and/or tag.

only way to distinguish asterisks coming at the beginning of a note-symbol-group from those coming at the end of the previous note-symbol-group.

In the Smijers edition of the works of Josquin there sometimes occur "suggested" ties, which are indicated by a broken line. These can be coded in I.M.L. by the use of parentheses around each asterisk involved. (If the multiple asterisks method of coding is employed (see next paragraph), the parentheses would surround a group of asterisks).

For example:

Insert 1

Would be coded as:

(*)A11,A,/*A11(*)/A1*bG1,MEN,/

The fact that a note is connected by ties to both the preceding and following notes is indicated by asterisks at both the beginning and the end of the group of symbols for that note's representation in I.M.L. It is possible to distinguish which tie is which by the use of multiple asterisks. This is not necessary, and makes no alteration in the interpretation of what is actually going on. For example, the above passage could be represented as:

(*)A11,A,/**A11(*)/A1**bG1,MEN,/

9. Dynamics:

No provision has yet been made for the inclusion of dynamics, accents, and other expressive indications except as non-analysable comments. If they occur, the note-symbol- group at which they are placed should be followed by the symbol =, this followed by the symbol for the expressive indication, and this in turn followed by another equal-sign.

10. Rests:

A rest is indicated in I.M.L. by the letter R followed by a number which indicates the duration (see b.4 above). Other indications are possible, such as those for triplets. Note that very often in the Smijers edition, for example, a breve rest is encountered which indicates a rest for an entire measure in 3/1 time. The rest is undotted, even though a note cccupying a whole measure would be dotted. Such a rest is represented by R11. A single rest occupying an entire measure is always interpreted as having the duration of the whole measure. (Therefore a triple breve rest when encountered can be coded as dotted double breve, even though in the musical notation it does not have a dot.)

There is at present no method in I.M.L. for indicating in abbreviated form the occurrence of several whole measures of rest in succession. If twenty measures of breve rest are encountered, each measure must be coded individually.

11. Barlines, end-of-staff barlines, and end-of-subsection barlines:

A barline is represented in I.M.L. by the character /. The end-of-staff barline is indicated by two slashes in succession. The end-of-sub-section barline, which is usually a double-barline in the music, is represented by three slashes in succession. (Spaces are optional).

The comment:

'=FINALbMOVEMENT' or '=FINALbSUB-SECTION'

will cause the normal three slashes at the end of a sub-section to be considered as four, representing the end of a complete section (of a Mass, say). This comment should appear immediately preceding the triple-slash of lyne 1 (only; not needed in other lynes).

12. Ligatures and Fermatas:

For a ligature, the following indication is used:

'$BRx'

where x indicates the number of notes connected by the bracket indicating the ligature (tied notes being counted as one note).

For a fermata, the indication

'$FERM'

is placed before the note-symbol-group which it governs, and before the bracket if any. No spaces are allowed in either "comment", but if both occur they should be separated by a space, and they should be similarly separated from the note-symbol-group which they precede.

13. End of input features:

The last card in the input deck for a single composition consists of $$$$$$ in the first 6 columns, to signify that the information about the composition has been completed. The last card in the whole input must have FILEbEND, beginning in column 1 of the punch-card.

14. "Sequencing" the cards:

All cards of I.M.L. input must be "sequenced" (i.e., given serial numbers) in columns 73 to 80, including the card with $$$$$$. For one composition each sequence number must be higher than the preceding. It is a good idea to use multiples of at least 10 when sequencing the input cards to allow for later insertions. The sequence numbers in any individual composition are completely independent of those of any other.

15. Character-sets:

There are several characters on the 026 keypunch (used to keypunch most of the data with 7094 computer in mind) which have different values when their punches are read on the 360 computer. The following characters if keypunched on the 026:

( ) = ' + become respectively
% < # @ & when read by the 360

but they will be interpreted by the 360 programs

as the former. E.g. a '=TIMEb(C-)' comment keypunched on an 026 will appear, if the holes in the card are interpreted on the 029 (i.e., printed on the card), as @TIMEb%C-<@, but either form will be correctly interpreted as '=TIMEb(C-)' by the programs designed to convert the I.M.L. data into material for information-retrieval purposes.

16. Tags:

A colon (:), or any number of successive colons, occurring immediately before the diatonic letter in a note-symbol-group will indicate that this note is 'tagged' for some purpose decided by the user. Different numbers of colons may be given different meanings, e.g. : equals non-harmonic tone, : : equals accented syllable, : : : equals both, etc.

17. Spaces:

To sum up the circumstances under which spaces may or may not occur:

Every program-analysable comment has a single space after the keyword.

Program-analysable comments given in a.1-3 may have internal spaces in the informational content of each comment.

There must be at least one space before and after each LYNE comment (a.4), but the symbol-group for lyne-number and -name (e.g., 1-SUPERIUS) must have no internal spaces.

Comments a.5 and a.6 must have no internal spaces in their informational content.

Comments a.7 and a.8 must have no spaces except the required one.

The RETIME comment (a.9) may have spaces on either side of the second = symbol.

Comments a.10 and a.11 must have no spaces except the required one.

mn$ indications may have internal spaces, except between the digits constituting a lyne- or

measure-number. The symbol-group for a measure alone (n) must have no internal spaces.

There should be at least one space separating individual note-symbol-groups from each other, this condition is not necessary (except for triplet and tie indications) but helps to make I.M.L. ouput easier to read, but note-symbol-groups should contain no internal spaces and should not be split between two cards.

For text (b.6) a space is optional between the 8 and the syllable that follows it. If any spaces are coded in the text that goes with one note, they will be automatically deleted.

There must be at least one space separating a note-symbol-group containing the triplet (b.7) indication from the groups on either side. But no spaces must separate the symbol +(3) from the first note of the triplet group, or the last note of the triplet group from the + that follows it.

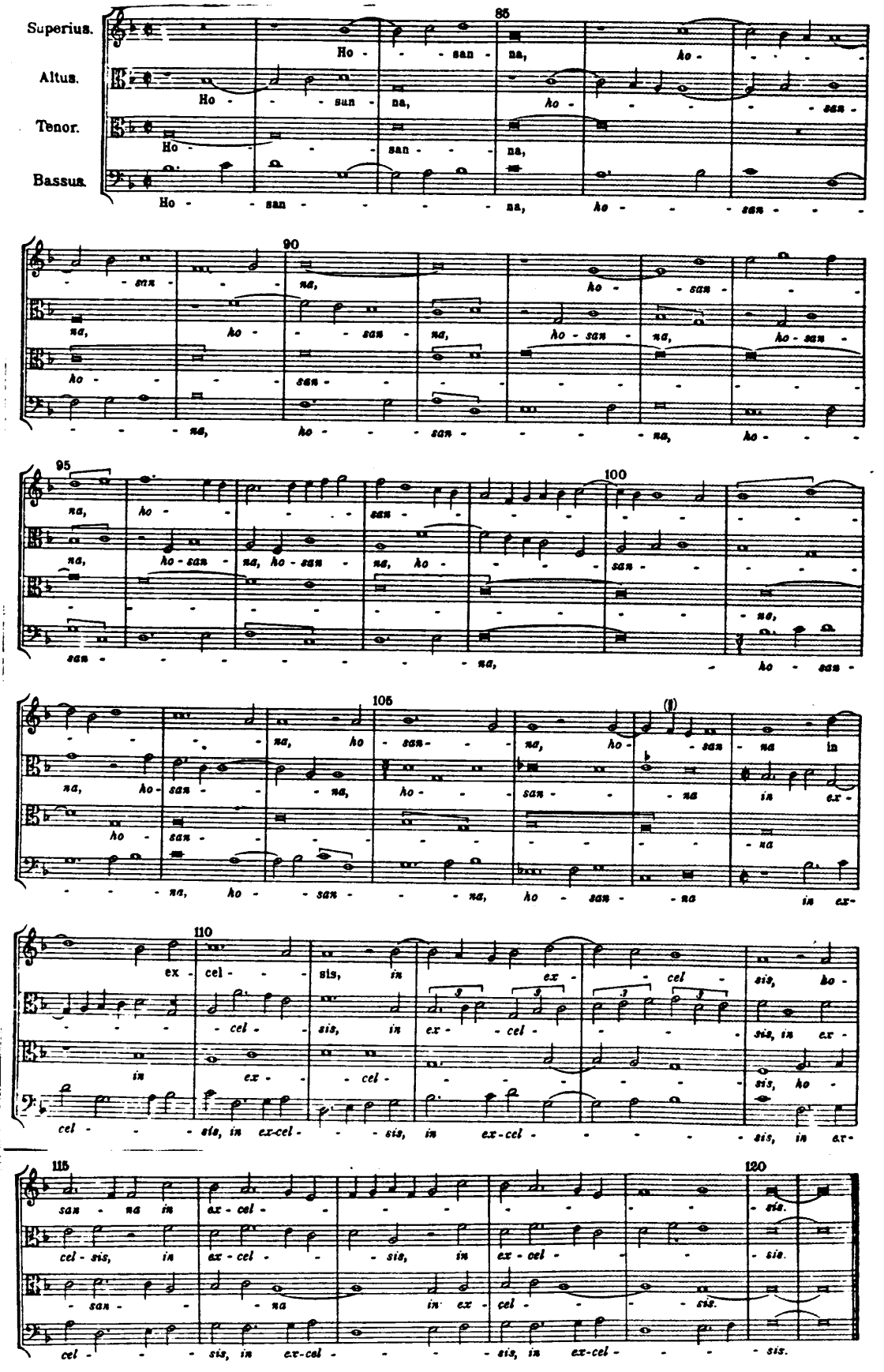
There must be at least one space separating a note-symbol-group containing the tie (b.8) indication from the groups on either side. But there must be no spaces within the note-symbol-group containing asterisks.

Spaces before, after, and between bar-line symbols (b.11), spaces are optional.

Ligature and fermata (b.12) comments should be separated by a space from each other (if both occur) and each should be separated by a space from the note-symbol-group it precedes. Neither comment can contain internal spaces.

$$$$$$ and FILEbEND indications (b.13) must contain no internal spaces, must always begin in column 1 of the punched card, and must be the only information on the card.

Insert 2


Superius.
Altus.
Tenor.
Bassus.
85
90
95
100
105
110
115
120
Ho - san - na, ho - san - na,
ho - san - na, ho - san - na,
ho - san - na, ho - san - na,
ho - san - na, ho - san - na,
ex - cel - sis, in ex - cel - sis,
san - na in ex - cel - sis.

To illustrate some of the features of the transcription of a piece of music into I.M.L., an example of music given in insert 2 from the Hosanna of the Missa Ave Maris Stella will be discussed at some length. It contains many features that one would run across in coding this type of music.

The first step in the coding of the example, or of any composition, is to make all the general program-analysable comments, referring to the entire sub-section. These include the comments mentioned under a.1 through a.6. At least one of the comments mentioned in a.1 through 3 must appear. The LYNE comment must appear. Comments a.1 through 3 can appear only at the beginning of a sub-section, but a.4 through 6 can appear elsewhere too.

The very first statements for the coding of insert 2 would be:

```
'=SUBTITLEbHOSANNA'
'=LYNEb1-SUPERIUSb2-ALTUSb3-TENORb4-BASSUS'
'MEASURE 82 HAS BEEN RENUMBERED AS 1 HERE'
```

$the names of all the lynes do not have to be given together, but lyne 1 must be defined in LYNE comment before the occurrence of material from that line in a mn$ character string. The code for the first lyne of music is:

```
$1$1$
'=CLEFbG' '=TIMEb(C-)' '=KEYbF'
R11/ R1b*B1,HO,/ B2*bC2 D1,SAN,/ G11,NA,/
R1b*C1/ C2*bB4 A4b*A1//
```

When there are no sharps or flats in the signature, no KEY comment is necessary, unless to cancel a previous KEY comment. (Note that the note-symbol-groups that contain tie-indications must include no blank characters and must be separated from adjacent note-symbol-groups by at least one space.)

Lyne 2, measure 21 (102 in the score) would be coded:

```
$2$21$
UF1 R2 F2/ LD2. B4b*B1/ B2*bG2 G1//
$2$24$
'=REDUCEb2-3/2' '=TIMEb3-1'
C1 F1 C1/ DT11 C1/ D(T)1 C11//
$2$27$
'=REDUCEb2-2/3' '=TIMEb(C/)'
```

```
A2. B4 C2b*F2//
```

Concerning the interruptions at mm.23-24 and 26-27, see b.1 above. The REDUCE statement is required since other lynes at this point are in ¢ meter.

The above was coded using the treble-clef convention. Continuation of the second lyne (m.109 in the score) using the TUCK convention would appear as follows (note the succession of triplets):

```
$2$28$
LG4*bA4 B4 UC4 D2 LG2/ A2 UA2. G4 E2/
F1. LB2/b+(3)B2. UC4 D2 LG2 E2 UC2/
D2 E2 F2 G2 D2 E2+/ F2 C1 F2//
```

The triplet-indication need not be repeated for each immediately succeeding occurrence unless desired.

The double-asterisk method of coding ties is illustrated in lyne 3, measure 7 (m.88 in the score):

```
$3$7$
'$BR2'bB11/ C11/ UE11/
'$BR2'bLD1 UE1/ *F11/ **F11*/ *F11**//
```

Note that ligature-comments come before the note-symbol-groups for the notes they govern.

The final system of the Hosanna, lyne 4, could appear as follows (in the TUCK convention):

```
$4$34$
A2 D2. E4 F2/ G2 F2. G4 A2/ D1 E2 F2/
G2 F2. G4 A2/ D1 E2. F4/  *G11/ G11*///
'HOSANNA ENDS HERE'
```

Spaces can occur between the elements of a note-symbol-group if that group does not involve ties or triplets. No space is required before the first asterisk (or double asterisk, or plus-sign) or after the second if a barline occurs there. Unanalysable comments can be placed anywhere they are appropriate.

The Conversion Process

Roughly explained, what happens is that the I.M.L. code indicates to the conversion program what the values of certain predefined categories, termed registers (the contents of which will be explained in Chapter II), are to be. Whenever a complete set of these for one note has been generated, it is written onto a tape which will be used as input data for the user's M.I.R. program. The tape can contain five different kinds of records. A record is all the information put onto the tape by any one command in the conversion to write onto the tape. The records are labelled HEDBLK, LYNBLK, NOTBLK, TALBLK and ENDBLK (N.B., these names are a carry-over from the 7094 computer 6-character-per-computer-word situation; the 360 allows only 4 characters per word and the actual labels for these records are discussed under the MIR register TYPE in Chapter II).

HEDBLK information consists of material taken literally from the program-analysable comments listed under a.1-3. Because each record on the tape has been limited to 40 computer-words, and because the information for HEDBLK would sometimes occupy more space than this, HEDBLK consists of two records (80 computer-words).

LYNBLK information consists of material extracted or derived from LYNE, CLEF, KEY, TIME, and RETIME comments.

NOTBLK information is generated from the information in note-symbol-groups as well as LYNE and FINALbSUB-SECTION comments, '$FERM' and '$BRx'. Attack-times are calculated not only from information about duration given in the note-symbol-groups but also from information in the comments: TIME, RETIME, and REDUCE. The value of various registers containing pitch-information is determined by the value of the staff-position and the comments: KEY, CLEF, TUCK and UNTUCK.

TALBLK signals the end of a composition. ENDBLK signals the end of all compositions on a tape.

All possible information about a particular note in a score is stored in one of the three types of records: HEDBLK, LYNBLK and NOTBLK. The order of the sequence is:

HEDBLK1 (2 records) LYNBLK1 NOTBLK1 --- NOTBLKn LYNBLK1 LYNBLK2
NOTBLK1 --- NOTBLKn LYNBLK2 LYNBLK3 . . . LYNBLKn HEDBLK1 (2 records

This is the arrangement for a sub-section of music. Note that the HEDBLKs for the section come at the very beginning and end. The LYNBLK for a particular lyne is repeated after all the NOTBLKs to which it belongs. This makes it possible to scan backwards and always have the correct HEDBLK and LYNBLK information. The last HEDBLK record for a composition is succeeded by a TALBLK record. The last composition on a tape file is followed by an ENDBLK record.

There are three programs associated with the "conversion process" - two to produce printout for diagnostic purposes, and one to put the registers calculated onto the tape. Each program analyses the I.M.L. data fed into it.

The first of these is called the Reformatting Program. It checks to see if some of the syntactical requirements of I.M.L. have been fulfilled, and also checks to see that information is not given which is clearly out of the question, such as clef-designations which are not allowed, or undefined symbolic time-signatures. It also checks to make sure that each measure of music lasts as long as it is supposed to, given the prevailing time-signature. Error messages of various kinds (see APPENDIX 2B p) are printed out if errors are detected.

The Reformatting Program produces a "reformatted" printout of the I.M.L. input which is used in proofreading. Example 1 (below) is such a printout for the example given in the first chapter. Here the information is reorganized visually to aid in proofreading by multiple-pass scanning. The top line of the reformatted printout contains triplet, tie, LYNBLK, bracket, and fermata information, as well as RETIME and REDUCE comments. All durations are given in the second line. All diatonic letter names, changes of register, and accidentals are given in the third line. The text accompanying the notes is given in the 4th line. Each general program-analyzable comment appears on a line by itself. Whenever a RETIME comment occurs, all time-signatures defined up to that point and their values are printed out for comparison. The registers in which symbolic signatures and their numerical values have been stored are called respectively RETSG and NRET. Lyne and measure indications (1m$) will appear at the beginning of the first line of the reformatted printout. Measure-number

indications alone (if keypunched) appear above line one.

In the printout, a set of four or five print lines corresponds wherever possible to a single staff of the score with its accompanying text. If the staff is too long for the printout page-width, it is continued on a succeeding set of lines, but the beginning of a staff is always imitated by the beginning of a set of lines. (See Example 1.)

The second form of the conversion process is called the register printout program. The values of all NOTBLK and LYNBLK registers are printed for every note as calculated by the program. Some of the registers have different names in the I.M.L. and M.I.R. programs. Since the user will normally use the M.I.R. names this should not cause any difficulty. Below is a list of the different names.

	I.M.L.	M.I.R.
LYNBLK		
	ICLEPH	CLEF
	XNSTRU	INSTRU
NOTBLK		
	NREGCL	REGSTR, REGCL
	NSEMIT	SEMITO
	NPRECA	PRECAC
	NSUGGA	SUGGAC
	NSTAFP	STAFPO
	NDURAT	DURAT
	NDOTND	DOTIND
	MAXLYN	NUMLYN
	NGRPNO	GRPNO
	NBARCT	BARLIN
	NBRACK	BRACK
	NSPECN	SPECSN
	NPHMRK	NO NAME
	NSYSNT	SYSINT
	NSYSNM	SYSNUM
	NSYSDN	SYSDEN
	NCOINT	ATINT
	NCONUM	ATNUM
	NCODEN	ATDEN
	NOTINT	DURINT
	NCTNUM	DURNUM
	NOTDEN	DURDEN
	NTIEND	TIEIND

This is illustrated below for the beginning of the Superius. (See Example 2.)

The third form of the conversion process, called the Tape-writing Program writes the values of the registers onto a tape and produces the same printout as the Reformatting program does.

The specific information needed to run any of these programs is given in APPENDIX 2A (page).

Example 1

Sample Print-out of Reformatting program

```
UBTITLE HOSANNA

'NE 1-SUPERIUS 2-ALTUS 3-TENOR 4-BASSUS

'ASURE 82 HAS BEEN RENUMBERED AS 1 HERE

)$1$ CLEF G TIME (C/) KEY F          *  *              *  *     *
                          11 1 1  2 2 1   11 1 1  2 4 4 1
                          R/ R B/ B C D/  G/ R C/ C B A A//
                               HO     SAN NA

$1$ KEY F CLEF C3 TIME (C/)    *   *                *   *      *  *
                          1 1   2 2  1   11  1 1   2 4  4 1  2 2 1
                          R LB/ B UC D/  LG/ R UC/ C LB A A/ A B UC//
                            HO       SAN NA

$1$ CLEF C3 KEY F TIME (C/)  *   *      *  *
                          11  11 11  11 11 11
                          LG/ G/ UD/ E/ E/ R//
                          HO     SAN NA

$1$ CLEF F KEY F TIME (C/)           *   *                      *
                          1. 2   1   1   2 2 1  11  1. 2  1  1
                          B  UC/ D   LG/ G A B/ UC/ LA B/ UC LF//
                          HO     SAN            NA
```

N.B. The different orders in which CLEF, TIME, and KEY information appears here in the different lynes reproduces the order in which the keypuncher(s) originally coded it: the order is optional.

Example 2

Sample Print-out of Register printout program

```
LYNE 1-SUPERIUS 2-ALTUS 3-TENOR 4-BASSUS

MEASURE 82 HAS BEEN RENUMBERED AS 1 HERE

XNSTRU      ICLEPH SIGKEY KEYPC                          REPKEY TSNUM TSDEN TSV
SUPERIUS       10            0 10  0   0   0   0   0   F       C/          2/  1
   MEASNO LYNENO NOTENO NREGCL NOTECL NTIEND
      1       1       1      14      14       0
NSEMIT DIA     NBARCT NPRECA NSTAFP NDURAT NDOTND
    0             1       0      55       2       0
   NUMLYN NGRPNO NSUGGA NBRACK NPHMRK NSPECN
     4       0       0       0       0       0
TEXT=
MEAS ATTACK TIME     0  0/  1
SYST ATTACK TIME     0  0/  1
COMP ATTACK TIME     0  0/  1
DURATION CUR. NOTE   2  0/  1

   MEASNO LYNENO NOTENO NREGCL NOTECL NTIEND
      2       1       1      14      14       0
NSEMIT DIA     NBARCT NPRECA NSTAFP NDURAT NDOTND
    0             0       0      55       3       0
   NUMLYN NGRPNO NSUGGA NBRACK NPHMRK NSPECN
     4       0       0       0       0       0
TEXT=
MEAS ATTACK TIME     0  0/  1
SYST ATTACK TIME     2  0/  1
COMP ATTACK TIME     2  0/  1
DURATION CUR. NOTE   1  0/  1

   MEASNO LYNENO NOTENO NREGCL NOTECL NTIEND
      2       1       2       4      10       0
NSEMIT DIA     NBARCT NPRECA NSTAFP NDURAT NDOTND
   58   B         1       0      31       3       0
   NUMLYN NGRPNO NSUGGA NBRACK NPHMRK NSPECN
     4       0       0       0       1       0
TEXT=HO
MEAS ATTACK TIME     1  0/  1
SYST ATTACK TIME     3  0/  1
COMP ATTACK TIME     3  0/  1
DURATION CUR. NOTE   1  0/  1
```

CHAPTER II

M.I.R. REGISTER INFORMATION

This chapter will discuss the data-structures generated by the conversion program; i.e., what the user has at his disposal.

The categories for each note are read from the tape into the M.I.R. registers through a call to a particular Fortran subroutine or function in his program. The various registers filled by the execution of any of the possible information-retrieval sub-program calls have been designed to include in many different forms the information usually associated with a note of music, providing as many different ways as possible of handling the basic information. The locations of the information are given symbolic names which refer to computer-memory locations. It is important to remember that at any one time in a M.I.R. program's operation the registers are filled with the information for the current note only, i.e., the note called up by the last executed M.I.R. command*.

Each M.I.R. register in the list below is assigned a specific amount of storage space and this is defined in terms of computer-words (each consisting of 4 bytes, a byte being 8 binary digits). A byte can represent one "character". A computer-word can represent 4 "characters". All M.I.R. registers are some whole number of computer-words long. For registers longer than one computer-word, a parenthetical numeral appears in the listing below after the the name of the M.I.R. register. M.I.R. registers storing information in the form of "characters" have been "declared" (identified) as such by a Fortran statement: REAL * 4 (here the asterisk does not mean "multiply by", as it does elsewhere in Fortran, but the whole expression means that the information is stored in one computer-word equal to 4 bytes, and with the capacity of storing 4 "characters").

---

* Thus the "previously current note" is not necessarily the preceding note in the music; it is the note that has just previously been under consideration (this is what current means) and might be at any distance before or after, above or below the current note, depending upon what sort of relation with the current note is being examined.

Registers storing information in the form of numbers have been declared as such by the Fortran statement: INTEGER * 4, meaning again that the information is stored in one computer-word, equal to 4 bytes, but with the capacity of storing only one number.

Occasionally "characters" are stored one per computer-word (instead of the 4 possible). In this case the information is "left-justified" in the computer word and 3 trailing blank characters are put in to fill up the rest of the word. Similarly, if the number of characters constituting the information for multiple-computer-word registers does not fill the space reserved for this register, the information is left-justified within the total space provided, and trailing blank characters are inserted to fill up that space.

It is necessary to know whether a M.I.R. register contains REAL or INTEGER representations and this is indicated in the list below. Computations can be done with INTEGERS only. "Characters" must be "logically" treated (as = or $\neq$ to something). For printing the information in a Fortran program one must specify the mode (A for REAL ("characters"), or I for INTEGER (numbers)), since the input-output procedures in printing the desired information will interpret the same binary numbers stored in the computer memory differently according to which mode is specified.

Label Registers

The conversion program incorporates into every record it generates information for the following M.I.R. registers (these registers are all declared as INTEGER * 4 except TYPE) :

TYPE (REAL * 4)

The first computer-word of every record contains the characters HEDB, LYNB, NOTB, TALB, or ENDB to identify the type of record read into computer storage during the execution of any MIR order.

NFIL (number of file)

Every record has the serial number of the current composition on the input tape. Used in connection with the TOCOMP subroutine. (See Chapter III.)

NSEC (number of sub-section)

Every record has the serial number of the sub-section of a composition to which it refers. Used with the TOSECN subroutine discussed in Chapter III.

NREC (number of record)

Every record is given a serial number in ascending order from 1, as it is generated. This register is used in conjunction with TOCURN (subroutine which accesses a note which was previously the current note) discussed in Chapter III.

HEDBLK Registers

The following registers contain information that has been read in via HEDBLKS. This information can change only at the beginning of a sub-section. Note that all these registers are REAL * 4 and the information in them is left-justified within the total space of the number of computer-words represented by the number in parentheses after the name of each register.

POSER (8):
: The name of the composer of the current note, if entered as an I.M.L. '=COMPOSERbXXX' program-analysable comment. Otherwise blanks.

TITLE (11):
: Similarly, the name of the composition containing the current note.

SUBTIT (6):
: Similarly, the subtitle of the sub-section containing the current note.

AUTHOR (8)
: Simlarly, the name of the writer of the text that accompanies the current note.

LISHER (6):
: Similarly, the name of the publisher of the composition containing the current note.

EDITOR (6):
: Similarly, the name of the editor of the composition containing the current note.

PUNXER (7):
: Similarly, the name of the keypuncher who punched the I.M.L. code for the current sub-section or composition.

RANGER (5):
: Similarly, the name of the arranger of the composition or sub-section containing the currrent note.

LYNBLK Registers

The LYNBLK registers, listed below, are available for use in a M.I.R. program at any time, but are not recalculated for every new current note unless in the process of scanning the data for retrieval of that note a new LYNBLK is encountered.

CLEF

According to the following table:

Contents of CLEF	
1	treble clef with 8ve-above meaning
2	treble clef with 8ve-below meaning
3	bass clef
4	baritone clef
5	tenor clef
6	alto clef
7	mezzo-soprano clef
8	soprano clef
9	G-clef on first staff line
10	treble clef
11	contra-bass clef

KEYPC (7)

There has been an error in the design of this register. If the key-signature of a composition was expressed in terms of a parenthetically enclosed list of sharps or flats, the results of that method of representation are stored in these 7 computer words in terms of the NOTECL (see NOTBLK registers p) of the altered note. Thus a key-signature with only one flat (B-flat) would have a 10 in the second of the 7 computer words representing the 7 diatonic note-classes (A, B, C, D, E, F, G) and a zero in each of the other 6 computer words. A key-signature with F-sharp, C-sharp and G-sharp would have a 6 in the sixth computer word, a 1 in the third, an 8 in the seventh, and zeros in the other four. But this then means that it is not possible to store the information of a key-signature in terms of a parenthetically enclosed list of sharps if that key-signature includes B-sharp.

ITSVNM, ITSVDN:

The "value" (considered as a rational number of whole-note units) of the time-signature

affecting the current note is stored as a lowest-terms fraction, numerator in ITSVNM, denominator in ITSVDN. E.g., a time signature of 12/8 would be stored as 3 in ITSVNM and 2 in ITSVDN. Remember that all time-signatures which are not "fractional" must be defined in I.M.L. with a RETIME comment, with the exception of the initial values of C, C/, and C-, which are all 2/1.

TSNUM (2) (REAL * 4), TSDEN (2) (REAL * 4):

The time-signature affecting the current note (actually the current lyne as well) is stored in TSNUM and TSDEN in alphameric form as follows:if the time-signature comprises a numerator and a denominator (e.g., 3/4), then the numerator is stored in TSNUM and the denominator in TSDEN; two characters apiece are allowed. (A single number will be left-justified). If a time-signature consists of a number only, or a combination of letters and numbers (enclosed within parentheses in I.M.L.), it will be left-justified within the total storage space contained in TSNUM and TSDEN together. If a symbolic time-signature has more than 4 characters in it, disregarding the parentheses, the excess over 4 characters will be lost.

REPKEY (2) (REAL * 4, each word contains one character left-justified):

If, however, the key-signature was given in the standard form of a letter representing the tonic of a major key, that result is stored in REPKEY. REPKEY(1) is the diatonic letter name of the key, and REPKEY(2) is the alteration, S for sharp, T for flat, or blank for no alteration.

INSTRU (3) (REAL * 4, left-justified):

The name of the musical instrument or voice scheduled to perform the current note is stored in adjacent computer words INSTRU(1), INSTRU(2) and INSTRU(3), left-justified with trailing spaces. If the name contains more than 12 characters it will be truncated.

SIGKEY (7) (REAL * 4, each word contains one character left-justified):

If the key-signature of a composition was

expressed in terms of a parenthetically enclosed list of sharps or flats, the results of that method of representation are stored in these 7 computer words with the characters S for sharp and T for flat, left-justified within each word. SIGKEY(1) stands for A, SIGKEY(2) stands for B, SIGKEY(3) for C . . . SIGKEY(7) for G. A blank character is inserted if there is no sharp or flat.

NOTBLK Registers

The following registers refer specifically to the current note, and can change every time a new note (NOTBLK) is made current. All registers are declared as INTEGER * 4 unless specified differently.

M.I.R. REGISTER	CONTENTS
MEASNO	The number of the measure in which the current note occurs.
LYNENO	The number of the lyne in which the current note occurs.
NOTENO	The number of the note within the current measure of the current lyne. (Remember that a rest is counted as a note.)
NOTECL	14 If the current note is a rest. Otherwise the note-class of the current note. All C's, B-sharps, and D-double-flats are assigned to note-class 0, all C-sharps, etc., to note-class 1, and so on.
REGSTR or REGCL*	14 If the current note is a rest. Otherwise the register of the current note. Middle C through the next higher B is assigned to register 4; notes in the next higher octave to register 5, etc.
SEMITO	The number of semitones that the current note is above an arbitrarily defined zero point which is the non-existent C four octaves below middle C. Middle C = 48. Since the note corresponding to this zero point would never appear in any score, the zero symbol never needs to be used for

---

* Programmers have used both terms REGSTR and REGCL, so these have been equivalenced.

representing pitch, and is used instead to represent a rest.

PRECAC

0 if the current note is not preceded immediately by an accidental; otherwise, according to the following table:

Contents of PRECAC	
1	sharp
2	flat
3	double-sharp
4	natural
5	natural and sharp
6	natural and flat
7	double-flat

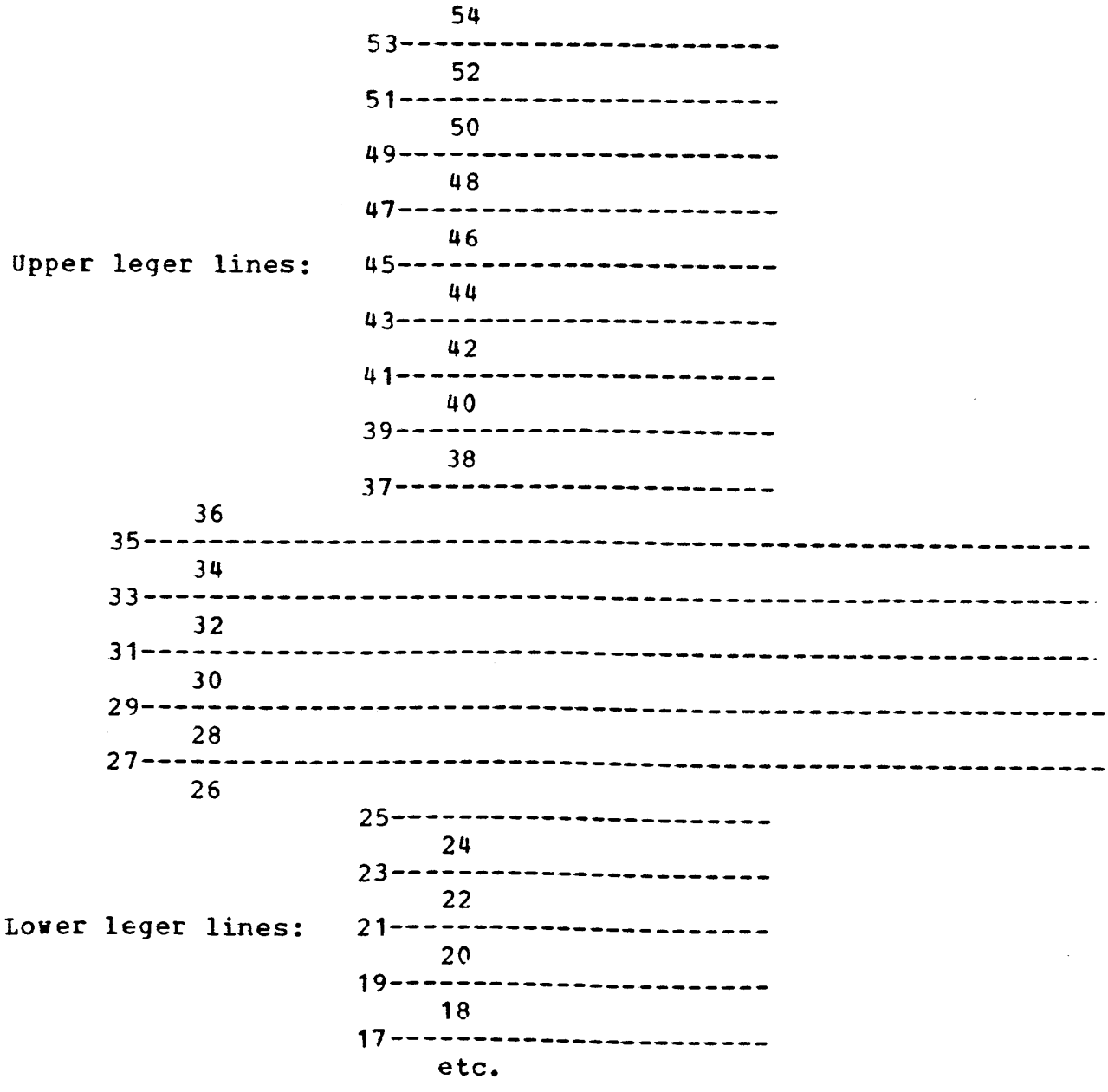
SUGGAC

0 if the current note is not affected by an accidental. Otherwise 1, 2, or 3 for a sharp, a flat, or a natural, respectively.

In the evaluation of NOTECL and SEMITO an accidental is considered as lasting through the measure. However, PRECAC and/or SUGGAC are not altered by the program to reflect such an assumption.

STAFFO

55 If the current note is a rest. Otherwise, the number of the staff-position of the current note according to the following diagram:



DURAT

According to the following table:

Contents of DURAT	
1	long
2	breve
3	whole (semibreve)
4	half (minim)
5	quarter (semi-minim)
6	eighth (fusa)
7	16th (semifusa)
8	32nd
9	64th
10	128th
15	grace-note

DOTIND

0, 1, 2, Or 3 according as the current note is undotted, single-dotted, double-dotted, or triple-dotted.

NUMLYN

This register contains the total number of lynes in the composition at the place where the current note is. (Not a value which could change for every note, but nevertheless part of each NOTBLK.)

GRPNO

0 If the current note is not an element of a triplet group; 3 if it is.

BARLIN

0 If the current note is not followed immediately by a barline indication; 1 if the current note is followed by a single barline that does not end a staff; 2 if the current note is followed by a single barline that ends a staff; 3 if the current ncte is followed by a double barline; and 4 to indicate the end of a complete section, as opposed to 3 which merely indicates the end of a sub-section.

BRACK

0 If the current note is not affected by any ligature-denoting brackets. Otherwise, 1, 2, or 3 according as the current note is affected by the

beginning, middle, cr end of a bracket.

SPECSN

This computer word contains 0 if no special signs affect the current note. If the current note is affected by a fermata, SPECSN contains the number 12. Symbols for other special signs are not yet established.

MESINT, MESNUM, MESDEN

MESINT is the number of whole-note units between the beginning of the current measure and the current note. MESNUM/MESDEN represents the fractional parts of a whole-note unit which when added to MESINT give the measure-attack-time of the current note. (Note that if no fractional parts exist, the numerator of a set of attack-time registers is 0 and the denominator is 1.)

SYSINT, SYSNUM, SYSDEN

In like manner, the number of whole-note units plus a fraction of a whole-note unit between the beginning of the current system of the score and the current note.

ATINT, ATNUM, ATDEN (or AT(3))

Similarly, the number of whole-note units plus some fraction of a whole-note unit between the beginning of the current composition and the current note.

DURINT, DURNUM, DURDEN

The duration of the current note, expressed in terms of units of whole-notes plus a fraction of the whole-note unit. (The attack-time of a note plus its duration equals the attack time of the next note in the current lyne.)

DNC (REAL * 4, left-justified)

The "diatonic note-class" of a note is the same as the letter-name of the note. Blank characters are inserted in this register if the note is a rest.

TEXT(4) (REAL * 4, left-justified)

Blank characters if no text accompanies the current note. Otherwise, that text, 16 characters maximum, left-justified with trailing blank

characters to make up the total 16, stored 4 characters per computer word.

TIEIND

1 If the current note is tied to the preceding note in the same lyne, 2 if there is a suggested tie from the preceding note, otherwise 0.

Interval Registers

The following M.I.R. registers are calculated not during the conversion process, but during the operation of a MIR program each time a new note is made current and this only if the subroutines PREP, CAL, and NGENU (see Chapter III) are included. The registers refer not just to the current note, but to the previously current note as well.

M.I.R. REGISTER	CONTENTS
INTVL	The pitch-interval between the current note and the immediately previous current note is stored in INTVL as a signed number of semitones: the sign is negative if the current note is lower in pitch than the immediately previous current note; otherwise the sign is positive. If the immediately previous current note is a rest, or there is no immediately previous current note (as at the start of a M.I.R. program) or if the current note is a rest, then INTVL contains zero.
INTDIR, INTREG, INTNOT	The interval between the current note and the immediately previous current note is represented in INTDIR, INTREG and INTNOT as follows: if the current note is lower in pitch than the immediately previous current note, then INTDIR contains 1; otherwise INTDIR contains 0. The size of the interval in terms of registral-span (0 for less than an octave (1-11 semitones), 1 for an octave or less than two octaves (12-23 semitones), etc., and NOTECL difference, is stored in INTREG and INTNOT respectively. For the computation of INTDIR, INTREG and INTNOT only, the "pitch" of a rest is considered to be register 0, NOTECL 0. E.g., if the immediately previous current note were middle C and the current note were an 8th-rest then INTDIR, INTREG, and INTNOT would contain 1, 4, and 0 respectively. If there is no immediately previous current note, then INTDIR contains 2, INTREG contains the register of the current note, and INTNOT contains

the note-class of the current note.

DINTVL (Diatonic interval), GENUS, DIOCT (Diatonic octave):
If the current note is a rest or if the immediately previous current note is a rest, or if there is no immediately previous current note, then DINTVL contains 0 as do also GENUS and DIOCT. Otherwise, the "diatonic" interval (that is the interval as measured on the staff, rather than by the number of semitones (note that DIOCT for the interval middle C to the first C-flat above it will be 1, even though it measures only 11 semitones)) between the immediately previous current note and the current note is represented in DINTVL, GENUS and DIOCT as follows:

Contents of DINTVL	Interval
1	unison, octave, double-octave, etc.
2	second, ninth, etc
3	third, tenth, etc.
4	fourth, eleventh, etc.
5	fifth, twelfth, etc.
6	sixth, thirteenth, etc.
7	seventh, 14th, etc.

Contents of GENUS	Interval
1	doubly diminished
2	Diminished
3	minor
4	Perfect
5	Major
6	Augmented
7	Doubly augmented
8	Any other interval

Contents of DIOCT	Interval
0	smaller than an octave
1	at least an octave (see remarks under INTREG above), but smaller than two octaves
n	n octaves or larger, but smaller than n+1 octaves.

DINDIR ("Diatonic" interval direction):
Let z denote the current note and y the previous current note. Then DINDIR contains:

1 if z is "diatonically" lower than y;
2 If z or y is a rest, or if y does not exist;
0 If z is "diatonically" higher than or at the same pitch as y.

Chapter III

M.I.R. Sub-programs

Once the I.M.L. version of some music has been interpreted and converted to the data-structure described in Chapter II, the problem becomes one of retrieving the information. A M.I.R. program consists of a body of pre-existing subprograms to which the user must add, at a designated point, a section devised by him (which is not a sub-program but completes the MAIN routine).

MAIN routine

This starts with certain storage declarations and statements, which the user will find ready made, to access the first note of the first lyne of the first composition of the tape file being read. Then follows the user's part of the program, which effects retrieval of some part of the data-structure on tape via calls to ready-made sub-programs. The names of the M.I.R. registers (see Chapter II) are variables in the MAIN routine. Those registers whose information comes from LYNBLKs or HEDBLKs are automatically "unpacked", i.e., can be referred to by name. Information from NOTBLKs is not unpacked except in specific cases noted below. Before the beginning of the user's own part of the program, provision has been made for accessing the first note, which is made the "current" note, and the NOTBLK registers applying to it are unpacked. From here on the user must supply all program statements and the MAIN routine must end with a Fortran END statement. Any call to an information-retrieval subprogram will access some note of music which will be termed the "current" note.

The M.I.R. system is modular, in that specific tasks are performed by specific sub-programs. It is therefore possible to call on each module separately. But this is not normally necessary or desirable, since sub-programs designed to perform combinations of tasks automatically call lower-order sub-programs to perform the individual tasks*.

The following subprograms are the only ones which the user will normally specify explicitly in his program. M.I.R. register names must not be used alone as arguments.

---

* Table VI at the end of this chapter shows the hierarchy of some of the sub-programs.

The following two subroutines, TOTITL and TOCOMP, can be used only with a M.I.R. data tape that contains all compositions combined into a single file.

TOTITL ('TITLE---')

This subroutine effects the retrieval of a composition within a tape file by name. If the composition cannot be found, an error message is printed out and the originally current note is restored. The argument must be enclosed within single quotes, must be the exact form of the TITLE comment in the I.M.L. version of the music, and must be, with as many trailing blank characters as necessary, 44 characters long. A table of the composition titles read in so far is automatically kept and as with TOSECT the search is converted to a search by composition number.

TOCOMP (N)

The serial-number of some composition on the input-tape file appears as the argument in a call to this subroutine. The first note of the first lyne of the desired composition is made current (by a call to TOSECN(1) which is built into TOCOMP). The value of ENDF (see Table V under subroutine READBK below) should be tested after a call to TOCCMP. If it is greater then 0, the composition desired does not exist.

TOSECT ('SUB-SECTICN NAME')

A sub-section within a composition may be retrieved by name, i.e., by the exact form of the SUBTITLE given in the I.M.L. coding of the music. (If the sub-section name cannot be found, an error message to this effect is printed and the originally current note is restored.) The argument of this subroutine must be enclosed within single quotes and must be, with as many trailing blank characters as necessary, 24 characters long. The value of the M.I.R. register SUBTIT as well as the sub-section number to which it refers is stored in a table for each sub-section read so far in the subroutine READFD (see below). This enables this TOSECT to convert the search for a section by name to one for a section by number, and subroutine TOSECN is called.

TOSECN (N)

A subsection within a composition whose serial-number is the value of the argument in the call to this subroutine is searched for and the first note in lyne 1 of the sub-section is made the current note. The value of ENDC (see subroutine SEARCB below) should be tested after a call to TOSECN. If it is greater than 0, the section desired does not exist.

The argument for TOSECN or TOCOMP must not be such that its value would turn out to be zero or negative. This could result in a vain search for a NOTBLK before record 1 on the tape. If there is a possibility of this, DEC (see subroutine SEARCB below) should be tested, because the backward search will have stopped at record 1. Such a failure does not result in an error message or in the restoration of the current note.

TONEXT, TOLYNE(N), TOMEAS(N), TONOTE(N)

A call to TONEXT gives the next note in the current lyne. Entry TOLYNE(N) finds the first note in the current measure of lyne N*.
Entry TOMEAS(N) finds the first note in measure N of the current lyne. Entry TONOTE (N) finds note N in the current measure and lyne. But if in TONOTE (N) the argument is given a value higher than the number of notes in the measure concerned, the notes in as many succeeding measures as necessary will be considered as members of a numbered series beginning with the first note of the original measure. Thus if a note-number 6 is given for a measure that has only five notes, the first note of the next measure will be found.

N may also be given the value of a relative reference, such as (NOTENO+M) or (NOTENO-M). If the value of (NOTENO-M) is zero, the last note of the preceding measure will be found; if it is -1 the next-to-last note will be found, etc. If the value of (MEASNO-M) is zero, the first note of the last measure of the preceding section will be

---

* ENTRY statements permit one to begin the execution of a sub-program at some point other than the beginning.

found, etc.

In the case of lyne-number, a value greater than the number of lynes or less than 1 is adjusted in a rotational manner. For example, a value of 5 in a three-voice section would find a note in lyne 2.

TOCURN(N), TOREC(N)

If N has previously been made equal to the value of NREC for a specific NOTBLK, the subroutine TOCURN(N) will make that NOTBLK current. The argument N may be a constant as well as a variable. Entry TOREC(N) performs the same operation but does not call PREP, CAL, or UNPACK (see below).

FUNCTIONs NOTE(N), ATTACK(J,K,L,N)

These subprograms are FUNCTIONs and are called by a statement such as M=NOTE(N). NOTE(N) finds the note in the lyne indicated by the argument N sounding simultaneously with the attack of the note that was current before the execution of this function. The entry ATTACK(J,K,L,N) finds a note in lyne N sounding at the composition attack time given in arguments J, K and L (but not necessarily attacked at that time.). Both NOTE(N) and ATTACK(J,K,L,N) have the same values: 1 if the attack-time for the note found is the same as that of the previously current note, and 0 if not.

FINDFD(N)

This subroutine finds a note whose attack-time is the closest following that of the current note, irrespective of what line it occurs in. The subroutine examines all notes in all lynes in the current measure and the note with the attack-time least greater than that of the current note is selected. In case there is no next note in the current composition, the value of N is set by FINDFD(N) to -1, otherwise it is set to zero. End of section may be detected in this and other cases by testing the value of BARLIN to see if it is greater than 2.

FINDBK(N)

This subroutine finds a note in any lyne whose attack-time is the closest previous to that of the current note. The process is the reverse of FINDFD. If there is no previous note, N is set to -1, otherwise 0. In this case and the corresponding situation with FINDFD, whenever N =-1 the note that was current before the subroutine was called is restored.

The following subprograms are not involved with information retrieval, but are useful in writing a M.I.R. program.

FUNCTION FRACOM(NA,NB,NC,ND,NE,NF) (Fraction comparison)

Two fractions may be compared to determine which is greater in value, or whether they are equal. The fractions (including whole-number portions which may be zero) are NA + (NB/NC) and ND + (NE/NF). The value of this function is 1, 2, or 3 depending on whether the first fraction is greater than, equal to, or less than the second. Arguments must not have negative values, and if they have whole-number portions their fractional portions must be less than 1.

FRALCD(J,K,L)

Fraction J + (K/L) is reduced to the lowest terms if possible.

FRACAD(J,K,L,M,N,NN)

Fraction J/K is added to fraction L/M and the result is placed in N/NN.

FRASUB(J,K,L,M,N,NN)

Fraction L/M is subtracted from J/K and the result is placed in N/NN.

FRAMPY(J,K,L,M,N,NN)

Fraction J/K is multiplied by fraction L/K and result is placed in N/NN.

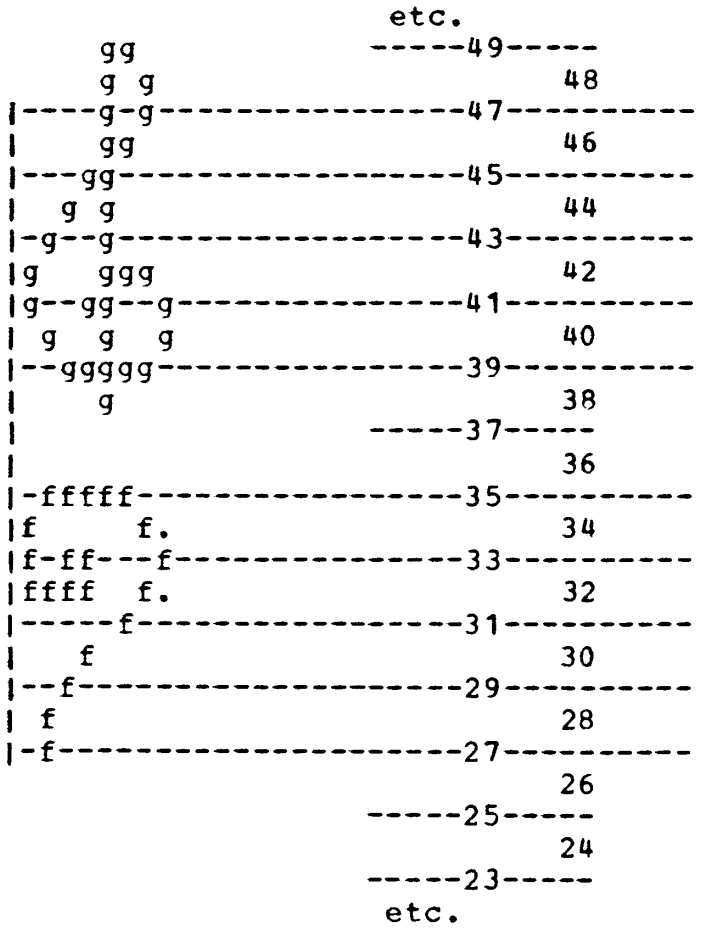
The printed output obtained during the execution of a Fortran program does not automatically have a desirable number of blank lines at the bottom of a page. This may be accomplished by the use of the following subprogram provided in M.I.R.:

SPACER(N),INIT,SKIP,LINE

Spacer(N) provides a means for keeping track of the number of lines that have been printed (or skipped) on a page during the execution of a Fortran program. The user argument N has a value equal to the number of lines that the user's program will print or skip before the next call to this subroutine. N is added by this routine to the total count of the number of lines on the current page and if that count is raised to a value greater than 60, the top of the next page will be accessed prior to the statements in the user's program which will actually cause the printing of the number of lines given in the argument N. Entry INIT skips immediately to the top of the next page and sets the count variable to zero. This should be called at the beginning of any program thet uses SPACER(N). Entry SKIP skips a line but does not count it. Entry LINE skips a line and counts it.

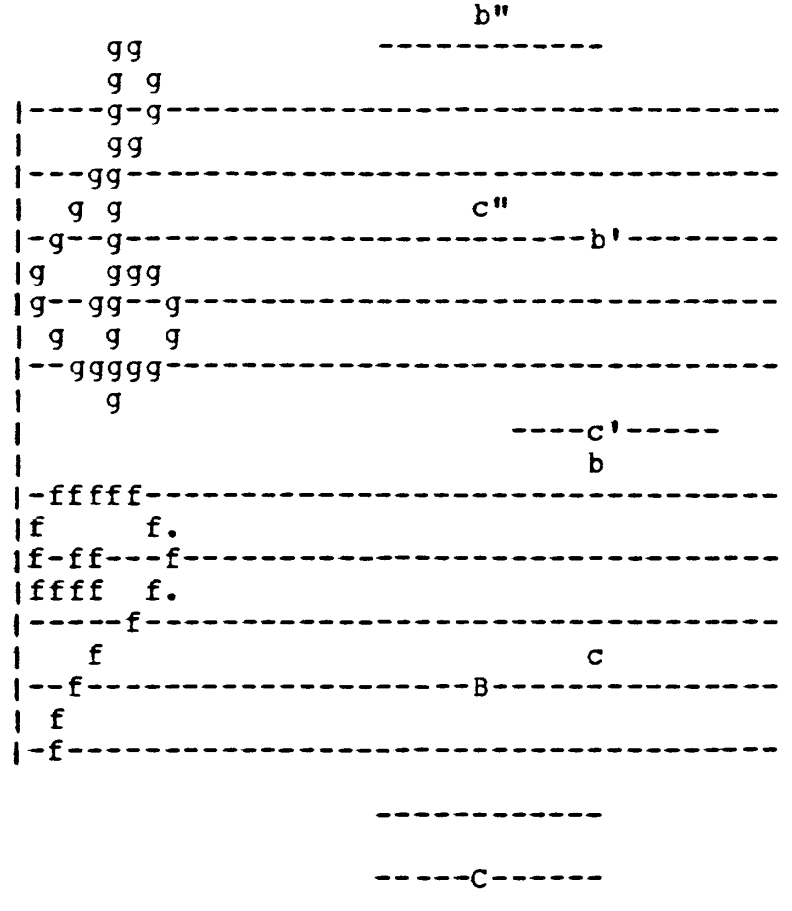
FUNCTION ABGS(N,M)

This function allows the comparison of diatonic values among notes governed by different clefs. N is the clef number from M.I.R. register CLEF, and M is the value of STAFFO for the note in question. The value of this function is zero for a rest.



DIABS(K,DUM)

DUM is an array of three computer words in which are stored three characters representing the absolute pitch of a note in terms of its letter-name. The value of DUM is calculated given the argument K which is the SEMITO value for the note. Upper-case letters are used to indicate the octave from C 2 octaves below middle-C to the B a 7th above. Lower-case letters indicate the C below middle-C to the B an octave and a 7th above middle-C. A single quote indicates middle-C to the B a 7th above. Double quotes indicate the C an octave above middle-C to the 7th above that. Sharps are indicated by the # character. Flats are indicated by lower-case b. Any notes which fall outside this range will result in erroneous formations.



The following 12 sub-programs are of lower order, and will therefore not normally be called explicitly by the user. Their descriptions are included as a matter of documentation. All are SUBROUTINEs except where otherwise specified.

UNPACK

This subroutine moves material pertaining to the NOTBLK into an array which can be accessed by the user via the M.I.R.-register names. This subroutine is called automatically by any of the following sub-programs: TONEXT, TONOTE, TOMEAS, TOLYNE, TOCURN, TOSECN, TOSECT, TOCOMP, TOTITL, NOTE, FINDFD, and FINDBK. No other retrieval subprograms call this routine, and if they are used by the programmer he must call UNPACK himself if he desires to utilise the M.I.R. registers which come from the NOTBLK record containing the information for the current note.

PREP (Prepare)

This subroutine saves the values of SEMITO, REGCL, and DNC which will be needed to calculate the M.I.R. registers pertaining to the interval between the current note and the previously current note. This routine is automatically called by the same subroutines as given in the description of UNPACK.

CAL (Calculate)

This subroutine finishes the work of calculating the M.I.R. registers which give the value of the pitch-interval between the current note and the previously current one. This routine is called at the conclusion of the operations performed by the retrieval subroutines mentioned in the description of UNPACK. The values and names of pertinent M.I.R. registers are given in Chapter II. Tables I and II (below) give the value of DINTVL in accordance with the information from which it is derived. Table III gives the values for GENUS, DIOCT and DINDIR.

If INTVL is positive, the values of DINTVL are according to the following table:

Table I

current note / previous note	A	B	C	D	E	F	G
a	*	2	3	4	5	6	7
b	7	*	2	3	4	5	6
c	6	7	*	2	3	4	5
d	5	6	7	*	2	3	4
e	4	5	6	7	*	2	3
f	3	4	5	6	7	*	2
g	2	3	4	5	6	7	*

* not calculated from this table

If INTVL is negative the values of DINTVL are according to the following table:

Table II

current note → / previous note ↓	A	B	C	D	E	F	G
a	*	7	6	5	4	3	2
b	2	*	7	6	5	4	3
c	3	2	*	7	6	5	4
d	4	3	2	*	7	6	5
e	5	4	3	2	*	7	6
f	6	5	4	3	2	*	7
g	7	6	5	4	3	2	*

* not calculated from this table

The values of GENUS are given in the following table.

Table III

DINTVL / INTNOT	=1	=2	=3	=4	=5	=6	=7
0	4	2	8	8	8	8	< 6
1	< 6	3	1	8	8	8	< 7
2	< 7	5	2	8	8	8	8
3	8	6	3	1	8	8	8
4	8	7	5	2	8	8	8
5	8	8	6	4	1	8	8
6	8	8	7	6	2	1	8
7	8	8	8	7	4	2	8
8	8	8	8	8	6	3	1
9	8	8	8	8	7	5	2
10	+ 1	8	8	8	8	6	3
11	+ 2	+ * 1	8	8	8	7	5

+means that DIOCT = INTREG +1

< means that DIOCT = INTREG - 1
In the absence of the signs + and <, the value of DIOCT is equal to that of INTREG

* means that DINDIR is the opposite of INTDIR; otherwise they are the same.

GENUS	Interval
1	doubly diminished
2	diminished
3	minor
4	perfect
5	major
6	augmented
7	doubly augmented
8	any other interval

NGENU (Table for GENUS)

This subroutine is called in MAIN before the user's part and puts the values for Table III into array NGEN.

Note that the use of PREP, CAL and NGENU slows down the retrieval process considerably. If the user's program does not make use of these routines they can be switched into "dummy mode" by specifying ISPREP = 0, ISCAL = 0, and ISNGEN = 0 at the beginning of the user's program.

NUMPIC(DIA,KX) (Diatonic pitch numbers)

This subroutine translates the diatonic letter-names A through G (argument DIA) into integers 1 through 7 (argument KX). The use of any other character for DIA will result in an error message.

GET(J,K,L)

This subroutine finds a note in the input stream according to the three arguments given it which correspond to the LYNENO, MEASNO and NOTENO of the desired note. This subroutine may be employed by the user in his program if he remembers that the NOTBLK found will not be unpacked. The arguments J, K and L must be declared implicitly or explicitly as INTEGER * 4, and the names of the M.I.R. registers cannot be used as arguments except as constituents of numerical expressions such as NOTENO + 1. This subroutine calls SEARCF and SEARCB and the value of ENDC is that returned by the last call it has made to either of those routines.

SEARCF (Search forward)

This subroutine finds the next NOTBLK. As in READFD, this does not mean necessarily the next NOTBLK in the same lyne; it means next in the order in which the records have been written on the tape by the conversion program.

SEARCB (Search backward)

This subroutine finds the immediately preceding NOTBLK (not necessarily in the same lyne). The settings of the variables in Table IV indicate the status of the input after each call to SEARCF or SEARCB. These variables available to the user in his program are, DEC (decrement), ENDC (end of composition) and ENDF (end of file).

Table IV

	DEC	ENDC	ENDF	Record just read
SEARCF	0	0	0	NOTBLK
	0	1	0	Attempt to read past end of composition; originally current note retained
SEARCB	1	0	1	Attempt to read back past 1st record on tape file; originally current note retained.

The settings of these variables will not indicate reading forward or backward into adjoining sections in one tape file. Testing for this can be done by examining the values of NSEC, or BARLIN (see page). An unsuccessful attempt to locate a particular NOTBLK results in the restoration of the previously current note.

SRCREC(N) (Search by record number)

This subroutine locates a record on the the tape with record number N.

READFD (Read forward)

This subroutine accesses the next portion of data (either a NOTBLK, a HEDBLK (two records), a LYNBLK, a TALBLK, or an ENDBLK). This is not necessarily the next NOTBLK in the same lyne. It is the next data block in the sequence put onto the tape by the conversion program.

READFD is automatically called by any M.I.R. subroutine in which forward scanning of the material is required. Because it accesses the next record, no matter what kind of information that record contains, if the user has occasion to

call it explicitly, he will have to make various kinds of tests to determine what kind of information it has just accessed. The HEDBLK and LYNBLK information is automatically unpacked into the M.I.R. registers, where it remains available until some other HEDBLK or LYNBLK is read.

READBK (Read backward)

This subroutine finds the record (2 records for HEDBLK) immediately preceding the current one.

To help determine what kind of information has been accessed by READFD and READBK, whether these routines have been called automatically by other retrieval-routines or explicitly by the user, they set certain variables to 1 under certain conditions and to 0 under all other conditions. The conditions determining their values are given in the following table:

Table V

	DEC	ENDC	ENDF	Record just read
READFD	0	0	0	NOTBLK, LYNBLK, or HEDBLK
	0	1	0	TALBLK
	0	0	1	ENDBLK
READBK	0	0	0	NOTBLK, LYNBLK or HEDBLK
	1	0	0	Terminal HEDBLKs of previous section
	1	1	0	TALBLK of previous compositio
	1	0	1	Attempt to read back past record on tape file

TPGET (Get a record from the tape)

At the beginning of the M.I.R. program's execution the first record on the tape is identified as a starting point. TPGET finds the next record. This routine may be entered at other points for the performance of other functions. ENTRY TPBACK moves back one record.

This subroutine actually reads the tape, or in most cases, since the records are written on the tape in blocks of 200, simply indicates which set of 40 computer words out of the input array of 200 records will constitute the next record

desired. This routine does not use Fortran input and output procedures, but instead utilizes Hale Trotter's routines TPREAD etc., whose descriptions are given in Appendix IV.**

Any manipulation of the tape must use these routines only. ENTRY TPSTAR calls TPOPEN which makes available a particular tape file. The M.I.R. system as it stands now initializes only the first file on whatever tape has been identified as being on device 15. Statements to do this occur in the MAIN routine before the user's part of the program.

**Mr. Trotter's input-output procedures are designed for any IBM 360 machine.

Table VI

M.I.R. SUBPROGRAM HIERARCHY

HIGH LEVEL (CALLED BY USER)	MIDDLE LEVEL	LOW LEVEL
TOTITL - TOCOMP	UNPACK CALLED BY ALL	GET
TOSECT - TOSECN	HIGH LEVEL SUBPROGRAMS, EXCEPT TOREC	SEARCF AND SEARCB FIND NEAREST NOTBLK
TONEXT ENTRY TOMEAS ENTRY TOLYNE ENTRY TONOTE	PREP, CAL, NGENU MAY BE IN "DUMMY STATE" DEPENDING ON SETTING OF SWITCHES: ISPREP, ISCAL, ISNGEN	READFD AND READBK FIND AND IDENTIFY NEAREST RECORD
TOCURN ENTRY TOREC		SRCREC FINDS NUMBERED RECORD
FUNCTION NOTE ENTRY ATTACK		TPGET ENTRY TPSKIP ENTRY TPBACK ENTRY TPSTAR THESE UTILISE TROTTER'S READING ROUTINES
FINDFD AND FINDBK		
FRALCD, FRACAD, FRASUB, FRAMPY, FUNCTION FRACOM		
ABGS AND DIABS		
SPACER ENTRY INIT ENTRY SKIP ENTRY LINE		

Chapter IV

Sample user program

In this chapter, a sample user program will be given to demonstrate the functions of various M.I.R. subprograms. The program is written in FORTRAN.* Retrieval of information is effected by CALLs to SUBROUTINEs or by the use of certain FUNCTIONs; both procedures cause some portion of the M.I.R. data-structure to be made available in the storage locations termed M.I.R. registers.

For demonstration purposes, we will use a program somewhat shorter and simpler than most user programs. The task, however, is typical: to locate all harmonic augmented fourths and diminished fifths and all harmonic fourths or fifths which have been "corrected"--that is, made perfect--by the addition of an accidental, either specified or editorial. The entire program and a sample of the output will be given.

There are several ways the problem might be attacked. Here, we will assume that the best approach is:

(1) to find every simultaneity** in a composition, and
(2) to examine every pair of simultaneously sounding notes, and
(3) to print out those interval pairs which are relevant.

We can find every simultaneity, as defined, in a composition by moving from an initial attack point to each successive attack point in any line, finding at each point the notes sounding simultaneously. This can be done with two M.I.R. subprograms:

ATTACK (see page)

FINDFD (see page)

---

*It is assumed that the reader knows FORTRAN.
**A simultaneity will be thought of here as the note occurring at a particular attack-time in at least one lyne, together with all the other notes in the remaining lynes which are sounding at the same time.

The program can be thought of as in four blocks:

Block 1

Gather all information about "current" simultaneity.

Block 2

Identify pairs of notes within that simultaneity.

Block 3

Print results, if any.

Block 4

Proceed to next simultaneity, make it "current" and go back to Block 1.

Thus blocks 1 and 4 form a loop which can be modified for use in other programs, and therefore they will be explained first.

The block 1 part of the program which stores the information about the current simultaneity is given below:

```
C     PROGRAM TO FIND SIMULTANEOUSLY SOUNDING AUG. 4THS AND DIM. 5THS
C     SWITCHES TO PUT PREP, CAL AND NGENU IN "DUMMY MODE"
      ISPREP=0
      ISCAL=0
      ISNGEN=0
      DIMENSION NMESNO(6),NLYNE(6),NNOTE(6),RDNC(6),NCL(6),NPRAC(6),
     1 NSGAC(6),NOTLST(6)
      DIMENSION DNCSAV(2)
      DATA DNCSAV/'B   ','F   '/
    1 WRITE(6,2)TITLE,SUBTIT
    2 FORMAT(1H1,10X,'SIMULTANEOUSLY SOUNDING AUG.4THS AND DIM.5THS',
     1//,1X,11A4,//,4X,6A4,//)
C     SWITCH SET TO 1 FOR EACH LYNE WHEN LAST NOTE OF SECTION HAS BEEN READ
      DO 3 J = 1,NUMLYN
    3 NOTLST(J) = 0
C
C      BLOCK 1
C
C     INITIALISE WITH LYNENO (=1)
  100 LIN=LYNENO
      LN=LIN
C     SAVE RECORD NUMBER OF NOTE IN LIN
      NAT=NREC
C     SAVE ATTACK TIME OF NOTE IN LYNE LIN
      NATIN = AT(1)
      NATNUM = AT(2)
      NATDEN = AT(3)
C     SAVE RELEVANT INFORMATION OF NOTE IN LYNE LN
  110 CONTINUE
```

```
      NMESNO(LN) =MEASNO
      NLYNE(LN) = LYNENO
      NNOTE(LN) = NOTENO
      RDNC(LN) = DNC
      NCL(LN) = NOTECL
      NPRAC(LN) = PRECAC
      NSGAC(LN) = SUGGAC
      IF (BARLIN .GE. 3) NOTLST(LN) = 1
C     INCREMENT LYNE NUMBER BY 1
  115 CONTINUE
      LN = LN+1
C     "CONVERTS" LN IF GREATER THAN NUMLYN
      IF (LN .GT. NUMLYN) LN = LN-NUMLYN
C     CHECK TO SEE IF ALL LINES HAVE BEEN EXAMINED.
      IF (LN .EQ. LIN) GO TO 120
C     FIND SIMULTANEOUS NOTE IN NEW LN
      NDUMY1 = ATTACK(NATIN,NATNUM,NATDEN,LN)
      GO TO 110
  120 CONTINUE
```

At the beginning of every M.I.R. user program, the "current" note is the first note in lyne 1 of the first composition on the tape. This is set up automatically by the MAIN routine.

If we call the note from which each simultaneity is referenced the "primary note", its lyne number, attack time, and tape-record number will need to be saved. This is done at the statements at 100.

The statements at 110 store all the relevant information for the "primary note", and for the notes in the other lines when they have been found, which will be needed to calculate the intervals in the simultaneity.

At 115 we proceed to the next line in the simultaneity, first reducing LN to LN - NUMLYN if it is greater than NUMLYN, and then checking to see that we have not returned to the line (LIN) of the "primary note". The simultaneous note in line LN is found by using the FUNCTION ATTACK. (The value of ATTACK is not required in this program and therefore it is made equal to a dummy variable NDUMY1.)

The program will move on from block 1 by means of statement 120 when all the lines of the current simultaneity have been read.

Although the program will at this point proceed to blocks 2 and 3, our description now jumps to block 4.

Having found the first simultaneity, how do we find the next?

```
C
C      BLOCK 4
C
C      TEST TO SEE IF LAST NOTE OF SECTION IN EACH LYNE HAS BEEN READ
  400 CONTINUE
      DO 410 J = 1,NUMLYN
      IF (NOTLST(J) .EQ.0) GO TO 420
  410 CONTINUE
      GO TO 460
C     RE-INITIALISE WITH NOTE IN LYNE LIN
  420 CONTINUE
      CALL TOCURN (NAT)
C     EXAMINE  ALL LYNES FOR FIRST SUCCEEDING ATTACK POINT
      CALL FINDFD(NDUMY2)
      GO TO 100
C  MOVE TO NEW SECTION
  460 CONTINUE
      CALL TOSECN(NSEC+1)
C     CHECK TO SEE IF CURRENT SECTION IS LAST IN CURRENT COMPOSITION
      IF (ENDC .GT. 0) GO TO 470
      GO TO 1
C  MOVE TO NEW MASS
  470 CONTINUE
      CALL TOCOMP(NFIL+1)
C     CHECK TO SEE IF CURRENT COMPOSITION IS LAST ON TAPE
      IF (ENDF .GT. 0) GO TO 480
      GO TO 1
  480 STOP
      END
```

Block 4 commences at statement 400 by checking to see if each of the notes of the current simultaneity is the last note of its line of the current section of music. If so, the program jumps to 460. If not, the program at 420 makes the "primary note" of the simultaneity current again and then using FINDFD the first succeeding attack point in any line is found, and the note at this point is then made the current "primary note" and the program returns to 100. If more than one line has a new note at this attack time, one of them will be automatically chosen as the "primary note". (Again the value calculated by FINDFD is not needed in this program, and it is also made equal to a dummy variable

NDUMY2.)

The statements at 460 are executed only when the end of a section has been determined by the test at 400. First we try to proceed to the next section. If, however, the original section was the last section in the composition this will not be possible and the value of ENDC will be set to 1 and we proceed to 470, instead of returning to statement 1. At 470 we similarly try to proceed to the next composition. If the original composition was the last composition on the tape this will not be possible and the value of ENDF will be set to 1 and we proceed to 480 and STOP, instead of returning to 1.

After gathering the information in block 1 on the simultaneity at a certain attack point, all the intervals between the various note-pairs are calculated and checked to see if they satisfy the conditions of the program task:

```
C
C
C      BLOCK 2
C
C  DETERMINING INTERVALS
  200 JX=NUMLYN-1
      DO 399 JJ = 1,JX
      IF(NCL(JJ).EQ.14)GO TO 399
      JY=JJ+1
      DO 398 JJJ = JY,NUMLYN
      IF(NCL(JJJ).EQ.14)GO TO 398
  210 INT=IABS(NCL(JJ)-NCL(JJJ))
      IF(INT.EQ.5.OR.INT.EQ.7)GO TO 250
      IF(INT.EQ.6) GO TO 260
      GO TO 398
  250 IAC=0
      IF(NPRAC(JJ).GT.0.OR.NSGAC(JJ).GT.0)GO TO300
      IF(NPRAC(JJJ).GT.0.OR.NSGAC(JJJ).GT.0)GO TO 300
      IF((RDNC(JJ).NE.DNCSAV(1)) .AND.(RDNC(JJ).NE.DNCSAV(2)))GO TO 398
      IF((RDNC(JJJ).NE.DNCSAV(1)).AND.(RDNC(JJJ).NE.DNCSAV(2)))GOTO 398
      IAC=1
      GO TO 300
  260 IAC = 0
      IF((RDNC(JJ).NE.DNCSAV(1)) .AND.(RDNC(JJ).NE.DNCSAV(2)))GO TO 126C
      IF((RDNC(JJJ).NE.DNCSAV(1)).AND.(RDNC(JJJ).NE.DNCSAV(2)))GOTO 126C
      GO TO 261
 1260 CONTINUE
      IF ( (NPRAC(JJ).NE.0) .OR. (NSGAC(JJ).NE.0) ) GO TO 261
      IF ( (NPRAC(JJJ).NE.0) .OR. (NSGAC(JJJ).NE.0) ) GO TO 261
```

```
      IAC=1
  261 GO TO 350
C
C     BLOCK 3
C
C  WRITE SECTION FOR CORRECTED 4THS AND 5THS
  300 CONTINUE
      WRITE(6,301)
  301 FORMAT (1X,'.....................................')
      IF (IAC .GT. 0) GO TO 1302
      WRITE(6,302)
  302 FORMAT (1H0,'CORRECTED FOURTH OR FIFTH BETWEEN')
      GO TO 310
 1302 WRITE(6,2302)
 2302 FORMAT(1H0,'CORRECTED FOURTH OR FIFTH ASSUMING')
      WRITE(6,303)
      WRITE(6,354)
      WRITE(6,303)
      WRITE(6,355)
  310 CONTINUE
      WRITE(6,303)
  303 FORMAT(1H+,'_____________________________________')
      NPAIR = 1
      NWRIT = JJ
  320 CONTINUE
      WRITE(6,321) NMESNO(NWRIT),NLYNE(NWRIT),NNOTE(NWRIT),RDNC(NWRIT),
     1 NCL(NWRIT),NSGAC(NWRIT),NPRAC(NWRIT)
  321 FORMAT (/,6X,'MEASNO LYNENO NOTENO',/,6X,3I6,/,6X'DNC NOTECL SUGGA
     1C PRECAC',/,7X,1A1,3I6,/)
      IF (NPAIR .GT.1) GO TO 390
      NPAIR = NPAIR + 1
      NWRIT = JJJ
      WRITE(6,322)
  322 FORMAT (1H ,20X,'AND')
      WRITE(6,323)
  323 FORMAT (1H+,20X,'___',/)
    . GO TO 320
C  WRITE SECTION FOR AUG.4THS AND DIM.5THS
  350 CONTINUE
      WRITE(6,301)
       IF (IAC.GT.0) GO TO 352
      WRITE(6,351)
  351 FORMAT (1H0,'TRITONE OR DIMINISHED 5TH  BETWEEN')
      GO TO 310
  352 WRITE(6,353)
  353 FORMAT(1H0,'TRITONE OR DIMINISHED 5TH ASSUMING')
      WRITE(6,303)
      WRITE(6,354)
```

```
354 FORMAT(1H ,'ACCIDENTAL LASTS THROUGH MEASURE')
    WRITE(6,303)
    WRITE(6,355)
355 FORMAT(1H ,12X,'BETWEEN')
    GO TO 310
390 CONTINUE
    WRITE(6,301)
398 CONTINUE
399 CONTINUE
```

Statement 210 subtracts one NOTECL from the other, determining the interval between them. Thus:

INT = 5 (perfect 4th)
INT = 6 (augmented 4th or diminished 5th)
INT = 7 (perfect 5th)

(N.B. In the evaluation of NOTECL in the conversion program (see Chapter II p), an accidental is assumed to last through the measure, but PRECAC or SUGGAC are not altered to reflect such an assumption.)

The statements at 250 check to see if the interval was made perfect by the addition of a specified or editorial accidental. If there are no accidentals it also checks to see if the fifth or fourth calculated at 210 consists of the pitch-classes B and F, if so, then one of the pitches must have been altered by a prior accidental within the measure.

Similarly at 260 any tritone or diminished fifth calculated at 210, except that between B and F, must be the result of alteration by an accidental. If there is no accidental with either of the notes then it must be the result of a prior accidental within the measure.

(The pitch-class test at 250 and 260 assumes a key of C major. If the key was F, the pitch-classes would have to be altered to E and B-flat; if the key was G, to C and F-sharp, etc.)

The write sections do not need explanation here. They demonstrate well the ease with which identifying information can be produced in M.I.R.

There now follows a complete listing of the program and a sample of the output.

```
C       PROGRAM TO FIND SIMULTANEOUSLY SOUNDING AUG. 4THS AND DIM. 5THS
C       SWITCHES TO PUT PREP, CAL AND NGENU IN "DUMMY MODE"
```

```
      ISPREP=0
      ISCAL=0
      ISNGEN=0
      DIMENSION NMESNO(6),NLYNE(6),NNOTE(6),RDNC(6),NCL(6),NPRAC(6),
     1 NSGAC(6),NOTLST(6)
      DIMENSION DNCSAV(2)
      DATA DNCSAV/'B   ','F   '/
    1 WRITE(6,2)TITLE,SUBTIT
    2 FORMAT(1H1,10X,'SIMULTANEOUSLY SOUNDING AUG.4THS AND DIM.5THS',
     1//,1X,11A4,//,4X,6A4,//)
C     SWITCH SET TO 1 FOR EACH LINE WHEN LAST NOTE OF SECTION HAS BEEN REAI
      DO 3 J = 1,NUMLYN
    3 NOTLST(J) = 0
C
C      BLOCK 1
C
C     INITIALISE WITH LYNENO (=1)
  100 LIN=LYNENO
      LN=LIN
C     SAVE RECORD NUMBER OF NOTE IN LIN
      NAT=NREC
C     SAVE ATTACK TIME OF NOTE IN LINE LIN
      NATIN = AT(1)
      NATNUM = AT(2)
      NATDEN = AT(3)
C     SAVE RELEVANT INFORMATION OF NOTE IN LINE LN
  110 CONTINUE
      NMESNO(LN) =MEASNO
      NLYNE(LN) = LYNENO
      NNOTE(LN) = NOTENO
      RDNC(LN) = DNC
      NCL(LN) = NOTECL
      NPRAC(LN) = PRECAC
      NSGAC(LN) = SUGGAC
      IF (BARLIN .GE. 3) NOTLST(LN) = 1
C     INCREMENT LINE NUMBER BY 1
  115 CONTINUE
      LN = LN+1
C     "CONVERTS" LN IF GREATER THAN NUMLYN
      IF (LN .GT. NUMLYN) LN = LN-NUMLYN
C     CHECK TO SEE IF ALL LINES HAVE BEEN EXAMINED.
      IF (LN .EQ. LIN) GO TO 120
C     FIND SIMULTANEOUS NOTE IN NEW LN
      NDUMY1 = ATTACK(NATIN,NATNUM,NATDEN,LN)
      GO TO 110
  120 CONTINUE
C
C     BLOCK 2
```

```
C
C  DETERMINING INTERVALS
  200 JX=NUMLYN-1
      DO 399 JJ = 1,JX
      IF(NCL(JJ).EQ.14)GO TO 399
      JY=JJ+1
      DO 398 JJJ = JY,NUMLYN
      IF(NCL(JJJ).EQ.14)GO TO 398
  210 INT=IABS(NCL(JJ)-NCL(JJJ))
      IF(INT.EQ.5.OR.INT.EQ.7)GO TO 250
      IF(INT.EQ.6)  GO TO 260
      GO TO 398
  250 IAC=0
      IF(NPRAC(JJ).GT.0.OR.NSGAC(JJ).GT.0)GO TO300
      IF(NPRAC(JJJ).GT.0.OR.NSGAC(JJJ).GT.0)GO TO 300
      IF((RDNC(JJ).NE.DNCSAV(1)) .AND.(RDNC(JJ).NE.DNCSAV(2)))GO TO 398
      IF((RDNC(JJJ).NE.DNCSAV(1)).AND.(RDNC(JJJ).NE.DNCSAV(2)))GOTO 398
      IAC=1
      GO TO 300
  260 IAC = 0
      IF((RDNC(JJ).NE.DNCSAV(1)) .AND.(RDNC(JJ).NE.DNCSAV(2)))GO TO 1260
      IF((RDNC(JJJ).NE.DNCSAV(1)).AND.(RDNC(JJJ).NE.DNCSAV(2)))GOTO 1260
      GO TO 261
 1260 CONTINUE
      IF ( (NPRAC(JJ).NE.0) .OR. (NSGAC(JJ).NE.0) ) GO TO 261
      IF ( (NPRAC(JJJ).NE.0) .OR. (NSGAC(JJJ).NE.0) ) GO TO 261
      IAC=1
  261 GO TO 350
C
C     BLOCK 3
C
C  WRITE SECTION FOR CORRECTED 4THS AND 5THS
  300 CONTINUE
      WRITE(6,301)
  301 FORMAT (1X,'.....................................')
      IF (IAC .GT. 0) GO TO 1302
      WRITE(6,302)
  302 FORMAT (1H0,'CORRECTED FCURTH OR FIFTH BETWEEN')
      GO TO 310
 1302 WRITE(6,2302)
 2302 FORMAT(1H0,'CORRECTED FOURTH OR FIFTH ASSUMING')
      WRITE(6,303)
      WRITE(6,354)
      WRITE(6,303)
      WRITE(6,355)
  310 CONTINUE
      WRITE(6,303)
  303 FORMAT(1H+,'_____________________________________')
```

```
      NPAIR = 1
      NWRIT = JJ
  320 CONTINUE
      WRITE(6,321) NMESNO(NWRIT),NLYNE(NWRIT),NNOTE(NWRIT),RDNC(NWRIT),
     1 NCL(NWRIT),NSGAC(NWRIT),NPRAC(NWRIT)
  321 FORMAT (/,6X,'MEASNO LYNENO NOTENO',/,6X,3I6,/,6X'DNC NOTECL SUGGA
     1C PRECAC',/,7X,1A1,3I6,/)
      IF (NPAIR .GT.1) GO TO 390
      NPAIR = NPAIR + 1
      NWRIT = JJJ
      WRITE(6,322)
  322 FORMAT (1H ,20X,'AND')
      WRITE(6,323)
  323 FORMAT (1H+,20X,'___',/)
      GO TO 320
C  WRITE SECTION FOR AUG.4THS AND DIM.5THS
  350 CONTINUE
      WRITE(6,301)
       IF (IAC.GT.0) GO TO 352
      WRITE(6,351)
  351 FORMAT (1H0,'TRITONE OR DIMINISHED 5TH  BETWEEN')
      GO TO 310
  352 WRITE(6,353)
  353 FORMAT(1H0,'TRITONE OR DIMINISHED 5TH ASSUMING')
      WRITE(6,303)
      WRITE(6,354)
  354 FORMAT(1H ,'ACCIDENTAL LASTS THROUGH MEASURE')
      WRITE(6,303)
      WRITE(6,355)
  355 FORMAT(1H ,12X,'BETWEEN')
      GO TO 310
  390 CONTINUE
      WRITE(6,301)
  398 CONTINUE
  399 CONTINUE
C
C     BLOCK 4
C
C     TEST TO SEE IF LAST NOTE OF SECTION IN EACH LYNE HAS BEEN READ
  400 CONTINUE
      DO 410 J = 1,NUMLYN
      IF (NOTLST(J) .EQ.0) GC TO 420
  410 CONTINUE
      GO TO 460
C     RE-INITIALISE WITH NOTE IN LYNE LIN
  420 CONTINUE
      CALL TOCURN (NAT)
C     EXAMINES ALL LYNES FOR FIRST SUCCEEDING ATTACK POINT
```

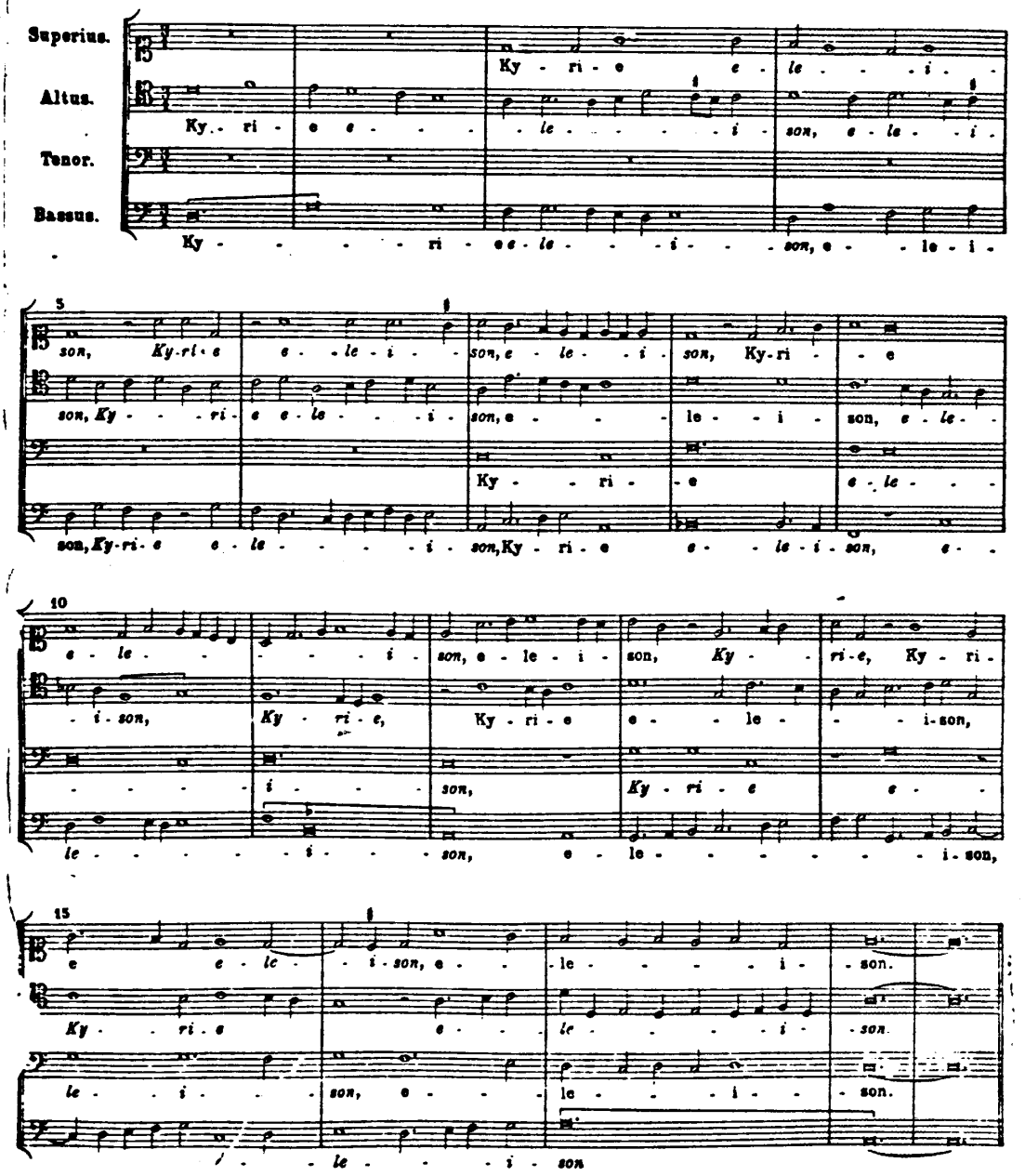
```
      CALL FINDFD(NDUMY2)
      GO TO 100
C  MOVE TO NEW SECTION
  460 CONTINUE
      CALL TOSECN(NSEC+1)
C     CHECK TO SEE IF CURRENT SECTION IS LAST IN CURRENT COMPOSITION
      IF (ENDC .GT. 0) GO TO 470
      GO TO 1
C  MOVE TO NEW MASS
  470 CONTINUE
      CALL TOCOMP(NFIL+1)
C     CHECK TO SEE IF CURRENT COMPOSITION IS LAST ON TAPE
      IF (ENDF .GT. 0) GO TO 480
      GO TO 1
  480 STOP
      END
```

Insert 3

Missa L'homme armé.

Super voces musicales.

Kyrie.



SAMPLE PRINTOUT

SIMULTANEOUSLY SOUNDING AUG.4THS AND DIM.5THS

MISSA L-HOMME ARME SUPER VOCES MUSICALES

KYRIE 1

.....

TRITONE OR DIMINISHED 5TH BETWEEN

MEASNO	LYNENO	NOTENO
3	1	3

DNC	NOTECL	SUGGAC	PRECAC
G	7	0	0

AND

MEASNO	LYNENO	NOTENO
3	2	6

DNC	NOTECL	SUGGAC	PRECAC
C	1	1	0

.....

.....

TRITONE OR DIMINISHED 5TH ASSUMING ACCIDENTAL LASTS THROUGH MEASURE BETWEEN

MEASNO	LYNENO	NOTENO
3	1	4

DNC	NOTECL	SUGGAC	PRECAC
G	7	0	0

AND

MEASNO	LYNENO	NOTENO
3	2	8

DNC	NOTECL	SUGGAC	PRECAC
C	1	0	0

.....

.....

CORRECTED FOURTH OR FIFTH BETWEEN

MEASNO	LYNENO	NOTENO

```
          8      3      1
     DNC NOTECL SUGGAC PRECAC
      F      5      0      0
```

AND

```
     MEASNO LYNENO NOTENO
          8      4      1
     DNC NOTECL SUGGAC PRECAC
      B     10      0      2
```

. . . . . .
. . . . . .

CORRECTED FOURTH OR FIFTH BETWEEN

```
     MEASNO LYNENO NOTENO
          8      3      1
     DNC NOTECL SUGGAC PRECAC
      F      5      0      0
```

AND

```
     MEASNO LYNENO NOTENO
          8      4      1
     DNC NOTECL SUGGAC PRECAC
      B     10      0      2
```

. . . . . .
. . . . . .

CORRECTED FOURTH OR FIFTH BETWEEN

```
     MEASNO LYNENO NOTENO
          8      3      1
     DNC NOTECL SUGGAC PRECAC
      F      5      0      0
```

AND

```
     MEASNO LYNENO NOTENO
          8      4      1
     DNC NOTECL SUGGAC PRECAC
      B     10      0      2
```

. . . . . .
. . . . . .

CORRECTED FOURTH OR FIFTH ASSUMING ACCIDENTAL LASTS THROUGH MEASURE

BETWEEN

```
MEASNO LYNENO NOTENO
     8      1      4
DNC NOTECL SUGGAC PRECAC
 F      5      0      0
```

AND

```
MEASNO LYNENO NOTENO
     8      4      2
DNC NOTECL SUGGAC PRECAC
 B     10      0      0
```

.....

ETC.

SUGGESTIONS FOR M.I.R. USERS

Whenever a tape error message appears, check the tape file by running some other program on it; or, re-run the original program. This error is usually caused by some temporary malfunction of the hardware or system and is not the fault of your program.

Save time by storing information whenever possible, to cut down on retrieval operations.

Whenever a program is such that it will eventually come to the end of a section or a lyne, insert some kind of check after the use of a pertinent retrieval operation to test the status of the current note. Otherwise one may get into a loop at the last note which will re-occur as the current note.

Some retrieval operations go from section to section without any indication except the change in the value of NSEC. This should be tested, or the value of BARLIN should be examined.

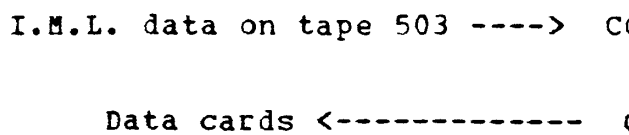
Use fractional values when examining durations or attack-times. The use of floating-point numbers or their equivalents is more trouble than it's worth. The subprograms which handle fractions will be handy for this purpose.

Any program designed by the user should be tried out on test data first. This data should be such as to cause every logical route through the program to be executed, not counting those which would halt the operation of the program. Such test data can be converted as the last file on the input tape normally used, thus avoiding extensive rearrangement of the M.I.R. program's job control cards.

APPENDIX 1A

THE CORRECTION PROCESS

1St program



1st card, column 1 - 3:
A number representing the number of files to be transmitted (note entire contents of tape must be transferred, even if only one file is being updated).

Successive cards:
A file identification card containing
A) asterisks in columns 1 - 4
B) file number in 5 - 7
C) SW1 setting in 8 - 10; (punch 000 or 001)
1 = Resequence file
0 = Do not resequence file
D) SW2 in 11 - 13
1 = Punch new deck
0 = Do not punch new deck
E) SW3 in 14 - 16
1 = Print entire file
0 = Print only correction cards, underscored.
F) SW4 in 17 - 19
1 = Delete entire file before inserting correction cards
0 = Merge correction cards in already existing file

2nd program

Updated I.M.L. on 158 ----> restore to ----> 503

One data card similar to 1st data card of the previous program

APPENDIX 1B

Coding of Triplets, etc., in I.M.L.

The procedure in I.M.L. for coding groups of notes which subdivide metric values by a number different from that provided by normal notation ("groupettes"), is similar to that for triplets. Instead of a 3, the parentheses following the + sign contain a value n, where n indicates the number of subdivisons into which a particular metric span is subdivided.

The conversion program translates the following numbers into the stipulated proportions. (Any other single number for the representation of a groupette will result in an error in calculating the durations for the note involved.)

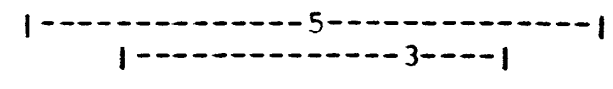
Groupette number	Proportion
3 (Triplet)	3 in the time of 2
4	4 in the time of 3
5	5 " 4
6	6 " 4
7	7 " 6
8	8 " 6
9	9 " 8
10	10 " 8
11	11 " 8
12	12 " 8
13	13 " 12
14	14 " 12
15	15 " 12
16	16 " 12

Groupettes may, however, be specified with more precision in the form:

+(n:m) . . . . +

where n groupette divisons in the time of m divisons is meant. Thus the factor m/n is the relative value of this groupette's notes as opposed to their value were there no groupette indicated at this point.

Nested groupettes, for example:



introduce a complication. Within any groupette field another field may be begun and ended. The example would be coded:

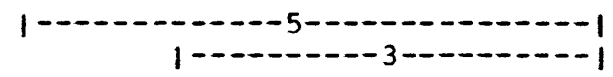
+(5) . . . . +(3) . . . + . . . +

For clarity, nested groupettes may be indicated by a different number of + symbols, for example:

+(5) . . . . ++(3) . . . ++ . . . +

but this is not required.

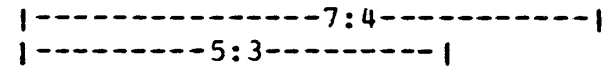
Simultaneous terminations (or beginnings) are indicated by the letter X in the following manner. Given the case:



the representation would be.

+(5) . . . . +(3) . . . . +X+

Given the case:



the representation would be.

+(7:4)X+(5:3) . . . . + . . . +

APPENDIX 2A
I.M.L. CONVERSION PROGRAM MODULES

MODULE A

MAIN

Handles processing of the I.M.L. data-stream into meaningful parsings which are then analyzed to produce information for the M.I.R. registers via appropriate subroutines.

PROGRM

Contains variables which are set by the user to indicate which version of the conversion program is required.

BCD

Translates 026 keypunched material into 029 characters.

ENDTS1

Tests for end of input card image.

ENDTS2

Tests for end of printed staff.

ENTRO1

Stores reformatted information about ties, triplets and other miscellaneous information.

ENTRO2

Same for rhythmic information.

ENTRO3

Same for pitch information.

ENTRO4

Same for text.

ENTROL

Same for lyne and measure number.

ENTROM

Same for measure number alone.

ENTROC

Same for comments.

PRINTR

Prints arrays containing reformatted I.M.L.

CONSUB

Identifies analysable comments as LYNE comment, or REDUCE comment, or TITLE comment, etc.

TABCOM

Helps set up calls to comment-analysing routines.

ENSTRU

Analyses LYNE comments.

ENSTRN

Handles (in the future, maybe) transposing

instruments.

(Rhythmic checking portion)

ENTTSV
: Calculates time-signature registers.

ENTRET
: Calculates RETIME information.

ENDRED
: Calculates REDUCE information.

ENTRHY
: Isolates rhythmic information.

CALRHY
: Analyses rhythmic information.

TRGPN
: Calculates triplet registers.

FRALCD
: Finds lowest common divisor and reduces fractions.

FRACAD
: Adds fractions.

FRAMPY
: Multiplies fractions.

MODULE B

ENTPIC
: Isolates pitch-information.

ENTUNT
: Sets flag to interpret the following data as "untucked", (from UNTUCK comment).

ENTTUC
: Sets flag to interpret the following data as "tucked", (from TUCK comment).

ENTCLF
: Calculates clef-register.

ENTEXT
: Processes text.

CALPIC
: Calculates pitch-registers.

ENTKEY
: Processes key-signature information.

ENTPC
: Calculates key-signature accidentals from key-name.

NUMPIC
: Translates diatonic pitch characters into

numerical values.

NTCTAB

Translates "untucked" diatonic values into real ones.

STATAB

Determines staff position number.

NSMTAB

Finds absolute semitone value number.

ENTNOT

Processes noteblock information; keeps track of pitch-values lasting for more than one note.

MODULE C
(Prints out lyne and note registers)

NT1

Prints MEASNO LYNENO NOTENO NREGCL NOTECL NTIEND.

NT2

Prints NSEMIT DNC NBARCT NPRECA NSTAFP NDURAT NDOTND

NT3

Prints MAXLYN NGRPNO NSUGGA NBRACK NPHMRK NSPECN.

NT4

Prints text.

NT5

Prints measure-attack time.

NT6

Prints system-attack time.

NT7

Prints composition-attack time.

TRAHED

Sets up HEDBLK information in AHED.

MODULE D
(Tape-writing routines)

WRIHED

Writes HEDBLK information onto tape.

WRITAL

Writes TALBLK information onto tape.

WRILYN

Writes LYNBLK onto tape.

WRISYS

Dummy.

NT8

Writes NOTBLK onto tape.

IOEND

Writes ENDBLK record onto tape (from FILE END statement).

Subroutine PROGRM contains two variables which are set by the user. The first variable, NPROG can be set to one of three values:

NPROG = 1
This will call the tape-writing version.
NPROG = 2
This will call the reformatting version.
NPROG = 3
This will call the register-printout version.

The second variable NFLCOM is only altered for the tape-writing version (for the reformatting and register-printout versions it should be set to 0). For the tape-writing version NFLCOM will become NFIL of the Label register and should be set to the "serial number of the current composition". Because of the use of subroutine PROGRM in this way, only one file can be converted at a time.

APPENDIX 2B
Messages in the Conversion program

Generated in MAIN

PUNCTUATION MISSING ON CARD X
: Some unexpected character has occurred in the I.M.L. character stream.

!
: Erroneous information for pitch or duration which this character replaces.

RHYTHMIC MISTAKE
: The duration of one measure of music does not correspond to the time-signature.

MEASURE-NUMBER EXCEEDS THREE DIGITS
: Program halts.

ERROR ON CARD X PERTAINING TO TRIPLETS OR TIES
: Missing or unexpected information when processing a triplet or tie.

TOO MANY LYNES, LYNENO= X
: More than 99 lynes are not allowed, program halts.

Generated in ENDIS1

INPUT OUT OF SEQUENCE (card number)
: Sequence-number on card was not greater than that of the previous card in the same composition, program halts. Note that the FILE END card must be in sequence with the last card of the composition which it follows.

END OF JOB
: Normal termination of a reformatting or conversion job has occurred.

Generated in ENSTRU

LYNE INFO MISSING AROUND CARD X
: Information in the lyne comment is not specified correctly, program halts.

Generated in CALRHY

IMPOSSIBLE RHYTHM (card image)
Numerical symbols for rhythmic indication are not possible ones.

Generated in ENTTSV

ERROR IN TIME-SIGNATURE (card image)
Incorrectly specified time-signature (missing parentheses, or /, or- symbols) -- check to see that only one space occurs after keyword.

TOO MANY SYMBOLS IN TIME-SIGNATURE (card image)
No more than 4 characters (not including parentheses) can define a symbolic time-signature.

TIME-SIGNATURE ERROR (card image)
A "fractional" time-signature has non-numeric characters in it (besides / or -).

Generated in ENTRET

TOO MANY DIGITS IN RETIME NUMERATOR (card image)
Check for missing =. Check for missing spaces or erroneous spaces.

TOO MANY DIGITS IN RETIME DENOMINATOR (card image)
Check for missing /. Check for missing spaces or erroneous spaces.

Generated in ENDRED

MISSING HYPHEN IN REDUCE COMMENT (card image)

TOO MANY DIGITS IN REDUCE NUMERATOR (card image)
Check for missing =. Check for missing spaces or erroneous spaces.

TOO MANY DIGITS IN REDUCE DENOMINATOR (card image)
Check for missing /. Check for missing spaces or erroneous spaces.

MISSING DENOMINATOR IN REDUCE STATEMENT (card image)
Check for missing spaces or erroneous spaces.

Generated in FRALCD

NEGATIVE DENOMINATOR (denominator)

Denominator of fraction being reduced to lowest terms is either zero or negative. Check for other errors in the data.

ERROR IN REDUCING FRACTION
More than 100 steps in the Euclidian algorithm were needed to find the lowest common denominator. Try to find other errors in the data.

Generated in NUMPIC
ERRONEOUS DIATONIC PITCH = X (card image)
Argument DIA not valid.

Generated in ENTPC
KEYREP ERROR (card image)
First character not A through G or second character not S or T.

Generated in ENTKEY
KEYSIG ERROR (card image)
Second character not S, T, or blank.

Generated in ENTNOT
NGRPNO ERROR (triplet number)
Missing triplet information.

Generated in ENTKEY
KEY INFORMATION ERROR (card image)
Missing space after keyword.

KEYSIG INFO ERROR (card image)
Letter name not A through G.

Generated in CALPIC
MISSING ACCIDENTAL (card image)
Suggested accidental symbols not S, T, or N.

Generated in NTCTAB, STATAB, NSMTAB
ERRONEOUS CLEF NUMBER = X
Clef information erroneous or missing.

Generated in ENTEXT

TEXT TOO LONG (card image)

No more than 16 characters can be processed.

Generated in ENTCLF

CLEF INFO MISSING (card image)

Space missing after keyword.

ERRONEOUS CLEF INFORMATION (card image)

First character not C F G L or U and 2nd character of C-clef indication not 1 2 3 or 4; or 2nd character of F-clef indication not 3, 4, 5, or blank.

APPENDIX 2C

Updating M.I.R. tapes

When the I.M.L. data for a composition are corrected the M.I.R. input tape has to be corrected as well. Since there is no way of correcting individual registers on the tape all the music bearing a single file-number has to be reconverted.

The tape-numbers refer to those at Princeton, and will obviously vary at other installations.

Tape — Contents

0503
I.M.L. data of twenty masses, each as a separate file.

3451
M.I.R. input tape of twenty masses each as a separate file.

0723
M.I.R. input tape of twenty masses combined as a single file.

0158 And 2186
Scratch tapes.

There are four programs associated with the updating procedure.

MIRP1 (The Conversion Program)
Converts file of 0503 and puts output onto file of 0158.

MIRP2
Merges file(s) of 0158 with old file(s) of 3451 and puts output onto 2186.

MIRP3
Copies 2186 onto 3451.

MIRP4
Copies 2186 onto 0723 and combines the separate

files as a single file.

Data cards required by the programs:

MIRP1

The use of subroutine PROGRM for this program is explained in Appendix 2A.

MIRP2

This program requires twenty pairs of data cards. The first card of each pair indicates the tape-number (normally 2186) and file-number that the file indicated on the succeeding card will be transferred to. Thus if files 1 - 2; 4 - 20 of 3451 are to be merged with a single file on 0158 (which already contains a corrected version of mass 3), the data cards will be arranged as follows.

Card columns	1 4	7
	2186	01
	3451	01
	2186	02
	3451	02
	2186	03
	0158	01
	2186	04
	3451	04
	2186	05
	.	.
	.	.
	.	.
	.	.
	2186	20
	3451	20

The data cards are read in I4,2X,I2 format.

MIRP3

This program requires a single data card which contains the total number of files that are to be copied from 2186 to 3451 (normally 20), the first three columns of the data card are read in I3 format.

MIRP4

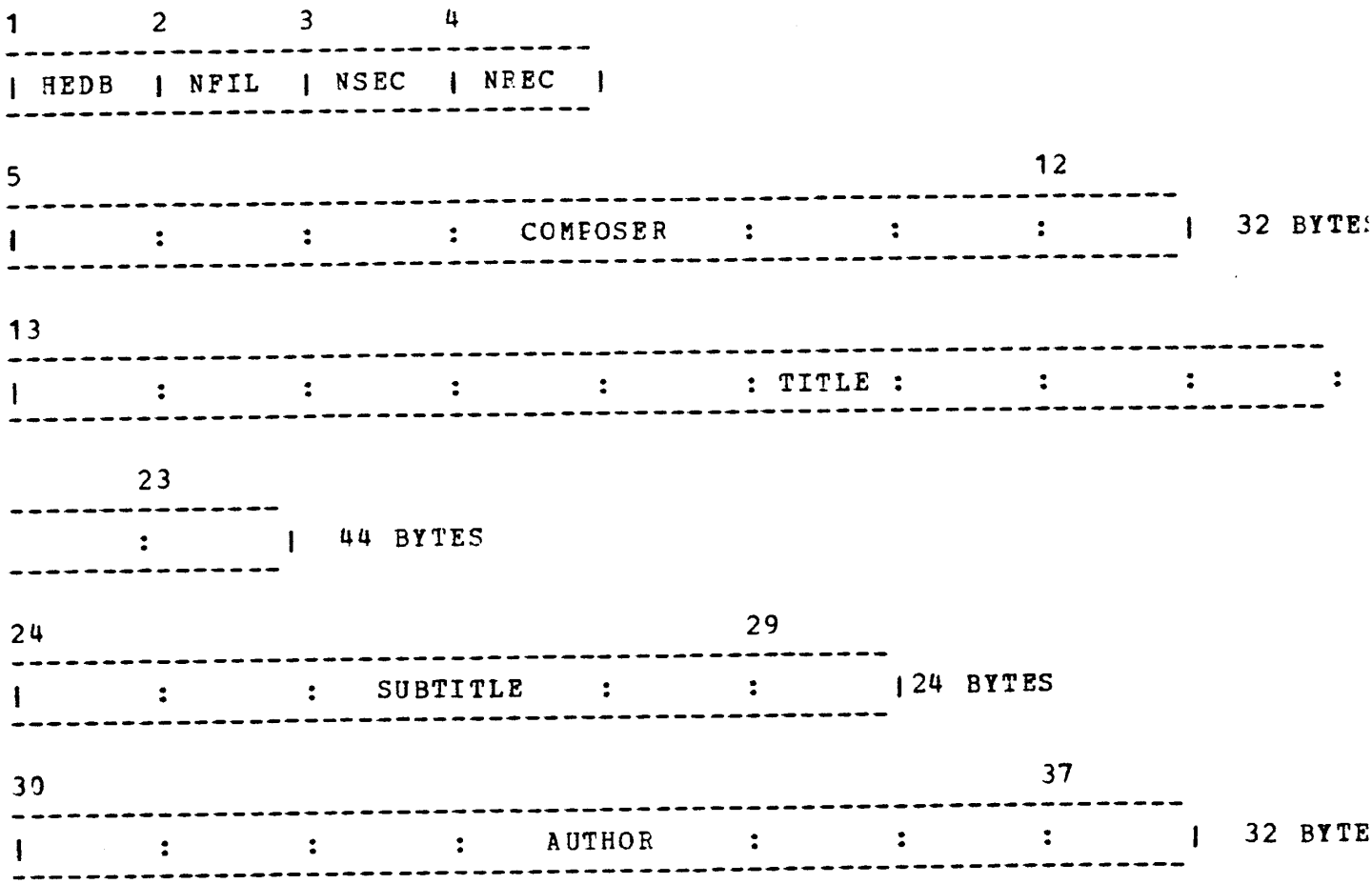
Requires no data card.

APPENDIX 3A

RECORD FORMAT FOR M.I.R. PROGRAM INPUT ON TAPE OR DISK

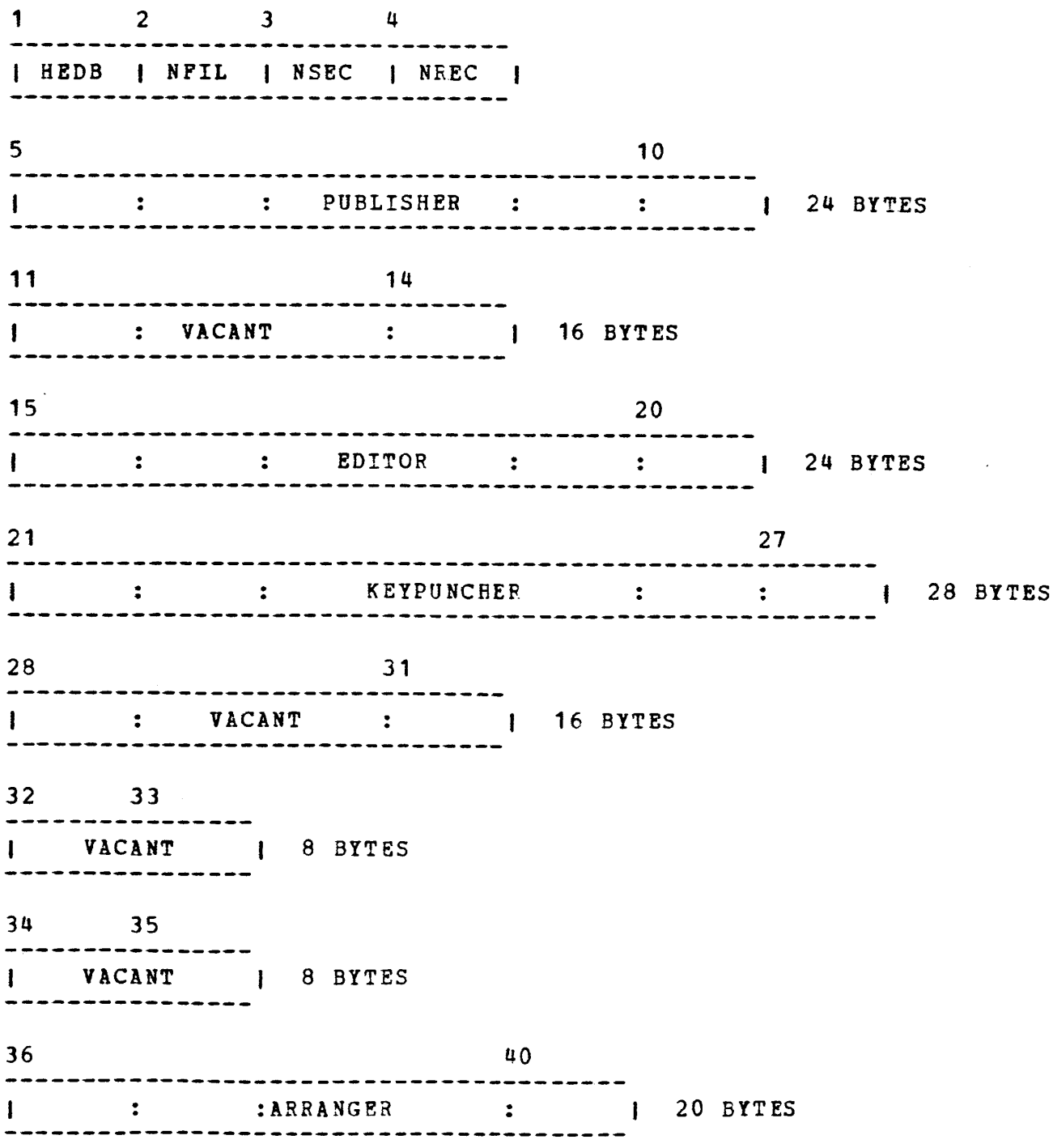
HEDBLK (2 PARTS)

PART 1

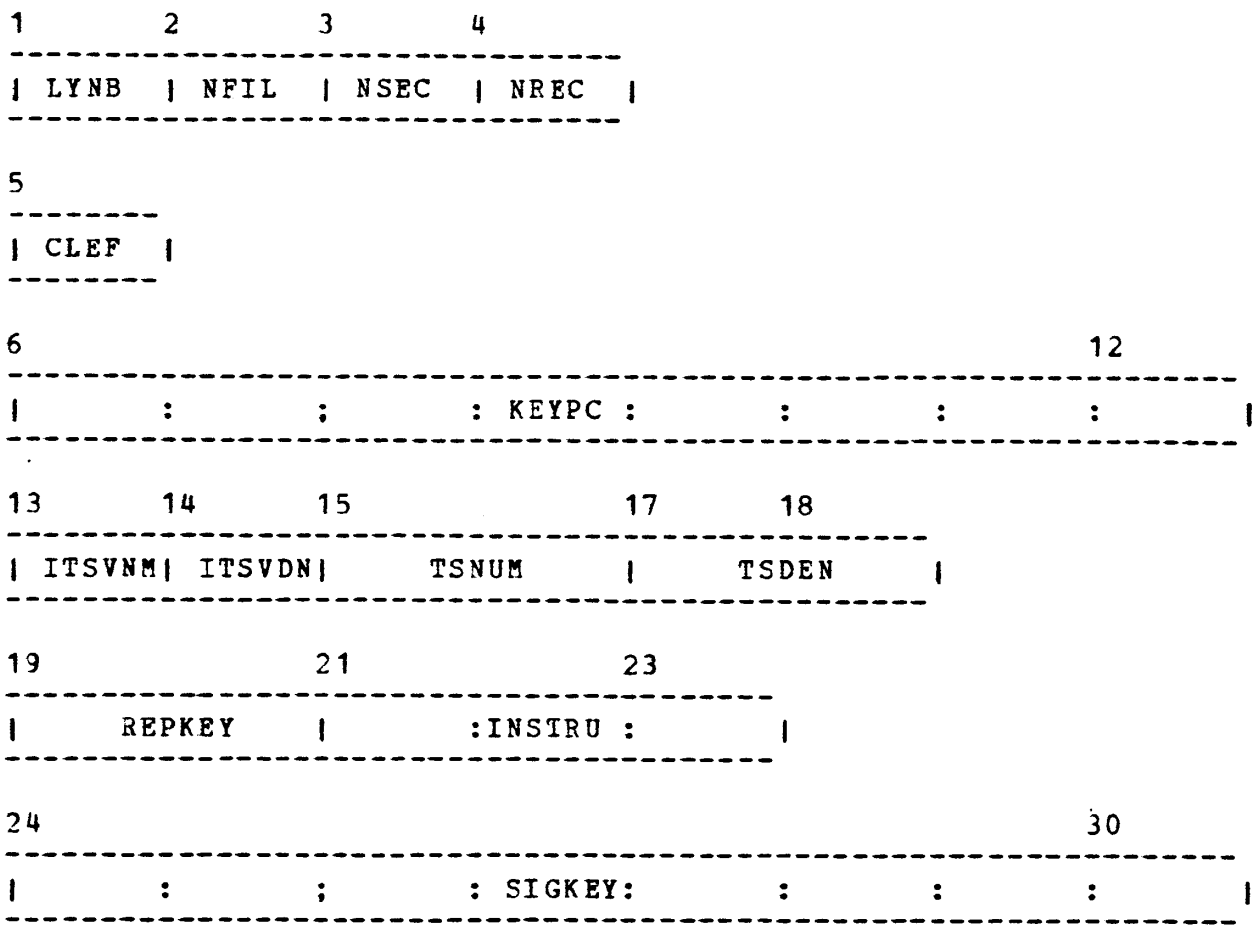


+ 3 Unused words = 40 word record written

PART 2


1 2 3 4
HEDB
NFIL
NSEC
NREC
5 10
PUBLISHER
24 BYTES
11 14
VACANT
16 BYTES
15 20
EDITOR
24 BYTES
21 27
KEYPUNCHER
28 BYTES
28 31
VACANT
16 BYTES
32 33
VACANT
8 BYTES
34 35
VACANT
8 BYTES
36 40
ARRANGER
20 BYTES

LYNBLK



+ 10 Unused words = 40 word record written

NOTBLK

1	2	3	4
NOTB	NFIL	NSEC	NREC

5	6	7	8	9	10	11
MEASNO	LYNENO	NOTENO	NOTECL	REGSTR	SEMITO	PRECAC

12	13	14	15	16	17	18
SUGGAC	STAFPO	DURAT	DOTINL	NUMLYN	GRPNO	BARLIN

19	20	21	22	23	24	25
BRACK	SPECSN	VACANT	MESINT	MESNUM	MESDEN	SYSINT

26	27	28	29	30	31	32
SYSNUM	SYSDEN	ATINT	ATNUM	ATDEN	DURINT	DURNUM

33	34	35	39
DURDEN	DNC	: TEXT :	TIEIND

+ 1 Unused word = 40 word record written

APPENDIX 3B

Messages in the M.I.R. program subprograms

Generated in READBK
IMPOSSIBLE RECORD
The label for a M.I.R. tape record is not one allowed, program halts.

Generated in TPGET
IND = (code) ERROR AFTER RECORD (record label)
The code given is that of the TPCHECK routine (see Appendix 4), program halts.

Generated in TOSECT
(subtitle) CANNOT BE FOUND
NFSEC is set to 1: no section bearing the name called for.

Generated in TOSECN
MISSING SECTION
Tape not generated correctly, program halts.

Generated in TOTITL
(title) CANNOT BE FOUND
NFCOM is set to 1: no composition on this tape file bearing the name called for.

Generated in TOCOMP
MISSING FILE
Search for a composition by number failed; either number too large or tape incorrectly generated, program halts.

APPENDIX 4

Routines in Machine Language
Written by Hale Trotter, used in the I.M.L.-M.I.R. - Fortran system

Input-Output routines for the M.I.R. system.

A subroutine package has been written which allows a Fortran program to exercise almost complete direct control over a tape unit. For example: it can be used to pass over a bad spot on a tape and to read the records which follow the bad spot. In some applications it can yield improved efficiency, since it bypasses some of the system overhead involved in standard Fortran I/O operations. Its disadvantage is that it provides no automatic blocking or error recovery or positioning features; these must be programmed in detail by the user.

REAL * 8 CA(14)

For each tape to be used under control of these routines, the user must provide an array of 14 REAL * 8 words. This array appears as an argument in every subroutine call, and serves to identify the tape to be used. The routines may thus be used with several tapes at once. In the following calling sequences, CA is used for the name of the control array, and it is asssumed that a REAL * 8 CA(14) statement has appeared in the program.

CALL TPOPEN (CA,NAME)

This call must be executed before any of the other routines can be used. It may be a Hollerith argument. Blanks must be added at the end if necessary to make a full 8 characters. The call opens the dataset and initialises CA with control information: DCB, IOB, and Channel Program -- used by other routines.

CALL TPREAD (CA,AREA,NBYTES)

This call initiates a reading of the next physical record. The record is read into the array AREA, and NBYTES is an integer specifying the maximum number of bytes to be read. (If the record contains more than this number of bytes,

the excess is ignored.) Any call to TPREAD must be followed by a call of TPCHEK before the data read is used, or any other operation is done on the same tape.

TPCHECK (CA,IND,LEN)

This call checks the completion of a previous read ot write. If the previous operation was a read, LEN is set to the number of bytes actually read (i.e., the smaller of NBYTES and the physical-record-length). IND is set to zero if the operation was completed normally.

Other possible values are:

1

Tape Mark (EOF) encountered while reading, or EOF encountered while writing.

4

Data check (bad tape); data may be in error. Usually worth backspacing and retrying a few times. A data check while writing indicates a bad spot on the tape; the IBM manual suggests backspacing, erasing a section using TPWGAP and retrying the write procedure.

8

Error condition; no point in retrying. Commonest causes are running off the end of the tape, or trying to write on a file-protected tape.

It is theoretically possible for more than one condition to occur at once; in this case the codes are added. For example, if a data check occurs as a tape-mark is read, a code of 5 will be set.

CALL TPBKSP (CA,IND)

Backspace one record (a tape mark counts as a record). No need to call TPCHEK afterwards.

Listing of Source Deck

```
TPZX    TITLE   'TAPE HANDLER--MODIFIED FOR 3-BYTE RECORD LENGTHS'
        MACRO
        ENTER &INT              LM&INT  IS EXTERNAL ENTRY
        ENTRY   TP&INT
        USING   *,15
TP&INT  B       COMSAV          GOTO COMMON ENTRY
        DC      X'07'           BYTE  OUNT
        DC      CL7'TP&INT'     AND ENTRY POINT NAME
        B      &INT             AND BRANCH TO REAL ENTRY
        DROP    15
        MEND
        MACRO
&LOC    UTIL    &CODE
&LOC    MVI     UCCW,X'&CODE'   SET UP CHANNEL OPCODE
        B       UTIL            BRANCH TO COMMON ROUTINE
        MEND
        MACRO
&LOC    TEST    &BYTE,&MASK,&MOD
&LOC    TM      &BYTE,X'&MASK'
        BZ      A&SYSNDX
        OI      ARG2+3,&MOD     OR IN SIGNAL BIT
A&SYSNDX        DS      0H
        MEND
TPZILCH  START
        ENTER   OPEN            CALL LMOPEN(ID,'DDDDNAME')   WHERE
*                               DDDDNAME  IS  8-CHARACTER  DDNAME.
*                               (PAD WITH BLANKS IF NECESSARY-- M U S T
*                               BE  E I G H T  CHARACTERS.)  THE DATASET
*                               IS OPENED, AND  ID  SET TO A MAGIC VALUE
*                               USED TO REFER TO THIS TAPE IN ALL OTHER
*                               UTILITY CALLS  (MORE THAN ONE TAPE MAY
*                               BE USED, WITH DIFFERENT  ID'S.)
*       OPEN IS NORMALLY DONE FOR INPUT;  IF A THIRD ARGUMENT IS GIVEN
*       (E.G.)  IN CALL TPOPEN(ID,DDNAME,0)   OPEN WILL BE FOR OUTPUT
        PRINT   NOGEN
 ENTER  CLOS    CALL LMCLOS(ID)  TO CLOSE DATASET.  TAPE WILL REWIND IS
*               DISP=DELETE  WAS SPECIFIED ON  DD CARD, WILL UNLOAD
*               IF  DISP=KEEP  WAS SPECIFIED.
 ENTER  READ    CALL TPREAD(ID,AREA,NBYTES)
 ENTER  WRIT    CALL TPWRIT(ID,AREA,NBYTES)
 ENTER  CHEK    CALL TPCHEK(ID,IND,LEN)
 ENTER  TYPE    CALL TPTYPE(ID,ITYPE)
 ENTER  BKSP    CALL LMBKSP(ID,IND)  TO BACKSPACE ONE RECORD
 ENTER  BKFL    CALL LMBKFL(ID,IND)  TO BACK ONE FILE
 ENTER  SKFL    CALL LMSKFL(ID,IND)  TO SKIP ONE FILE FORWARD
```

```
 ENTER WGAP    CALL LMWGAP(ID,IND)  TO WRITE GAP
 ENTER WEOF    CALL LMWEOF(ID,IND)  TO WRITE END-OF-FILE (TAPE MARK)
 ENTER REWD    CALL LMREWD(ID,IND)  TO REWIND TAPE
*      IND =  1   MEANS  END-OF-FILE ON A READ OPERATION
*                        WRITE AFTER END-OF-TAPE, ON WRITE OPERATION
*             2   MEANS  BACK MOTION AT LOAD-POINT
*             4              DATA CHECK (REDUNDANCY, NOISE, ETC.)
*             8              NOT READY, OR WRITE ON FILE-PROTECTED TAP
*                            OR ANY OF A NUMBER OF MACHINE ERRORS.
*      SIMULTANEOUS CONDITIONS ARE CODED AS SUM OF INDIVIDUAL CODES
 ENTER GETM    CALL TPGETM(ID,MODE,DENS)
 ENTER SETM    CALL TPSETM(ID,MODE,DENS)
       LTORG
       PRINT  GEN
       USING   IHADCB,2       SET DSECT BASES
       USING   ARG2,3
       USING   ARG3,4
       USING   ARG4,5
COMSAV STM     14,12,12(13)   SAVE ALL
       BALR    12,0           SET  12  AS BASE
       USING *,12
BASE   LR      2,13           SET UP SAVEAREA
       LA      13,SAVEA
       ST      13,8(2)        AND POINTERS
       ST      2,4(13)        BACK
       LM      2,5,0(1)       PICK UP ARGUMENT ADDRESSES
       B       12(0,15)       BRANCH TO ENTRY-POINT BRANCH
SAVEA  DS      18F            SAVEAREA
ARG1   DSECT
ARG2   DSECT
ARG3   DSECT
ARG4   DSECT
       DCBD    DSORG=PS,DEVD=TA
       ORG     IHADCB+60
ECB    DS      F              EVENT CONTROL BLOCK
IOBASE DS      0D
IOFLG  DS      2CL1           FLAGS
IOSNS  DS      2CL1           SENSE BYTES
IOECB  DS      F              ECB  ADDR
IOCSW  DS      D              CSW  (FIRST BYTE IS FLAGS)
IOCP   DS      F              CHANNEL PROGRAM ADDRESS
IODCB  DS      F              DCB  ADDRESS
       DS      2F             RESERVED?
UCCW   DS      D              FIRST WORD OF CHANNEL PROGRAM
CCW2   DS      D              SECOND WORD OF CHANNEL PROGRAM
LIOB   EQU     *-IHADCB
JFCB   DS      22D            JFCB AREA
JEXLST DS      F              ADDRESS OF JFCB AREA
```

```
JFCBVOL EQU   JFCB+118      VOLUME SERIAL NUMBERS
JFCBVP  EQU   JFCB+77       BIT X'80' ON MEANS VOL.LAB.PROC. REQ'D
JFCBLTYP EQU  JFCB+66       ...1 0000   FOR  BLP
TPZILCH CSECT               BACK TO REALITY
TYPE    L     7,DCBDEBAD    LOAD  DEB  ADDRESS
        L     7,32(7)       LOAD  UCB  ADDRESS
        LA    8,9(0)        ASSUME 9-TRACK TAPE
        TM    17(7),X'80'   TEST  0-BIT  OF BYTE 2 OF  UCBTYP
        BZ    TAPE9         SKIP IF 9-TRACK
        LA    8,7(0)        ELSE SET  7
TAPE9   ST    8,ARG2        STORE RESULT
        B     EXIT
GETM    MVC   ARG2(1),DCBTRTCH        MOVE TRTCH BYTE TO ARG
        MVC   ARG3(1),DCBDEN          MOVE DENSITY BYTE TO ARG
        B     EXIT
SETM    MVC   DCBTRTCH,ARG2 MOVE ARGUMENT TO TRTCH BYTE
        MVC   DCBDEN,ARG3   MOVE ARGUMENT TO DENSITY BYTE
        B     EXIT
OPEN    DS    0H
        XC    IHADCB(LIOB),IHADCB   CLEAR AREA
        MVC   DCBDDNAM,ARG2  ENTER  DDNAME
        MVI   DCBDSORG,X'02'   MARK AS PHYSICAL SEQUENTIAL
        MVC   DCBOFLGS(4),IDCBF  SET OTHER FLAGS
        ST    2,IODCB       AND  DCB  ADDR
        LA    0,ECB         AND
        ST    0,IOECB       ECB ADDR
        MVI   IOFLG,X'C2'   MARK FOR CHAINING AND UNRELATED PROG
        LTR   3,3           WAS SECOND ARGUMENT THE LAST ONE?
        BP    ROPEN         IF NOT, OPEN FOR OUTPUT
        OPEN  ((2))         OPEN FOR INPUT
        B     POPEN
ROPEN   DS    0H
        OPEN  ((2),(OUTPUT))  OPEN FOR OUTPUT
POPEN   DS    0H
        LTR   15,15         REST RETURN CODE  (I.E. TEST IT)
        BZ    OPENOK        AND CONTINUE IF ZERO
        MVC   ERRDN,DCBDDNAM  ELSE SET UP ERROR MESSAGE
        CALL  FERROR,(OPERR)  PRINT ERROR MESSAGE
        ABEND 333           AND DIE (NO DUMP)
OPERR   DC    C' FAILURE TO OPEN '          START OF ERROR MESSAGE
ERRDN   DC    CL8'?DDNAME?'       REPLACED BY  DDNAME
        DC    C'.  CHECK FOR MISSING OR INCORRECT  DD  CARD. '
OPENOK  DS    0H            BRANCH HERE  IF NO OPEN ERROR
        L     7,DCBDEBAD    GET DEB
        MVC   DCBTRTCH,32(7)          AND THE MODESET BYTE
        MVC   DCBDEN,32(7)            BOTH PLACES
        NI    DCBTRTCH,X'3B'          AND SEPARATE
        NI    DCBDEN,X'C3'            THE FUNCTIONS
```

```
        MVI     UCCW+7,1        WITH A COUNT OF  1
        MVI     CCW2+4,X'20'    SET  SLI  BITS IN  CCW2
        MVI     CCW2+7,1         AND A COUNT OF 1
        LA      0,UCCW          GET THE CHANNEL PROGRAM ADDRESS
        ST      0,IOCP          AND PUT IT AWAY
        B       EXIT
IDCBF   DC      X'02'           FOR DCBOLFLGS
        DC      X'0C'  DCBIFLG  (SET TO EVADE SUPVR ERROR RTS)
        DC      X'D008'         DCBMACR  FOR  EXCP, 0 APP, 5 WORD DEVD
CLOS    CLOSE   ((2))
        B       EXIT            AND GET OUT
WRIT    ST      3,CCW2          PLANT ADDRESS
        MVI     CCW2,1          AND OP CODE
        B       RW
READ    ST      3,CCW2          PLANT ADDRESS
        MVI     CCW2,2          AND OP CODE
RW      MVC     CCW2+5(3),ARG3+1   MOVE IN 3-BYTE COUNT
        MVC     UCCW(1),DCBTRTCH        SET MODE
        OC      UCCW(1),DCBDEN          SWITCH
        MVI     UCCW+4,X'40'    SET COMMAND CHAIN BIT
        NI      ECB,0           ZERO WAIT AND COMPLETE BITS FOR LUCK
        EXCP    IOBASE          GO
        B       EXIT            BUT DON'T WAIT FOR IT
BKSP UTIL 27    BACKSPACE
        PRINT NOGEN
BKFL UTIL 2F    BACKFILE
SKFL    UTIL 3F SKIP FILE
WGAP    UTIL    17              WRITE GAP
WEOF UTIL 1F    WRITE  EOF
REWD UTIL  07    REWIND
        PRINT GEN
UTILB DS        0H
UTIL    DS      0H
        NI      ECB,0           ZERO WAIT AND COMPLETE BITS FOR LUCK
        MVI     UCCW+4,X'20'    SET SLI BITS IN UCCW
        MVI     CCW2,0          CANCEL OPCODE, SO LENGTH WON'T BE CALCULTD
        EXCP    IOBASE          GOGOGO
        WAIT    ECB=ECB         WAIT FOR CHANNEL END
        MVI     UCCW,X'CB'      SET NO-OP
        NI      ECB,0           ZERO WAIT AND COMPLETE BITS
        EXCP    IOBASE          ISSUE THE NO-OP (AND WAIT FOR DEVICE END)
CHEK    DS  0H
TPCH    TM      ECB,X'40'       CHECK IF WAIT NEEDED
        BNZ     NOWAIT          SKIP IF PROGRAM ALREADY COMPLETED
        WAIT    ECB=ECB
NOWAIT  XC      ARG2(4),ARG2    CLEAR INDICATOR WORD
        CLI     CCW2,2          TEST FOR A READ OPERATION
        BNE     NOLENG          IF NOT, SKIP LENGTH CALULATION
```

```
        L       0,CCW2+4        GET ORIGINAL COUNT
        S       0,IOCSW+4       MINUS FINAL COUNT
        SLL     0,8             AND TAKE RESULT
        SRL     0,8             MODULO  2**24
        ST      0,ARG3          AND STORE RESULT
NOLENG  DS      0H
        TEST    IOCSW+4,01,1    UNIT EXCEPTION
        TM      IOCSW+4,02      UNIT CHECK
        BZ      EXIT            SKIP IF NO STATUS
        TEST    IOSNS+1,08,2    LOAD-POINT
        TEST    IOSNS+1,80,4    NOISE ON WRITE OR SKIP OPERATION
        TEST    IOSNS,08,4      DATA CHECK
        TEST    IOSNS,F7,8      ANYTHING IN  BYTE 0  EXCEPT DATA CHECK
EXIT    L       13,4(13)
*WHAT ABOUT RETURN INDICATOR BYTE?????
        LM      14,12,12(13)
        SR      15,15           0  FOR RETURN CODE
        BR      14              GO BACK
        END
```

Character-examination routines used in the conversion program, and available for use in the M.I.R. user program

INTEGER FUNCTION LOOK(A,N,LOC,INIT,NEXT,R)

Input parameters

A	string (e.g. LOGICAL * 1 array)
N	integer - length of A in bytes
LOC	INTEGER - scan begins at A(LOC)

Operation

Initial blanks are skipped. Scan looks for a number in I, F, E, or D format (decimal point required with E or D, no blanks allowed within a number). If first non-blank character is a letter, scan continues over all consecutive letters or digits, stopping at first blank or special character. Otherwise scan takes single character only.

Output values

LOOK = 0
if remainder of string is blank. In this case values of INIT, NEXT, R are meaningless.

LOOK = 1
If I-format number is found.

LOOK = 2
If F-, E-, or D-format number is found.

LOOK = 3
If E-, or D-format number found with exponent positive and greater than 72.

LOOK = -1
If letter followed by letters and digits found.

LOOK = -2
If special character found.

INIT INTEGER
A(INIT) is first non-blank found.

NEXT INTEGER
A(NEXT) is first character not covered in scan (NEXT - INIT gives length of substring found).

R REAL * 8
Value of number found (in REAL * 8 form) if LOOK = 1 or 2.
0.0 If LOOK = 3.
String found, truncated or padded with blanks on right to make 8 characters, if LOOK = -1.
Cleared to 0 with character found inserted in 4th byte (so integer arithmetic may be done on 1st half of) if LOOK = -2.

Subroutine CHLOC (CHAR,STRING,N,LOC)

Input parameters

CHAR
Single character (LOGICAL * 1 or Hollerith)

STRING
Character string (LOGICAL * 1 array or Hollerith)

N
INTEGER - length of STRING in bytes

Output parameter

LOC
Set to location of first occurrence of CHAR in STRING, or to N+1 if CHAR does not occur in STRING.

Listing of Source Deck

```
LUKC    TITLE   'SCANNER AND SEARCHER'                          LOOK
*INTEGER FUNCTION  LOOK(A,N,LOC,INIT,NEXT,R)
*
*INPUT PARAMETERS
*       A         STRING   (E.G., LOGICAL*1 ARRAY)
*       N         INTEGER          LENGTH OF  A  IN BYTES
*       LOC       INTEGER          SCAN BEGINS AT  A(LOC)
*OPERATION
*                 INITIAL BLANKS ARE SKIPPED.  SCAN LOOKS FOR A NUMBER IN
*       I,F,E, OR D  FORMAT (DECIMAL POINT REQUIRED WITH  E  OR  D ,
*       NO BLANKS ALLOWED WITHIN A NUMBER).  IF FIRST NON-BLANK
*       CHARACTER IS A LETTER, SCAN CONTINUES OVER ALL CONSECUTIVE
*       LETTERS OR DIGITS, STOPPING AT FIRST BLANK OR SPECIAL
*       CHARACTER.  OTHERWISE SCAN TAKES SINGLE CHARACTER ONLY.
*OUTPUT VALUES
*       LOOK   = 0    IF REMAINDER OF STRING IS BLANK.  IN THIS CASE
*                     VALUES OF  INIT,NEXT,R  ARE MEANINGLESS.
*                1    IF  I-FORMAT NUMBER FOUND
*                2    IF  F-,E-, OR D-FORMAT NUMBER FOUND
*                3    IF E- OR D-FORMAT NUMBER FOUND WITH EXPONENT
*                     POSITIVE AND GREATER THAN  72
*               -1    IF LETTER FOLLOWED BY LETTERS AND DIGITS FOUND
*               -2    IF SPECIAL CHARACTER FOUND.
*       INIT      INTEGER          A(INIT) IS FIRST NON-BLANK FOUND
*       NEXT      INTEGER          A(NEXT)  IS FIRST CHAR NOT COVERED IN SCAI
*                 ( NEXT - INIT  GIVES LENGTH OF SUBSTRING FOUND)
*       R         REAL*8           VALUE OF NUMBER FOUND (IN REAL*8 FORM) IF
*                                  LOOK = 1 OR 2 .
*                                  0.0  IF  LOOK = 3.
*                                  STRING FOUND, TRUNCATED OR PADDED WITH
*                                  BLANKS ON RIGHT TO MAKE  8 CHARS,
*                                  IF  LOOK = -1.
*                                  CLEARED TO  0  WITH CHAR FOUND INSERTED IN
*                                  4TH BYTE  (SO INTEGER ARITHMETIC MAY BE
*                                  DONE ON 1ST HALF OF  R ) IF  LOOK = -2 .
*SUBROUTINE  CHID(CHARA,CHARB,ID)
*INPUT PARAMETERS
*       CHARA,CHARB    SINGLE CHARACTERS (LOGICAL*1 OR HOLLERITH)
*OUTPUT PARAMETER
*       ID        INTEGER OR LOGICAL*4    SET TO  1 (.TRUE.)  IF CHARA
*                                  AND  CHARB  ARE EQUAL, ELSE TO 0 (.FALSE.)
*SUBROUTINE CHLOC(CHAR,STRING,N,LOC)
*INPUT PARAMETERS
*       CHAR      SINGLE CHARACTER (LOGICAL$1  OR HOLLERITH)
*       STRING    CHARACTER STRING (LOGICAL*1 ARRAY  OR HOLLERITH)
```

```
*       N       INTEGER         LENGTH OF STRING IN BYTES
*OUTPUT PARAMETER
*       LOC     SET TO LOCATION OF FIRST OCCURRENCE OF  CHAR  IN
*               STRING , OR TO  N+1  IF  CHAR  DOES NOT OCCUR IN STRING.
        MACRO
&L      T       &A,&B,&C,&D,&E,&F
&L      DC AL2(&A-B)
        DC AL2(&B-B)
        DC AL2(&C-B)
        DC AL2(&D-B)
        DC AL2(&E-B)
        DC AL2(&F-B)
        MEND
LOOK    START
*REGISTERS
AR      EQU     1               BASE OF ARRAY (MODIFIED)
CR      EQU     2               POINTS AT CURRENT CHARACTER
IR      EQU     3               FIRST NON-BLANK CHAR
LR      EQU     7               LIMIT ON SCAN
W       EQU     8               WORKING (ALWAYS CLEAR EXCEPT LAST BYTE)
ER      EQU     9               HOLDS EXPONENT
TR      EQU     10              TABLE BASE REGISTER
XR      EQU     11              TRANSFER REGISTER
*FLAGS
PB      EQU     X'01'           DEC PT FOUND
SB      EQU     X'02'           NEG SIGN
EDB     EQU     X'04'           DIGIT FOUND IN EXPONENT
ESB     EQU     X'08'           SIGN          "     "     "
ENB     EQU     X'10'           NEG EXP
CHAR    DSECT
C       DS      CL1
ARG4    DSECT
ILOC    DS      F
ARG5    DSECT
NLOC    DS      F
ARG6    DSECT
R       DS      D
        USING   CHAR,2
        USING   ARG4,4
        USING   ARG5,5
        USING   ARG6,6
LOOK    CSECT
        STM     1,12,20(13)     (SAVE AREA NOT NEEDED FOR THIS PROGRAM)
        BALR    12,0            SET BASE REG
        USING   *,12
B       LM      1,6,0(1)        LOAD ARGUMENT ADDRESSES
        BCTR    AR,0            POINT  AR  AT  A(0)
        L       2,0(2)          PICK UP  N
```

```
        LA      LR,0(AR,2)      SET ADDR OF  A(N)  IN  LR
        L       3,0(3)          PICK UP  LCC
        LA      CR,0(AR,3)      SET INITIAL VALUE FOR  CR
        SR      W,W             CLEAR  W
        BCTR    CR,0            SET BACK
ENDCH LA       CR,C+1           ADVANCE POINTER
        CR      CR,LR           OVER THE END?
        BNH     NEND            SKIP IF NOT
        SR      0,0             ELSE RETURN  0
        B       EXIT
NEND    CLI     C,C' '          BLANK?
        BE      ENDCH           IF SO, CONTINUE SCAN
        LA      TR,INIT         ELSE SET FOR INITAL TEST
        MVI     F,0             CLEAR FLAGS
        LR      IR,CR           NOTE CURRENT POSITION
        SR      IR,AR           GET RELATIVE VALUE
        ST      IR,ILOC         SET VALUE IN CALLER
        LR      IR,CR           AND SAVE AGAIN FOR USE HERE
        B       GETC            BEGIN
NEXT    LA      CR,C+1          ADVANCE POINTER
GETC    IC      W,C             PICK UP CHARACTER
        IC      W,TAB(W)        TRANSLATE
        CR      CR,LR           TEST FOR END
        BNH     NTE             SKIP IF NOT
        IC      W,=AL1(10)      ELSE TAKE CODE FOR SPECIAL CHARACTER
NTE     LH      XR,0(TR,W)      PICK UP TRANSFER VALUE
        B       B(XR)           AND BRANCH ON IT
INIT T ID,IW,IW,IP,IS,SPEC      INITIAL CHARACTER
ID      SDR     0,0             INITIAL DIGIT.  CLEAR  FR0
        LD      2,=D'10.'       10  TO  FR2
        LD      4,=D'1.'        1  TO  FR4
        LA      TR,DIG          SET DIGIT MODE
DG      MVN    NUM+1(1),C       INSERT DIGIT IN DUMMY NUMBER
        TM      F,PB            BEFORE DEC?
        BO      FRAC            SKIP IF NOT
        MDR     0,2             MULT BY 10
        AD      0,NUM           ADD NEW DIGIT
        B       NEXT
FRAC    DDR     4,2             DIVIDE POWER BY 10
        LDR     6,4             COPY
        MD      6,NUM           MULT BY DIGIT
        ADR     0,6             ADD TO VALUE
        B       NEXT
IW      LA      TR,WORD         INITIAL LETTER, SO SET WORD MODE
        B       NEXT
IP      TM      C+1,X'F0'       DIGIT FOLLOWS?
*** NOTE ASSUMPTION THAT CERTAIN NON-DIGIT (NON-EBCDIC) BYTES DO
*               NOT OCCUR.
```

```
        BNO     SPEC            IF NOT, TREAT POINT AS SPECIAL CHAR
        CR      CR,LR           AT END OF STRING?
        BE      SPEC            IF SO, TREAT AS SPECIAL CHARACTER
        OI      F,PB            ELSE SET FLAG
        B       NEXT            CONTINUE, STILL IN  INIT  MODE
IS      CLI     C,C'-'          INITIAL SIGN.  MINUS?
        BNE     POS             SKIP IF NOT
        OI      F,SB            ELSE SET BIT FOR NEGATIVE SIGN
POS     CLI     C+1,C'.'        PERIOD FOLLOWS?  (SEE NOTE UNDER  IP)
        BE      NEXT            IF SO STAY IN  INIT  MODE
        TM      C+1,X'F0'       DIGIT NEXT?  (SEE NOTE UNDER IP )
        BO      NEXT            IF SO STAY IN INITIAL MODE
SPEC    LR      CR,IR           NOT WORD OR NUMBER. SET  POINTER  BACK
*                               (MAY HAVE BEEN ALTERED BY  IP  OR  IS)
        LA      0,2(0)          SET RETURN VALUE
        LNR     0,0             OF  MINUS  2
        XC      R,R             CLEAR  R
        MVC     R+3(1),C        PUT CHARACTER IN  4TH  BYTE
        LA      CR,C+1          POINT AT NEXT CHARACTER
TM      SR      CR,AR           GET RELATIVE LOCATION
        ST      CR,NLOC         STORE IN CALLER
EXIT    LM      1,12,20(13)     RESTORE
        SR      15,15           0  RETURN CODE
        BR      14              AND LEAVE
DIG T   DG,DE,DT,DP,DT,DT       WE'RE WORKING ON A NUMBER
DE      TM      F,PB            D  OR  E  FOUND.  PREVIOUS DEC PT?
        BZ      DT              QUIT IF NOT, ELSE
        SR      ER,ER           CLEAR EXPONENT REGISTER
        LR      IR,CR           NOTE LOCATION FOR POSSIBLE RESET
        LA      TR,EXP          SET EXPONENT MODE
        B       NEXT
DP      TM      F,PB            DEC PT.  GOT ONE ALREADY?
        BO      DT              QUIT IF SO
        OI      F,PB            ELSE SET FLAG
        B       NEXT            CONTINUE IN DIGIT MODE
DT      LA      0,1(0)          SET RETURN CODE OF  1
        TM      F,PB            DEC PT?
        BZ      DFT             SKIP IF SO
        AR      0,0             ELSE MAKE CODE  2
DFT     TM      F,SB            NEGATIVE?
        BZ      RPOS            SKIP IF NOT
        LNDR    0,0             ELSE CHANGE SIGN
RPOS    STD     0,R             STORE RESULT
        B       TM              (CR  PRESUMABLY POINTS BEYOND END OF #)
EXP T   ED,ET,ET,ET,ES,ET       WORKING ON AN EXPONENT
ED      IC      W,C             PICK UP DIGIT
        N       W,=F'15'        REMOVE ZONE
        MH      ER,=H'10'       MULST PREV VALUE BY 10
```

```
        AR      ER,W            ADD IN DIGIT
        OI      F,EDB           FLAG FOR EXPONENT DIGIT FOUND
        B       NEXT
ES      TM      F,EDB+ESB       EXP SIGN.  SIGN OR DIGIT ALREADY?
        BC      5,ET            TERMINATE IF SO
        OI      F,ESB           ELSE NOTE SIGN FOUND
        CLI     C,C'-'          IS IT  - ?
        BNE     NEXT            SKIP IF NOT
        OI      F,ENB           ELSE SET FLAG
        B       NEXT
ET      TM      F,EDB           END OF EXP. WERE THERE ANY DIGITS?
        BO      ETT             SKIP IF SO
        LR      CR,IR           ELSE POINT BACK TO THE  E OR D
        B       DT              AND TERMINATE
ETT     LA      0,2(0)          PREPARE TO RETURN  2
        C       ER,=F'72'       TEST EXPONENT SIZE
        BH      ERRA            ERROR IF TOO BIG
        LTR     ER,ER           CHECK SIGN (COULD COME FROM OVER FLOW)
        BNM     EOK             OK IF NOT NEG
ERRA    TM      F,ENB           NEG EXP?
        BO      ENG             IF SO GIVE ZERO WITHOUT ERROR FLAG
ERR     LA      0,3(0)          ELSE SET RETURN CODE OF  3
ENG     SDR     0,0             GIVE ZERO RESLUT
        B       RPOS            AND GO
ETES    TM      =AL1(1),0       EXECUTED TO TEST LAST BIT OF REG
EOK     LD      4,=D'1.'        SET  1  IN  FR4
ELP     EX      ER,ETES         LAST BIT OF  ER  ON?
        BZ      NOBIT           SKIP IF NOT
        MDR     4,2             ELSE MULTIPLY
NOBIT   MDR     2,2             SUQARE POWER OF 10
        SRA     ER,1(0)         SHIFT RIGHT
        BNZ     ELP             AND DO IT AGIAN IF NECESSARY
        TM      F,ENB           NEG EXP?
        BZ      POSEX           SKIP IF NOT
        DDR     0,4             ELSE DIVIDE
        B       DT              AND   GO
POSEX   MDR     0,4             MULTIPLY
        B       DT              AND GO.
WORD T  NEXT,NEXT,NEXT,WT,WT,WT     WORD COLLECTING MODE
WT      MVC     R,BLKS          BLANK OUT  R
        LR      ER,CR           USING  ER, GET
        SR      ER,IR           LENGTH OF STRING
        BCTR    ER,0            DECREMENT TO MAKE LENGTH CODE
        C       ER,=F'7'        COMPARE WITH LENGTH  8
        BNH     WEX             AND SKIP IF NOT OVER  8
        L       ER,=F'7'        ELSE SET TO  8
WEX     EX      ER,WMVC         AND MOVE WITH THE COMPUTED LENGTH
        LA      0,1(0)          SET RETURN VALUE
```

```
        LNR     0,0             TC  MINUS 1
        B       TM
WMVC    MVC     R(0),0(IR)      EXECUTED MOVE STATEMENT
        ENTRY   CHID
        USING   *,15
CHID    STM     1,3,20(13)      SAVE
        LM      1,3,0(1)        LOAD ARGUMENT ADDRESSES
        XC      0(4,3),0(3)     SET RESULT TO  0
        CLC     0(1,1),0(2)     COMPARE  1  CHARACTER
        BNE     IDCX            SKIP IF NOT EQUAL
        MVI     3(3),1          ELSE SET RESULT TO 1
IDCX    LM      1,3,20(13)      RESTORE
        SR      15,15           0  RETURN CODE
        BR      14              RETURN
        ENTRY   CHLOC
        USING   *,15
CHLOC   STM     2,7,24(13)      SAVE
        LM      2,5,0(1)        LOAD ARGUMENT ADDRESSES
        LR      7,3             COMPUTE
        A       7,0(4)          ADDR + N
        BCTR    7,0             -1  IN  R7  AS SEARCH LIMIT
        LA      6,1(0)          SET  1  IN  R6  FOR INCREMENT
        LNR     4,3             SAVE MINUS ORIGINAL ADDRESS
LCHL    CLC     0(1,2),0(3)     COMPARE SINGLE CHARACTERS
        BE      LCHX            BRANCH IF EQUAL
        BXLE    3,6,LCHL        ELSE ADVANCE POINTER AND LOOP
LCHX    LA      4,1(4,3)        RES = (FINAL ADR) - (INIT ADR) + 1
        ST      4,0(5)          STORE RESULT
        LM      2,7,24(13)      RESTORE
        SR      15,15           0  RETURN CODE
        BR      14              RETURN
TAB     DC      256X'0A'        FILL TABLE WITH 10'S
        ORG     TAB+75
        DC      X'06'           PERIOD
*       DC      X'08'  AT  TAB+80  FOR  BCD  '+'
        ORG     TAB+78
        DC      X'08'           PLUS
        ORG     TAB+96
        DC      X'08'           MINUS
        ORG     TAB+129
        DC      3X'04'          ABC
        DC      2X'02'          DE
        DC      4X'04'          F-I
        ORG     TAB+145
        DC      9X'04'          J-R
        ORG     TAB+162
        DC      8X'04'          S-Z
        ORG     TAB+193
```

```
         DC        3X'04'          ABC
         DC        2X'02'          DE
         DC        4X'04'          F-I
         ORG       TAB+209
         DC        9X'04'          J-R
         ORG       TAB+226
         DC        8X'04'          S-Z
         ORG       TAB+240
         DC        10X'00'         0-9
         ORG
NUM      DS        0D              DUMMY NUMBER WITH EXP OF  2
         DC        X'4200000000000000'
BLKS     DC        CL8' '          BLANKS
F        DS        CL1             FLAG BYTE
         END
```

INDEX