INFORMATION TO USERS

This material was produced from a microfilm copy of the original document. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the original submitted.

The following explanation of techniques is provided to help you understand markings or patterns which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting thru an image and duplicating adjacent pages to insure you complete continuity.

2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.

3. When a map, drawing or chart, etc., was part of the material being photographed the photographer followed a definite method in “sectioning” the material. It is customary to begin photoing at the upper left hand corner of a large sheet and to continue photoing from left to right in equal sections with a small overlap. If necessary, sectioning is continued again — beginning below the first row and continuing on until complete.

4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from “photographs” if essential to the understanding of the dissertation. Silver prints of “photographs” may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.

5. PLEASE NOTE: Some pages may have indistinct print. Filmed as received.

University Microfilms International
300 North Zebe Road
Ann Arbor, Michigan 48106 USA
St. John’s Road, Tyler’s Green
High Wycombe, Bucks, England HP10 8HR
SAALFELD, Richard Elmer, 1947-
AN INTERACTIVE SYSTEM FOR COMPUTER-AIDED
COMPOSITION AND SOUND SYNTHESIS.

The Ohio State University, Ph.D., 1977
Music

University Microfilms International, Ann Arbor, Michigan 48106

© 1977

RICHARD ELMER SAALFELD

ALL RIGHTS RESERVED
PLEASE NOTE:

Pages 280-355 have small and indistinct print. Best copies available. Filmed as received.

UNIVERSITY MICROFILMS INTERNATIONAL
AN INTERACTIVE SYSTEM
FOR COMPUTER-AIDED COMPOSITION
AND SOUND SYNTHESIS

DISSERTATION

Presented in Partial Fulfillment
of the Requirements
for the Degree Doctor of Philosophy
in the Graduate School.
of the Ohio State University

By
Richard Elmer Saalfeld, B.A., M.A.

The Ohio State University
1977

Reading Committee:
Professor Burdette Green
Professor Norman Phelps
Professor William Poland

Approved By

William Poland
Adviser
School of Music
To my wife and children
who have given and continue to give me
so much joy.
ACKNOWLEDGMENTS

I wish to thank my adviser, Dr. William Poland, for the many ways in which he has helped me bring this dissertation to completion. During the course of the project he has made himself most available to me. I am proud to have had him both as an adviser and as a friend.

I also wish to extend my thanks to the members of my reading committee, who, along with Dr. Poland, were Dr. Burdette Green and Dr. Norman Phelps. Their concern for correct diction and rigorous logic have done much to improve the quality of this dissertation. During my studies at Ohio State, these three men have given me many insights both with respect to music and with respect to clarity of thought and expression.

I wish to thank the faculty and staff of the Department of Electrical Engineering and of the Biomedical Engineering Center of the Ohio State University for allowing me to use the PDP-11/45 computer, which is part of their Hybrid Computer Facility. Special thanks go to Dr. Robert Lacke, who initially granted me permission
to use the machine, and to Mr. David Shultheis and Mr. Michael Kramer, whose intimate knowledge of the PDP-11 was of immense help to me during program development.

I also wish to thank the Instruction and Research Computer Center (IRCC) of the Ohio State University for making possible the computer-aided editing and printing of this dissertation. Special thanks go to Professor Thomas Whitney of IRCC, who developed and modified the text writing program.

I also wish to thank my wife, Barbara, for being a constant source of encouragement during this undertaking.
VITA

May 26, 1947

Born -- Cincinnati, Ohio

1969

B.A., Philosophy, St. St. Mary's Seminary, Cincinnati, Ohio

1971

B.A., Church Music, St. Joseph's College, Rensselaer, Indiana

1971-1975

University Fellow, The Ohio State University, Columbus, Ohio

1972-1973

Graduate Teaching Associate, School of Music, The Ohio State University, Columbus, Ohio

1973

M.A., Music Theory, The Ohio State University, Columbus, Ohio

FIELDS OF STUDY

Major Field: Music Theory

Studies in Analysis: Professors William Poland and Jay Huff

Studies in Composition: Professors Marshall Barnes, Jay Huff, and Mary Beth Nelson

Studies in Computer Applications to Music: Professors William Poland and Kenneth Breeding

Studies in Music History: Professors Richard Hoppin, Alexander Main, and Keith Mixter

Studies in Choral Music: Professor Maurice Casey

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ................................................................. iii
VITA ....................................................................................................... v
LIST OF EXAMPLES ........................................................................ viii
INTRODUCTION ...................................................................................... 1

Chapter

I. OVERVIEW OF COMPUTER-BASED ELECTRONIC MUSIC PRODUCTION ......................................................... 8
   Sound Synthesis ........................................................................... 8
   Composition .................................................................................. 25
   General Reflections ...................................................................... 42
   Summary ........................................................................................ 46

II. CONCEPTUAL AND PHILOSOPHICAL BASES .......................................................................................... 48

   Introduction to Computer Structure ........................................... 49
   Some Assumptions and Conclusions ............................................ 53
   Summary of System Operation ..................................................... 59
   Concepts for Phase Two .............................................................. 62
   Concepts for Phase Three ........................................................... 69
   Summary ....................................................................................... 74

III. A SERIES OF TUTORIAL EXAMPLES .............................................................................................. 76

   SCR001 ....................................................................................... 78
   SCR002 ....................................................................................... 89
   SCR003 ....................................................................................... 93
   SCR004 ....................................................................................... 96
   Summary ...................................................................................... 151
IV. CAEM: SYSTEMATIC DESCRIPTION OF PRESENT IMPLEMENTATION .......................... 152

General Characteristics of Master Score Format ........................................... 153
General Characteristics of Sound Synthesis Subprogram ............................. 154
Master Score Format .................................................................................. 154
MASCH Command Language ......................................................................... 174
Summary of Computer-Generated Messages .................................................. 191

V. EVALUATION AND SUGGESTIONS FOR FUTURE IMPLEMENTATIONS ............ 197

Design of Master Score Handler ..................................................................... 197
Sound Description and Production ................................................................. 210
Summary .................................................................................................... 214

APPENDICES ............................................................................................... 217

A. GLOSSARY ............................................................................................. 218

B. TECHNICAL DESCRIPTION OF PROGRAM ............................................ 229

C. PROGRAM LISTINGS ............................................................................. 279

LIST OF REFERENCES ................................................................................. 357
<table>
<thead>
<tr>
<th>Example</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Direct sound synthesis system</td>
<td>11</td>
</tr>
<tr>
<td>2. Simple instrument with envelope</td>
<td>13</td>
</tr>
<tr>
<td>3. Indirect sound synthesis system</td>
<td>21</td>
</tr>
<tr>
<td>4. Typical &quot;tendency mask&quot;</td>
<td>33</td>
</tr>
<tr>
<td>5. a) Clough's figure (not rotated)</td>
<td>35</td>
</tr>
<tr>
<td>b) Clough's figure rotated counterclockwise 45 degrees</td>
<td>35</td>
</tr>
<tr>
<td>6. Arrays used by Howe in composition given in Example 7</td>
<td>37</td>
</tr>
<tr>
<td>7. Portion of Howe's Canons derived from the arrays in Example 6</td>
<td>38</td>
</tr>
<tr>
<td>8. Sample events</td>
<td>64</td>
</tr>
<tr>
<td>9. Use of modifiers</td>
<td>65</td>
</tr>
<tr>
<td>10. Group in master score</td>
<td>67</td>
</tr>
<tr>
<td>11. Group in version</td>
<td>67</td>
</tr>
<tr>
<td>12. First violin part of the opening measures of Elgar's Enigma Variations</td>
<td>79</td>
</tr>
<tr>
<td>13. SCR001: Representation in master score format of the opening measures of the first violin part of Elgar's Enigma Variations</td>
<td>80</td>
</tr>
<tr>
<td>14. Entering master score with macro</td>
<td>87</td>
</tr>
<tr>
<td>15. Sample of an error message for unidentified opcode</td>
<td>89</td>
</tr>
<tr>
<td>Example</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>16. SCRO02: Representation in master score format of the opening measures of the first violin part of Elgar's <em>Enigma Variations</em></td>
<td>90</td>
</tr>
<tr>
<td>17. Graph of Envelopes 1 and 2 of SCRO02</td>
<td>92</td>
</tr>
<tr>
<td>18. SCRO03: Representation in master score format of the opening measures of the first violin part of Elgar's <em>Enigma Variations</em>; use of envelope to vary the amplitude continually</td>
<td>94</td>
</tr>
<tr>
<td>19. Graph of Envelope 1 of SCRO03</td>
<td>95</td>
</tr>
<tr>
<td>20. SCRO04 in musical notation</td>
<td>97</td>
</tr>
<tr>
<td>21. SCRO04 in master score format</td>
<td>100</td>
</tr>
<tr>
<td>22. Graph of Envelope 1 of SCRO04</td>
<td>103</td>
</tr>
<tr>
<td>23. Graph of Envelope 2 of SCRO04</td>
<td>104</td>
</tr>
<tr>
<td>24. Graph of Envelope 3 of SCRO04</td>
<td>106</td>
</tr>
<tr>
<td>25. Graph of Envelope 4 of SCRO04</td>
<td>107</td>
</tr>
<tr>
<td>26. Graph of Envelopes 5 and 6 of SCRO04</td>
<td>108</td>
</tr>
<tr>
<td>27. SCRO04 in musical notation with item and group numbers</td>
<td>111</td>
</tr>
<tr>
<td>28. Items of frequency opcodes in second nine-second segment</td>
<td>115</td>
</tr>
<tr>
<td>29. Comparison of ratios obtained by equally spaced frequencies with ratios obtained by equally spaced o/p numbers</td>
<td>117</td>
</tr>
<tr>
<td>30. Items, choices, and values for SCRO04</td>
<td>118</td>
</tr>
<tr>
<td>31. Groups for SCRO04</td>
<td>122</td>
</tr>
<tr>
<td>Example</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>32. Opening composer/computer dialogue and</td>
<td>128</td>
</tr>
<tr>
<td>Version 1 of SCR004</td>
<td></td>
</tr>
<tr>
<td>33. Version 1 of SCR004 in musical notation</td>
<td>132</td>
</tr>
<tr>
<td>34. Use of PRINT PROB and listing of</td>
<td>135</td>
</tr>
<tr>
<td>probability table for Items 1 and 2</td>
<td></td>
</tr>
<tr>
<td>35. Listing of probability table for Item 1</td>
<td>136</td>
</tr>
<tr>
<td>36. Partial listing of Sequence 1 of SCR004</td>
<td>137</td>
</tr>
<tr>
<td>37. Composer/computer dialogue to select best expressions of Groups 2,</td>
<td>140</td>
</tr>
<tr>
<td>3, and 4 for SCR004</td>
<td></td>
</tr>
<tr>
<td>38. Commands to combine groups from different versions into a new</td>
<td>141</td>
</tr>
<tr>
<td>version</td>
<td></td>
</tr>
<tr>
<td>39. Command to release only those items which are operands of frequency</td>
<td>145</td>
</tr>
<tr>
<td>instructions in the three nine-second segments for</td>
<td></td>
</tr>
<tr>
<td>purposes of evaluating frequency content of that part of SCR004</td>
<td></td>
</tr>
<tr>
<td>40. Fixing items in first nine-second segment</td>
<td>146</td>
</tr>
<tr>
<td>(SCR004)</td>
<td></td>
</tr>
<tr>
<td>41. Partial listing of edited SCR004</td>
<td>147</td>
</tr>
<tr>
<td>42. Relation of typical envelope contours to the amplitude instructions</td>
<td>160</td>
</tr>
<tr>
<td>available in this master score format</td>
<td></td>
</tr>
<tr>
<td>43. Relative position of opcodes within events</td>
<td>173</td>
</tr>
<tr>
<td>Example</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>44. Map of memory usage</td>
<td>232</td>
</tr>
<tr>
<td>45. Details of structure of internal tables for (a) items, (b) groups, and (c) sequences</td>
<td>233</td>
</tr>
<tr>
<td>46. Table of frequencies available to the sound synthesis subprogram</td>
<td>235</td>
</tr>
</tbody>
</table>
INTRODUCTION

Since the 1950's computers have been used in the production of music. The task of producing music can be analyzed into three stages: (1) the formation of the compositional concept, (2) the composing process, during which the concept is given concrete form by means of the score, and (3) the realization of the score into sound. The speed and ease with which a computer handles logical relationships makes it a valuable tool for testing new compositional methods (stage 2). The usefulness of a computer for sound synthesis (stage 3) makes it especially valuable to a composer of electronic music. He can use a computer in both stages 2 and 3. If the composition and sound synthesis are done on the same computer, the composer may be able to achieve very rapid feedback concerning the realization of his concept.

This dissertation describes such a system—a Compositional Aid for Electronic Music (CAEM). CAEM does not do the actual composition. All compositional decisions are under the control of the composer. The system is intended to provide the composer with one more tool. It operates by relieving the composer of some of the burden of specific, low-level decision making by
allowing him to translate his concept into a form called a **master score**, which is a vehicle for expressing compositional alternatives. In a master score the composer has the option of specifying quantities either by single values or by ranges of values. The possibility of specifying ranges of values makes one master score capable of generating many versions, each of which is an expression of the compositional concept. The composer evaluates the versions presented to him and controls the manner of presentation. The composer has control over the probability of selection of choices and can suspend the selection process when and where desired. Using the available commands, the composer can shape and combine versions of the master score in order to arrive at the optimum expression or expressions of the compositional concept. The decision-making tasks of the composer are aided by rapid audio and visual presentation of score-related material. During the process the computer automatically keeps records of all products of composer-computer interaction.

Temporarily giving over some low-level decision making to a computer allows the composer to give more mental energy to evaluation. The computer may make the composer aware of previously unexplored possibilities since it may make decisions that the composer would not
have conceived of by himself. Programming a computer to take over some compositional decisions may be useful to some composers because of the many decisions that need to be made in an electronic composition. In addition to the decisions that all composers must make, the composer in the electronic medium must create his own instruments and do his own interpretation.

CAEM has the same advantages of other computer-based sound synthesis systems; that is, it overcomes some of the difficulties which arise from producing electronic music with synthesizers and multiple-track tape recorders. With a computer-based system all changes can be made by altering the symbols in the score. In this way the actual sound generating apparatus can be controlled more accurately and in more dimensions simultaneously than would be possible otherwise. The fact that this system incorporates elements of both composition and sound synthesis does not make it unique. Other systems do this. What is unique about this system is the way in which it enables the composer to work with the score, the final symbolic form of the composition.

Computer-based composition/sound synthesis programs generally accept compositional information of a more abstract nature. The concept of the composition is structurally embodied in the composition section of the
program. The program uses the information supplied by the composer to create the score. The information resembles an analysis of the composition. Feedback from the synthesis section is used by the composer to re-evaluate and alter the information he supplies to the composition section. This altered information is then used to produce an altered score, and so on. The composer does not have direct access to the score.

Programs of this kind are valuable. The very act of writing one helps a composer clarify his thinking. It is to be hoped that such programs will eventually increase the store of knowledge of the compositional process. However, the possible applications of computers in this field are many, and it appears that some of the benefits to be derived today from computer-based systems are being ignored. It should be possible to develop a program to aid the composer in developing an electronic composition without embodying in itself part of the compositional concept. It would do this by operating closer to the score than to the concept. A particular compositional concept should be capable of more than one expression. Some improvements in a composition should be achievable without changing the basic structure. A computer-based system providing a means for rapid feedback and alteration of the score should help in the "polishing" process. This
dissertation is intended to present one possible way of implementing such a system—a computer-based system for electronic music production which facilitates the composer's exploration of various compositional alternatives. Such a system would probably not add anything significant to the store of knowledge about the compositional process. That is not its purpose. It is intended to be of practical benefit to a composer investigating various expressions of a compositional concept. The system has the following features:

1. A high degree of control of the details of a composition;

2. The facilities for encoding a score in such a way that it is a set of scores;

3. The highly selective control of chance;

4. Rapid storage and retrieval of all necessary information, including all versions of a score generated during a session;

5. A structure of a general compositional aid, useful to more than the one composer who wrote it;

6. Rapid audio and visual feedback of composer/computer choices;

7. The means of recording and reacting to composer preferences.

The system described here, as well as any computer-based system, is becoming easier to implement as the cost of some kinds of electronic equipment continues to decrease. Computer systems are becoming affordable by individuals as well as by institutions. This system could
be used by itself or as part of a larger system.

Chapter I will provide an overview of the present state of computer sound synthesis and composition. This background will enable the system described in succeeding chapters to be seen in proper perspective. Chapter II will present the conceptual basis of CAEM by giving a series of assumptions and conclusions and by giving definitions of system concepts. Chapter III will elaborate on the system by means of a series of examples. The information presented discursively in Chapters II and III will be systematized in Chapter IV. In Chapter V CAEM and its implementation will be evaluated and suggestions will be given for future implementations.

During the course of this dissertation it is necessary to use various computer-related technical terms. This may cause a problem for some readers, most of whom are probably musicians whose acquaintance with computer-related terms cannot be assumed. In an effort to meet the needs of these people, terms which may be unfamiliar are defined when they first appear, unless their meaning is obvious in context. These terms are also included in a Glossary in the Appendix. The Glossary will include, in addition, other terms that have a special meaning in this dissertation. All terms included in the Glossary are underlined when they first appear in the
References will be given using a format based on that of the American Psychological Association (1974). In this format the publication cited is identified by a parenthetical insertion into the text, as in the previous sentence, consisting of one or more of the following: the last name of the author, the year of publication, and the page number or numbers (see below, p. 10 for an example). The last name of the author is omitted if it appears immediately before the reference, as it does above and also on p. 14. The page number or numbers are omitted if the reference is to the entire book or article. The year of publication is qualified by a letter of the alphabet if two or more works by one author are given in the List of References and were published in the same year (below, p. 37). In the List of References the works cited are listed alphabetically by author and then chronologically.
Chapter I

OVERVIEW OF COMPUTER-BASED ELECTRONIC MUSIC PRODUCTION

This chapter outlines the present state of computer-based electronic music production. In the light of this account it will be easier to understand the function of the system described in this dissertation. It will then also be easier to present the system in detail.

The applications of computers to electronic music production fall into two general categories: (1) those which are primarily related to composition and (2) those which are primarily related to sound synthesis. Since the latter is the more straightforward, it makes the better place to begin.

Sound Synthesis

Sound synthesis utilizes the power of computers to manipulate data rapidly and to control processes in real time; that is, a computer can be connected to external devices in such a way that it can control physical events through these devices. Industry has for some time recognized this and has used computers to control tools and even entire assembly lines. Digital computers are
designed to manipulate discrete packages of data. An operation that involves continuous changes over a period of time must be treated by a computer as a large number of very short, single-valued segments. If these segments are short enough and if a computer can supply values for them fast enough, the operation appears to be continuous, occurring as it normally would. One might compare this to a movie, which is stored on film in individual frames which are presented to the viewers in such rapid succession that their eyes cannot detect the individual pictures but see only one continuous moving image.

Computer sound synthesis programs can be divided into (1) those which generate the sound wave itself and (2) those which generate the control signals for external sound generating equipment. The former will be referred to as direct synthesis programs; the latter, as indirect synthesis programs.

Direct Synthesis

In the case of direct synthesis, the sound wave is calculated in segments, known as samples, as small as 1/40,000th of a second. Such small segments are necessary in order to address the full range of human hearing, which is approximately 16 to 20,000 hertz (cycles per second). It is necessary to supply twice as many samples per second as the highest frequency that might be generated. If this
is not done, the output will be distorted (Mathews, 1969, 5).

The immediate result of a direct synthesis procedure is a string of numbers that represent the instantaneous amplitudes of the sound wave for equal intervals of time in the range of $1/40,000$th of a second. This representation of sound is analogous to the grooves in a record or to the magnetic patterns on recording tape. In the course of being transformed into sound these latter representations are transformed into a variable voltage by suitable electronic equipment, and this signal is used to drive loudspeakers. The same is true for the digital representation. The string of numbers is transformed into a variable voltage by a digital-to-analog converter (DAC). The conversion is done at a uniform rate which is the reciprocal of the sampling rate, or in the range of 40,000 samples per second. The output of the DAC is filtered to remove frequencies higher than half the sampling rate. A computer is usually used to control the conversion because of the precise timing that is required (Example 7) (Mathews, 1969, 31-33).

Many synthesis techniques use direct synthesis procedures. In the better known direct synthesis programs, the composer describes the sounds he wants to produce by symbolically patching together sound generating
Example 1. Direct sound synthesis system.

*Conventional recording tape running on a conventional tape recorder.
and sound shaping modules as one would do with a typical electronic music synthesizer (Example 2). In the program these modules are simulated by subprograms called unit generators, one for each kind of module. Only one unit generator is required to simulate each kind of module since all calculations are made sequentially. For example, the same unit generator is used for every calculation involving an oscillator. Consequently, there is no physical limit to the number of modules of any one kind which can be used. A complete patch giving the characteristics for a class of sounds is called an instrument. Information for playing the instruments is given by the notes—data consisting of pitch, amplitude, and time specifications. The instrument definitions and notes provide the sound synthesis program with a complete description of the sounds of the composition. Two programs of this kind are Mathew’s MUSIC IV (Mathews, 1961; Tenney, 1963) and MUSIC V (Mathews, 1969) which were developed during the 1960’s at Bell Telephone Laboratories.

Although such programs are capable of producing practically any sound which a composer can specify in their terms, it is not always easy to specify sounds with the sonic variety and richness associated with sounds produced by physically vibrating bodies. Incorporation of
Example 2. Simple instrument with envelope (Mathews, 1969, 54). The block diagram (a) is the instrument definition that is represented in the computer score (e) by statements 1 to 5. The instrument consists of two oscillators and an output block. The lower of the two oscillators is the audio source. F2 indicates the waveform that the oscillator is generating. This waveform is shown at (c) and is defined in computer score statement 7. The upper oscillator generates one cycle of the envelope function F1, shown at (b), for each note. F1 is defined in score statement 6. P5 and P6 control amplitude and frequency for the envelope (upper) oscillator, whose output becomes the amplitude input for the audio (lower) oscillator. P7 is the frequency input for the audio oscillator. The two notes shown in musical notation at (d) are represented in score statements 8 and 9.
richer sounds into computer synthesized music is aided by using waveforms derived from computer analysis of naturally occurring sounds. Work in this field has uncovered the fact that the human ear responds in the same way to certain simplified approximations of natural sounds as to the natural sounds themselves (Moorer, 1977, 12). Substituting the approximation for a more detailed analysis reduces the amount of computer time required to generate sounds using the derived waveform. This savings makes the use of derived waveforms more possible. Sounds are presented to the computer for analysis by means of an analog-to-digital converter (ADC), which transforms the representation of the sound as a variable voltage into a string of numbers with variable magnitude.

A technique similar to this, which has recently produced some interesting sonic results, is the use of computer analysis/synthesis to produce what Moorer (1977, 8) calls a new form of musique-concrete. Both Charles Dodge (1975), and Tracy Lind Peterson (1975; 1976) have done computer analysis of human speech in order to derive the formants which the dynamically varying shape of the vocal tract imposes on the harmonically rich waveform produced by the vocal chords. They have then used this information to synthesize the speech but using a sound source other than the vocal chords. In one of his works,
"Everything and Nothing" (1976), Peterson's sound source is a symphony orchestra. The illusion created can be described as an orchestra "speaking" with varying degrees of clarity. Using the same technique one instrument can be "played" through the formants of another. This is only one example of the manipulations which digital processing of audio signals makes possible.

While direct synthesis techniques are useful for creating and manipulating sounds, their usefulness comes at a price. Digital processing of complex audio spectra requires large amounts of computing time. With increasing complexity even modern computers cannot always calculate the samples as fast as required to convert them immediately to sound. That is, it may take longer than one second to compute one second of sound. This makes it necessary to store the digital representation of the sound on tape as it is calculated until the calculations for the whole run are completed. Then the tape can be rewound and converted by means of a DAC. Sound synthesis done in this way can be of excellent quality, but the cost in computer time is sometimes prohibitive.

Another disadvantage is the length of time required between the submittal of a job and the return of the converted tape. Direct synthesis programs are usually run at large computer facilities which operate on a batch
system, where users submit jobs on decks of cards, or their equivalent, and the order in which jobs are run depends partially on their projected length—the smaller jobs running first. There is no opportunity for operating the computer oneself; nor is there any feedback until the job is completed. Even small errors cannot be detected until then, which may be an hour, several hours, or even days later.

One must also take into account the special requirements of a direct synthesis program. Large computers allocate computing time to many jobs simultaneously—this is known as time sharing. The precise timing required for the conversion by the DAC does not permit control by a time-sharing computer. It requires, rather, the services of a smaller machine which can be dedicated to this one function during the conversion process. Ideally, the large and the small computers should both be connected to the same digital tape unit. The small one could even be controlled by the larger. Time sharing would permit this. However, what often happens is that the digital tape needs to be transported to another location, sometimes many miles away, in order to be processed. Or, it may happen that the machine equipped with the DAC has a tape unit which is not compatible with the other computer. This situation
requires that the tape be reformatted for the second machine—another intermediate step. This is the current state of affairs at the Ohio State University, where the usefulness of its many computers is hampered by the lack of any universally compatible medium for transferring data at the rates required.

Some of the disadvantages of direct synthesis have been and are being abated. Direct synthesis programs which translate directly into a computer's machine language, such as MUSIC 360 (Vercoe, 1971), have proven more efficient than programs written in a high-level, machine-independent language such as FORTRAN. Technological advances constantly increase computing speed. Facilities for high-quality digital-to-analog conversion of audio signals are becoming cheaper and more available. Some composers have been fortunate enough to work at computer installations which are dedicated to musical purposes, such as the Experimental Music Studio at M.I.T. (Haflich, 1976) and the Center for Computer Research in Music and Acoustics at Stanford (Smith, 1973; Moorer, 1977). In such situations the feedback, while not always immediate, is as rapid as possible.

The size of the feedback loop can be reduced by using algorithms which require less computation. A giant step in this regard has been taken by John Chowning (1973) in
his development of sound synthesis by frequency modulation. Modulating one audio frequency with another audio frequency produces a rich spectrum with much less computation than that required to produce the same spectrum by summing the component partials. Using this algorithm the system at M.I.T. can generate four voices in real time (Moorer, 1977, 19). Steve Saunders (1975) has written a similar algorithm which computes the waveform at an even faster rate.

Other improvements for direct synthesis are being made in the computers themselves. Computers designed specifically for sound synthesis could do their job more efficiently than the general purpose computers currently being used, which are required to do a wider range of tasks. Hubert Howe (1975b, 255-256) has spoken of the possibility of special-purpose digital modules which could be incorporated into such machines and which could return values at significantly faster speeds than equivalent unit generators. Barry Vercoe has designed such a machine—one that has been described as a "hardware version of MUSIC 360" (Rogers, 1975, 284). There have been and are projects under way to develop digital devices that could be "slaved" to a computer and configured by it to assume any of the sound generating or shaping functions of conventional sound synthesizer modules (Smoliar and
Willson, 1974, 40; Gross, 1975). Each one of these devices could be a satellite computer in itself, each representing one sound processing module or one "voice". In this way the data for sound synthesis could be processed in parallel rather than sequentially.

**Indirect Synthesis**

While a direct synthesis system requires the division of time into very small segments, an indirect synthesis system allows the division of time into considerably larger segments. While full-fidelity direct synthesis requires segments as small as 1/40,000th of a second, indirect synthesis permits segments as large as 1/50th of a second (Gabura and Ciamaga, 1968, 9). This allows the computer much more time to calculate each segment, making it possible for the machine to supply them as rapidly as they are needed. The segments can be much larger because the computer is free from the burden of calculating the sound wave. For indirect synthesis the computer generates the signals that control the sound generating and shaping equipment external to it. Therefore, the factor determining the size of the segment is no longer the size of the audio spectrum but the resolving power of the human ear. Similar, contiguous events as "long" as 1/50th of a second are much too short for humans to distinguish. Consequently, even at these speeds a computer can simulate
continuity; and even with several control signals per channel, a large number of devices can be handled simultaneously.

However, with indirect synthesis there are limitations imposed by the need for physically separate modules. These limitations can be partially overcome by computer control of the patching. This procedure allows the same modules to be used in more than one instrument in those cases where the instruments do not have to sound simultaneously (Example 3).

Computer control of sound synthesis is the latest link in a chain of events which began with the development of the kind of voltage-controlled sound producing and shaping modules that make up present-day sound synthesizers. A sequencer is one such module. Its purpose is to supply voltages at specific times to control other modules in the system. The first sequencers were analog devices; that is, they functioned by internally representing time intervals and control information as measurable electrical quantities, such as voltage, amperage, and capacitance. The cost and size of analog sequencers increase in proportion to their maximum sequence length. With improvements in production techniques, digital sequencers have been developed that are capable of storing control information as collections
Example 3. Indirect sound synthesis system.
of electrical states. The electrical states are used to represent binary numbers. These digital sequencers are so designed that cost and size do not increase in proportion to the maximum sequence length (Hemsath, 1975, 11-12). Sequencers of this kind capable of storing over two hundred entries are common. New departures in input format have been designed into some of these devices. Several of them can store information played to them from a kind of organ keyboard. However, information is usually accessible only in sequential order, and information cannot be added or deleted from the sequence without changing all subsequent locations.

It is not a very big step from a digital sequencer to a small digital computer. The digital sequencer already has memory, a timing device, and the interface to external devices. All that is needed is the capacity to read and execute instructions and to do arithmetic. This capacity is supplied by a central processing unit (CPU). It and its supporting devices can be purchased for the price of a medium-quality tape recorder (Truax, 1975, 20). This "sequencer turned computer," instead of reading through a set of timing information and control settings stored in memory and giving only one kind of response, can scan instructions (that are stored in memory in the same fashion) and react in a unique way to each one. It is
able to access randomly any memory location. This makes
addition and deletion of control information much easier.
The digital computer can perform the same functions as the
sequencer, but can be programmed to do them much more
easily and efficiently. The computer can be instructed to
accept control information in the way most suited to the
composer and to adapt this information to the form
required by the external sound producing devices. Because
it is programmable, many alterations and improvements in
the controlling functions can be made without purchasing
additional equipment. A suitably equipped computer can
assume all control functions, eliminating the need for
separate envelope generators, sample-and-hold units, and
similar devices, and allowing more of the budget to be
spent for sound generating and shaping equipment.

As a computer can control equipment by means of a
DAC, so it can be controlled by means of an ADC. Incoming
voltages from devices manipulated by the composer can be
read by a computer via an ADC in order to provide more
flexible means of control. Bell Telephone Laboratories
has developed a very elaborate system using both ADC's and
DAC's for this purpose. GROOVE, as it is called, allows
the composer to indicate which modules of a synthesizer he
wants to be controlled by each of a series of dials and
levers. On command the computer reads all voltages coming
from these controls via the ADC. This results in a
digital representation of every manipulation the composer
has made, a record which contains nuances of expression
that are very difficult to notate. The composer has
complete control over the tempo. He can hold or reverse
the work at any point or run it at any speed. He can
perform as well as compose. The stored performance can be
edited as he sees fit (Mathews and Moore, 1969; 1970).

At Iowa State University, interdepartmental work
among faculty and students in computer and information
science, electrical engineering, and music is being done
on the Iowa State Computerized Music System (ISMUS)
(Silverston and White, 1975). This is an indirect
synthesis system interfaced with teletype, graphic display
terminal, and keyboard. The Buchla digital keyboard gives
the composer a means of real-time control. The system
uses a PDP-11/20 computer to control a Buchla synthesizer.
Such an interdisciplinary approach is very fruitful. The
possibilities for many musicians using new parts of the
system as they are developed helps to spark enthusiasm and
keeps those designing the electronics and the programs in
touch with the needs of musicians.

The indirect synthesis systems described so far have
used digital computers to control analog devices. When
these analog devices are parts of a conventional
synthesizer, some difficulties can arise. Even if high-precision DAC's are used to control them, conventional synthesizer oscillators often exhibit instability in their response to control signals. Since the human ear is sensitive to very small changes in frequency, such instability can be a serious problem in some applications. In order to avoid this problem, some indirect synthesis systems have been designed to include digital oscillators, which can be controlled with an extremely high degree of accuracy (Alonso et al., 1977, 58). Sidney Alonso and others have designed such a computer-controlled digital synthesizer as part of a computer-based music instructional system now being marketed. Other systems of this kind have been developed by Peter Zinovieff (Rogers, 1975, 284) of Electronic Music Studios, London, and G. Di Giugno (1976) of the Institut de Recherche et Coordination Acoustique/Musique (IRCAM) in Paris.

**Composition**

Since the 1950's a number of experiments have been done involving computers in musical composition. These experiments have been undertaken for reasons both musical and non-musical. Behavioral scientists have used computer composition to develop a model of human creative activity.
Researchers in artificial intelligence have approached computer composition as one of many test situations for studying machine-oriented problem solving. Some musicians have used computer composition to check the validity of analyses. Others have used it as just one more compositional tool. Herbert Brun (1966, 32) has found a computer to be an unbiased and powerful helper in the exploration of new compositional ideas. He finds this kind of work increasingly important today since he believes that the symbol system of Western music has entered what he calls the "administrative stage," in which the sound universe becomes so ordered that it loses its power to communicate new messages.

How can a computer assist in composition? Its ability to calculate rapidly and follow a set of instructions permit a composer to explore strategies that may have proven too time consuming if he were working without assistance. The degree to which a computer is used in the work is entirely up to the composer.

Computer composition experiments can be divided into the following categories:

1. Experiments with stochastic processes;
2. Experiments based on the similarities between music and language;
3. Experiments with sets of sound objects;
4. Pedagogical applications.
This four-way division is not mutually exclusive. The fourth category cannot help but overlap with the others. However, the division is adequate for this treatment of the subject, and it allows a desirable emphasis, since several composition programs are related to computer-based instructional systems.

1. Experiments with Stochastic Processes

In the late 1950's and early 1960's the information theory model—a mathematical model which "deals with the likelihood of accurate transmission of messages subject to transmission failure, distortion, and noise" (Graf, 1972, 285)—gave people in many disciplines a new way of looking at their respective fields. Music theorists were among those who saw promise in it.

Of primary importance in information theory is the concept of a stochastic process which is defined as "a system which produces a sequence of symbols according to certain probabilities" (Shannon and Weaver, 1949, 102). For example, imagine a communication system which has a repertoire of twenty-seven symbols: the letters of the alphabet and the space. An analysis of messages sent using these symbols reveals that certain symbols are used more often than others, for example, the letter "e". Another way of saying this is to say that the probability of occurrence of some symbols is higher than that of
others. Generating messages using these symbols is a stochastic process.

Another related concept is the \textit{Markoff process} or \textit{Markoff chain} which is defined as "the special case of a stochastic process in which the probabilities depend on the previous events" (Shannon and Weaver, 1949, 102). Using the same example, imagine that in the course of transmitting an English sentence, the letter "g" is transmitted. It is extremely likely that the next symbol to be transmitted will be a "u". The probability of a "u" is increased by the previous occurrence of a "g". The dependence of the probability of "u" on the occurrence of "g" is an attribute of a Markoff process.

At first glance the Markoff process appears to be a usable model for musical composition, since, as anyone knows who has tried to write a fugue, probabilities of subsequent events do depend to a great extent on the choices one has already made. Intrigued by this idea, researchers collected data from bodies of music concerning the probabilities of musical events and the probabilities of sequences of musical events. Data collected from the music was tested by reversing the process, that is, programming a computer to generate musical examples using this information. By making comparisons between the original compositions and the computer-generated ones, the
researchers were able to assess the extent to which a musical style could be abstracted using such methods.

Although the results of these experiments tend to verify the analyses to some degree, they also indicate that stochastic analysis alone does not yield sufficient information to generate music of artistic significance (Cohen, 1962, 154; Slawson, 1968, 109). One of the chief researchers in this field is Lejaren Hiller. Although he did not intend his works, such as Illiac Suite or Computer Cantata to be anything more than "laboratory notebooks," (Hiller and Baker, 1962, 426) some reviewers have criticized his work on artistic grounds. Others have felt that the high level of exposure given his work may have temporarily stifled the imaginations of other composers and may have diverted research away from areas with equal promise, while at the same time making computer music synonymous in the minds of many laymen with "music" that is "random, incoherent, or mechanistic" (Howe, 1975a, 282-283).

The lack of artistic quality in the products of such experiments can be traced to three causes. First, although "probabilities of subsequent events do depend to a great extent on the choices one has already made" (above, p. 28), previously made choices usually include a plan for the entire composition. Events depend,
therefore, both on what has gone before and on what is yet to come. Second, the composer is kept at a distance from the final product. In most cases he merely sets the operation in motion. There is no composer-computer interaction programmed into the system. Third, musical processes and Markoff chains are inherently different. Many believe music to be closer to a linguistic than to a stochastic model. Both language and music are capable of embedded structures—relative clauses contained within relative clauses, chord progressions contained within longer-range chord progressions (Slawson, 1968, 110). Markoff chains, on the other hand, are of fixed length. With such a model, embedded structures cannot be considered. This is not to say that statistical procedures are not valid as analytical tools. Chi-square analyses of both musical and literary works have been quite useful in determining questions of disputed authorship, for example (Paisley, 1964, 231). The only assertion being made here is that the Markoff chain is not a good model for computer-aided composition.

2. Experiments Based on the Similarities between Music and Language

Analogies of music to language have spawned some promising research, although the degree to which the speculations have been realized is unclear. The work of
Stephen Smoliar is based on an analogy between music and computer programming languages. He has published reports on his EUTERPE language (Smoliar and Willson, 1974; Smoliar, 1971a; 1971b, 1973) in which musical processes are represented by their computer equivalents. For example, a motive is represented by a subroutine; a note, by a computer word. Smoliar hopes that the same data structure used for music production may also be used for analysis.

A great deal of research in this area has been done by Otto Laske (1973a; 1973b; 1974a; 1974b; 1974c; 1975) of the Institute of Sonology at Utrecht. He has written a number of papers describing a procedural theory of music. He believes that music, like language, is a "process", and that a procedural model would be more fruitful than an analytical one. However, his style of writing is complex and not easily understood by one not versed in computational linguistics and artificial intelligence. Both he and Smoliar make reference to the work of Terry Winograd of M.I.T., who has done extensive work in developing systems capable of understanding human language and of simulating logical thought processes (Winograd, 1972). Winograd has tried to apply his work with language to the domain of music by developing a program capable of doing harmonic analyses of traditional tonal chord
progressions (Winograd, 1968). It appears that insights from computational linguistics will continue to overflow into music theory and composition for some time to come.

3. Experiments with Sets of Sound Objects

Experiments with sets of sound objects involves a computer in the manipulation of pre-existing structures supplied by the composer. The structures serve as generative models for the composition. The work of Barry Truax (1973a; 1973b) offers a good and well-documented example of this kind of program. As input to one of his programs the composer defines a set of abstract sound objects and uses various subroutines to arrange the events derived from them according to time and fundamental frequency. The programs use a graphic display to allow the composer to indicate which areas of the frequency spectrum are to be available to the previously defined sound objects. The use of the display provides a convenient way of indicating variations over time in the allowable frequency areas (Example 4). Truax (1973a, 1) calls such a graph a "tendency mask." Using this information and other information about the density of events per second, the program computes the sound by random and weighted choices from the sound object repertoire (Truax, 1973a, 18f). One of the strengths of the programs is the way in which the composer and computer
Example 4. Typical "tendency mask" (Truax, 1973a, 1).
interact. The composer can hear the results of his specifications immediately by means of a limited direct synthesis system and adjust his next specifications accordingly. The feedback loop is contained within the system.

John Clough (1969) has reported on his application of the mathematical theory of groups to musical composition. He has experimented with two-dimensional co-ordinate systems similar to the one shown below (Example 5), which he used as his example. The X and Y axes define time and pitch respectively. Position left and right determines spacial placement in a stereo environment. Random values are chosen for the co-ordinates with the stipulation that they fall on one of the lines. The selection process is weighted in such a way that there are more selections in the lower half of the graph than in the upper half. Thus, vertical position is also an indication of relative density, density increasing from top to bottom. The information developed in this "group" is used as input to a MUSIC V sound synthesis program. Variations are introduced by rotating the figure while holding stationary the pitch/time axis. The stereo placement information remains tied to the points as they originally stood. Rotating the figure changes the relation of pitch and time to density and spatial position. The contribution of the
Example 5a. Clough's figure (not rotated) (Clough, 1966, 16).

Example 5b. Clough's figure rotated counter-clockwise 45 degrees.
composer to this program comes at the beginning with the
selection of the figure and the ways in which it is
rotated.

Hubert Howe (1975a) has reported on a program of his
which uses multi-dimensional arrays to define horizontal
and vertical pitch structures and rhythm. Howe defines an
array as "a unit that specifies musical characteristics
... of pitch classes ... in more than one dimension
simultaneously ... [An] array determines relationships
between groups of pitches that have certain properties in
common" (Howe, 1975a, 283). He uses arrays to define
abstract structural models which are to be realized by the
program (Examples 6 and 7). His design is a reaction to
other composition programs that make extensive use of
pseudo-random number generation. In contrast to Clough's
program, Howe's does not supply any unspecified values in
this way. The function of the program is to realize the
implications inherent in the arrays. This results,
according to Howe, in a very coherent musical structure.

Howe finds this kind of program of value because he
believes writing it has helped him clarify his composi-
tional technique and allows him to spend more time
working at higher structural levels. He feels he then can
more easily express "emotion" and "abstract ideas" in his
work than he could if he were spending more time working
<table>
<thead>
<tr>
<th>SCR</th>
<th>A G B Bb</th>
<th>G A F C</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>F# G B</td>
<td>G# E A F</td>
</tr>
<tr>
<td>F#</td>
<td>Eb E D</td>
<td>E C# F# G#</td>
</tr>
<tr>
<td>REG</td>
<td>A Eb G D</td>
<td>G C# A G#</td>
</tr>
<tr>
<td>Bb</td>
<td>F# Eb G</td>
<td>C E C# A</td>
</tr>
<tr>
<td>F#</td>
<td>B E Bb</td>
<td>E F F# C</td>
</tr>
<tr>
<td>ARR</td>
<td>E F# D Eb</td>
<td>F# E G# C#</td>
</tr>
<tr>
<td>B</td>
<td>G F# D</td>
<td>F A E G#</td>
</tr>
<tr>
<td>G</td>
<td>Bb A B</td>
<td>A C G F</td>
</tr>
</tbody>
</table>

Example 6. Arrays used by Howe in composition given in Example 7. (Howe, 1975a, 286). SCR stands for "screen", part of pitch selection operation; REG stands for "register" (each row represents a different octave); ARR stands for "arrangement", part of rhythmic determination.
Example 7. Portion of Howe's Canons derived from the arrays in Example 6 (Howe, 1975a, 287f).
Example 7 (continued).
in "manual activity" at the note level (Howe, 1975a, 289).

4. Pedagogical Applications

Some work has been done with computer-aided composition in conjunction with computer-based instruction. The products of this research fit into the second or third categories discussed above, but their emphasis is different since their primary purpose is to teach music through composition. At this time there are efforts in computer composition for educational purposes at institutions including the following: Oberlin Conservatory (Nelson, 1977), Iowa State University (Silverston, and White, 1975), Dartmouth College (Hofstetter, 1976, 13), and York University (Truax, 1975, 24-25; Beckwith, 1975).

The Dartmouth system uses a unique approach. Developed by Jon Appleton and Sydney Alonso,

this system consists of a 16-channel synthesizer which is driven by a NOVA minicomputer. Up to four students can use the system simultaneously; they learn to perceive musical structures by trying to compose. The compositional exercises begin with a limited sound vocabulary, and as the students learn to manipulate sounds, the vocabulary is increased until the student's are given complete compositional freedom ... (Hofstetter, 1976, 13).

At York University in Toronto, under the auspices of the Interactive Music Project, directed by Dr. Sterling Beckwith, some work has been done in music education with the LOGO system, developed by the M.I.T. Artificial
Intelligence Project. Barry Truax (1975, 24-25) has reported on it in an article on the state of computer music in Canada. He bemoans the fact that systems with more acoustic sophistication have not shown the same degree and kind of compositional sophistication:

A terminal at York ... is in phone contact with a PDP-10 at the [National Research Council], Ottawa, where the program runs on a time-sharing basis. Since the terminal is portable and only phone line connection is needed, the system has also been used in local schools for various experiments. The emphasis has been on developing a powerful, accessible tool for music education, and the project has been supported by a research grant from the Ontario Ministry of Education. Many innovations have been added by Beckwith to the basic system, including "gestural rhythm input, tactile-visual input of number strings, and, most recently, limited degrees of control of an ultra-small synthesizer by digital characters normally used to operate the music box" [a sound output device under computer control].

With the extremely limited sound resources of the "music box" (which can produce four voices over a five-octave range in steps, with optional percussion accompaniment), the emphasis has clearly been on music structural thinking, an emphasis sadly lacking in systems of greater acoustic sophistication. According to Beckwith, children and other users "are using it to gain some measure of control over the relationships among larger musical units, or to transform those units by defined, repeatable operations, much more than to manipulate the physical sound itself" (Truax, 1975, 24-25).

It is also possible to use computer composition in an indirect manner in computer-aided music education. Gary Nelson (1973) has speculated about a counterpoint program that would present musical examples for examination and correction by the user. The program would use these corrections to formulate a model of a style or to instruct
a counterpoint student in the avoidance of unstylistic procedures.

**General Reflections**

After this examination of the present state of computer sound synthesis and composition, some general observations are in order. From the consideration of computer sound synthesis it is evident that there is a trade-off between direct and indirect synthesis. Direct synthesis, when fully implemented, provides flexibility at the expense of computer time and immediacy of feedback. Indirect synthesis provides more immediate feedback while sacrificing flexibility. Composition programs which have a high priority on interaction need the more immediate control potential of indirect synthesis or a limited form of direct synthesis. New algorithms and technological improvements may eliminate the need for such distinctions (above, pp. 17f).

The usefulness of some of the most powerful direct synthesis programs is reduced by the processing delays which sometimes occur. In these cases we can speak of a long feedback loop between the running of the original job and the return of the processed tape. Reducing the power of the program so that much of the calculation can be done in real time, making it possible for the composer to hear
his composition without leaving the terminal, seems to gain more by reducing the length of the feedback loop than it looses in sound synthesizing power. Composers can produce art with less than complete control over all musical dimensions. Reducing the delay between encoding and audio feedback helps the composer learn more quickly how to specify the sound he has in mind. A system which incorporates the feedback loop within it can provide many more feedback and correction cycles per unit time.

Another comparison emerges from a consideration of the domains of computer sound synthesis and computer composition. The line between them cannot always be easily drawn. First, although a synthesis program plus its data constitute a score for an electronic composition, the realization into sound is as essential to the compositional process as is the score, since the form of the score is so uncongenial to human information processing. Second, some sound synthesis programs are so open ended that compositional subroutines can be added to them. In the course of realizing sound events, a synthesis program must translate information from a human-oriented format into a machine-oriented format. In the course of translation the computer performs certain operations on the data. These may be very straightforward, such as executing a request for a sine
wave at a frequency of 440 hz for a duration of two seconds. However, they may be as complex as the composer is able to and chooses to make them. They could include such things as (1) generating a scale with twenty-five equal steps to the octave, (2) inverting a melody, (3) transforming one melody into another, (4) generating a melody in a certain style, or (5) generating a fugal exposition in the style of J.S. Bach. Not all of these tasks may be programmable at this time, but if they are, they can be made part of one of the current general-purpose sound synthesis programs.

These considerations have been detailed in order to show how computer-aided electronic composition and computer sound synthesis flow into each other. They are both part of the larger process of electronic music production. One may represent this process by a continuous line with the composer's concept at one end and the realization into sound at the other. The application of computers at some points along this continuum have not been explored.

Examples of various kinds of compositional programs have been presented. The differences among them reflect different compositional approaches. However, it is characteristic of many of them that they accept information at levels higher than the individual note. By
so doing they allow the composer to give more thought to overall structure. That is their purpose. But because of this structural approach, the composer may not easily be able to make direct changes in the final form of the score. This could be a disadvantage to someone who does not want to make structural changes but only wants to polish one specific part of an expression of a compositional concept.

Structural programs are effective to the extent that they mirror and extend the composer's thinking. Programs such as the ones described in this chapter should help musicians, psychologists, and others investigate the compositional process. This process is not well understood. Composers cannot always explain how they developed a composition. Neither composer nor listener can always predict what will please or displease them. The choices which must be made to fill a form in an aesthetically satisfying way are not now programmable.

A program to aid electronic composition need not operate several levels away from the final form of the score. It need not make aesthetic judgements. It can operate within its present competence. A program to aid electronic composition can operate at the note level—at the level of the score—in order to use the power of the computer to help the composer evaluate and change specific
note-level values. It can function by providing a means for selecting and presenting to the composer various note-level possibilities implied in the concept, recording and reacting to composer evaluations, and keeping complete records of the interaction so that possibilities of interest may be retrieved later. Such a program would almost of necessity have the feedback loop within it. This dissertation will describe such a program. Most of the emphasis will be on the note-level presentation and evaluation. Input to the program could come from a program operating at a higher level as well as directly from the composer.

Summary

This chapter has presented an overview of the present state of computer-based electronic music production in both composition and sound synthesis. The relative benefits of direct and indirect synthesis were discussed. A high value has been placed on rapid audio feedback of compositional decisions even if this is at the expense of some control over the sound resources. The intimate connection between computer composition and computer sound synthesis has also been discussed. It has been pointed out that the design of many composition programs keeps the composer from making direct changes in the score. The
unprogrammable nature of aesthetic response has been mentioned in order to emphasize the desirability of a program that would allow greater access to note-level events and that would allow both composer and computer to do what each does best: the composer to evaluate and select; the computer to present, record evaluations and decisions, and control sound synthesis. The next chapter will provide the basic assumptions and concepts upon which such a program can be built.
Chapter II

CONCEPTUAL AND PHILOSOPHICAL BASES

In the last chapter a summary of the current state of computer-based electronic music production was presented in order to put in perspective and to show the need for the CAEM (Compositional Aid for Electronic Music) system. The following points were made: (1) the design of many composition/sound synthesis programs does not permit the composer to have direct access to the score; (2) this condition is not altogether desirable because of our limited knowledge of the compositional process and the personal nature of aesthetic response; (3) computer technology could be useful at the note level just as it has been and will be at more abstract structural levels. The system presented in these pages is an attempt to fill the need for such a note-level system. It frees the composer from other tasks so that he can function more fully as a critic—an arbiter of musical taste—in the shaping of the final product.

This chapter presents fundamental information about CAEM. First, a general description of computer architecture is given for the benefit of those with little acquaintance with computers. Then a number of assumptions
and conclusions are presented in order to establish a philosophical basis for the system. This is followed by a summary of the system's use in the compositional process. The chapter ends with definitions of important concepts in each of the system's two computer-based phases. In this way a general picture of CAEM is presented which does not consider its implementation on any particular computer. Those details will be considered later.

Introduction to Computer Structure

CAEM was developed to run on a small computer that can be completely controlled by one user at any one time. Before receiving more specific information about the system it is necessary for the reader to have a basic understanding of the design of such a computer. Readers already acquainted with computer operations may want to skip this section.

The structure of a computer was mentioned briefly in Chapter I (above, p. 22). The computer was described as a machine capable of reading and executing instructions stored in its memory. The memory of a computer, divided into words and then into bits, is that part of the machine which stores both the series of instructions known as the program and the data that is processed during the execution of the program. The central processing unit
(CPU) is the controlling element in the structure (above, p. 22). It reads and executes the instructions and in so doing carries out arithmetic operations and initiates the input and output of data.

A computer is usually connected to a number of external devices which facilitate the input and output of data. These are called peripherals. They include the familiar typewriter-like terminal which is usually the primary link between user and computer, some kind of high-speed printer, a paper tape reader and punch, some kind of high-speed storage device, and the ADC (analog-to-digital converter) and DAC (digital-to-analog converter) already discussed (above, p. 10 and 14).

The electronic devices including the computer itself and its peripherals are known as hardware. This term differentiates the electronics from the programs which direct it. The programs are collectively known as software. Some distinctions need to be made about the various kinds of software.

In the first chapter brief mention was made of different kinds of programming languages (above, p. 17). Different kinds of programming languages are designed to meet different needs. A particular CPU understands only its own repertoire of machine language instructions. A machine language instruction is usually contained in one
computer word as a series of ones and zeros. Humans find it very inefficient to write programs directly in this form. Series of ones and zeros are not easily remembered. Mistakes in encoding are easily made and finding errors is difficult. For these reasons programs are most often written in a symbolic form using alphabetic and numeric characters to form words (mnemonics) which can be more easily remembered. Computers have special programs for reading and translating these words into the binary machine language.

The complexity of the task of translation varies with the kind of language. Two kinds of languages can be distinguished: (1) those in which each expression corresponds to a machine language instruction and is, consequently, specific to a particular kind of computer and (2) those in which expressions correspond to procedures and only indirectly to machine language instructions and are machine independent. The first kind of language is known as assembly language, and the process of translation is known as assembly. The second kind of language is known as procedure-oriented or high-level language, and the process of translation is known as compilation. One language of the second kind is FORTRAN (FORMula TRANslator). FORTRAN expressions have much in common with algebraic notation. The language is used to
define mathematical procedures. A program written in FORTRAN can be run with only minor modifications on many different computers. Mathematical programs can be written much more easily in FORTRAN than in assembly language. One FORTRAN expression may need to be compiled as many machine-language instructions in order to be executed by a particular computer. Whether one writes in assembly language or in a procedure-oriented language, the program that is written is known as the source program. The machine code which is generated by the assembler or compiler is known as the object or binary program.

Another distinction to be made involves the source of the software. Computer manufacturers supply with their hardware a set of programs known as systems software. These programs provide services of a general nature for all users. The monitor is a part of the systems software which allows the user to communicate with the computer via the terminal. The assembler and compilers also form part of the systems software. There are also utility programs for performing functions such as passing data between peripherals and creating and editing source programs and data sets. All these programs are distinguished from the applications programs which are written by individual users for their particular applications.
The computer system outlined above is similar in design to the one on which CAEN was developed. It is of the kind known as a minicomputer, so called because of its small physical size and limited memory. Large central computers differ from minicomputers in that they have larger memories and additional hardware and software for time sharing and for batch processing (above, p. 15f).

Some Assumptions and Conclusions

In the course of this discussion certain assumptions have been made, some of which have been explicit, others not. This is due to the introductory nature of the material. Before proceeding further, it is necessary to make explicit all of these assumptions and the conclusions that follow from them, since they have helped determine the form of the system.

**Assumption 1a. Aesthetic response is not always predictable or explainable.** Aesthetic response is value judgment based almost totally on one's experience (Apel, 1969, 14; Meyer, 1956, 60-64; Meyer, 1967, 272f). Aesthetic responses necessarily shape compositional strategies. If aesthetic responses are not understood, they cannot be programmed.

**Assumption 1b. The compositional process itself is as individual as are the responses to its products.** It is
not well understood, either, although attempts to program it are increasing the understanding of it.

**Conclusion 1.** If a computer cannot at this time be programmed to compose aesthetically, and if the system under discussion intends to optimize the best qualities of computer and composer, then the composer should be responsible for the compositional ideas and should remain the final arbiter of musical value.

**Assumption 2.** While looking for one thing, a person may find something else very pleasing and useful. When one is researching one topic, he often finds an article on another topic of interest. In the electronic music studio a composer occasionally comes upon something by accident which is a valuable addition to the composition. It is conceivable that such an accidental "find" could substantially change the outcome of the composition, if the composer considered it valuable enough (Weinland, 1969, 724). If chance can be productive in this way, may it not also be productive in another? Could chance be consciously employed by the composer with some of the same favorable results?

**Conclusion 2.** The possibility of an affirmative answer leads to the conclusion that some space should be reserved in the system for the controlled use of chance.
Assumption 3a. Electronic compositions that are realized completely in the studio may require a greater degree of specification by the composer than would be required with a conventional instrumental composition. With an instrumental composition, the end product is a score in musical notation. Indications of timbre are given by the instrumentation. Rhythmic indications are given by the note values, articulations, and tempo markings. Pitch is notated within relatively strict limits, but dynamic level is dependent upon the nature of particular instruments and upon the acoustics of the hall as well as written indications. The composer may choose to give many suggestions for interpretation or to give few. In either case, the composer relies, to a lesser or to a greater extent, on the good judgment of the performer.

In the electronic music studio the composer is also the performer. The details of the sound are worked out by the composer/performer during the final recording. Interpretation of an intermediary is eliminated. If the composer is working in a computer-controlled environment he must in some way spell out the details of his concept or allow the computer to supply them. It is also up to the composer to specify the characteristics of his own instruments.
Assumption 3b. It is as valid for the composer to use a computer as it is for a painter to use a brush. One of the distinguishing characteristics of man is that he is a user of tools. Man has often found himself using tools that were of such complexity that they required an expert—often someone other than himself—to design, construct, and maintain.

Conclusion 3. It is valid for a composer to use a computer to aid himself in making compositional decisions.

Assumption 4. Since it is the object of the composer to create a work in sound, it is desirable that the design be as close to the sound as possible. The overall design of a work should be, in some degree, audible. Designs that are not reflected in the sound cannot be musical and should not be used.

Conclusion 4. A computer program designed to be a compositional aid should allow close contact with the sounds themselves and, to this end, should provide for quick audio feedback of the products of composer/computer interaction.

Assumption 5a. Stochastic procedures in themselves are not sufficient for artistic composition (above, p. 29).

Assumption 5b. Processes suitable to one musical dimension are not necessarily suited to all or any other
musical dimension. For example, a set of values specifying pitches may not be suitable for specifying note durations. Graphs of pitch patterns do not typically resemble graphs of note durations. Each varies in its own way with durations usually being more redundant. Attempts to force values taken from one musical dimension onto another can only give pseudo-random results. Such a procedure is analogous to arranging the numbers from one to ten in alphabetical order; the ordering criterion has nothing directly to do with the objects ordered (Krenek, 1960, 216). It is on these grounds and on those above (Assumption 4) that Clough's program (above, p. 34) and others like it are questionable. Not only are pitches used to control durations, but the values for both come from a pseudo-random process. It is hard to see how the selection of a geometric figure can define a musical structure.

Conclusion 5. The design of a computer program to be used as a compositional aid should avoid models which result in unavoidably unmusical decisions.

The system to be presented here is designed with the preceding assumptions and conclusions in mind. It is an attempt to facilitate the compositional process by allowing both composer and computer to do what each does
best (Assumption and Conclusion 1) (AC1). The program uses chance in the selection process, but chance is always under the composer's control (AC2). The program assumes that the computer is a valid tool for a composer to use, especially in the light of the extra decisions that may be made in creating an electronic composition (AC3). The program is designed to give the composer control of the score and to provide rapid audio feedback in order to keep him in close contact with the sounds he is manipulating (AC4). An attempt has been made to avoid designs that make arbitrary decisions about musical dimensions (AC5). These considerations are being applied to a system that deals with the composition of electronic music, since only in the electronic medium can the computer control the production of the sound. The system is concerned with that subset of electronic music compositions that are realized completely in the studio, although it is not inconceivable that such a system, with some modifications, could be used to generate music in a concert setting. With this background it is now possible to proceed to a more detailed view of the system's operation.
Summary of System Operation

The use of CAEM is divided into three phases:

Phase One: Formation of the Compositional Concept

Phase One is the formation of the compositional concept. This is the activity with which any compositional task begins. It corresponds to the first stage in music production, which was discussed in the Introduction (above, p. 1)—the process by which the composer comes to decide on the general shape of the composition: the divisions and lengths of sections, frequency contours, kinds of envelopes, kinds of timbres, proportions of the sections, use of motives, harmonic considerations, rhythmic patterns, and all those considerations which precede quantitative decisions at the note level.

Phase Two: Encoding the Master Score

It is during Phase Two that the use of a computer begins. This phase consists of translating the composer's concept into a form that both he and the computer can manipulate. This phase consists of two activities: (1) making the translation to the system format and (2) entering the score onto the computer's storage device.
The word "score" has been used to describe the symbolic representation of the sounds of an electronic composition at the point right before sound synthesis. The score represents the final form of the work. Each symbol in it has only one meaning. On the other hand, in Phase Two the composer encodes a master score. Although a master score resembles an ordinary score, quantitative statements need not be made unequivocally. Rather than give one specific value at a certain point, the composer has the option of giving a range of possible values from which the computer selects, given a specific number of possible values equally spaced through the range. Using this option the composer can embody in the master score a number of possible scores, or versions, which he can explore under computer control. In this way he can approach the computer with a general concept of the composition and work with the computer to find the best expression of this concept. The computer can help him explore the concept more efficiently. While encoding the master score the composer must make decisions about what parts of the piece he wants to make variable, the ranges for each of these variables, and the number of choices for each one.

Once these decisions have been made, the composer is ready to go to the terminal and enter the master score.
onto the computer's storage device. This can be done using the editing program supplied with the computer system.

**Phase Three: Interactive Exploration of Master Score**

Once the master score has been stored on the computer's storage device, it can be loaded into computer memory under control of the Phase Three program MASCH (Master Score Handler). This program allows the composer to experiment with his concept through a process in which he evaluates versions of the master score that are presented to him and by which he directs their manner of presentation. By means of this procedure the composer arrives at either of two goals: (1) the best possible version of a master score—in fact, a score in the sense defined on p. 60—or (2) a master score which is capable of generating a number of acceptable versions.

**Re-Encoding: Return to Phase Two**

After working with a master score in Phase Three the composer may decide that more fundamental changes are necessary than are possible under control of MASCH. In this case he can return to Phase Two in order to alter the master score itself. After working with CAEM for a while, the composer learns how to encode so that little re-encoding is necessary. However, encoding errors made
during Phase Two may generate error messages during Phase Three that will require a return to Phase Two for correction.

This has been a summary of the operation of CAEM through its three phases. What follows is a presentation of the concepts used in the two computer-based phases.

**Concepts for Phase Two**

The composer sees the master score as a series of words and numbers that describe sounds. However, the computer deals with the master score as a string, which is defined as an ordered group of symbols taken from the set of characters available on typewriter keyboards, including the letters of the alphabet, the numbers, and the various marks of punctuation, including the dash, period, comma, slash, asterisk, parentheses, angle brackets (<>), and square brackets ([ ]). The letters of the alphabet are referred to as alphabetic characters; the digits from 0 to 9, the minus (−) and plus (+) signs are referred to as numeric characters; the marks of punctuation, the space character, and the non-printing line-feed and carriage-return characters are referred to as delimiters. A score can now be defined as a string organized into substrings, called instructions, which defines an ordered arrangement of sounds in a form readable by both composer
and computer.

The instructions that constitute the score are of various kinds. They can be categorized according to what they specify. The most important kinds of instructions are those which are the components of events. An event is the smallest unit of a score that can describe a sound with respect to time, frequency, and amplitude. It consists of a time instruction, specifying the beginning of the event, and the following frequency and amplitude instructions up to but not including the next time instruction. Instructions are composed of three kinds of units: opcodes, operands, and delimiters. An opcode is that part of an instruction which indicates the kind of operation to be specified. It is composed of at least three alphabetic characters. An instruction has only one opcode. An operand is that part of an instruction which yields a quantity needed by the opcode in order to specify the operation. The number of operands in an instruction is determined by the opcode. Operands follow the opcode with which they are associated. A delimiter is a string of one or more characters used to separate operands and parts of operands from one another as well as operands from opcodes. Delimiters are marks of punctuation, spaces, and non-printing line-feed and carriage-return characters.
Example 8. Sample events.

(1) TIME 0; FREQ 440; LEVEL 500;
(2) TIME 100; FREQ 500; LEVEL 500;

Example 8 above shows two sample events. The event in line 1 describes an output of a function defined elsewhere at a fundamental frequency of 440 herz (hz) at an amplitude level of 500 amplitude units (au) for a duration of one second. Time is given here in hundredths of a second. A TIME instruction gives the time for the beginning of an event. The duration of an event is determined by taking the difference between times for adjacent events, in this case, 1.00 - 0.00 = 1.00 seconds. TIME, FREQ, and LEVEL are opcodes. The numbers are operands. The delimiters are spaces, semicolons, and the non-printing line-feed and carriage-return characters.

Operands are of two kinds: constants and variables. A constant is an operand whose value is not changeable during Phase Three. A variable is an operand whose value is changeable during Phase Three. It is indicated by reserved delimiters which enclose its descriptors. It is in the variable that much of the power of CAEM is concentrated.

The descriptors of a variable are terms and modifiers. A term is a descriptor which is used to define the set of possible values of a variable. The terms are
used to create tables within computer memory in order to deal with each of the possible values of each variable. A **modifier** is a descriptor which alters the value of a variable after that value has been selected from the set of possible values as defined by its terms. The information supplied by a modifier does not affect the internal tables established by the terms. A modifier can alter the value of a variable directly or indirectly, after it has already been altered by another modifier. A modifier consists of an **arithmetic operator** and an operand which itself can be either a constant or a variable. When used in the proper context, modifiers provide a way of defining relationships for a group of operands.

**Example 9. Use of modifiers.**

1. TIME 0; FREQ 440; LEVEL 500;
2. TIME 100; FREQ (440,880,3); LEVEL 500;
3. TIME 200; FREQ [(1 ADD 110)]; LEVEL 500;
4. TIME 300; FREQ [(1 ADD (100,200,5)))] LEVEL 500;

In the example above, the operands in the first line are all constants. The frequency instruction in the second line has an operand which is a variable. It has three possible values: 440, 660, and 880. The three numbers in parentheses are the terms. In the third line, **ADD 110** is a modifier which adds 110 to the selected value of the variable. **ADD** is the arithmetic operator; 110 is its operand, a constant. In the fourth line, the **ADD** is
followed by an operand which is a variable; it has five possible values: 100, 125, 150, 175, and 200.

Variables are of three kinds: items, groups, and extenders. An item is a variable having a specific number of possible values equally spaced within a specific range. Its reserved delimiters are parentheses which enclose three terms. The first two terms give the upper and lower bounds of the range; the third, the number of equally spaced values available within the range. The final term may be followed by one or more modifiers. "(440,880,3)" (Example 9) is an item. A group is a set of a specific number of variables, known as members, whose values fall within a specific range and which are arranged in the master score in order of increasing value. The first member, the primary member, of a group is delimited by angle brackets which enclose three terms. The first term indicates the number of members in the group. The next two terms indicate the range of the group. All members of a group other than the primary member, that is, the secondary members, are indicated in the master score by a series of asterisks terminated by a semicolon. The program fills the specifications given by the primary member by generating a series of pseudo-random numbers, one for each member, within a specific range, and inserting these numbers into the score at the locations
marked by the asterisks in ascending order. They are usually used in time instructions.

Example 10. Group in master score.

(1) TIME <4:0,300>; FREQ 440; LEVEL 500;
(2) TIME ******; FREQ (440,880,3); LEVEL 500;
(3) TIME ******; FREQ [(1 ADD 110)]; LEVEL 500;
(4) TIME ******; FREQ [(1 ADD (100,300,3))]; LEVEL 500;

Example 10 above shows a group of four members. The first time instruction contains the primary member. It requests four times, the first of which is 0 and the last of which is 300. One possible version of this section of a master score is shown below in Example 11:


(1) TIME <0 >; FREQ 440; LEVEL 500;
(2) TIME * 90; FREQ (660 ); LEVEL 500;
(3) TIME * 210; FREQ [(770 )]; LEVEL 500;
(4) TIME * 300; FREQ [(860 )]; LEVEL 500;

An extender is a variable which can assume the value of an item or member. It is used to extend the influence of an item or member. By using it one can duplicate the value of one variable in other variables. By using an extender with a modifier, one can establish a relationship between two variables. An extender has one term which is the number of the item or member whose value is to be inserted in place of it. Items are numbered consecutively from the beginning of the master score. The first item is
Item 1; the second, Item 2; and so on. Group members are numbered in the same way. An extender is delimited by square brackets inside of which are placed parentheses or angle brackets depending on whether the term is the number of an item or a member. In Example 10 (p. 67) \( [ (1 \text{ ADD } 110) ] \) is an extender used with a modifier. The 1 is the term giving the number of the variable whose current value is to be inserted here. The parentheses indicate that the variable is an item. If the variable in question, in this case, the one directly above it, is valued at 660, the value appearing in the third line at the extender will be \( 660 + 110 = 770 \). In the fourth line the extender requests a value which is the sum of the current value of the first item (line 2) (660) and the item contained within the extender (line 4) (200): \( 660 + 200 = 860 \).

It should be noted that operands are defined without regard to what they signify. It makes no difference to the program what a variable is used for. Operands are simply numbers. It is only after variables are evaluated that they are given significance by their operators; that is, it is only after \( (440,680,3) \) is evaluated at 660 that the program determines that the number refers to 660 hertz.
Concepts for Phase Three

Once a master score has been encoded and entered onto the computer's storage device, MASCH is loaded into computer memory, and from then on the master score can be manipulated by the composer under its control.

MASCH has a monitor-oriented structure. The word is used here in the same sense as it was earlier in the chapter (above, p. 52). The monitor allows communication with the computer through the terminal. Just as the monitor in the systems software decodes and executes commands from the terminal relating to system operations, so the MASCH monitor decodes and executes commands from the terminal relating to its operations. When MASCH is loaded, control is given to its monitor. Control is relinquished by the monitor only during the execution of a command. When the command is completed, control is given back to the monitor. This kind of structure allows the composer a degree of flexibility; he can execute commands in many possible sequences. In a program where user and computer interact, a monitor-oriented structure is more appropriate than a tutorial structure in which the user is interrogated by the computer and can enter information in only one set order (Truax, 1973, 18).
Modes of Variation

After MASCH scans a master score, it requests a mode of variation from the composer. The selection of a mode tells MASCH how to make selections from the choices for each variable. Each time a mode selection is made, a version of the master score is generated. The choices made for each version are stored in computer memory as a sequence. The choices for Version 1 are stored in Sequence 1; the choices for Version 2, in Sequence 2; and so forth. There are four modes of variation. Two of these are concerned with choices of values of items; the other two are concerned with choices of sequences. ITEM FREE directs the computer to make a pseudo-random selection for each item based on the probability of occurrence of each choice. ITEM SERIES directs the computer to make a pseudo-random selection for each item based on the probability of occurrence of each choice, but adds a condition that no choice be made that has previously been made until all choices for that item have been made. If for a certain item a choice is made that has previously been made, MASCH moves from choice to choice until one is found that has not previously been made. Once all choices for an item have been made, the process begins again—any choice able to be selected until actually being selected removes it from the available
choices.

A sequence is stored for each version that is generated. The composer can select from among the versions by selecting these sequences. This is done by means of the modes SEQUENCE FREE and SEQUENCE SERIES. This pair of modes is similar to the pair just described. The first allows a free pseudo-random choice; the second imposes the condition that no sequence is to be chosen which has already been chosen until all remaining unchosen sequences have been chosen.

Probabilities and the Selection Process

When a master score is first scanned, the probability of any one choice being made for a particular item is equal to that of the other choices for that item. MASCH allows these probabilities to be altered by means of a grading system. A favorable grade increases the probability that the choice will be made again. Likewise, an unfavorable grade decreases the probability.

The probabilities that are established are operative only if the mode is ITEM FREE. Only in this mode is the selection process allowed to proceed unencumbered by conditions. Use of ITEM SERIES mode allows the composer to react to many choices for each item before allowing the probabilities he has established to assert themselves in further versions under ITEM FREE mode. Use of either
SEQUENCE FREE or SEQUENCE SERIES mode does not generate new versions. These two modes select only from previously existing versions. Consequently, neither do they reflect the current probabilities for items.

If a composer has inserted many variables into the master score, he may not want to work with all of them at once; or, as he works with the master score, he may make some decisions about certain variables that exclude all choices but one. In these cases he can remove the variables in question from the selection process. He can virtually make them constants. Three commands have been provided for this purpose. FORCE causes a variable to assume one of its values. HOLD causes a variable to maintain its present value. RELEASE causes a variable to enter again the selection process. These three commands are for ITEM modes, since they are concerned with generating new versions. FORCE can also be used with SEQUENCE modes in order to return to a previously generated version.

Audio and Visual Feedback

In order to make the decisions required in using the above commands, the composer must be able to hear the sounds he is working with. This is made possible by the PLAY command, which causes the current version to be calculated and sent to the DAC and thence to an amplifier,
filter, and either headphones or loudspeaker.

It is also necessary that the composer receive various kinds of visual information about the master score and its versions. This is provided by several commands. PRINT SCORE causes the current version of the score to be listed as it is being generated. If the NOPRINT command is given, only error messages are listed. The listing of a version differs from that of the master score in that all variables have their terms and modifiers replaced by the values selected for them. LAST causes the current version to be listed in abbreviated form as a sequence. PRINT SEQUENCE causes the listing of any or all sequences. PRINT PROB causes the listing of the probability table for any or all items.

Data Storage and Retrieval

The composer must be able to make permanent records of various kinds of information that are generated during Phase Three. SAVE SCORE allows the composer to copy onto the storage device the version or versions which he thinks best express his concept. A permanent copy of a version allows an electronic composition to be stored indefinitely without committing it to recording tape. The computer under control of MASCH provides the playback mechanism.

Other commands allow the composer to save information that may have permanent or semi-permanent value. If a
composer wants to resume an interrupted session at a later time, **SAVE PROB** and **SAVE SEQUENCE** allow him to copy onto the storage device the probability tables for each item and all the sequences generated up to that time. The complementary commands, **UNSAVE PROB** and **UNSAVE SEQUENCE** allow him to read these tables back into memory when he is ready to resume the session.

**Summary**

This chapter has presented information designed to acquaint the reader with the concepts of CAEM. The concepts presented have been those necessary to a description of the system without all the considerations arising from its implementation on a particular computer.

The first section acquainted the reader with some of the concepts needed in order to discuss typical minicomputer systems. The second section consisted of a series of assumptions and conclusions which together form the philosophical basis for the design of CAEM. This was followed by an overall view of its operation. The concept of the master score was introduced. It was shown how the master score is the pivotal concept in the system: how Phase One leads up to its encoding and how Phase Three leads away from it to the production of one or more optimum versions. The chapter was concluded with two
sections dealing with the operation of the two computer-based phases of the system. These sections included a discussion of the master score format with emphasis on the variable and a discussion of the Phase Three manipulation of the master score by means of MASCH. In so doing, the most important Phase Two instructions and Phase Three commands were introduced.

The next chapter will present the commands and instructions which were developed for CAEM as implemented on a particular minicomputer system.
Chapter III

A SERIES OF TUTORIAL EXAMPLES

The last chapter presented the foundation upon which CAEM (Compositional Aid for Electronic Music) is built. It covered several different subjects: (1) basic computer design, (2) some basic assumptions and conclusions which defined a point of departure and a direction for development, (3) an overview of the operation of CAEM, and (4) a presentation of its important concepts. The present chapter continues the exposition of the concepts of the system by means of a number of examples. The examples are necessary to an understanding of CAEM, and in order to present the examples it is necessary to present CAEM as implemented on a specific computer. Consequently, this chapter is also an introduction to such an implementation. In keeping with the explanatory and introductory nature of the chapter, various aspects of CAEM will be presented as they arise in the course of the examples. Chapter IV will present much of this same information in a systematic form designed for easy reference. This dual presentation—first in examples and then in a systematic form—follows the lead of Max Mathews who used a similar format in his book The Technology of Computer Music (1969).
The elements of CAEM are not all equally abstract. That is, some elements are so basic that they will be represented in every implementation, while others, some of which have been discussed in the preceding chapter, occur only in some implementations. It is most essential to CAEM that two elements be present: (1) some means of representing compositional alternatives and (2) some means of exploring the possibilities inherent in this representation. These two requirements are met by the master score, encoded during Phase Two, and by MASCH (MASTER SCORE Handler), the program operating in Phase Three which aids the composer in the exploration of the master score. Other elements discussed in the last chapter are not quite so essential to CAEM. Some elements represent the beginning stages of implementation, although different implementations could share them without being alike. This condition applies to the structure of events and to the various kinds of operands. Another step toward an implementation is taken when a specific instruction set is established. The examples in this chapter deal at all three of these levels. This is necessary because CAEM is intimately connected with the form of the score and with the specific operands which are used.
The examples which follow represent the basic principles of the system implemented in a particular way. This particular implementation is given at a fixed moment in its development. In Chapter V the rationale is given for the form of this implementation.

The computer on which this implementation has been made is a PDP-11/45 which is manufactured by the Digital Equipment Corporation. At the time this work was done, it had 24K (24,576) sixteen-bit words of core memory. Its peripherals include a DECwriter terminal, a paper tape reader and punch, a twelve-bit DAC and ADC made by Datel, and a pair of small digital tape drives (DECTapes). The DECTape drives, also supplied by Digital Equipment Corporation, provide a random-access storage device for both user programs and the systems software. The master score uses the character set available on the DECwriter, and MASCHE is written entirely in PDP-11 MACRO assembly language.

The first example to be considered is taken from Elgar's Enigma Variations, Op. 36 (1899). Example 12 (p. 79) shows the first violin part of the theme in conventional music notation. Example 13 (p. 80) presents the same music encoded in master score format. This is
Example 12. First violin part of the opening measures of Elgar's Enigma Variations.
Example 13. SCR001: Representation in master score format of the opening measures of the first violin part of Elgar’s Enigma Variations.

COM SCR001

CHAN 1; FUNC 0;
TIME 0; FREQ 4/20; LEVEL 500;
TIME 50; FREQ 4/14; LEVEL 500
TIME 100; FREQ 5/0; LEVEL 500;
TIME 200; FREQ 4/18; LEVEL 500;
TIME 300; FREQ 4/18; LEVEL 0;
TIME 400; FREQ 5/4; LEVEL 500;
TIME 500; FREQ 4/20; LEVEL 500;
TIME 600; FREQ 4/18; LEVEL 500;
TIME 650; FREQ 5/0; LEVEL 500;
TIME 700; FREQ 5/0; LEVEL 0;
TIME 800; FREQ 4/20; LEVEL 500;
TIME 850; FREQ 5/4; LEVEL 500;
TIME 900; FREQ 5/14; LEVEL 500;
TIME 1000; FREQ 4/18; LEVEL 500;
TIME 1100; FREQ 4/18; LEVEL 0;
TIME 1200; FREQ 5/10; LEVEL 500;
TIME 1300; FREQ 4/14; LEVEL 500;
TIME 1400; FREQ 4/18; LEVEL 500;
TIME 1450; FREQ 4/20; LEVEL 500;
TIME 1500; FREQ 4/20; LEVEL 500;
TIME 1600; FREQ 4/18; LEVEL 500;
TIME 1650; FREQ 4/14; LEVEL 500;
TIME 1700; FREQ 5/4; LEVEL 500
TIME 1800; FREQ 4/20; LEVEL 500;
TIME 1900; FREQ 4/20; LEVEL 0;
TIME 2000; FREQ 4/20; LEVEL 500;
TIME 2100; FREQ 4/14; LEVEL 500;
TIME 2200; FREQ 4/18; LEVEL 500;
TIME 2250; FREQ 4/14; LEVEL 500;
TIME 2300; FREQ 4/22; LEVEL 500;
FINE 2700;
EXIT
not really a master score since it has no variables (above, p. 64); and because it has no variables, it has no versions (above, p. 60). However, this lack of variables does not prevent Example 13 from being interpreted and played by MASCH, which will accept it as if it were a version of a master score.

This and all the following examples present one implementation of the ideas sketched in Chapter II (above, pp. 59f). Features such as the design of the instruction set and the limitations of the sound synthesis subprogram are characteristics of this implementation. These limitations are caused by the kind of hardware and software being used. They are not limitations of the basic concepts of the system, which could be implemented with many different kinds of instruction sets and sound synthesis techniques.

The first few lines of Example 13 begin with COM. This is short for COMMENT. As with all opcodes in this implementation, only the first three of its characters are decoded. More of the opcode can be used if the composer finds it helpful, but only the first three are necessary. COMMENT does not take the usual operands. It is only an indication to MASCH that what follows, up to the first carriage return, is a comment. Comments are disregarded by MASCH since their only purpose is to convey messages to
the composer. Comments usually come at the beginning of a score so that the composer can remind himself of pertinent information. Information in comments usually includes the name of the score by which MASCH recognizes it on the storage device, in this case, the tape mounted on DECTape drive number 1 (DT1). The name of a score is a string of six characters, the first three of which are SCR and the last three of which are decimal digits. The name of the score in this example is SCR001.

CHAN (line 3) is short for CHANNEL. CHAN 1 indicates that the following events are for Channel 1. MASCH will continue to assign events to Channel 1 until a CHAN 2 instruction is encountered. The sound synthesis subprogram in this implementation allows for two channels of audio output. Only one event per channel can be sounding at any one time.

FUNCT (line 3) is short for FUNCTION. FUNCT 0 indicates that Function 0 is the stored function which is to be used in generating the following events. Function 0 is a sine wave.

TIME, as mentioned in the last chapter (above, p. 64), gives the time of the beginning of an event, which is composed of instructions following the TIME instruction up to but not including the next instruction designating a time. Durations of events are calculated by taking the
differences between time instruction operands for adjacent events. Time is given in hundredths of a second. Since the tempo is sixty beats to the quarter note, the duration of an eighth note is 0.50 seconds and the duration of a quarter note is 1.00 seconds. This is reflected in the score. The first two events (lines 4 and 5, Example 13, p. 80), corresponding to the first two notes of Example 12 (p. 79), begin at 0.00 and 0.50 seconds. The next two events (lines 6 and 7, Example 13), corresponding to the next two quarter notes of Example 11, begin at 1.00 and 2.00 seconds. These are followed by a one-second silence, representing a quarter rest, which begins at 3.00 seconds (line 8), and so on.

Although FREQ (short for FREQUENCY) was mentioned in the last chapter (above, p. 64), the pitch designation used here was not. FREQ (beginning in line 4) takes its one operand in either of two forms: (1) in herz, as it was in the last chapter, or (2) in a two-part designation composed of octave number and pitch number separated by a slash (/). This form will hereafter be referred to as the octave/pitch (o/p) number form. The operands of FREQ in Example 13 are all in this form. The number before the slash indicates the octave; the number after the slash indicates the pitch within the designated octave on a twenty-four tone scale. In this form 1/0 corresponds
approximately to the bottom C on the piano (33 hz.); 4/0 corresponds approximately to middle C (266 hz); and 4/18 corresponds approximately to A above middle C (446 hz) (see Appendix B, p. 235, for a table of available frequencies). The sound synthesis subprogram of MASCH in this implementation is capable of frequencies up to 2000 hz, which means an octave number no greater than 6.

It should be understood that the sound synthesis subprogram in this implementation is not capable of generating all pitches within its frequency spectrum. There is roughly a quarter tone between most available pitches. For this reason a pitch designation form is used which allows twenty-four pitches to the octave. However, the ratios between all pairs of adjacent pitches are not the same. Since the program is not capable of generating all pitches within its spectrum, the operand of the FREQ instruction is not an indication of the exact frequency, but is a request for a frequency. MASCH responds to the request by selecting the available frequency which is closest to the requested frequency. For example, 4/0 is a request for a frequency of 264 hz. MASCH responds by supplying the available frequency of 266 hz. The closest available frequency below 264 hz is 260 hz (see below, p. 258, for a more complete discussion of the frequency generating algorithm).
LEVEL is an amplitude instruction which takes one operand, the maximum amplitude of the sound wave in amplitude units (au), a scale from 0 to 2047 (the highest positive number which can be represented by twelve binary digits). LEVEL sets the amplitude for a duration equal to the duration of the event of which it is a part. In Example 13 (p. 80) the level is 500 au for all eighth notes and quarter notes. A quarter rest is represented by a level of 0 au. Even though no sound is heard during the rests, a frequency instruction is necessary in order to satisfy the program. In those events with LEVEL 0 (e.g., line 8), the frequency has been left at what it was in the previous event.

At the end of the score (lines 34 and 35) are two instructions which have not been encountered before. FINE is an instruction designating time. FINE takes one operand which is the time of the end of the last event on a particular channel. EXIT takes no operands but indicates the physical end of the score.

In this example no mention has been made of Phase One since the encoding is of a pre-existing composition. After the score has been encoded, it is entered by means of the terminal onto DT1, where it resides as a file. The entering is done by means of EDIT, a computer system program which allows the user to create files, change
them, and combine them in computer memory, and then store them as files on a storage device, in this case, DTI. EDIT has commands for text manipulation which allow the user to insert new material, change old material, move around blocks of text, and define a series of text manipulation tasks. This last capability is important for entering strings, such as master scores, where the same substrings repeatedly occur. For example, SCR001 could be entered in two ways. The first way would be simply to type in the score from beginning to end. The second way employs the macro-instruction capability of EDIT. A macro-instruction, or simply macro, is an instruction which causes the execution of a series of instructions that have been previously defined and given a macro designation. EDIT allows the user to define a macro, which in this case is a series of EDIT commands, and then specify with another command how many times he wants the macro executed. For example, the user, after entering the first three lines of the score, could define a macro to insert "TIME; FREQ; LEVEL; carriage return". He could then instruct EDIT to execute this macro thirty times. This would result in a string similar to that in Example 14 below:
Example 14. Entering master score with macro.

TIME; FREQ; LEVEL;
TIME; FREQ; LEVEL;
TIME; FREQ; LEVEL;

The user could then define another macro to find each occurrence of TIME and position the cursor (a pointer used by EDIT to indicate its current place within the string) after the final E in preparation for inserting an operand. Using this macro the user could with one command find the location for inserting each operand of a TIME instruction. The same procedure could be followed with the operands for FREQ and LEVEL. This second method can significantly reduce the fatigue which sets in while entering long scores, and by reducing fatigue this method can also reduce encoding errors.

After the score has been entered on DT1 the user is ready to load MASCH into computer memory. MASCH, too, must be residing on one of the DECTapes. After MASCH is loaded and running, it will indicate its readiness to proceed by typing [] on the terminal. These two characters will be printed every time the MASCH monitor is ready to accept a command from the terminal. The first command will be LOAD 001, which causes MASCH to search the directory of DT1 for the location of SCR001, to find it, and to scan it. During the scanning, MASCH determines
that the score has no variables. It therefore locks out those commands which deal with variables. During the scanning MASCH decodes the score and enters into a block of memory the internal representation of it that will be used by the sound synthesis subprogram. This is called by the PLAY command, which is one of the few commands applicable to a score without variables. During the execution of PLAY the user listens to the signal from two channels of the DAC over headphones that are connected to the DAC via a stereo amplifier and a smoothing filter (below, p. 268).

SCR001 as it is shown in Example 13 (p. 80) is error free. However, a number of errors that can be made during Phase Two can be detected by MASCH. One such error is the misspelling of an opcode. Another is the supplying of less than the required number of operands. If an error is detected, MASCH indicates this by printing at the terminal a string of question marks and a carriage return followed by the line of the score which contains the questionable text. The string of question marks will be above the text, and the series will end above that part of the line that MASCH was scanning when it detected the error. A typical error message is shown below in Example 15:
Example 15. Sample of an error message for unidentified opcode.  

TIME 100; PRIQ 4/4; LEVEL 500;  

In this case PRIQ has been misspelled FRIQ. MASCH does not recognize PRIQ and indicates an error. A return to Phase Two (above, p. 61) is necessary in order to make the correction. Not all errors will be detected by the program. Some will only become obvious when the user hears the resulting sound.  

When the user wants to end the session he can do so by typing the EXIT command, which returns control to the system MONITOR. This EXIT command, which is part of MASCH's command language, is not to be confused with the EXIT instruction which comes at the physical end of every score.

SCR002  

SCR002 (Example 16, p. 90) is identical to SCR001 in that it represents the same sequence of pitches. It also has no variables. It has been included because it contains three new instructions: ENVLP, SYNCH, and HOLD. ENVLP is short for ENVELOPE. It permits the user to assign a reference number to a set of amplitude instructions so that the set can be used repeatedly in many events while being indicated only by the reference
Example 16. SCR002: Representation in master score format of the opening measures of the first violin part of Elgar's Enigma Variations.

```
(1) COM SCR002
(2) COM
(3) CHAN 1; FUNCT 0;
(4) ENVLP 1: LEVEL 500; HOLD 6000;
(5) ENVLP 2: LEVEL 0; HOLD 6000;
(6) COM
(7) TIME 0; SYMCH 1; FREQ 4/20;
(8) TIME 50; SYMCH 1; FREQ 4/14;
(9) TIME 100; SYMCH 1; FREQ 5/0;
(10) TIME 200; SYMCH 1; FREQ 4/18;
(11) TIME 300; SYMCH 2; FREQ 4/18;
(12) TIME 400; SYMCH 1; FREQ 5/4;
(13) TIME 500; SYMCH 1; FREQ 4/20;
(14) TIME 600; SYMCH 1; FREQ 4/18;
(15) TIME 650; SYMCH 1; FREQ 5/0;
(16) TIME 700; SYMCH 2; FREQ 5/0;
(17) TIME 800; SYMCH 1; FREQ 4/20;
(18) TIME 850; SYMCH 1; FREQ 5/4;
(19) TIME 900; SYMCH 1; FREQ 5/14;
(20) TIME 1000; SYMCH 1; FREQ 4/18;
(21) TIME 1100; SYMCH 2; FREQ 4/18;
(22) TIME 1200; SYMCH 1; FREQ 5/10;
(23) TIME 1300; SYMCH 1; FREQ 4/14;
(24) TIME 1400; SYMCH 1; FREQ 4/18;
(25) TIME 1450; SYMCH 1; FREQ 4/20;
(26) TIME 1500; SYMCH 2; FREQ 4/20;
(27) TIME 1600; SYMCH 1; FREQ 4/18;
(28) TIME 1650; SYMCH 1; FREQ 4/14;
(29) TIME 1700; SYMCH 1; FREQ 5/4;
(30) TIME 1800; SYMCH 1; FREQ 4/20;
(31) TIME 1900; SYMCH 2; FREQ 4/20;
(32) TIME 2000; SYMCH 1; FREQ 4/20;
(33) TIME 2100; SYMCH 1; FREQ 4/14;
(34) TIME 2200; SYMCH 1; FREQ 4/18;
(35) TIME 2250; SYMCH 1; FREQ 4/14;
(36) TIME 2300; SYMCH 1; FREQ 4/22;
(37) FINE 2700;
(38) EXIT
```
number. The one argument of ENVLP is the reference number, which applies to the amplitude instructions that follow. In line 4 Envelope 1 is defined as "LEVEL 500; HOLD 6000". This envelope is referred to in line 7 by the SYNCH 1 instruction. SYNCH takes one operand which is the reference number of the envelope referred to. Envelope 2 is defined in line 5. It is referred to in line 11. In this example Envelope 1 is used for all sounding events, and Envelope 2 is used for all rests. The two envelopes are represented graphically in Example 17 (p. 92).

In line 5 the definition of Envelope 1 consists of two instructions. LEVEL 500 sets the amplitude level of the first event to 500 au. In SCR001 (Example 13, p. 80) the LEVEL instruction had to be repeated in each event since its duration equals the duration of the event of which it is a part. In SCR002 the HOLD command makes this unnecessary. HOLD takes one operand which is the duration for which the last specified amplitude level is to be held. In lines 4 and 5, 6000 is an arbitrary number which was chosen to indicate a time (60.00 seconds) which is longer than any duration for which the instruction would be used. When SYNCH 1 is encountered, the sound synthesis subprogram starts executing Envelope 1. The program will continue to execute Envelope 1 until the duration of the envelope is expired, at which time the program will go on
Example 17. Graph of Envelopes 1 and 2 of SCR002.
to the next amplitude instruction, or until another SYNCH instruction is encountered. Then the program will stop executing the current envelope and start executing at the beginning of the envelope specified in the SYNCH instruction. Accordingly, Envelope 1 is executed until 3.00 seconds, at which time Envelope 2 is initiated, causing a rest to be perceived. Envelope 1 begins again at 4.00 seconds, and so on.

**SCR003**

SCR003 (Example 18, p. 94) is identical to the first two scores in all respects except the amplitude envelope for each event. Lines 4 and 5 define Envelope 1. These instructions are interpreted as follows: an initial rise from 0 to 1024 au in .10 seconds; a subsequent fall to 512 au in another .10 seconds; a constant amplitude at this level for another .10 seconds; followed by a decay to 0 au in .70 seconds; a level of 0 au is then to be held indefinitely. The rates of change of amplitude are all linear, that is, they can be graphed by straight lines (Example 19, p. 95). With this envelope it is possible to specify the rest which occurs at 3.00 seconds without any additional instructions, since the envelope reaches 0 au at that time and holds it until the SYNCH 1 instruction at 4.00 seconds.
Example 18. SCR003: representation in master score format of the opening measures of the first violin part of Elgar's Enigma Variations; use of envelope to vary the amplitude continually.

(1) COM SCR003
(2) COM
(3) CHAN 1; FUNCT 0;
(4) ENVLP 1: RISE 10, 1024; FALL 10,512; HOLD 10;
(5) DECAY 70; HOLD 6000;
(6) COM
(7) TIME 0; SYNCH 1; FREQ 4/20;
(8) TIME 50; SYNCH 1; FREQ 4/14;
(9) TIME 100; SYNCH 1; FREQ 5/0;
(10) TIME 200; SYNCH 1; FREQ 4/18;
(11) TIME 400; SYNCH 1; FREQ 5/4;
(12) TIME 500; SYNCH 1; FREQ 4/20;
(13) TIME 600; SYNCH 1; FREQ 4/18;
(14) TIME 650; SYNCH 1; FREQ 5/0;
(15) TIME 800; SYNCH 1; FREQ 4/20;
(16) TIME 850; SYNCH 1; FREQ 5/4;
(17) TIME 900; SYNCH 1; FREQ 5/14;
(18) TIME 1000; SYNCH 1; FREQ 4/18;
(19) TIME 1200; SYNCH 1; FREQ 5/10;
(20) TIME 1300; SYNCH 1; FREQ 4/14;
(21) TIME 1400; SYNCH 1; FREQ 4/18;
(22) TIME 1450; SYNCH 1; FREQ 4/20;
(23) TIME 1600; SYNCH 1; FREQ 4/18;
(24) TIME 1650; SYNCH 1; FREQ 4/14;
(25) TIME 1700; SYNCH 1; FREQ 5/4;
(26) TIME 1800; SYNCH 1; FREQ 4/20;
(27) TIME 2000; SYNCH 1; FREQ 4/20;
(28) TIME 2100; SYNCH 1; FREQ 4/14;
(29) TIME 2200; SYNCH 1; FREQ 4/18;
(30) TIME 2250; SYNCH 1; FREQ 4/14;
(31) TIME 2300; SYNCH 1; FREQ 4/22;
(32) FINE 2700;
(33) EXIT.
Example 19. Graph of Envelope 1 of SCR003.
The three previous scores were intended to familiarize the reader with the format of the master score and with the basic instruction set used in this implementation. All the operands in these three scores are constants. The present score, SCR004, includes several different kinds of variables; consequently, it is a true master score.

**Phases One and Two**

SCR004 begins in a way similar to the previous scores. However, only the first two measures of the first violin part are used. This material is stated as a theme and then used in what might be called a development (Example 20, p. 97). After the first two measures, the second measure is repeated; then the last two notes are augmented and repeated two more times. At this point (17.00 seconds), a sound with less well defined pitch enters on Channel 1 while activity similar to what has been going on enters on Channel 2. On this second channel three nine-second segments are described, beginning at 17.00, 26.00, and 35.00 seconds. The material of these segments is composed of pitches in the same range as that of the theme, but the exact pitches have not been specified. This is indicated by the position of the
Example 20. SCR004 in musical notation.

Specific duration, unspecified center frequency.

Unspecified duration, unspecified frequency

Unspecified duration, specific pitch
squares on the staff, which encompass the range of the opening measures of Channel 1. The entrance times have not been specified. This is the reason that conventional notes have not been used. Each event has simply been given a designation number. Both the exact pitches and the entrance times are to be explored during Phase Three. What has been specified in Phase Two is that each subsequent nine-second segment contains approximately twice as many events as the previous one, thus shortening the duration of the later events and causing an increase in activity as the composition moves to its conclusion.

In order that the three segments may have more in common than simply their frequency space, the pitches of the four events of the first segment have been carried over into the second segment and new pitches have been interspersed between them. Material from the first and second segments is carried over into the third segment in the same manner. This is shown in Example 20 (p. 97) by the numbering of the events of Channel 2. Beginning at 17.00 seconds there are four events labeled 1, 2, 3, and 4. The second segment, beginning at 26.00 seconds, also contains events 1, 2, 3, and 4, but new ones have been added around them. The material from the second segment—events 1, 5, 2, 6, 3, 7, 4, 8, and 9—is put into the third segment beginning at 35.00 seconds, in the same manner.
While the third segment is in progress on Channel 2, the theme re-appears in unspecified durations at 39.00 seconds on Channel 1. The final section, beginning at 47.00 seconds, is a restatement of the theme in its original duration pattern.

Example 21 (p. 100) shows the composition as encoded in master score format. All material for Channel 1 is entered first. This is terminated by the PINE instruction in line 62, and is followed by the material for Channel 2. Two envelopes are specified at the beginning of the score (lines 4 to 7) using a number of items in the place of exact, unequivocal specifications. The durations for Envelope 1 are in the range from 0.10 to 2.00 seconds. Those for Envelope 2 are in the range from 0.05 to 0.50 seconds. Graphs of these two envelopes are shown in Examples 22 (p. 103) and 23 (p. 104). The purpose of this encoding scheme is to provide two envelopes with one having a more gradual rise and fall than the other. The exact specifications of the envelopes are to be determined during Phase Three when they can be heard in context. An item is used as an operand in many of the SYNCH instructions so that either Envelope 1 or Envelope 2 can be assigned to each of them.

At 17.00 seconds (line 30) a new envelope is defined for the new sound which enters at this point, the less
Example 21. SCR004 in master score format.

(1) COM SCR004
(2) COM
(3) CHAN 1; FUNCT 0;
(4) ENVL P 1: RISE (10,110,5), 1024; FALL (10,110,5)
(5) HOLD (10,110,5); DECAY (50,200,4); HOLD 6000;
(6) ENVL P 2: RISE (5,50,5), 1024; HOLD (5,50,5);
(7) DECAY (5,50,5); HOLD 6000;
(8) COM
(9) COM ITEM 8 IS NEXT
(10) COM
(11) TIME 0; SYNCH (1,2,2); FREQ 4/20;
(12) TIME 50; SYNCH (1,2,2); FREQ 4/14
(13) TIME 100; SYNCH (1,2,2); FREQ 5/0
(14) TIME 200; SYNCH (1,2,2); FREQ 4/18
(15) TIME 400; SYNCH (1,2,2); FREQ 5/4;
(16) TIME 500; SYNCH (1,2,2); FREQ 4/20;
(17) TIME 600; SYNCH (1,2,2); FREQ 4/18;
(18) TIME 650; SYNCH (1,2,2) FREQ 5/0;
(19) TIME 800; SYNCH (1,2,2); FREQ 5/4;
(20) TIME 900; SYNCH (1,2,2); FREQ 4/20;
(21) TIME 1000; SYNCH (1,2,2) FREQ 4/18;
(22) TIME 1050; SYNCH (1,2,2); FREQ 5/0;
(23) TIME 1200; SYNCH (1,2,2); FREQ 4/18;
(24) TIME 1300; SYNCH (1,2,2); FREQ 5/0;
(25) TIME 1400; SYNCH (1,2,2); FREQ 4/18;
(26) TIME 1600; SYNCH (1,2,2); FREQ 5/0;
(27) COM
(28) COM ITEM 24 IS NEXT
(29) COM
(30) TIME 1700; ENVL P 3: SYNCH 3; FMI (75,150,5), 30,[ (24DIV2) ];
(31) RISE 10, 1024; FALL 10, 256; DECAY 1000;
(32) TIME 2700; ENVL P 4: SYNCH 4; FMI [ (24) ],30,[ (24DIV2) ];
(33) RISE 95,1024; DECAY 5;
(34) TIME 2800; SYNCH 3; FMI (75,150,5);30;[ (25DIV2) ];
(35) TIME 3800; SYNCH 4; FMI [ (25) ],30,[ (25DIV2) ];
(36) TIME 3900;
(37) SEC 3900;
(38) COM
(39) COM GROUP 1 IS NEXT
(40) COM
(41) TIME <9:0,500>; ENVL P 5; SYNCH 5; FMI0; FREQ 4/20;
(42) LEVEL 600; HOLD 1000;
(43) TIME*****; FREQ 4/14;
(44) TIME*****; FREQ 5/0;
95) TIME********; SYNCH \((1,2,2)\); FREQ \([37/1]\);
(96) TIME********; SYNCH \((1,2,2)\); FREQ \((4/14,5/14,13)\);
(97) TIME********; SYNCH \((1,2,2)\); FREQ \([47/1]\);
(98) TIME********; SYNCH \((1,2,2)\); FREQ \((4/14,5/14,13)\);
(99) TIME********; SYNCH \((1,2,2)\); FREQ \([39/1]\);
(100) TIME********; SYNCH \((1,2,2)\); FREQ \((4/14,5/14,13)\);
(101) TIME********; SYNCH \((1,2,2)\); FREQ \([50/1]\);
(102) TIME********; SYNCH \((1,2,2)\); FREQ \((4/14,5/14,13)\);
(103) TIME********; SYNCH \((1,2,2)\); FREQ \([41/1]\);
(104) TIME********; SYNCH \((1,2,2)\); FREQ \((4/14,5/14,13)\);
(105) TIME********; SYNCH \((1,2,2)\); FREQ \([53/1]\);
(106) TIME********; SYNCH \((1,2,2)\); FREQ \((4/14,5/14,13)\);
(107) TIME********; SYNCH \((1,2,2)\); FREQ \([55/1]\);
(108) TIME********; SYNCH \((1,2,2)\); FREQ \((4/14,5/14,13)\);
(109) TIME********; SYNCH \((1,2,2)\); FREQ \((4/14,5/14,13)\);
(110) TIME********;
(111) TIME 4400; ENVL P 6; SYNCH 6; FREQ 100; LEVEL 0;
(112) FINE 5400;
(113) EXIT.
Example 22. Graph of Envelope 1 of SCR004.
Example 23. Graph of Envelope 2 of SCR004.
definitely pitched frequency-modulated (FM) sine wave (Example 24, p. 106). The envelope is specified by the amplitude instructions in line 31; however, the ENVLP instruction which identifies these instructions as Envelope 3 appears in line 30. The event is entered this way because in this implementation the ENVLP instruction has to precede the SYNCH instruction which refers to it. The SYNCH instruction is in its position because it must precede the frequency instruction (FMI) in its event (below, p. 173). Envelope 4 (line 32; also Example 25, p. 107), Envelope 5 (line 41), and Envelope 6 (line 111; also Example 26, p. 108) require the same sort of positioning of the instructions, i.e., "ENVLP x; SYNCH x; frequency instruction, amplitude instruction(s);".

The FM sound is specified by the FMI (PM IN) opcode, which takes three operands: the center frequency, the modulating frequency, and the maximum deviation from the center frequency. The center frequency is to be any of five equally spaced frequencies from 75 to 150 hz; the modulating frequency is set at 30 hz; and the maximum deviation from the center frequency is to be one half of the center frequency. This is done by means of an extender and modifier combination (above, pp. 67 and 65). The first operand after the FMI operator, "(75,150,5)", which specifies the range of the center frequency is Item
Example 24. Graph of Envelope 3 of SCR004.
Example 25. Graph of Envelope 4 of SCR004.
Example 26. Graph of Envelopes 5 and 6 of SCR004.
24. (The composer has used the comment in line 28 to remind himself that the next item is Item 24.) The third operand after the FMI operator, "[(24DIV2)],[" is the extender which carries over the value assigned to Item 24 to this position. "24" specifies the item number. "DIV2" is a modifier and its operand which instruct MASC to take the value of Item 24 and divide it by 2 and use the value obtained as the value for this operand. The same technique is employed at 28.00 seconds (line 34) with Item 25, which is the first operand of the FMI operator in line 34. At both 27.00 (line 32) and 38.00 seconds (line 35) the same sound is carried down from the previous event, but the envelope is changed. In line 32 the first operand of the FMI operator is an extender which specifies the value of Item 24, which is the first operand after the FMI operator in line 30. The third operand after the FMI operator in line 32, an extender, accomplishes the same thing as the corresponding extender in line 30. At 38.00 seconds (line 35) the same procedure is followed with Item 25 (line 34) which has just been described for Item 24.

To clarify the relationship between the representation of SCR004 in master score format (Example 21, p. 100) and the representation of the same in musical notation (Example 20, p. 97), another representation of the score in musical notation has been provided in Example
27 (p. 111), but in this one the designation numbers for the incompletely specified events have been replaced by the item and group numbers which these events have in the master score (Example 21, p. 100f). For example, at 17.00 seconds on Channel 1, the box representing the FM sound has the number 24 in it. This indicates that the center frequency of this event is given by Item 24.

A new command appears at 39.00 seconds (line 37). The instruction SEC (SECTION) can be used whenever it is necessary or desirable to start counting time again from 0.00 seconds. Restarting time at 0.00 seconds is necessary whenever the time, as stored in hundredths of seconds, exceeds the capacity of the sixteen-bit computer word, which occurs at $(2 \times 15)-1$ hundredths of a second or 327.67 seconds or 5.46 minutes. It is convenient to do so at the beginning of a new compositional section, as here. Time at 39.00 seconds is restarted at 0.00 seconds. SEC takes one operand, indicating the time at which one wants to reset to 0.00 seconds.

Beginning in line 41 is that part of the composition in which the pitches of the theme are repeated but with unspecified durations. The TIME operators in this section (line 41 and following) take operands that are members of a group (above, p. 66). The primary member is identified by the angle-brackets ($<>$) in line 41. It requests nine
Example 27. SCR004 in musical notation with item and group numbers.

Specific duration, unspecified center frequency.

Unspecified duration, unspecified frequency.

Unspecified duration, specific pitch.
values in the range from 0 to 500. The first value will be 0 and will be the operand of the TIME opcode in line 41, replacing the terms of the primary member. The ninth value will be 500 and will be the operand of the TIME opcode in line 50, replacing the string of asterisks (*). The other seven values will be inserted in place of the asterisks in lines 43 to 49 in ascending order (above, p. 66). Used in the context of a TIME instruction, these values will be interpreted in the range from 0.00 to 5.00 seconds. The nine values give the durations for eight events. The last member of the group (line 50) is used to specify the end of the eighth event. It is used in the same way as a FINE instruction (above, p. 85).

Line 41 contains an instruction that has not yet been encountered. This is the FMO (FM OUT) instruction, an opcode which takes no operands. It signals MASCH to discontinue the frequency modulation that was initiated by the PMI instructions beginning in line 30. If the FMO instruction was not in line 40, MASCH would continue to use frequency modulation on the sine wave. It would use the operand of the FREQ instruction in each event as the center frequency and would continue with the same modulating frequency and maximum deviation from the center frequency which were given in the last PMI instruction (line 35).
Channel 1 ends with a recapitulation of the theme as it was originally heard (line 54 and following). The information for Channel 2 begins at line 66, after the information for Channel 1 has been completed. Channel 2 is silent for the first 17.00 seconds. This silence is indicated by LEVEL 0 in line 67. FREQ 100, also appearing in line 67, is a dummy instruction used to hold the frequency place in that event.

At 17.00 seconds the three nine-second segments begin (above, p. 96). The primary member for the first segment (line 70) specifies the times for four events beginning at 17.00 seconds and ending at 26.00 seconds. There is one more time value than there are events. This is the same situation as in line 50 (above, p. 112). The primary member for the second segment (line 78) specifies with ten values nine events beginning at 26.00 seconds and ending at 35.00 seconds. The primary member for the third segment (line 91) specifies with twenty values the nineteen events beginning at 35.00 seconds and ending at 44.00 seconds. In all cases the values selected will replace the terms of the primary members and the asterisks representing the secondary members.

Beginning at 17.00 seconds (line 70), items are used as operands of FREQ opcodes in order to describe the developmental section (above, pp. 96f). The terms of
these items specify the range of values from which the pitch selection is to be made. The range is the same as that of the theme. The items involved in the first nine-second segment are Items 35, 37, 39, and 41. The composer has used the comment in line 69 to remind himself that the next item is Item 34. This is the item for the SYNCH opcode in line 70. The next item, the operand for the FREQ opcode in line 70, is therefore Item 35.

The values selected for these four items are carried over into the next nine-second segment (beginning line 78) by means of extenders in lines 78, 80, 82, and 84. These extenders are also operands of FREQ opcodes and contain within their square brackets the number of the items whose values are to be inserted there. In each case the number of the item is followed by a slash (/). This is to indicate that the value is in the octave/pitch number format. If the slash is not included, the value will not be converted to herz.

In the second nine-second segment (lines 78f), the values carried over from the first segment are surrounded by items which generate new values (above, Example 20, p. 97; Example 27, p. 111; p. 98). The second segment, by means of its items and extenders, presents nine frequencies whose values have the item numbers given in
Example 28 (below):

Example 28. Items of frequency opcodes in second nine-second segment.

<table>
<thead>
<tr>
<th>line</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>78</td>
<td>35</td>
</tr>
<tr>
<td>79</td>
<td>44</td>
</tr>
<tr>
<td>80</td>
<td>37</td>
</tr>
<tr>
<td>81</td>
<td>47</td>
</tr>
<tr>
<td>82</td>
<td>39</td>
</tr>
<tr>
<td>83</td>
<td>50</td>
</tr>
<tr>
<td>84</td>
<td>41</td>
</tr>
<tr>
<td>85</td>
<td>53</td>
</tr>
<tr>
<td>86</td>
<td>55</td>
</tr>
</tbody>
</table>

The values of these items are carried over into the third segment (beginning line 91) by means of extenders, and new items are added around them, as done before in the second segment. Item 35 is carried over into line 91; Item 44 into line 93 and so on.

In this score the great majority of the frequency instruction operands use the octave/pitch number format. This format has been used here for more than mere convenience. The items that are frequency operands on Channel 2 are used to select pitches from the same scale as that of the theme. An item requests a number of equally spaced choices. What is required in this case is a number of equally spaced intervals, each of which approximates the ratio of a semitone. If the frequencies in the items were requested in herz, MASC would respond with thirteen values between 264 and 528 hz, each separated from the next by 22 hz but with no two intervals
having the same ratio. By using the octave/pitch number format this difficulty is overcome. Since this format requests a pitch from a twenty-four tone scale, equally spaced octave/pitch numbers will result in a set of intervals which are closer to being equal (above, p. 83). In this case, requesting thirteen values between 4/14 and 5/14 results in thirteen values each separated by one o/p number. Since the difference between adjacent o/p numbers is approximately one quarter tone, the request results in twelve semitones. Below in Example 29 (p. 117) is a comparison of the values and ratios obtained by using the two available formats for the frequency instruction.

Phase Three

When SCR004 is scanned by MASCH, MASCH detects the presence of the variables and reserves space in memory for information concerning the values, selection, and probability of each choice for each variable. A representation of the stored information is shown in Examples 30 and 31 (pp. 118 and 122). Example 30 shows the items in SCR004. The item numbers read across, and the choice numbers for each item read down. By comparing this example with the listing of the master score in Example 21 (p. 100), the reader can see that the first seven items, each of which has either four or five choices, deal with the variables in Envelopes 1 and 2.
Example 29. Comparison of ratios obtained by equally spaced frequencies with ratios obtained by equally spaced o/p numbers.

<table>
<thead>
<tr>
<th>Req. hz</th>
<th>Act. hz</th>
<th>Ratio</th>
<th>O/P # = Req. hz</th>
<th>Act. hz</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>264</td>
<td>266</td>
<td>4/0</td>
<td>264</td>
<td>266</td>
<td>.957</td>
</tr>
<tr>
<td>284</td>
<td>280</td>
<td>4/2</td>
<td>280</td>
<td>266</td>
<td>.933</td>
</tr>
<tr>
<td>304</td>
<td>296</td>
<td>4/4</td>
<td>296</td>
<td>298</td>
<td>.952</td>
</tr>
<tr>
<td>329</td>
<td>314</td>
<td>4/6</td>
<td>314</td>
<td>313</td>
<td>.951</td>
</tr>
<tr>
<td>347</td>
<td>333</td>
<td>4/8</td>
<td>333</td>
<td>329</td>
<td>.948</td>
</tr>
<tr>
<td>379</td>
<td>352</td>
<td>4/10</td>
<td>352</td>
<td>347</td>
<td>.943</td>
</tr>
<tr>
<td>391</td>
<td>373</td>
<td>4/12</td>
<td>373</td>
<td>368</td>
<td>.941</td>
</tr>
<tr>
<td>417</td>
<td>396</td>
<td>4/14</td>
<td>396</td>
<td>391</td>
<td>.938</td>
</tr>
<tr>
<td>446</td>
<td>419</td>
<td>4/16</td>
<td>419</td>
<td>417</td>
<td>.935</td>
</tr>
<tr>
<td>463</td>
<td>444</td>
<td>4/18</td>
<td>444</td>
<td>446</td>
<td>.963</td>
</tr>
<tr>
<td>481</td>
<td>470</td>
<td>4/20</td>
<td>470</td>
<td>463</td>
<td>.963</td>
</tr>
<tr>
<td>510</td>
<td>498</td>
<td>4/22</td>
<td>498</td>
<td>500</td>
<td>.926</td>
</tr>
<tr>
<td>532</td>
<td>528</td>
<td>4/24</td>
<td>528</td>
<td>532</td>
<td>.940</td>
</tr>
</tbody>
</table>

**LEGEND:**
- Req. hz = Requested hertz
- Act. hz = Actual hertz (value returned to score by NASCH)
Example 30. Items, choices, and values for SCR004.

<table>
<thead>
<tr>
<th>CHOICE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>50</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>100</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>150</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>85</td>
<td>85</td>
<td>85</td>
<td>200</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td></td>
<td>49</td>
<td>49</td>
<td>49</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHOICE</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHOICE</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>75</td>
<td>75</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>93</td>
<td>93</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>111</td>
<td>111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>129</td>
<td>129</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>147</td>
<td>147</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHOICE</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
<th>32</th>
<th>33</th>
<th>34</th>
<th>35</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4/18</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4/20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4/22</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5/0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5/2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5/4</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5/6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5/8</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5/10</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5/12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5/14</td>
<td></td>
</tr>
<tr>
<td>CHOICE</td>
<td>ITEM 37</td>
<td>ITEM 38</td>
<td>ITEM 39</td>
<td>ITEM 40</td>
<td>ITEM 41</td>
<td>ITEM 42</td>
<td>ITEM 43</td>
<td>ITEM 44</td>
<td>ITEM 45</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
</tr>
<tr>
<td>7</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
</tr>
<tr>
<td>9</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
</tr>
<tr>
<td>11</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
</tr>
<tr>
<td>12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHOICE</th>
<th>ITEM 46</th>
<th>ITEM 47</th>
<th>ITEM 48</th>
<th>ITEM 49</th>
<th>ITEM 50</th>
<th>ITEM 51</th>
<th>ITEM 52</th>
<th>ITEM 53</th>
<th>ITEM 54</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
<td>5/0</td>
</tr>
<tr>
<td>7</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
<td>5/2</td>
</tr>
<tr>
<td>9</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
<td>5/6</td>
</tr>
<tr>
<td>11</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
<td>5/10</td>
</tr>
<tr>
<td>12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
<td>5/12</td>
</tr>
<tr>
<td>CHOICE</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>--------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4/18</td>
<td></td>
<td></td>
<td>4/18</td>
<td></td>
<td></td>
<td>4/18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4/20</td>
<td></td>
<td></td>
<td>4/20</td>
<td></td>
<td></td>
<td>4/20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4/22</td>
<td></td>
<td></td>
<td>4/22</td>
<td></td>
<td></td>
<td>4/22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5/0</td>
<td></td>
<td></td>
<td>5/0</td>
<td></td>
<td></td>
<td>5/0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5/2</td>
<td></td>
<td></td>
<td>5/2</td>
<td></td>
<td></td>
<td>5/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>5/4</td>
<td></td>
<td></td>
<td>5/4</td>
<td></td>
<td></td>
<td>5/4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>5/6</td>
<td></td>
<td></td>
<td>5/6</td>
<td></td>
<td></td>
<td>5/6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5/8</td>
<td></td>
<td></td>
<td>5/8</td>
<td></td>
<td></td>
<td>5/8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5/10</td>
<td></td>
<td></td>
<td>5/10</td>
<td></td>
<td></td>
<td>5/10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>5/12</td>
<td></td>
<td></td>
<td>5/12</td>
<td></td>
<td></td>
<td>5/12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>5/14</td>
<td></td>
<td></td>
<td>5/14</td>
<td></td>
<td></td>
<td>5/14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
</tr>
<tr>
<td>CHOICE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4/18</td>
<td></td>
<td></td>
<td>4/18</td>
<td></td>
<td></td>
<td>4/18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4/20</td>
<td></td>
<td></td>
<td>4/20</td>
<td></td>
<td></td>
<td>4/20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4/22</td>
<td></td>
<td></td>
<td>4/22</td>
<td></td>
<td></td>
<td>4/22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5/0</td>
<td></td>
<td></td>
<td>5/0</td>
<td></td>
<td></td>
<td>5/0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5/2</td>
<td></td>
<td></td>
<td>5/2</td>
<td></td>
<td></td>
<td>5/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>5/4</td>
<td></td>
<td></td>
<td>5/4</td>
<td></td>
<td></td>
<td>5/4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>5/6</td>
<td></td>
<td></td>
<td>5/6</td>
<td></td>
<td></td>
<td>5/6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5/8</td>
<td></td>
<td></td>
<td>5/8</td>
<td></td>
<td></td>
<td>5/8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5/10</td>
<td></td>
<td></td>
<td>5/10</td>
<td></td>
<td></td>
<td>5/10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>5/12</td>
<td></td>
<td></td>
<td>5/12</td>
<td></td>
<td></td>
<td>5/12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>5/14</td>
<td></td>
<td></td>
<td>5/14</td>
<td></td>
<td></td>
<td>5/14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHOICE</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
<td>81</td>
</tr>
<tr>
<td>--------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4/18</td>
<td>2</td>
<td>4/18</td>
<td>2</td>
<td>4/18</td>
<td>2</td>
<td>4/18</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>5/0</td>
<td>2</td>
<td>5/0</td>
<td>2</td>
<td>5/0</td>
<td>2</td>
<td>5/0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>5/2</td>
<td>2</td>
<td>5/2</td>
<td>2</td>
<td>5/2</td>
<td>2</td>
<td>5/2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>5/4</td>
<td>2</td>
<td>5/4</td>
<td>2</td>
<td>5/4</td>
<td>2</td>
<td>5/4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5/6</td>
<td>2</td>
<td>5/6</td>
<td>2</td>
<td>5/6</td>
<td>2</td>
<td>5/6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>5/8</td>
<td>2</td>
<td>5/8</td>
<td>2</td>
<td>5/8</td>
<td>2</td>
<td>5/8</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>5/10</td>
<td>2</td>
<td>5/10</td>
<td>2</td>
<td>5/10</td>
<td>2</td>
<td>5/10</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>5/12</td>
<td>2</td>
<td>5/12</td>
<td>2</td>
<td>5/12</td>
<td>2</td>
<td>5/12</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>5/14</td>
<td>2</td>
<td>5/14</td>
<td>2</td>
<td>5/14</td>
<td>2</td>
<td>5/14</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHOICE</th>
<th>82</th>
<th>83</th>
<th>84</th>
<th>85</th>
<th>86</th>
<th>87</th>
<th>88</th>
<th>89</th>
<th>90</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>4/14</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>4/16</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4/18</td>
<td>2</td>
<td>4/18</td>
<td>2</td>
<td>4/18</td>
<td>2</td>
<td>4/18</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>5/0</td>
<td>2</td>
<td>5/0</td>
<td>2</td>
<td>5/0</td>
<td>2</td>
<td>5/0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>5/2</td>
<td>2</td>
<td>5/2</td>
<td>2</td>
<td>5/2</td>
<td>2</td>
<td>5/2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>5/4</td>
<td>2</td>
<td>5/4</td>
<td>2</td>
<td>5/4</td>
<td>2</td>
<td>5/4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5/6</td>
<td>2</td>
<td>5/6</td>
<td>2</td>
<td>5/6</td>
<td>2</td>
<td>5/6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>5/8</td>
<td>2</td>
<td>5/8</td>
<td>2</td>
<td>5/8</td>
<td>2</td>
<td>5/8</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>5/10</td>
<td>2</td>
<td>5/10</td>
<td>2</td>
<td>5/10</td>
<td>2</td>
<td>5/10</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>5/12</td>
<td>2</td>
<td>5/12</td>
<td>2</td>
<td>5/12</td>
<td>2</td>
<td>5/12</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>5/14</td>
<td>2</td>
<td>5/14</td>
<td>2</td>
<td>5/14</td>
<td>2</td>
<td>5/14</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Example 31. Groups for SCR004.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>MEMBER</th>
<th>VALUE</th>
<th>PRIMARY MEMBER</th>
<th>LINE #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>★&lt;9:0,500&gt;</td>
<td>(41)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1700</td>
<td>★&lt;5:1700,2600&gt;</td>
<td>(70)</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>2600</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>2600</td>
<td>★&lt;10:2600,3500&gt;</td>
<td>(78)</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>3500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>3500</td>
<td>(&lt;20:3500,4400&gt;) (91)</td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>-----</td>
<td>------</td>
<td>--------------------------</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>4400</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
All the items of SCR004 (Example 21, p. 100f) shown in Example 30 (p. 118) with only two choices are operands in SYNCH instructions for choosing one of these two envelopes. The items, beginning with Item 35, which have thirteen choices are operands of frequency instructions on Channel 2. Example 31 (p. 122) shows the groups in SCR004. All the groups in this score are associated with instructions specifying time. This is the usual case. The example gives the group and member numbers, the fixed and variable values, the contents of the primary member of each group, and the line number in the score on which the primary member is found. The first and last member of each group is fixed. The intervening secondary members (represented by the asterisks) are generated by the pseudo-random number subroutine and arranged in ascending order, so that for any group in any version the numbers will increase as one reads down the page.

When MASCH has completed the initial scanning of the score it requests a mode of variation (above, p. 70) by printing EVAL MODE (EVALUATION MODE?) on the terminal. (EVALUATION MODE is synonymous with MODE OF VARIATION; it is part of an older terminology which reflected an emphasis on the evaluation by the composer rather than on the selection by the program. The term MODE OF VARIATION better reflects the function of the modes.) At this point
the composer can select from one of the two ITEM modes (above, p. 70), ITEM FREE or ITEM SERIES. A SEQUENCE mode, either SEQ FREE or SEQ SERIES, cannot be used since there are as yet no sequences in memory to select from. A selection of one of the two SEQUENCE modes will generate an error message. When the composer has made his selection and has typed the corresponding command, MASCH will scan the master score again and use the information derived from it to generate the first version of the master score and to write in memory the internal representation of that version which will be used as input to the sound synthesis subprogram during execution of the PLAY command. The version whose internal representation is currently in memory is known as the current version. Whenever a version is generated or recalled, the number of that version, the current version, is printed at the terminal.

Whenever a variable is encountered during the scanning, MASCH will use its pseudo-random number generator to make a selection from the choices available for that variable. In the case of items, the choice will be made according to the probability of each choice. Until they are made otherwise, all choices are equally probable. In the case of groups, the required number of entries will be generated within the specified range and
entered at the appropriate places in the score in ascending order.

**Preliminary explorations.** Due to the monitor-oriented structure of MASCH (above, p. 69), the composer could proceed in several different ways. One way is simply to generate a number of versions in one of the ITEM modes. This would give him a general impression of the available possibilities. Since at this point the probabilities of all choices for any item are equal, the composer can obtain a good sampling of choices regardless of which of the two ITEM modes he selects. However, if the composer chooses ITEM SERIES he will assure that no previously made choice will be selected again until all choices for that particular item have been made (above, p. 71). Since SCR004 has no item with more than thirteen choices, all choices for all items will have been heard at least once after thirteen versions in ITEM SERIES mode. These choices will have been heard in only a few of the many possible combinations, but the information gathered from this procedure is adequate enough to indicate to the composer some directions in which to proceed.

Regardless of which of the two ITEM modes is selected, MASCH will treat the groups (above, p. 66) in the same way. Since no restrictions have been placed on them, MASCH will generate a new set of members for each
group with each new version. If the composer hears thirteen versions, he will have thirteen sets of durations to consider.

During the course of the exploration the composer will probably want to see a listing of at least some of the versions. In order to obtain a listing he must give the PRINT SCORE command (above, p. 73) before the mode selection command for the first version of the score that he wants to see. Once he has given this command, all subsequent versions will be printed at the terminal until the NOPRINT command is given. This will happen regardless of which of the four modes is in effect; that is, it will happen whether a new version is generated, as in an ITEM mode, or an old version is regenerated, as in a SEQUENCE mode. Versions are printed as they are scanned (above, p. 60) and as the internal representation of the version is being stored for the sound synthesis subprogram. As was stated before (above, p. 67), the listing of a version is identical to that of a master score except that all descriptors (above, p. 64) are replaced with the values selected for each variable.

Example 32 (p. 128) is a listing of a typical version along with composer/computer dialogue which precedes and follows the version. Line numbers have been added to the listing of the version as they have been to the master
Example 32. Opening composer/computer dialogue and Version 1 of SCR004.

*RUN DT1: MUSIC
[LOAD 100
EVAL NODE?
[PRINT SCORE
[ITEM SEQUENCE
SEQUENCE 1

(Load MASC)
(Request for mode)
(Request to see listings of versions)
(Mode request command)
(MASC responds that Version
(Sequence) 1 is being created.)

(1) COM SCR004
(2) COM
(3) CHAN 1; FUNCT 0;
(4) ENVP 1: RISE (10 ), 1024; FALL (110 ), 512;
(5) HOLD (85 ); DECAY (150 ); HOLD 6000;
(6) ENVP 2: RISE (15 ), 1024; HOLD (5 );
(7) DECAY (27 ); HOLD 6000;
(8) COM
(9) COM ITEM 8 IS NEXT
(10) COM
(11) TIME 0; SYNCH (1 ); FREQ 4/20;
(12) TIME 50; SYNCH (1 ); FREQ 4/14
(13) TIME 100; SYNCH (1 ); FREQ 5/0
(14) TIME 200; SYNCH (1 ); FREQ 4/18
(15) TIME 400; SYNCH (2 ); FREQ 5/4;
(16) TIME 500; SYNCH (1 ); FREQ 4/20;
(17) TIME 600; SYNCH (2 ); FREQ 4/18;
(18) TIME 650; SYNCH (2 ); FREQ 5/0;
(19) TIME 800; SYNCH (1 ); FREQ 5/4;
(20) TIME 900; SYNCH (1 ); FREQ 4/20;
(21) TIME 1000; SYNCH (2 ); FREQ 4/18;
(22) TIME 1050; SYNCH (1 ); FREQ 5/0;
(23) TIME 1200; SYNCH (1 ); FREQ 4/18;
(24) TIME 1300; SYNCH (1 ); FREQ 5/0;
(25) TIME 1400; SYNCH (1 ); FREQ 4/18;
(26) TIME 1600; SYNCH (2 ); FREQ 5/0;
(27) COM
(28) COM ITEM 24 IS NEXT
(29) COM
(30) TIME 1700; ENVP 3: SYNCH 3; PMI (111 ), 30,[ (55 ) ];
(31) RISE 10, 1024; FALL 10, 256; DECAY 1000;
(32) TIME 2700; ENVP 4: SYNCH 4; PMI [(111 )], 30,[ (55 ) ];
(33) RISE 95, 1024; DECAY 5;
(34) TIME 2800; SYNCH 3; PMI (129 ), 30,[ (64 ) ];
(35) TIME 3800; SYNCH 4; PMI [(129 )], 30,[ (64 ) ];
(36) TIME 3900;
(37) SEC 3900;
(38) COM
(39) GROUP 1 IS NEXT
(40) COM
(41) TIME <0 >; ENVP 5; SYNCH 5; PMO; FREQ 4/20;
       LEVEL 600; HOLD 1000;
(42) TIME*21 ; FREQ 4/14;
(43) TIME*57 ; FREQ 5/0;
(44) TIME*115 ; FREQ 4/18;
(45) TIME*140 ; FREQ 5/4;
(47) TIME*275 ; FREQ 4/20;
(48) TIME*360 ; FREQ 4/18;
(49) TIME*490 ; FREQ 5/0;
(50) TIME*500 ;
(51) COM
(52) RESTATEMENT OF THEME ITEM 26 IS NEXT
(53) COM
(54) TIME 500; SYNCH (2 ); FREQ 4/20;
(55) TIME 550; SYNCH (1 ); FREQ 4/14;
(56) TIME 600; SYNCH (2 ); FREQ 5/0;
(57) TIME 700; SYNCH (1 ) FREQ 4/18;
(58) TIME 900; SYNCH (2 ) FREQ 5/4;
(59) TIME 1000; SYNCH (2 ); FREQ 4/20;
(60) TIME 1100; SYNCH (1 ); FREQ 4/18;
(61) TIME 1150; SYNCH (1 ); FREQ 5/0;
(62) TIME 1500;
(63) COM
(64) BEGINNING OF CHANNEL 2 ITEM 34 IS NEXT
(65) COM
(66) CHAN 2; PUNCT 0;
(67) TIME 0; FREQ 100; LEVEL 0;
(68) COM
(69) GROUP 2, MEMBER 10, ITEM 34
(70) TIME <1700 >; SYNCH (1 ); FREQ (5/4 );
(71) TIME*1956 ; SYNCH (2 ); FREQ (4/16 );
(72) TIME*2206 ; SYNCH (1 ); FREQ (5/4 );
(73) TIME*2541 ; SYNCH (1 ); FREQ (5/8 );
(74) TIME*2600 ;
(75) COM
(76) GROUP 3, MEMBER 15, ITEM 42
(77) COM
(78) TIME <2600 >; SYNCH (2 ); FREQ [(5/4)];
(79) TIME*2721 ; SYNCH (2 ); FREQ (5/4 );
(80) TIME*2893 ; SYNCH (1 ); FREQ [(4/16)];
(81) TIME*2963 ; SYNCH (1 ); FREQ (4/18 );
(82) TIME*3030 ; SYNCH (1 ); FREQ [(5/4)];
(83) TIME*3153 ; SYNCH (1 ); FREQ (5/6 );
(84) TIME*3323 ; SYNCH (2 ); FREQ [(5/8)];
(85) TIME*3347 ; SYNCH (1 ); FREQ (5/10 );
(86) TIME*3374 ; SYNCH (1 ); FREQ (4/18 );
(87) TIME*3500 ;
(88) COM
(89) COM GROUP 4, MEMBER 25, ITEM 56
(90) COM
(91) TIME<3500 >; SYNCH (2. ); FREQ [ (5/4) ];
(92) TIME*3621 ; SYNCH (1 ); FREQ (5/0 ) ;
(93) TIME*3756 ; SYNCH (2 ); FREQ [ (5/4) ];
(94) TIME*3793 ; SYNCH (2 ); FREQ (4/20 ) ;
(95) TIME*3812 ; SYNCH (1 ); FREQ [ (4/16) ];
(96) TIME*3847 ; SYNCH (2 ); FREQ (4/6 ) ;
(97) TIME*3863 ; SYNCH (1 ); FREQ [ (4/18) ];
(98) TIME*3891 ; SYNCH (2 ); FREQ (5/10 ) ;
(99) TIME*3930 ; SYNCH (1 ); FREQ [ (5/4) ];
(100) TIME*3965 ; SYNCH (2 ); FREQ (4/22 ) ;
(101) TIME*4047 ; SYNCH (1 ); FREQ [ (5/6) ];
(102) TIME*4053 ; SYNCH (1 ); FREQ (4/16 ) ;
(103) TIME*4063 ; SYNCH (2 ); FREQ [ (5/8) ];
(104) TIME*4122 ; SYNCH (1 ); FREQ (4/20 ) ;
(105) TIME*4131 ; SYNCH (1 ); FREQ [ (5/10) ];
(106) TIME*4168 ; SYNCH (2 ); FREQ (4/18 ) ;
(107) TIME*4223 ; SYNCH (1 ); FREQ [ (4/18) ];
(108) TIME*4247 ; SYNCH (2 ); FREQ (5/10 ) ;
(109) TIME*4274 ; SYNCH (1 ); FREQ (4/20 ) ;
(110) TIME*4400 ;
(111) TIME 4400; ENVL P 6; SYNCH 6; FREQ 100; LEVEL 0;
(112) FINE 5400;
(113) EXIT.

[ PLAY ]
[ NOPRINT ]
[ NEXT ]

SEQUENCE 2
[ PLAY ]
[ NEXT ]

SEQUENCE 3
[ PLAY ]
score. In this and in all future examples of this kind, that which is typed by the composer will be underlined. Each command string must be terminated with a carriage return, which will be assumed and not otherwise indicated. Example 33 (p. 132) shows the same version as Example 32 represented graphically. The reader is encouraged to compare this representation of a version to the graphic representations of the master score shown in Examples 20 (p. 97) and 27 (p. 111).

Near the end of Example 32 (p. 129f) a command appears that has not previously been discussed. The command NEXT is a substitute for any of the four "mode" commands. It causes the generation or recall of a version according to the mode selected last. In this case it corresponds to ITEM SERIES.

Composer evaluation: restricting the selection process: items. At some time during the audition of the versions, the composer will form some opinions about what choices are working better than others. In response to these opinions, the composer may want to weight the selection process in favor of certain choices, or he may want to eliminate entirely some of the choices. There are two commands for accomplishing these two objectives. EVAL (EVALUATE) (above, p. 71) is used to give a grade or a weight to a choice made for the current version of the
Example 33. Version 1 of SCR004 in musical notation.
master score. It takes two arguments: the item number (above, pp. 66, 67) and the grade, a positive or negative number with an absolute value less than 15. A positive number corresponds to an increase in weight, and a negative number corresponds to a decrease in weight. The effect the weights have on the probabilities of choice varies with the number of choices that the item has. A typical command string using EVAL would be EVAL 1,1; 2,-10. The command string means "increase the probability of the current choice for Item 1 by 1 unit and decrease the probability of the current choice for Item 2 by 10 units." MASCHE knows that the current choice for Item 1 is Choice 4 and that the current choice for Item 2 is Choice 5. Both of these items refer to variables associated with Envelope 1. Item 1 (line 4, Example 32, p. 120f; Example 21, p. 100f) is the operand for the RISE instruction and Item 2 (line 4) is the operand for the FALL instruction. These two have been chosen because decisions regarding the envelopes are probably among the first that the composer will make, since they affect many events in the score.

Each of these items has five choices, and, initially, all choices are equally probable; that is, all choices have a probability of .200. After the EVAL command, the probability of Choice 4 of Item 1 is increased to .333 while the probability of the other four choices is
decreased to .167. For Item 2, the probability of Choice 5 is reduced to .00024 while the probability of the other four is increased to .24993. In effect, Choice 5 is eliminated.

SET is identical to EVAL except that it can be used to set the probability of any choice, whether or not that choice was selected for the current version. Because it deals with all choices it needs an extra argument to specify the choice number. In order to use SET to accomplish the same changes in probabilities that were accomplished above with EVAL, the following command string would be required: SET 1,4,1; 2,5,-10. This means "increase the probability of Item 1, Choice 4 by 1 unit, and decrease the probability of Item 2, Choice 5 by 10 units."

It should be noted that to use EVAL the composer does not need as much information as he does to use SET. With EVAL he needs to know only that the choice he wants to affect has been made in the current version (he has probably just heard it). With SET the composer also has to know the choice number. EVAL makes possible a rapid sequence of selections, auditions, and evaluations with a minimum of input information.

In order to check on the execution of the above commands, the composer may want to see a listing of the
probability tables for those two items. The listing is generated by the PRINT PROB n command, where n is the number of the item in question. The command strings and resulting output are shown in Example 34. Giving the command without n results in the listing of the probability tables for all items in the master score.

Example 34. Use of PRINT PROB and listing of probability tables for Items 1 and 2.

[ ] PRINT PROB 1
ITEM 1 1 .167 2 .167 3 .167 4 .333 5 .167
[ ] PRINT PROB 2
ITEM 2 1 .250 2 .250 3 .250 4 .250 5 .000
[ ]

If during the course of exploring SCR004, the composer alters the probabilities of certain choices while working in ITEM SERIES mode, the changes will not become evident in subsequent versions until the composer selects ITEM FREE mode. This happens because ITEM SERIES mode (above, p. 70) overrides the weights of the choices. This feature of ITEM SERIES allows the composer to hear and evaluate choices which have a very low probability of selection. For example, if in Item 2 the composer had chosen to give a weight of 10 rather than -10 to Choice 5, the probability table would look like this:
Example 35. Listing of probability table for ITEM 2.

| ITEM 2 | 1 | .001 | 2 | .001 | 3 | .001 | 4 | .001 | 5 | .996 |

In subsequent versions in ITEM FREE mode it would be highly unlikely that any choice other than Choice 5 would be selected because of its very high probability. If the composer still wanted to hear and evaluate the other four choices, he would have to use ITEM SPRIES mode, which would prohibit Choice 5 from being selected until each of the other four choices had been selected.

While EVAL has been provided in order that alteration of the probability tables may be done as efficiently as possible by the composer who is reacting to choices he has just heard, he may want to do even more involved manipulations of the table. For example, he may want to change the weights of choices other than the one in the current version. SET has been provided for this purpose. But, in order to use SET the composer needs more information about which values correspond to which choice numbers. This information is accessible by means of the PRINT SEQ and LAST commands. Both of these cause the contents of a particular sequence or sequences (above, p. 70) to be listed at the terminal. LAST lists the sequence of the current version. PRINT SEQ n lists the sequence for the nth version. Giving the command without
specifying n causes all the sequences thus far generated and in memory to be listed. Example 36 is a sample of a listing of the sequence of the version given in Example 32 (p. 128f).

Example 36. Partial listing of Sequence 1 of SCR004.

[ ]PRINT SEQ 1
SEQUENCE 1

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

......

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>140</td>
<td>1</td>
<td>6</td>
<td>275</td>
</tr>
</tbody>
</table>

......

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>1700</td>
<td>34</td>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>36</td>
<td>2</td>
<td>37</td>
<td>2</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>39</td>
<td>8</td>
<td>2</td>
<td>13</td>
<td>2541</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>2600</td>
<td>3</td>
<td>15</td>
<td>2600</td>
</tr>
</tbody>
</table>

......

[ ]

Each line of the listing is divided into twelve fields of fixed length. The twelve fields are divided into four collections of three fields each, with six spaces in each field, with an extra space separating collections. Each collection represents either an item or a group member. The collections which have the middle field blank, such as those in the first two lines, indicate an item number and the corresponding choice number. The collections which have a number in all three fields indicate a group number, a member number, and the value of the member. Members are numbered consecutively.
from the beginning of the score to the end, even where there is more than one group. The member numbers are those which are used in the extenders (above, p. 67). The group number increments by one for every new primary member. The group numbers are used in group-oriented commands, such as FORCE TIME (below, p. 140). The order in which the items and members occur in the sequence listing is the same as their order in the score. The items discussed so far, Items 1 and 2, can be seen in the first line of the listing. This listing shows the choice numbers of these items that were selected for Version 1.

At some point in working with the master score the composer may decide that only one set of values for Envelopes 1 and 2 meets his needs. In this case he will want the selection process to cease for those items which are operands in the envelope instructions. He could stop the selection process by giving a very high grade to the one choice in each item that meets his needs. For example, if he wanted to hold the rise time of Envelope 1 at .85 seconds, he could give the following command: SET 1,4,10 (above, p. 134). However, if he used SET or EVAL he would have to be sure that there were no other choices which also had high positive grades. If there were, he would have to neutralize them by giving them negative grades. In addition, he would have to use ITEM FREE mode
from then on, since ITEM SERIES would override the weights in the probability tables (above, p. 136). It is much easier simply to bypass the selection process. This can be done by means of the FORCE ITEM or HOLD ITEM commands (above, p. 72). FORCE is to HOLD as SET is to EVAL; that is, HOLD requires less information (above, p. 134). If the current version contains the desired value, say 85 for Item 1, HOLD ITEM 1 will suspend the selection process for that item so that in all subsequent versions, 85 will be its value. FORCE, on the other hand, is more flexible. It is not limited to the values of the current version, but it also requires more information, namely, the choice number, in this case, 4. The FORCE command to accomplish the same as the above HOLD command would be FORCE ITEM 1,4. The other items affecting the first two envelopes can be fixed in the same way. The composer can find the choice number corresponding to a particular value by using the PRINT SEQ and LAST commands (above, p. 73). Once an item is fixed, it will remain so until the proper RELEASE command is given. RELEASE ITEM 1 allows Item 1 to vary once more as before. Not specifying an item number in the command string will cause all items to be released (RELEASE ITEM). In the same way, not specifying an item number in the HOLD command (HOLD ITEM) will cause all items to be fixed at the values of the current version.
Composer evaluation: restricting the selection

process: groups. The composer may at some time focus his attention on the three groups on Channel 2 which control the durations during the three nine-second segments (above, p. 96). These are Groups 2, 3, and 4. Assume that up to this point the composer has generated thirteen versions. He has thirteen sets of values for these three groups from which to choose. He could decide to review the thirteen versions using one of the SEQUENCE modes (above, p. 70f), probably SEQ SERIES, so that all thirteen would be selected once in a random order. Listening to these thirteen versions could enable the composer to decide. However, in these versions the items are varying, too. In order to limit the number of variables that are varying the composer may choose to fix the values of all the items and apply the groups as found in the first thirteen sequences to thirteen new sequences, all having the same values for any item. This could be done with a series of commands similar to those in Example 37.

Example 37. Composer/computer dialogue to select best expressions of Groups 2, 3, and 4 for SCRO004.

```
[ ]NOPRINT           (Inhibit printing of versions)
[ ]FORCE SEQ 5       (Make Version 5 the current version)
SEQUENCE 5
[ ]PLAY              (Listen to Version 5)
[ ]HOLD ITEM         (All items fixed)
[ ]FORCE TIME 1,2    (Force Groups 2, 3, 4 of Version 1)
[ ]FORCE TIME 1,3
```
Proceeding in this manner, when the composer reaches Sequence 26, he will have generated the thirteen new versions. At this point the composer may choose to take an entire version as is, or he may decide to select groups from multiple versions. He may, for example, decide on Group 2 from Version 14, Group 3 from Version 15, and Group 4 from Version 20. He could put these all into a new version, Version 27, with the following series of commands:

Example 38. Commands to combine groups from different versions into a new version.

[ ]FORCETIME 14,2  
[ ]FORCETIME 15,3  
[ ]FORCETIME 20,4  
[ ]ITEM FREE  
SEQUENCE 27  
[ ]PLAY

Everything else that had been fixed earlier will be the same as in Version 26.
Dealing with limited storage. During an extended session at the computer terminal, a large number of versions can be generated, and the memory available for storing the associated sequences can be exhausted. While generating a new version, the composer may receive the appropriate warning (NO MORE ROOM). At this point he has the option of (1) saving the sequences thus far generated on DT1, or (2) simply clearing the sequence storage area and starting over. Proceeding in the first way, the composer makes use of the SAVE SEQ command, which transfers the contents of the sequence area to DT1, and of the ERASE command, which allows the sequence area to be overwritten. The next sequence after the ERASE command is Sequence 1. The SAVE SEQ command can be used more than once if the sequence area is filled repeatedly. If this happens, the composer can specify a page number in the command string, e.g., SAVE SEQ 2. If no page number is specified, page 1 is assumed. If the sequence area were filled a second time and the sequences were saved without specifying a page number, page 1 on DT1 would be overwritten. Pages of the sequence area stored on DT1 can be read back into memory by means of the complementary command, UNSAVE SEQ n, where n is the page number. If no page number is specified, page 1 is assumed.
The second option, that is, simply clearing the sequence area and starting over by means of the ERASE command, is not as self-defeating as it may at first appear. It will usually be the case that the versions generated last are those which correspond closest to the composer's wishes, since they have been the product of the kind of selection demonstrated in the examples above. Composer decisions are implemented by weighting the selection process and by fixing items and groups. The weights are stored in the item area. Information about items that have been fixed is also stored in the item area. Information about groups that have been fixed is stored in the group area. All this information is used in version generation, but it is not stored in the sequence area. When the sequence area is cleared, the only thing the composer has lost is the ability to regenerate versions from those sequences. If HOLD ITEM and HOLD TIME are given before the ERASE command, the first sequence of the new page will be identical to the last sequence of the old page. Not saving unnecessary sequences will also conserve valuable space on DT1.

Use of extenders. Up to this point the exploration of the pitch content of the events of Channel 2 has not been discussed. The use of extenders to unify the material has already been mentioned (above, pp. 98 and
144). However, it is desirable to emphasize that determining the pitches of the events of Channel 2 from 17.00 to 26.00 seconds determines eight more pitches during the next two nine-second segments, and that determining the five items from 26.00 to 35.00 seconds determines five items during the last nine-second segment. The segment from 35.00 to 44.00 seconds has, then, nineteen events, but only ten new pitches; nine have been carried over from the previous two nine-second segments.

Determining the final settings of these pitches can be done using procedures similar to those described above. The composer could decide to fix everything except the items in question and let them vary by means of an ITEM mode in order to get an idea of the various possibilities. This could be done with the commands in Example 39 (p. 145).

He could also decide that, in the interest of greater unity, the four pitches from 17.00 to 26.00 seconds should be the same as the first four notes of the theme. This goal could be accomplished easily by fixing those four items, as is shown in Example 40 (p. 146). The values of the four choices are shown in parentheses.
Example 39. Commands to release only those items which are operands of frequency instructions in the three nine-second segments, for purposes of evaluating frequency content of that part of SCR004.

[ ]HOLD ITEM
[ ]RELEASE ITEM 35

(Fix all items)

[ ]RELEASE ITEM 37
[ ]RELEASE ITEM 39
[ ]RELEASE ITEM 41
[ ]RELEASE ITEM 44
[ ]RELEASE ITEM 47
[ ]RELEASE ITEM 50
[ ]RELEASE ITEM 51
[ ]RELEASE ITEM 53
[ ]RELEASE ITEM 55
[ ]RELEASE ITEM 58

(Release operands of FREQ opcodes in nine-second segments.)

[ ]RELEASE ITEM 84
[ ]ITEM FREE
SEQUENCE 15
[ ]PLAY
[ ]NEXT
SEQUENCE 16
[ ]PLAY

(Start generating new sequences)

[ ]NEXT
SEQUENCE 24
[ ]PLAY
Example 40. Fixing items in first nine-second segment (SCR004).

On the other hand, the composer could conclude that keeping the pitches of all three nine-second segments in the same octave does not provide enough variety. He could decide that it is desirable to alter the range of each nine-second segment by some unspecified amount. This could be done by inserting modifiers (above, p. 65) into these events with items as their operands. In this way the composer could experiment with different pitch ranges in order to find the most satisfactory combinations.

Inserting the modifiers cannot be done without a return to Phase Two (above, p. 61). This is so because changes have to be made to the master score. Example 41 (p. 147) shows part of the affected area of the master score after the changes have been made.

In the revised master score each pitch in any of the nine-second segments is controlled by two modifiers, ADD and SUB, each of which has an item for an operand. Only one pair of items is used as operands for the modifiers in any one nine-second segment. For example, Items 36 and 37 (Example 41, line 71) are used for this purpose in the first nine-second segment. Their influence is carried
Example 41. Partial listing of edited SCR004.

(64) COM BEGINNING OF CHANNEL 2 ITEM 34 IS NEXT

(65) COM

(66) CHAN 2; FUNCT 0;

(67) TIME 0; FREQ 100; LEVEL 0;

(68) COM

(69) COM GROUP 2, MEMBER 10, ITEM 34

(70) TIME <5:1700,2600>; SYNCH (1,2,2);

(71) FREQ (4/14,5/14,13 ADD (0,11,12) SUB (0,11,12));

(72) TIME****; SYNCH (1,2,2);

(73) FREQ (4/14,5/14,13 ADD (36) SUB (37));

(74) TIME****; SYNCH (1,2,2);

(75) FREQ (4/14,5/14,13 ADD (36) SUB (37));

(76) TIME****; SYNCH (1,2,2);

(77) FREQ (4/14,5/14,13 ADD (36) SUB (37));

(78) TIME****;

(79) COM

(80) COM GROUP 3, MEMBER 10, ITEM 44

(81) COM

(82) TIME <10:2600,3500>; SYNCH (1,2,2);

(83) FREQ (35/ADD (0,11,12) SUB (0,11,12));

(84) TIME****; SYNCH (1,2,2);

(85) FREQ (4/14,5/14,13 ADD (45) SUB (46));

(86) TIME****; SYNCH (1,2,2);

(87) FREQ (39/ADD (45) SUB (46));

.....
over to the other pitch items in the segment by extenders. By setting the items associated with the ADD and SUB modifiers for each nine-second segment, the composer can request the transposition of all the pitches in that segment up or down the given number of quarter tones. A pair of modifiers is used, i.e., ADD and SUB, because in this implementation the terms of items must be greater than or equal to zero. In order to obtain a negative transposition, it is necessary to include the SUB modifier, so that its operand can be given a higher value than the accompanying ADD modifier, yielding a negative result.

Adding these items to the master score changes the numbering of the items in the latter part of it. The numbering of the extenders that refer to the items is also changed. Because the number of variables has been changed, any probability tables or sequence area pages that have been saved on DT1 cannot be used with the revised score. However, this is a defect in this implementation only. More sophisticated implementations of the system could make it possible to avoid this problem (below, p. 206).

**Interrupting a session.** A composer may have to end a session at the computer with his work on a master score incomplete. He will be interested in saving the work he
has done so that he can resume it later. Mention has already been made of the SAVE SEQ command (above, p. 142) by which the sequence area can be saved on DT1. The SAVE PROB command can be used in a similar way to make a permanent copy of the item area. After these commands have been given, the composer can leave MASCH by giving the EXIT command, thus returning control to the computer system MONITOR. After the EXIT command has been given, a number is printed at the terminal which represents the current state of the pseudo-random number generator. If the composer is still doing considerable evaluating of computer selections he may want to avoid the same sequence of numbers that were generated during the preceding session. If when he returns to the computer, he gives the RAND n command, where n is the number that was printed at the end of the last session, he will continue the pseudo-random sequence from where it left off. In fact, any number can be used for n, and the command can be given at any time during a session in order to give the generator a new seed. If the RAND command is not used, MASCH starts the pseudo-random number generator at the default value. Upon return to the computer, the composer need only load the appropriate master score and give the UNSAVE PROB and UNSAVE SEQ commands, followed by the optional RAND command, in order to return MASCH to the
state in which he last left it.

**Saving the end product.** As stated before (above, p. 61f), the exploration of a master score can yield two results: (1) a single version that best expresses the composer's concept, or (2) a master score, along with its tables, that can generate an indefinite number of acceptable versions. Whether the composer's aim is the first or the second, he will at some time want to make a permanent copy of one or several versions. Copying is accomplished with the `SAVE SCORE n` command, where `n` is the number of the version to be saved. Giving this command causes the appropriate sequence to be selected and the corresponding version to be regenerated. Then a copy of the version is written onto DT1. The file name for the version consists of the name of the master score followed by a period and three digits giving the number of the version. The command `SAVE SCORE 25` will create a file on DT1 which contains Version 25 and which has the name `SCR004.025`. If the composer at some future time wants to play this version, he can do so by first loading the version with the `LOAD 004.025` command and then giving the `PLAY` command. A saved version such as this is treated in the same way as the scores without variables discussed earlier in the chapter: `SCR001`, `SCR002`, and `SCR003` (above, pp. 78f).
Summary

This chapter has explained the kinds of composer/computer interaction that CAEM is designed to supply. The explanation was accomplished by means of a number of examples in which compositional concepts were explored in order to find their best expression. In addition to clarifying the function of CAEM, this chapter has provided an introduction to the actual use of the system in one implementation. The examples have gradually introduced the instructions and commands that are available during Phases Two and Three. The next chapter will systematize this information and add additional detail. It is needed to organize the information for the composer using the system rather than for the composer learning the system.
CHAPTER IV

CAEM: SYSTEMATIC DESCRIPTION OF PRESENT IMPLEMENTATION

In the previous chapter the workings of CAEM (Compositional Aid for Electronic Music) were explained by minute examinations of a number of sample scores and the ways in which these scores were explored. While this format is suitable for explaining the system, it is not very useful as a reference, and it blurs the major outlines. These defects are remedied by the present chapter. The chapter summarizes CAEM in a systematic way. It is designed as a reference for someone intending to use this implementation or to design his own. Since this chapter deals with the implementation of the two computer-based phases of the system, little attention will be paid to Phase One (above, p. 59), the formation of the compositional concept, since that phase precedes the utilization of a computer. For an overall description of all three phases the reader is directed to Chapter II (above, p. 59). The present chapter will concentrate on the characteristics of the master score format and on the MASCH (MASTER SCORE Handler) command language. Since the present chapter is a summary, it will not include examples
of its own when examples are available in an earlier chapter.

The chapter begins with lists of general characteristics for the master score format and sound synthesis subprogram used in this implementation. These lists are followed by discussions of the following topics:

1. The master score format, including a list of available opcodes and a discussion of the various kinds of operands and delimiters;

2. The position of instructions within events;

3. The MASCH command language, including a list of available commands;

4. A summary of computer-generated messages (Phase Three).

**General Characteristics of the Master Score Format**

1. A master score is a character string which is composed of substrings known as instructions (above, p. 62).

2. Most instructions are grouped into units known as events, the smallest unit of a master score that can describe a sound with respect to time, frequency, and amplitude (above, p. 63).

3. Instructions are composed of opcodes, operands, and delimiters (above, p. 63).

4. Instructions are of variable length and can begin anywhere on a line.

5. More than one instruction can appear on a line.

6. An instruction must be complete on one line.

7. The parts of an instruction have positional
significance. The opcode comes first, followed by the proper number of operands. The quantities specified by the operand or operands receive their significance from their position (first, second, third, etc.) after the opcode (above, p. 63).

8. Delimiters separate operands and parts of operands from each other (above, p. 63).

9. Unreserved delimiters may separate opcodes from operands, although they are not necessary (below, p. 169).

10. Instructions themselves have positional significance (below, p. 172). Events must be encoded in the order of their execution. Within events, frequency instructions generally precede amplitude instructions, and time instructions precede both.

General Characteristics of the Sound Synthesis Subprogram

1. The master score format of this implementation is used to specify sounds for a sound synthesis subprogram which can produce two channels with a maximum frequency of 2000 hz in real time.

2. One event per channel is possible at any one time.

3. There is the capability for periodic frequency and amplitude modulation, but an increase in the complexity of specifications causes a decrease in the maximum frequency limit (below, p. 185f).

Master Score Format

Below is a discussion of the components of instructions in this master score format: opcodes, operands, and delimiters.
Opcodes

An opcode is that part of an instruction which indicates the kind of operation to be specified (above, p. 63). It is composed of alphabetic characters forming a word or words that are meaningful to the composer in remembering what the opcode specifies. It may be written out or abbreviated, but the first three characters must be used. Only the first three characters are decoded. Opcodes come at the beginning of an instruction. Opcodes may be separated from operands by an unreserved delimiter (below, p. 169), such as a space or a comma, but an unreserved delimiter is not necessary. The user is advised to choose a method of punctuation, such as the one used in the examples of Chapter III, and use it consistently. This precaution makes the scores easier to read. Opcodes must be satisfied with the proper number of operands. Less than the required number causes an error message during version generation: "NO ARG?". Extra operands are ignored.

Below is a list of available opcodes grouped by function. Included in the list are the function, syntax, associated error messages (Errors:), and any additional information (Note:) useful to the composer. Some opcodes have their common abbreviations printed next to them in parentheses. Errors detected during version generation
cause a line of question marks to be printed (above, p. 89) stopping at the questionable instruction, which is printed on the next line. If the error is other than the misspelling of an opcode, an error message is printed above the line of question marks. In the case of an unrecognized opcode only the line containing the questionable instruction and the line of question marks is printed.

The first three categories of opcodes are those of time, frequency, and amplitude. These are the categories of opcodes which constitute events.

**Opcodes of Time**

**TIME**

**Function:** Specifies beginning and end times for events (above, p. 64).

**Syntax:** TIME t

where t = time of beginning or end of event in hundredths of a second.

**Note:** TIME usually specifies the beginning of an event, but it can also specify the end of the last event of a series of events (above, p. 112). The assumption behind this format is that it is easier to specify the moments when events begin and end than to specify durations.

**FINE**

**Function:** Specifies the time for the end of the last event on a channel (above, p. 85).

**Syntax:** FINE t

where t = time of the end of the last event on a channel in hundredths of a second.
Opcodes of Frequency

FREQUENCY (FREQ)

Function: Specifies frequency for an event (above, p. 83).

Syntax: FREQUENCY f
where f = frequency in herz or o/p form (above, p. 83).

Note: See note under FMIN below.

Errors: BAD FREQ--FREQON
The subroutine which converts frequency in herz into a form suitable for the sound synthesis subprogram is about to return bad data.

FMIN (FMI)

Function: Supplies frequency information for periodic frequency modulation (above, p. 105).

Syntax: FMIN c, m, d
where c = center frequency in herz or o/p form;
       m = modulating frequency in herz or o/p form;
       d = maximum deviation from center frequency.

Note: FMIN begins frequency modulation, which continues until an FMOUT instruction is detected. Subsequent FREQUENCY instructions provide a new center frequency. The modulating frequency and maximum deviation from center frequency are retained from the last FMIN instruction (above, p. 112).

Errors: BAD FREQ--FREQON
The subroutine which converts frequency in herz into a form suitable for the sound synthesis subprogram is about to return bad data.

FMIE ERROR
While translating an FMIN instruction, bad frequency data were detected.
FREQ TOO HIGH
This instruction results in a frequency which is above the limit of the sound synthesis subprogram.

FMOUT (FM0)

Function: Cancels current frequency modulation, if any (above, p. 112).

Syntax: FMOUT

Note: See note under FMIN (above).

GLISSANDO (GLISS)

Function: Specifies continuous pitch change in one direction during an event.

Syntax: GLISSANDO t, f
where t = duration of pitch change in hundredths of a second; must be less than the duration of event;
f = destination frequency, toward which change is being made.

Note: Frequency from which change is begun is the last frequency generated by the previous event.

When frequency (f) has been attained, it is held for the rest of the event.

Errors: BAD FREQ--PRECON
The subroutine which converts frequency in herz into a form suitable for the sound synthesis subprogram is about to return bad data.

GLISS TOO SLOW
There is a limit slower than which a particular glissando cannot go. This limit is determined by duration (t) and by the number of available frequencies separating the initial and destination (f) frequencies. This limit has been exceeded.
Opcodes of Amplitude

LEVEL

Function: Sets amplitude level for duration of current event (above, p. 85).

Syntax: LEVEL a
where \( a \) = amplitude level in amplitude units (au).

CRESCEndo (CRESc)

Function: Specifies an amplitude level continuously changing in one direction.

Syntax: CRESCEndo t, a
where \( t \) = duration over which amplitude is changing in hundredths of a second;
\( a \) = new level which is to be attained at end of the specified duration.

Note: The duration \( t \) should be less than the duration of the event in which the instruction is found.

Once the new level \( a \) is attained, it is held for the remainder of the event.

Despite the name of this opcode, it may be used to diminish the amplitude level as well as to increase it.

The following opcodes have been provided to describe typical envelope contours as shown in Example 42 below. The names of the opcodes have been printed opposite those parts of the envelope to which they refer.

RISE

Function: Specifies the duration of the initial rise of an amplitude envelope (see Example 42, p. 160; see also p. 93).
Example 42. Relation of typical envelope contours to the amplitude instructions available in this master score format.
Syntax: RISE \( t, a \)

where \( t \) = duration of initial rise from 0 au in hundredths of a second;

\( a \) = instantaneous level in au at end of rise duration.

Note: RISE always initializes level to 0 au before increasing amplitude, regardless of previous level.

FALL

Function: Specifies a continuously changing amplitude level from the previous level, usually the initial peak, to a new level, usually the steady state level (see Example 42, p. 160; p. 93), and the duration of the change.

Syntax: FALL \( t, a \)

where \( t \) = duration of continuous amplitude change in hundredths of a second;

\( a \) = instantaneous level in au at end of duration (\( t \)).

HOLD

Function: Indicates the duration for which the most recent amplitude level is to be maintained (above, Example 42, p. 160; p. 89).

Syntax: HOLD \( t \)

where \( t \) = duration for which most recent amplitude level is to be maintained in hundredths of a second.

Note: HOLD may be used to maintain the amplitude level past the boundaries of one event, since it, like RISE, FALL, and DECAY, is not dependent on the time of the event in which it is found.

HOLD is generally used for the steady state durations after the FALL and DECAY opcodes (above, Example 42, p. 160).

Use of HOLD after DECAY insures that the sound synthesis subprogram will not finish executing the envelope in question until the composer wishes (below, p. 173).
DECAY

Function: Specifies duration of continuously diminishing amplitude level from most recent level to 0 au (above, Example 42, p. 160).

Syntax: DECAY t
where \( t \) = duration over which amplitude is to diminish continuously from most recent amplitude to 0 au in hundredths of a second.

The following two opcodes are used to refer to amplitude instructions which have been grouped into amplitude envelopes.

ENVLOPE (ENVLP)

Function: Marks the beginning of a series of envelope instructions with an identifying number so that they can be referred to later in the score (above, p. 89).

Syntax: ENVLOPE n
where \( n \) = a positive integer between 1 and 20.

Note: The envelope numbers should be assigned in ascending order, since the last number assigned is assumed to be the highest number assigned (see SYNCH, below).

SYNCH

Function: Starts execution of amplitude instructions at the beginning of the envelope referred to (above, p. 89).

Syntax: SYNCH n
where \( n \) = a positive integer between 1 and 20 which has been previously assigned in an ENVLOPE instruction.

Errors: ENVLP # NOT ASSIGNED
The number referred to in this instruction is
higher than the last envelope number assigned. Numbers should be assigned in ascending order. Therefore this number has most probably not been assigned.

**AMIN**

Function: Specifies information needed for periodic amplitude modulation.

Syntax: \texttt{AMIN c,f,d}

where \( c = \text{center amplitude in au}; \)

\( f = \text{modulating frequency}; \)

\( d = \text{maximum deviation from center amplitude in au}. \)

Note: \texttt{AMIN} begins amplitude modulation. Amplitude modulation continues until an \texttt{AMOUT} instruction is detected. This is similar to the way in which \texttt{FM IN} operates (above, p. 157).

**AMOUT**

Function: Cancels any amplitude modulation initiated by an \texttt{AMIN} instruction.

Syntax: \texttt{AMOUT}

**Control Opcodes**

The following opcodes do not regularly occur within events. Their appearance in the score is occasional or irregular.

**FUNCTION**

Function: Specifies the stored waveform which is to be used in the following events (above, p. 82).

Syntax: \texttt{FUNCTION n}

where \( n = \text{a non-negative integer to which a waveform has been assigned}. \)

Note: At this time only Function 0 has been assigned. It is a sinc wave. See below, p. 249 for
directions on how to add functions.

CHANNEL (CHAN)

Function: Specifies the channel to which the following instructions apply (above, p. 82).

Syntax: CHANNEL n

where \( n = 1 \) or 2

Note: Channel 1 must be specified first.

COMMENT (COM)

Function: Clarifies the score by allowing the composer to insert messages to himself (above, p. 81).

Syntax: COMMENT text cr

where text = message the composer desires to include in the score;

\( cr = \) carriage return, with which the comment must end.

Note: A comment cannot be continued on the following line. Each line of a comment must begin with the opcode COMMENT.

COMMENT takes no operands in the usual sense, but only indicates to MASCH that the following characters up to the first carriage return constitute a comment, and is to be disregarded by it.

SECTION (SEC)

Function: Indicates the beginning of a new section, at which point time is to be counted again from 0.00 seconds (above, p. 110).

Syntax: SECTION t

where \( t = \) time in hundredths of a second at which the section begins.

Note: It is necessary to use this command in a composition which lasts longer than 5.46 minutes (327.67 seconds), since this is the largest duration which can be reckoned in this implementation. The composer may find it convenient to use this command when he begins a new compositional section (above, p. 110).
EXIT

Function: Marks the physical end of a score.

Syntax: EXIT

Note: EXIT must be the last instruction in any score.

Errors: READ PAST END OF SCORE
An EXIT instruction was not detected before the end of the character buffer was detected.

Operands

An operand is that part of an instruction which yields a quantity needed by the opcode in order to specify an operation (above, p. 63). The number of operands in an instruction is determined by the opcode. Operands follow the opcode with which they are associated. Operands have positional significance in that, in the case of instructions which take multiple operands, the object quantified depends on the position of the operand after the opcode—whether it is first, second, or third after the opcode.

An operand can be either a constant or a variable. Little need be said about constants. They are simply numbers which are entered during Phase Two and read during Phase Three. Their value always remains the same (above, p. 64). Variables, on the other hand, due to their complexity and to their importance, deserve more extended treatment.
A variable is an operand whose value is changeable during Phase Three (above, p. 64). It is indicated by reserved delimiters which enclose its descriptors. Descriptors define the variable. They are of two kinds: terms and modifiers. A term is a descriptor which is used to define the set of possible values of the variable that is stored in the computer's internal table for that variable. A modifier is a descriptor which describes an arithmetic operation to be performed on the selected value of the variable. Terms affect the internal tables for the variable; modifiers do not. Modifiers operate on the value after it has been selected from the available values but before it is returned to the version being generated. The number of terms is determined by the kind of variable. The terms of a variable are mandatory. Modifiers, on the other hand, are optional additions.

Each modifier consists of a three-character arithmetic operator—ADD, SUB, MUL, or DIV—and an operand which supplies the value by which the variable is to be changed (above, p. 65). The operand in a modifier has all the characteristics of any other operand. It can be either a constant or a variable. Thus, it is possible to have a variable contained within another variable. Modifiers can be used singly or in groups. When used in groups the order of operation matches the sequence in
which they are encoded.

Modifiers were initially intended for use with extenders (above, p. 67) to establish a mathematical relation between the values of two variables, both of which are controlled by the same item or member. However, they can be used with any variable and can take any kind of variable as an operand. SCR004 (above, p. 100) uses modifiers in pairs to control the amount of transposition in the three nine-second segments (above, p. 146). They are used there in both items and extenders. They can also be used within items to provide relations between the values of an item that are unobtainable in the usual way. For example, the following item "(1,5,5)" has the five possible values of 1, 2, 3, 4, and 5. If a modifier with an extender for an operand is added to this item in the following way "(1,5,5 MUL [(1)])", and, assuming that the item in question is Item 1, the modifier multiplies the value of the item by itself before inserting it into the version being generated. This procedure has the effect of making the five values of the item exponential rather than linear in range. The five values are now 1, 4, 9, 16, and 25.

When using combinations of variables, care should be taken that all opening parentheses, square brackets, and angle brackets have their complementary closing characters
and that they have them in their proper order.

There are three categories of variables. Their definition and the syntax for their terms are given below:

Item:

A variable having a specific number of possible values equally spaced within a specific range (above, pp. 66, 96).

Syntax: (a,b,c)
where a = lower bounds of the range;
       b = upper bounds of the range;
       c = number of equally spaced values within the range.

Group:

A set of a specific number of variables, known as members, whose values fall within a specific range and which are arranged in the master score in order of increasing value. Descriptors are given in the primary member. The terms of the primary member specify values for the primary member and for the secondary members. It is usually used with opcodes of time (above, pp. 64, 110, 113).

Syntax of primary member: <a,b,c>
where a = number of members in the group;
       b = lower bounds of the range;
       c = upper bounds of the range.

Syntax of secondary members: ******;

Note: Secondary members can be represented by two or more asterisks, depending on the size of the largest expected value. However, it is recommended that six always be used (below, p. 171).

All secondary members must end with a semicolon (below, p. 170).

The primary member is the first to occur. The program fills the specifications given by the
primary member by generating a series of pseudo-random numbers, one for each member, within the specified range, and inserting these values into the score at the locations marked by the members in ascending order (above, p. 66).

Extender:

A variable which can assume the value of another variable, either an item or a group member (above, p. 67, 114, 143).

Syntax: 

\[
[ (i) ] \quad \text{[<m>]} \\
\text{where } i = \text{item number;} \\
\text{m = member number.}
\]

Errors: SQ BRCKT ERM
Something is wrong with the form of this extender.

Delimiters

Delimiters are characters other than those which are alphabetic or numeric. Their function is to separate operands and parts of operands from each other and opcodes from operands. Special delimiters, known as reserved delimiters, indicate specific kinds of variables or have other special functions. Of the unreserved delimiters the most commonly used are the space, comma, period, colon, and semicolon. Opcodes need not be separated from operands by anything; operands and parts of operands can be separated by any unreserved delimiter. However, it is desirable for ease of encoding and reading that unreserved delimiters be used in a consistent manner (above, p. 155). The following recommendations are made regarding the use of unreserved delimiters. These recommendations have been
followed, for the most part, in the examples given in Chapter III.

1. A space should separate an opcode from its first operand.

2. Commas should be used to separate operands and parts of operands.

3. A semicolon should end most instructions. It must end every secondary member (above, p. 160).

4. A colon should be used to end an ENVELOPE instruction.

5. A period may be used at the end of each event or at the end of a channel.

Below is a list of reserved delimiters and their functions. The directions given concerning their use are not optional but are mandatory for the proper functioning of the program. In using delimiters associated with variables it is necessary to insure that there are enough positions between the delimiters to accommodate the largest value that the variable can assume. This precaution must be taken because in the process of printing a listing of a version, the characters in memory (the character buffer) which are the descriptors of the variable, are replaced by the characters representing the value that the variable is to assume. This is usually no problem since the descriptors of an item or primary member will always occupy more space than a specific value, but in extenders and secondary members, extra spaces or, in the case of secondary members, extra asterisks, will have to be used
to accommodate all the digits of the largest possible value. If a value will not fit in the positions available, an error message, NO ROOM IN CH BUP, will be printed along with the line containing the offending variable. This condition requires a return to Phase Two to add the additional space. When the descriptors occupy more positions than the value, a listing of the version will show the excess positions filled with spaces (above, p. 128).

Parentheses ()

Mark the beginning and end of an item. When encoded in the master score they contain the descriptors of the item. In a listing of a version they contain the current value of the item. Unused positions between the parentheses are replaced by spaces (above, p. 128).

Angle Brackets <>

Mark the beginning and end of a primary member of a group. When encoded in the master score, they contain the descriptors of the group. In a listing of a version they contain the current value of the primary member. Unused positions between angle brackets are replaced by spaces.

Asterisks *****

Mark the beginning and hold spaces for the values of secondary members. A string of asterisks is used to represent each secondary member. There should be enough asterisks to accommodate all digits of the largest possible value to be inserted there. Since the largest possible value in this implementation is a five-digit number, and since one asterisk is left in the listing of a version to indicate a secondary
member, it is therefore recommended that every secondary member be represented by a string of six asterisks (above, p. 168). In a listing of a version all asterisks except the first are replaced by the digits of the value the member has taken.

Semicolon ;

This delimiter has the dubious distinction of being both a reserved and an unreserved delimiter. This is so because a semicolon must mark the end of every secondary member. However, a semicolon need not be used to terminate any other variable or instruction. It is good practice to end every instruction (except the ENVELOPE instruction, which should end with a colon) with a semicolon, although any other delimiter can be used (above, p. 170). This will insure that every time instruction containing a secondary member ends with a semicolon.

Square brackets [ ]

Mark the beginning and end of an extender. These square brackets are used in combination with parentheses and angle brackets (above, p. 67) to indicate whether the enclosed term is the number of an item or a member (above, p. 168f); that is, [( )] is used to indicate an item number, and [< >] is used to indicate a member number. During version generation when the listing is being prepared, the term is replaced with the value taken by the extender. As was said above, the composer should be sure to leave sufficient spaces between the delimiters to accommodate the largest possible value the extender can assume (above, p. 170).

Position of Instructions Within Events

In this implementation MASCH decodes and writes the internal representation of the version one instruction at a time. Frequency instructions are entered one after another in one part of memory, while amplitude instructions are entered one after another in another part of memory. Time instructions are used to establish
durations for frequency and amplitude instructions separately. The limited capabilities of this implementation make it imperative that the instructions be encoded in a strict order. Below is a diagram giving the relative positions of opcodes occurring within events.

Example 43. Relative position of opcodes within events.

<table>
<thead>
<tr>
<th>TIME</th>
<th>SYNCH</th>
<th>FREQUENCY</th>
<th>ENVELOPE</th>
<th>LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>FINE</td>
<td></td>
<td>FHOUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>GLISSANDO</td>
<td>CRESCENDO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RISE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FALL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HOLD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DECAY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AMIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AMOUT</td>
</tr>
</tbody>
</table>

ENVELOPE precedes the envelope instructions to which it refers. Envelopes do not have to be defined within events. They can be defined at the beginning of a score and from then on just referred to by SYNCH instructions (above, p. 162), but in either case, ENVELOPE must precede the instructions to which it refers.

It can be seen that frequency instructions generally precede amplitude instructions. The exception to this rule is the SYNCH instruction, which must precede the frequency instruction in the event where it occurs.

SCR003 and SCR004 (above, pp. 94, 100) in Chapter III have envelopes that end with a HOLD instruction in which the operand is 6000—a number representing 60.00 seconds, a duration longer than would ever be needed. It is good
practice to end an envelope with such an instruction because during Phase Two the composer is not always certain what the durations of the events which will use a particular envelope will be. The durations may be controlled by groups or other variables. If an event in which an envelope is used is longer than the sum of the durations of all the amplitude instructions in its envelope, the sound synthesis subprogram will malfunction either by halting sound synthesis prematurely or by beginning execution of whatever amplitude instruction happens to be next in memory.

Control opcodes (above, p. 163) are not included in the above list. Their placement is not as critical. FUNCTION (above, p. 82) occurs in the score before the event to which it refers. COMMENT (above, p. 81) can occur anywhere in the score, but good practice dictates that it begin a line. Everything on that line after COMMENT is a comment. EXIT must come at the physical end of every score. Omission of this instruction will cause a READ PAST END OF SCORE error to be printed at the terminal (above, p. 165).

**MASCH Command Language**

When MASCH is ready to receive a command, it prints [ ] on the terminal. This symbol is used to distinguish
the MASCH monitor from the computer system MONITOR, which prints a period (.) when it is ready to receive a command, and from some of the utility programs of the system, which use the asterisk (*). The operation of MASCH is directed by the composer, who uses the terminal to present MASCH with various commands which MASCH decodes and executes. Every command begins with one or more alphabetic words. In addition, some commands require one or more numeric arguments. The same rules apply to commands as apply to opcodes; that is, only the first three letters of each alphabetic word are required in order to be decoded, although the composer may wish to use more than the first three letters if it helps him to remember what the command signifies. Command words may be separated from their arguments by a delimiter although it is not mandatory. Command words must be separated from each other by a delimiter, and a delimiter must be used to separate multiple arguments. Spaces and marks of punctuation may be used as delimiters, but spaces and commas are preferred. Every command is terminated with a carriage return.

The following is a description of the commands available in this implementation. The commands are grouped according to function into the following categories:
1. Control of Execution
2. Selection of Mode of Variation
3. Weighting the Selection Process
4. Restricting the Selection Process
5. Sound Synthesis
6. Visual Feedback
7. Data Storage and Retrieval

Some of these categories deal with manipulation of information supplied to MASCH from variables. These categories can only be used with scores that contain variables, that is, scores that are true master scores (above, p. 78, 96). The categories in question are numbers 2, 3, 4, 6, and 7. Attempting to execute commands in one of these categories upon a score without variables will cause an error message (?? ILL CMND ??).

1. Control of Session

RUN

Function: Loads and starts execution of MASCH (above, p. 128).

Syntax: RUN DT1: MUSIC

where DT1 = device on which the load module MASCH is to be found;
MUSIC = name of the load module.

Note: Immediately after MASCH has been successfully loaded it will print [] on the terminal to indicate that it is ready to accept a command.

RUN is necessary but nevertheless external to MASCH. The reader is directed to the computer system manual (Digital Equipment Corp., 1974), for further information on the command repertoire of the computer system MONITOR.
EXIT

Function: Ends a session and transfers control back to the computer system MONITOR (above, p. 149).

Syntax: EXIT

Note: After this command has been given, a number will be printed at the terminal which indicates the final state of the random number generator. This number can be re-entered at the beginning of the next session with the RAND command (above, p. 149; below, p. 179), if the composer desires to continue with the pseudo-random sequence from where he left off rather than let NASCH restart the random number generator at the default value.

2. Selection of Mode of Variation

The four modes of variation (above, p. 124) control the way in which material is selected for presentation to the composer. The two ITEM modes control the selection of values for individual variables. When operating in an ITEM mode, values selected for individual variables are used to generate new versions. The two SEQUENCE modes control the selection of previously generated versions. When in a SEQUENCE mode no new versions are generated. Old versions are regenerated from their sequences. ITEM modes are used first to generate compositional alternatives. In the course of working with the master score, the composer may find SEQUENCE modes useful for reviewing previously generated material. A mode selection command precedes the generation or regeneration of any version. Below is a list of the commands associated with
the mode selection process along with information pertaining to each.

ITEM FREE

Function: Directs MASCH to make a pseudo-random selection from the choices available for each free item (below, p. 182) according to the probability of occurrence of each choice; to generate new values for all group members; and to use the values so obtained to generate a new version (above, p. 124).

Syntax: ITEM FREE

ITEM SERIES

Function: Directs MASCH to make a pseudo-random selection for each free item as in ITEM FREE mode, but in addition to pass over any choice that has previously been made until all choices for that item have been made. New groups are generated as in ITEM FREE mode. The values so obtained are used to generate a new version (above, p. 124).

Syntax: ITEM SERIES

SEQUENCE FREE

Function: Directs MASCH to make a pseudo-random selection from among the sequences that have thus far been generated, according to the probability of occurrence of each sequence; to regenerate the version associated with that sequence; and to make that version the current version (above, p. 140).

Syntax: SEQUENCE FREE

Note: Neither SEQUENCE command can be used until at least one sequence has been generated.

SEQUENCE SERIES

Function: Directs MASCH to make a pseudo-random selection from among the sequences that have
thus far been generated as in SEQUENCE FREE mode, but directs MÄSCH to pass over any selection that has already been made until all sequences thus far generated have been selected. Once a valid selection has been made, MÄSCH is to regenerate the version associated with it and make it the current version (above, p. 140).

Syntax: SEQUENCE SERIES

Note: Neither SEQUENCE command can be used until at least one sequence has been generated.

NEXT

Function: Causes a version to be generated or regenerated according to the current mode (above, p. 137).

Syntax: NEXT

Note: This command is a "wild card" in that it can take the place of any of the four mode selection commands.

This command should be used only when the composer is certain of the mode he is using.

RAND

Function: Allows the composer to change the seed of the pseudo-random number generator and thus to change the number sequence which it generates (above, p. 149).

Syntax: RAND n

where n = any positive number less than 2**15.

Note: This command can be used at any time, although it is most useful when resuming an interrupted session. At the beginning of the new session, the composer can use this command to enter the number that was printed after the EXIT command at the end of the last session. This results in a continuation of the same pseudo-random sequence. It reduces the chance that versions similar to those of the last session will be generated due to the use of the default seed for both sessions. However, even if the default seed is used when
the session is continued, the likelihood of generating versions similar to those generated during the last session is very small.

3. Weighting the Selection Process

The following two commands are used to alter probabilities of selection of choices for items and probabilities of selection of sequences. Both EVALUATE and SET are used with items; only EVALUATE is used with sequences. In the case of items, both EVALUATE and SET perform the same function, but while EVALUATE requires less information, SET is more flexible.

EVALUATE

Function: Allows the composer to give a grade and a corresponding weight to the current choice of a particular item or items while in an ITEM mode, and to the current sequence while in a SEQUENCE mode (above, pp. 71, 131).

Syntax: EVALUATE i g i g i g ... (ITEM mode)
         EVALUATE s     (SEQUENCE mode)
where
i = item number;
g = grade for current choice, with absolute value less than 15;
s = grade for current sequence with absolute value less than 15.

Note: While in an ITEM mode, EVALUATE takes two arguments in order to grade the current choice of a particular item. When in a SEQUENCE mode, it takes one argument, the grade for the current sequence.

Multiple pairs of arguments (i g i g i g) indicate that while in an ITEM mode, more than one item can be graded in one command string (above, p. 133).
The most useful function for this command in a SEQUENCE mode is in conjunction with SEQUENCE FREE mode, in order to eliminate from the selection process a sequence that is no longer under consideration. This is done by giving the sequence in question a very low grade.

**Errors: EVAL LIMIT**
The greatest weight that can be given to a choice or sequence is a positive or negative 14. If that limit is exceeded for a particular choice or sequence, this warning message will be printed. If the limit is exceeded while giving a negative grade, the warning message may be printed more than once. This happens because giving a negative grade causes the weights of all choices or sequences other than the one in question to be increased. Any time during this process that a choice or sequence reaches its maximum weight, the message is printed.

**NO GRADE**
The second of the two arguments for grading a choice has been omitted. When using EVALUATE in an ITEM mode, the number of arguments must be even (two arguments for every grade). The message may mean that the number of arguments is odd. The composer may have intended to grade a sequence but was in an ITEM mode.

**NO SUCH CHOICE**
An attempt has been made to grade a nonexistent choice.

**NO SUCH ITEM**
An attempt has been made to grade a choice for a nonexistent item.

**SET**

*Function:* Allows the composer to give a grade and a corresponding weight to any choice of a particular item or items (above, p. 134).

*Syntax:* SET i c q i c q i c q ...

*Note:* Since any choice can be specified rather than
just the current choice, it is necessary to include an additional argument (c) to supply the choice number.

Multiple groups of arguments (icg icg icg) indicate that more than one item can be graded in one command string.

The composer can obtain information about which choice numbers correspond to which values by using the PRINT SEQUENCE and LAST commands (above, p. 136, 139; below, p. 187).

Errors: EVAL LIMIT
NO GRADE
NO SUCH CHOICE
NO SUCH ITEM

See EVALUATE (above).

4. Restricting the Selection Process

The following commands are used to suspend the selection process for items and groups (above, pp. 72, 139f). When the selection process has been suspended for an item or group, that item or group is said to be fixed. Otherwise the item or group is free, that is, free to vary according to the probabilities of selection, if any, that pertain to it. The HOLD and FORCE commands have the same relation to each other as do the EVALUATE and SET commands (above, p. 134; 180), that is, the HOLD commands require less information than the FORCE commands, although the FORCE commands are more flexible.

HOLD ITEM

Function: Fixes an item at the value it has in the current version (above, p. 139).
**Syntax:** HOLD ITEM i  
where i = item number.

**Note:** Since the command implies that the choice to be fixed is the current choice, only the item number is required.

After this command is given, the item is fixed for all subsequently generated versions until the appropriate RELEASE command is given (above, p. 139).

Giving the command without specifying i results in all items being fixed at the current values.

---

**FORCE ITEM**

**Function:** Fixes an item at any of its possible values (above, p. 139).

**Syntax:** FORCE ITEM i c  
where i = item number;  
c = choice number.

**Note:** A choice number (c) is required here because any choice can be specified (above, p. 139).

This command fixes the item for all subsequently generated versions until the appropriate RELEASE command is given.

---

**RELEASE ITEM**

**Function:** Frees the item specified (above, p. 139).

**Syntax:** RELEASE ITEM i  
where i = item number.

**Note:** This command nullifies the effect of any previous FORCE ITEM or HOLD ITEM command on the specified item.

If no item (i) is specified, all items are freed.

---

**HOLD TIME**

**Function:** Fixes the members of the specified group at the values they have in the current version.
Syntax: \texttt{HOLD TIME }g
\texttt{where }g = \text{ group number.}

Note: Giving the command without specifying \emph{g} fixes all members of all groups at their current values. The group specified is fixed until the proper RELEASE command is given (below).

FORCE TIME

Function: Fixes the members of the specified group at the values they have in the designated sequence (above, p. 139).

Syntax: \texttt{FORCE TIME }s g
\texttt{where }s = \text{ sequence number;}
\texttt{g = group number.}

Note: The specified group is fixed until the proper RELEASE command is given (below).

RELEASE TIME

Function: Frees the specified group.

Syntax: \texttt{RELEASE TIME }g
\texttt{where }g = \text{ group number.}

Note: This command nullifies the effect of any previous HOLD TIME or FORCE TIME command on the specified group.

Giving the command without specifying \emph{g} frees all groups.

FORCE SEQUENCE

Function: Causes the specified sequence to be used to regenerate the corresponding version and make it the current version (above, p. 140).

Syntax: \texttt{FORCE SEQUENCE }s
\texttt{where }s = \text{ sequence number.}

Note: This command has no corresponding HOLD or RELEASE commands.

Giving this command puts MASCII in a SEQUENCE mode.
5. Sound Synthesis

The following two commands control the sound synthesis subprogram which reads and realizes the internal representation of the current version (above, p. 88).

PLAY

Function: Transfers control to the sound synthesis subprogram which reads and realizes the internal representation of the current version.

Syntax: PLAY

Note: Versions are always played from the beginning, but the playing can be halted at any time by setting the register switches (below, p. 269) on the computer console to 0.

Errors:

ERROR 1
During sound synthesis the digital-to-analog conversion subroutine (DACNV) attempted to read data not yet calculated for it. Calculations can not be made fast enough for real-time synthesis. Try increasing value of c (see CSET, below).

ERROR 2
During sound synthesis DACNV was called by an interrupt (below, p. 269) from the real-time clock before it was able to finish servicing its last interrupt request. This indicates that DACNV has accepted bad data or that the sound synthesis subroutine has entered an endless loop and has ceased calculating new data for DACNV.

ERROR 3
An attempt was made during sound synthesis to generate a frequency which was out of range (above 2000 hz with c = 25) (below, p. 186). This can happen if the composer has requested a frequency that is too high by a FREQUENCY command or has made an error in determining
what would be the highest frequency generated by an FMIN command, or if the composer has used CSET (below) to reduce the upper frequency limit after receiving an ERROR 1.

CSET

Function: Changes the minimum interrupt duration (below, p. 268) given to the real-time clock for controlling the timing of the digital-to-analog conversion routine (DACNV).

Syntax: CSET c

where c = minimum interrupt duration in tens of microseconds.

Note: When MASC is loaded, c is equal to 25. If the synthesis specifications are complex, the sound synthesis subprogram may not be able to compute values for DACNV fast enough. If this happens, an error message will be printed at the terminal (ERROR 1). Increasing c may solve the problem if the version does not request frequencies close to the upper limit (2000 Hz), since increasing c also reduces the frequency range.

6. Visual Feedback

The following commands produce listings at the terminal which are of value to the composer in his exploration of the master score:

PRINT SCORE

Function: Causes listing of all versions subsequently generated or regenerated.

Syntax: PRINT SCORE

Note: Versions will be listed (above, p. 127) at the terminal as they are being generated or regenerated until the NOPRINT command is given (below).
N O P R I N T

Function: Inhibits listing of versions at the terminal (above, p. 130).

Syntax: N O P R I N T

Note: This command is included because it is not always desirable to list every version. This is mainly due to considerations of time, since nontrivial scores may take over a minute to list (using a DECwriter; above, p. 78).

When N A S C H is first loaded, N O P R I N T is in effect.

P R I N T  P R O B

Function: Lists the probability table of the specified item (above, p. 135).

Syntax: P R I N T  P R O B  \( i \)
            where \( i \) = item number.

Note: If \( i \) is not specified, the probability tables of all items are listed.

P R I N T  S E Q

Function: Lists the sequence information for the designated sequence.

Syntax: P R I N T  S E Q  \( s \)
            where \( s \) = sequence number.

Note: The sequence information consists of the choice numbers for all items and the values for all group members (above, p. 137).

Giving the command without specifying \( s \) will cause all sequences thus far generated in the current page (above, p. 137) to be listed.

L A S T

Function: Lists the sequence information for the sequence of the current version (above, p. 136).

Syntax: L A S T
7. Data Storage and Retrieval

This last category of commands contains those which control the transfer of information to and from computer memory and DT1, the storage device through which permanent files of score-related information can be maintained.

LOAD

Function: Causes the specified score to be located on DT1, scanned and otherwise prepared for processing in Phase Three (above, p. 87).

Syntax: LOAD nnn.vvv
where n = score number digit;
     v = version number digit.

Note: When a score is being loaded that has been directly encoded by the composer during Phase Two, it is necessary only to enter nnn, which is a three-digit number corresponding to the last three characters of the six-character file name, the first three of which are SCR (above, p. 82). Versions of a master score which have been saved by SAVE SCORE (below) have an additional three-digit number which is their version number (above, p. 150). This is separated from the score number by a period. If a saved version is to be loaded, it is necessary to specify both nnn and vvv separated by a period.

SAVE SCORE

Function: Causes the specified version to be regenerated and stored as a file on DT1.

Syntax: SAVE SCORE v
where v = version number.

Note: The name of the resulting file is composed of SCR plus the three-digit number of the master score plus a period and another three-digit number which is the same as that of the version (above, p. 150; also under LOAD).
Versions saved in this way can be retrieved with the LOAD command.

SAVE PROB

Function: Saves the item area (above, p. 149) as a file on DT1.

Syntax: SAVE PROB

Note: The name of the resulting file is the same as that of the master score plus a period and the three letters PRB (SCRnnn.PRB, where nnn are the three digits of the score number).

UNSAVE PROB

Function: Retrieves the probability table stored on DT1 by the SAVE PROB (above) command and reads it into the item area of memory (above, p. 149).

Syntax: UNSAVE PROB

Note: This command cannot be used until the proper master score has been loaded via the LOAD command.

Probability tables can be successfully retrieved only as long as the number and relative position of the variables have remained unaltered since the time when the table was originally saved. This excessive dependence upon position is a defect in this implementation which can be eliminated in future implementations (below, p. 206).

SAVE SEQUENCE

Function: Saves the sequence area as a file on DT1.

Syntax: SAVE SEQUENCE p

where p = page number.

Note: It is necessary to specify p only if the sequence area is filled more than once. If p is not specified, page 1 is assumed (above, p. 142). If the contents of the second filling of the sequence area is to be saved, page 2 should be specified. Specifying page 2 will cause the
second filling to be stored on DT1 below the first filling. If no page number had been specified, the first filling of the sequence area would be overwritten by the second.

After this command is executed, the message "ERASE?" will be printed at the terminal. This is to remind the composer that merely giving the SAVE SEQUENCE command does not clear the sequence area but only makes a permanent copy of it. If further sequences are to be generated, the sequence area needs to be cleared with the ERASE command (below).

The command SAVE SEQUENCE is usually given either at the end of a session or in response to the message "NO MORE ROOM" which indicates that the sequence area is full (above, p. 142).

UNSAVE SEQUENCE

Function: Retrieves from DT1 the specified page of the sequence area which has been saved previously by SAVE SEQUENCE (above, p. 142).

Syntax: UNSAVE SEQUENCE p
where p = page number.

Note: A page number need be specified only if the sequence area has been filled more than once (above, note under SAVE SEQUENCE).

If no page number is specified, page 1 is assumed.

The sequence area can be successfully retrieved only if the score has remained unaltered since the area was originally saved. This is a defect in this implementation which can be eliminated in future implementations.

ERASE

Function: Clears the sequence area (above, p. 142).

Syntax: ERASE

Note: The first sequence generated after giving the ERASE command is Sequence 1.

This command is used in conjunction with SAVE
SEQUENCE to record more sequences than the allocated memory space can handle.

It is sometimes desirable to clear the sequence area without saving the stored sequences first (above, p. 143).

In the process of transferring data to and from DT1, a number of errors can occur. Such errors are indicated by these error messages:

- ENT ERR
- LOOKUP ERR
- WRITE ERR
- READ ERR
- BAD DATA TO ENTER
- BAD DATA TO LOOKUP
- BAD DATA TO WR
- BAD DATA TO READ

These error messages refer to the four subroutines that perform the data transfer tasks: ENTER, which prepares DT1 for receiving a new file; LOOKUP, which locates a previously written file; READ, which reads data from a file; and WRITE, which writes new data into a file. The first four error messages can indicate either a hardware malfunction or invalid information supplied by the composer. The last four indicate some kind of composer error.

Summary of Computer-Generated Messages

During Phase Three, MASCH is capable of transmitting to the composer various kinds of unsolicited information. Some of these messages are reminders. Most of them,
however, are indications that the user has supplied erroneous information. The following is an alphabetical list of these messages. Most of them have been discussed earlier; a few have not. Those that remain to be discussed indicate error conditions that are not as directly under the control of the composer. They were inserted into the program to aid in locating programming errors during system development. They have not been mentioned before because it has not been necessary to do so. References to more complete information concerning them have been included.

A number of errors are detected during version generation. When an error occurs, MASCN always gives the place in the score which was being scanned when the error was detected. MASCN indicates the location of the error by printing the line that was being scanned and above it a string of question marks ending at the point on the line where the scanning stopped (above, p. 89). The error message in question is printed above this. If no error message is printed, then the location marked by the string of question marks contains an opcode which was not recognized as valid.

A string of five question marks is printed at the terminal if the preceding command string was not recognized as beginning with a valid command.
BAD DATA
An operand has an illegal value.

BAD DATA TO ENTER
BAD DATA TO LOOKUP
BAD DATA TO READ
BAD DATA TO WR

These four messages indicate error conditions encountered by the subroutines manipulating the files on DTI. ENTER prepares the device for a new file. LOOKUP finds a previously written file. READ and WRITE read data from a file and write it into a file (above, p. 191).

BAD FREQ—FRECON
FRECON is the name of a subroutine which translates frequency in hertz into the form used by the sound synthesis subprogram. The message indicates that the subroutine is about to return erroneous values. Probably bad input data are the cause (above, p. 157).

ENT ERR
Error condition detected while ENTER was preparing DTI for a new file (above, p. 191).

ERASE?
A reminder that after saving the sequence area the composer must clear it with the ERASE command before he can generate any new versions (above, p. 190).

ENVP # NOT ASSIGNED
A SYNCH instruction has just tried to refer to an envelope that has a number higher than the number of the last envelope assigned. Numbers should be assigned in ascending order. Therefore, this number has most probably not been assigned (above, p. 162).

ERROR 1
During sound synthesis DACNV (above, p. 185) attempted to read data that was not yet calculated for it. Calculations were too complex to be made fast enough for real-time synthesis. Try increasing the value of c (default value = 25) with the CSET command (above, p. 185; below, p. 268).
ERROR 2
During sound synthesis DACNV was called by an interrupt from the real-time clock before it was able to finish servicing the last interrupt request. This indicates that DACNV has accepted bad data or that the sound synthesis subroutine has entered an endless loop and has ceased calculating new data for DACNV (above, p. 185).

ERROR 3
An attempt was made during sound synthesis to generate a frequency which was out of range (above 2000 hz with c = 25). This can happen if the composer has requested a frequency that is too high with a FREQUENCY instruction or has made an error in determining what would be the highest frequency generated by an FMIN instruction or if the composer has used CSET to reduce the upper frequency limit due to receiving an ERROR 1 (above, p. 186).

EVAL MODE
Reminder given to composer after LOAD command has been executed on a score with variables. It reminds the composer to supply a mode of variation (evaluation) so that the first version can be generated (above, p. 124).

EVAL LIMIT
During an EVALUATE or SET command the maximum weight has been given to a choice. If a negative grade was given, the message may be printed more than once since it is given every time a choice reaches the maximum weight and because, for a negative grade, the weights of all choices except the one being graded are increased (above, p. 181).

FINDIT BAD DATA
FINDIT BAD ERR
FINDIT CHOICE ERR
FINDIT is a subroutine used to locate a desired item. It is used by many of the subroutines which make up MASCH. These messages indicate that the subroutine received bad data which in some cases caused the subroutine to search for an item outside of the item area or caused the program to look for a non-existent choice.

FMI ERROR
While translating an FMIN instruction, bad frequency data were detected (above, p. 157).
**FREQ TOO HIGH**
During version generation a frequency above the range of the sound synthesis subprogram has been detected (above, p. 158).

**GLISS TOO SLOW**
For a particular GLISSANDO command the limit for slowness of a pitch change has been exceeded. If the offending operand is a variable, it is necessary to change the master score (Phase Two) or try another choice (above, p. 158).

**ILL CMD (?? ILL CMD ??)**
The last command may not be given at this time (above, p. 176).

**LOOKUP ERR**
Error condition detected during lookup of a file on DT1 (above, p. 191).

**NEG NUM IN TIME FORCE**
A negative number has been detected in a word used to indicate whether a group is fixed or free (above, p. 182). This number should always be positive or zero (below, p. 250).

**NO ARG?**
Missing operand or argument (above, p. 155).

**NO DELIMITPR**
During version generation the program has detected a variable with no final delimiter (above, p. 252).

**NO GRADE**
During a grading command (EVALUATE or SET) an argument specifying a grade was omitted (above, p. 181).

**NO MORE ROOM**
KASCH has just attempted to write sequence information in the area of memory reserved for the internal representation of a version which is read by the sound synthesis subprogram. The sequence area was full with the last sequence. It is time to erase the sequence area and/or save it (above, pp. 142, 190).

**NO ROOM IN CH BUP**
While encoding the master score, not enough spaces were left between the reserved delimiters of a variable to accommodate all the digits of the largest
possible value that this variable can assume. Return to Phase Two and add the necessary spaces (above, p. 171).

NO SUCH CHOICE
An attempt has been made to reference a nonexistent choice (above, p. 181).

NO SUCH ITEM
An attempt has been made to reference a nonexistent item (above, p. 181).

0 IN SEQ GRADE
A zero has been detected in the word reserved for storing a sequence grade. Grades should be non-zero and positive (below, p. 265).

READ ERR
Error condition detected while reading a file from DT1 (above, p. 191).

READ PAST END OF SCORE
MASCH has scanned outside of the character buffer. Score may lack EXIT instruction (above, p. 165).

SEQFND BND ERR
While looking for a sequence, MASCH has left the sequence area (below, p. 265).

SEQMOD BND ERR
SEQMOD ERR
While in a SEQUENCE mode an error has been detected while attempting to locate a choice for an item. Either MASCH has scanned outside of the sequence area or the number has been found to be negative or zero when it should be positive (below, p. 247).

SQ BRCKT ERR
An error has been detected in the syntax of an extender (above, p. 169).

TIME BND ERR
While in a sequence mode an error has been detected in an attempt to locate a group member. MASCH has scanned outside of the group area (below, p. 250).

WR ERROR
Error condition detected while writing a file to DT1 (above, p. 191).
CHAPTER V

EVALUATION AND SUGGESTIONS FOR FUTURE IMPLEMENTATIONS

The preceding chapters have presented the historical background, philosophy, design, and implementation of a Compositional Aid for Electronic Music (CAEM). The present chapter reflects upon the work done so far and offers some suggestions for improving future implementations. The subject is treated under two major headings: (1) the design of the Master Score Handler (MASCH) and (2) the master score format and sound synthesis subprogram.

Design of the Master Score Handler

Speed of Version Generation

CAEM is primarily a means by which a composer can explore compositional alternatives, or versions, embodied in a master score. Once the decisions have been made concerning what operands are to vary and within what ranges, and once the master score has been encoded and entered, versions can be generated rapidly. The speed of version generation aids in the process of comparing and evaluating, since the time interval between auditions of different versions can be quite short. Thus, multiple
versions can be fresh in the composer's mind.

**Computer Selection and Composer Evaluation**

CAEM augments the benefits of computer sound generation by automating the selection process. For initial exploration the composer does not have to decide which selections he wants to make next. He only has to decide to hear something else. Individual selections may be left to the computer. Even so, the ranges of selection are established by the composer. The computer only particularizes general decisions already made by the composer. All computer decisions are filtered through the composer. Unwanted decisions can be rejected. The possibility that the computer will make unwanted choices is balanced by the possibility that the computer will choose possibilities that the composer not only will find desirable but would not have conceived of on his own.

There has been no attempt to program compositional procedures. This is exemplified in the structure and handling of variables. For example, if the composer does not have in mind specific time intervals for a particular section of a composition, he may direct the computer to present to him various alternatives. He can do this by means of a group (above, p. 66). The generation of members is basically random. Yet the mindlessness of the process is complemented by the speed with which a number
of groups can be generated, from which the composer can choose the one which best suits his needs and then suspend further group generation.

Record of Versions

It has proved helpful to be able to refer to any version that has previously been generated. The best parts of previous versions can be easily combined into new versions. In this way the list of sequences becomes a kind of history of the development of the composition.

Modes of Variation

The modes of variation have proven effective, each in its own way. ITEM SERIES has been best used to insure selection of all choices of each item in preliminary explorations. ITEM FREE has been best used to suppress or emphasize certain choices, since in ITEM FREE mode the probability tables are operative. SEQUENCE SERIES has been best used to review all previously generated versions. SEQUENCE FREE has been best used to review selected versions by allowing the composer to make the selection of undesired sequences very unlikely.

Range of Items

The terms of an item allow the composer to establish a range and a number of values that are to be selected from that range. The values of an item always have equal
or nearly equal numbers of units between them. While this has proven sufficient in most cases, sometimes other distributions would be preferred. When items are used with opcodes of frequency, it is often desirable to have a logarithmic distribution, in which values are separated by equal ratios rather than by equal numbers of units. Equal ratios between adjacent available frequencies result in perception of equal intervals. In the current implementation this is done with o/p numbers (above, p. 83), which refer to frequencies in a distribution which is as close to logarithmic as the frequencies available from the sound synthesis subprogram will permit. By using extenders in combination with modifiers other distributions are possible (above, p. 167), but it would be desirable to be able to select from several commonly used distributions without being forced to use extenders and modifiers. Future implementations should include options for choice of at least three distributions: (1) linear, (2) logarithmic, and (3) sets of arbitrarily spaced values.

**Trial Values**

In the current implementation all specifications for variables must be entered during Phase Two. There is no means by which a trial value not in the tables can be tested. This has not been a problem, but during Phase
Three it would be convenient to be able to enter a value which was not derived from the tables. Then if the experimental value were found to be acceptable, it could be included within the item and group specifications.

**Numbering of Variables**

In the present implementation all numbers in the master score not within comments are considered by MASCH to be operands. For this reason item, group, and member numbers have not been included in the master score. This exclusion causes no difficulties in scores with few variables, but where there are many, it becomes difficult to determine the number of a particular item or member. In these cases comments can be added at appropriate points to indicate what item or member is next. Such comments have been included in SCR004 (above, p. 100). However, including comments is not as helpful as computer numbering of the variables would be. In future implementations a routine should be added so that the numbering of the variables is done by computer and indicated in the master score and in all versions (below, p. 205).

**Sets of Variables**

Each variable has its own set of terms which is used by MASCH to establish a table of possible values. It often happens that many items have the same or similar
terms. Similar terms occur when a set of items are all operands of the same kind of opcode (above, SCR004, p. 100). Repeated occurrences of the same or similar numbers is not a serious problem since the master score must be entered only once; and with a powerful editing program, identical strings can be entered with one command. However, from a conceptual point of view, it would be desirable to find some way to link terms or entire variables into a set which could be referred to with less effort and which would simplify both the encoding procedure and any changes which needed to be made later.

**Chaining Variables**

In the present implementation it is cumbersome to link variables in a chain fashion. Sometimes it is desirable to have variables arranged in ascending or descending order. Arrangement in ascending order can be done with groups, but with groups there is no control over the intervals between members. This kind of ordering cannot be done easily and simply with items. Given three variables it is not possible to arrange them such that

\[ B = A + b \]

\[ C = B + c \]

where \(A, B,\) and \(C\) are the values of three variables returned to the score, and \(b\) and \(c\) are operands of
modifiers contained within the second and third variables.

Instead, a set of variables can be related only by
referring all back to the value of one variable:

\[ B = A + b \]
\[ C = A + b + c \text{ or } \]
\[ C = A + x \text{ where } x = b + c \]

Below is an example of this kind of relation. \((10,100,5)\)
is the first variable and is considered Item 1.

\[
(10,100,5) \quad A \\
[(1 \text{ ADD } 20)] \quad B = A + b \\
[(1 \text{ ADD } 20 \text{ ADD } 40)] \quad C = A + b + c \text{ or } \\
[(1 \text{ ADD } 60)] \quad C = A + x
\]

A reference to the second variable cannot be made in the
third variable because an extender can only refer to the
value of an item or group; it cannot refer to the value
returned to the score by a complex variable, such as \(B\),
which is an extender plus a modifier. In future
implementations it should be possible to reference a value
of a complex variable directly without referring to its
parts. Perhaps this could be done by symbolic assignment:

\[(10,100,5)\]
\[
$2 = [(1 \text{ ADD } (10,50,5))] \\
[(2 \text{ ADD } (10,50,5))]
\]

Making items more flexible in a manner such as this might
make groups unnecessary.
Groups

If groups are used in future implementations it might be helpful to make provisions for avoiding generation of intervals between members which are excessively long or short. A composer might start a session with no such specifications in mind but become aware as the session progresses that intervals smaller than a certain minimum or larger than a certain maximum are always undesirable. Although a composer may do as he does now and simply reject groups which do not suit his needs, some way of adding optional terms to group specifications for eliminating certain intervals would lessen the number of unusable selections.

Means of Entering and Editing Master Scores

In this first implementation the decision was made to use a utility program belonging to the systems software of the PDP-11 to enter and edit master scores so that the time needed to develop a score entering and editing program specifically for CAEM would be saved. This decision has had a number of effects on the functioning of CAEM. The editing program makes possible the entering of long scores with minimum fatigue by means of its macro feature (above, p. 86); but using this program makes the entering and editing a separate step and thus dictates the
division of CAEM into Phases Two and Three. This division slows down the functioning of CAEM to a small degree because information sometimes has to be saved before returning to Phase Two, and because both MASCH and the editing program have to be loaded separately. In addition, there is no way to link together sections of a long composition except by means of this editing program. In future implementations an editing program should be written specifically for CAEM which would include the capability for linking together sections of a long master score or version. This program could reside in memory simultaneously with MASCH. If the editing program was too large for this, it could share memory space with a part of MASCH that was not needed at the same time. The computer system could then swap them in and out as needed. In this way the distinction between Phases Two and Three would be eliminated.

This proposed editing program could be designed to assign reference numbers to variables as they were entered and to print them in the score so that the composer would not have to keep track of the numbering himself. It could also have the capability of assigning to a set of variables a collective name so that operations could be performed on all variables in the set, such as changing descriptors, setting probabilities, fixing and freeing
(above, p. 182). Another potentially useful feature that could be incorporated into the editor would be the ability to shift the ranges of the variables by arithmetic operations if their ranges were too high, low, wide, or narrow.

**Format of Internal Tables**

Tables are generated by MASCH to store information obtained from scanning the variables. These tables contain information about which choices have been made and what their probability of selection is. The tables are designed to use as little memory as possible. It was felt that position alone was sufficient to indicate which tables pertained to which variables. The order in which variables occur in the score is matched by the order in which they occur in the tables. This system works well as long as the master score is not substantially changed (above, p. 148). Experience shows that most of the changes made in the master score are made early in the interactive procedure. Most of these are for correction of encoding errors. Changes made early in the interactive process—before any of the probabilities have been altered and before any information is saved—cause no problem. But information on altered probabilities which has been saved may be made unreadable if the number and/or arrangement of variables are changed by a return to Phase
Two. This information is made unreadable because the tables do not have any indication other than their relative position what variable they pertain to. If the number and/or arrangement of the variables is changed, the only thing the composer can do is re-enter all the changes in probabilities that have been made.

The table for each item and group needs a symbol so that it can be identified in spite of changes in the arrangement of variables. Assigning identification symbols would be easier if, as was suggested above (p. 205), the editing program was written especially for handling master scores and was made part of the same program with MASCH so that entering a score, changing a score, and exploring a score were all parts of one computer-controlled process.

**Visual Display of Data**

The computer on which this implementation was made has only one printer—the DECwriter terminal. While the DECwriter is perfectly capable of handling most composer/computer dialogue, it is not fast enough when the volume of information requested by the composer is high. Substantial master scores may take the greater part of a minute to list. This discourages the composer from requesting a listing of each version even when it would be helpful to do so. Waiting for the listing simply takes
too much time. A cathode-ray tube (CRT) terminal would be ideal for displaying versions, sequences, and other kinds of lists. Provision would have to be made, however, for unlimited scrolling up and down the listing, since the whole would probably not fit on the screen at the same time. A separate line printer or other hard copy device could be used to obtain permanent copies of data when required.

**MASCH Command Language**

Experience with the MASCH command language has brought several things to light. The specification of commands by words and combinations of words rather than by single letters or all single words has worked well. Single letters would have been too hard to remember although the number of commands might have made such a system possible. Single-word commands would have been easier to remember than single letters and easier to implement than multiple words, but in several instances similar procedures are done with more than one class of things; for example: versions, sequences, and probability tables are all listed, saved, and retrieved. It seemed easier during development, and it still does, to use the same words to describe the same procedure and to use other words to define the object of the procedure. Thus it seems easier for the composer to remember PRINT SCORE.
PRINT SEQUENCE, PRINT PROB, SAVE SCORE, SAVE SEQ, and SAVE PROB than to remember one word for each of these six commands.

The EVAL command (above, p. 180) was designed so that the composer would have to enter as little information as possible in order to grade particular choices or sequences. It is for this reason that it was thought better to let the command be simply EVAL rather than EVAL ITEM and EVAL SEQ and let the meaning of the command be determined by the current mode. However, sometimes this design has been more confusing than helpful. The composer may not realize that he has put the program into a SEQUENCE mode by a FORCE SEQ command. If he then tries to grade a choice, he will instead grade a sequence. The fact that the composer has entered two arguments rather than the one needed by EVAL in a SEQUENCE mode does not cause an error message. The second argument is simply ignored (above, p. 155). An error message is only given if there is a need for two arguments and only one is entered. EVAL should be changed to EVAL ITEM and EVAL SEQ. Perhaps EVAL could always default to EVAL ITEM.

Another option might be to count the number of arguments after the command. If one, use it to designate a sequence; if two, use the first as the item number and the second as the grade.
In the present implementation values of items are accessible in listings of versions, and choice numbers are accessible in listings of sequences. While a composer would normally be interested in a choice number if the choice number identified a value which he has already seen in a version, and which he could access through the corresponding sequence, he might find helpful a single command which listed all the values and corresponding choice numbers together.

**Sound Description and Production**

**Sound Generating Power**

One of the goals of CAEM is to provide immediate feedback concerning composer/computer choices. This goal has been achieved with this sound synthesis subprogram but at a high cost in sound producing power. When the project was originally conceived, it was intended for use with an indirect synthesis system. At that time it was thought very necessary to have computer control of patching (above, p. 20) if an indirect synthesis system were used. However, the amount of time required to design and build the interface and the cost of materials and sound producing/shaping hardware caused the author to decide instead on a direct synthesis system. Considerable time was spent in developing the sound synthesis subprogram,
and the versatility expected did not materialize. In retrospect, it would have been better to invest in even an inexpensive synthesizer and forego computer control of the patching. In that event more than two channels could have been controlled, and the frequency range could have been wider. In future implementations the high priority placed on immediate feedback and the need for a powerful sound generating subsystem dictate that some form of indirect synthesis be used—either control of an analog synthesizer via DAC or direct digital control of a digital synthesizer. The versatility of CAEM could possibly be enhanced by implementing the Chowning frequency-modulation algorithm (above, p. 17) or one similar to it. Using this simple technique, it might even be possible, as it is at M.I.T., to operate with a powerful direct synthesis system in an interactive environment. CAEM would be well suited for use with the Chowning algorithm since it would give composers a vehicle for experimenting with the algorithm's unconventional parameters (Moorer, 1977, 19f).

**Synchronization**

In the present implementation the two channels do not always stay synchronized. This is true especially when periodic frequency modulation is used and the maximum deviation from center frequency is larger than an octave. Even when the synchronization is good, it is sometimes
hard to follow the sound in the listing of the version since all of Channel 1 is listed before Channel 2 and because some of the events are so short. In future implementations it would be desirable to strive for better synchronization among the channels by a better algorithm and more precise arithmetic—in this implementation all calculations are done with sixteen-bit fixed-point "single-precision" numbers. The sound should also be synchronized with a display of the score. Perhaps the display could blink the lines of the version currently being played, or perhaps the score could be realized graphically in some way making it easier to follow. Events happening simultaneously on different channels should somehow be displayed close together. Perhaps a form of expanded musical notation could be used, in which the voices are shown in parallel as in a musical score.

**Playing Partial Versions**

Versions can be played only from the beginning. They can be stopped before normal termination at any time by setting the console switches (below, p. 269) to zero. This arrangement dictates to the composer that, if he wants to divide the score into temporal parts and work with one such part at a time, he must start at the beginning. This is not an unacceptable arrangement, but it takes away some flexibility. In future implementations
it would be desirable to have the capability of listening
to any portion of a version by itself. Perhaps this could
be done by specifying line numbers in the command, such as
"play from line 5 to line 10." Or perhaps it would be
better to specify a time and a channel, for instance,
"play Channel 1 from 1.00 to 2.00 seconds."

Power of Master Score Format

This master score format was sufficient for the kind
of sound synthesis subprogram which it controlled, but in
future implementations greater sophistication would be
desirable. There should be some easier means for defining
classes of sounds. New functions should be capable of
being entered under control of CAEM. At present adding a
function is a separate step (below, p. 249). There should
be some way of indicating that sections of a composition
are to be repeated either as is or after some kind of
permutation. Operands which do not change between events
should retain their value from the previous event.

Noise

One last consideration is the noise generated by the
computer's cooling fans and other moving parts. The
author worked in the same room with all the computer
equipment. This made it necessary to use headphones in
order to isolate the sounds produced by MASCH from the
rest of the sounds in the room. Something similar to the arrangement at M.I.T. would be preferred, where the computer's front panel with its various control switches and blinking lights is the only part of the computer which is seen or heard. The rest of the hardware is located in another room.

Summary

The purpose of this dissertation has been to present an interactive system for composition and sound synthesis (CAEM—Compositional Aid for Electronic Music) which provides a composer, by means of a computer, with a vehicle for exploring compositional alternatives.

The system was presented in the following way:

1. An overview of the present state of computer-based composition and sound synthesis was given in order to provide background and to show the need for the system (above, Chapter I).

2. A set of assumptions and conclusions flowing from the assumptions were presented. These have been guiding factors during system development (above, Chapter II).

3. Concepts related to the system were defined. The concepts are applicable to the system independent of the computer on which the system is implemented. The concepts presented included some pertaining to this particular system and others pertaining to computer-based systems in general (above, Chapter II).

4. The system was then presented as implemented on a specific computer. The presentation took the form of a series of examples in which the computer—
specific aspects of the system and the modes of composer/computer interaction were examined in detail (above, Chapter III).

5. The system in its current implementation was evaluated and suggestions were given for improving future implementations (above, Chapter V).

The objectives achieved by the system were the following:

1. An interactive system was developed to help the composer explore compositional alternatives embodied in a master score. The system helps the composer by relieving him of some of the burden of low-level decision making in order to allow him to spend more of his mental energy in evaluation.

2. The system is effective in allowing multiple versions to be generated rapidly, evaluated, compared, and combined.

3. Both composer and computer function in the areas of their greatest competence: the composer uses his creative and critical faculties; the computer functions as a sophisticated record-keeper and sound producer/controller.

4. A useful instruction set was developed for describing sounds.

5. An effective command language was developed for manipulating the versions generated during Phase Three.

In order for the system to reach its full potential, the following improvements need to be made:

1. Improvements need to be made in the Master Score Handler (HASCH):
   a. In items, multiple distributions of values should be available (above, p. 199).
   b. Provision should be made for inserting trial values (above, p. 200).
   c. Variables should be capable of being structured into sets (above, p. 201).
   d. It should be possible to reference variables
in a chain fashion (above, p. 202).

e. Groups should include optional descriptors for maximum and minimum intervals (above, p. 204).

2. Phases Two and Three need to be merged by incorporating into MÄSCH a program for creating and editing master scores. This program should have features to help it deal specifically with master scores, such as keeping track of the numbering of items and group members and doing the data management necessary when the order and/or arrangement of variables is changed (above, pp. 204-207).

3. The subsystems for audio and visual output need to be upgraded:

   a. The sound synthesis hardware and software need to be more versatile (above, pp. 210-213).

   b. The speed with which visual information is displayed needs to be increased (above, p. 207f).

   c. The format of the master score needs to be rethought in order to better exhibit parallel processes (above, pp. 212-213).

4. The site of the system needs to be free from excessive noise (above, p. 213f).

5. The system needs to be evaluated by many users. This could be accomplished both by questionnaire and by keeping thorough records of the composer/computer interaction. For this purpose what is needed is, not only a record of the sequences that were generated, but a record of the commands that were given by the composer as well as any abnormal system states that resulted from them, such as error messages or program malfunctions. With this kind of information the system described here can become a very effective compositional tool.
APPENDICES
Appendix A

GLOSSARY

ADC. See Analog-to-digital converter.

algorithm. "A set of rules or processes for solving a problem in a finite number of steps" (Graf, 1972, 21); a way of writing a computer program without necessarily committing it to a specific programming language.

alphabetic. Pertaining to the letters of the alphabet.

amplitude unit. A linear scale from 0 to 2047 used to specify amplitudes to the sound synthesis subprogram.

analog. Pertaining to measurable, continuously-varying electrical quantities such as voltage, amperage, and capacitance.

analog-to-digital converter. A device which transforms voltages into numbers varying in step with them.

angle brackets <>. Reserved delimiters for primary members.

applications programs. Those programs which are written by individual users for their particular needs.

argument. Part of a command string; a string of numeric characters yielding a number needed by the command to completely specify its procedure.

arithmetic operator. Part of a modifier which specifies what operation (addition, subtraction, multiplication, or division) is to be performed on the value of the variable by the operand of the modifier.

assembler. The program which does the translation from assembly language into machine language.

assembly. The process of translating an assembly-language program into machine language.

assembly language. Computer programming language in which
each expression corresponds to a machine language instruction and is, consequently, specific to a particular kind of computer.

au. See amplitude unit.

batch processing. A term used in electronic data processing to refer to a system in which the users' "jobs" are grouped in certain ways in order to make best use of available facilities. Users receive their output after their jobs have been processed. They have no direct access to the computer during program execution.

binary. Pertaining to a two-state system or to a numbering system which uses a base of 2.

binary program. See object program.

buffer. An area of memory to and from which data is transferred by means of input and output instructions.

CAEM. Acronym for Compositional Aid for Electronic Music, the system described in this dissertation. The term includes all three phases (Phase One, Phase Two, and Phase Three).

central processing unit. That part of a digital computer which reads and causes execution of instructions.

character buffer. See buffer.

character string. See string.

choice. One of the possible alternatives that an item can assume. Choices are numbered. An item with five choices has Choices 1, 2, 3, 4, and 5.

command. The opening word or words of a command string, composed of alphabetic characters, which specify the procedure to be performed.

command string. A string entered at the terminal by the composer during Phase Three to request MASCH to perform certain procedures.

compiler. A program which translates a procedure-oriented language into the machine language of a particular computer.
compositional concept. The image of a composition as it is first developed in the composer's mind—the general shape: the divisions and lengths of sections, frequency contours, kinds of envelopes, kinds of timbres, number of events per section, and all those considerations which precede exact quantitative note-level decisions.

concept. See compositional concept.

constant. An operand whose value is not changeable during Phase Three.

CPU. See central processing unit.

current version. The version whose internal representation is currently in memory.

cursor. A pointer used by the editing program to indicate its current place within the string.

DAC. See digital-to-analog converter.

DECtape. A small random-access storage device using tape; it is manufactured by the Digital Equipment Corporation.

delimiter. Part of an instruction. A string of one or more characters used to separate operands and parts of operands from one another as well as operands from opcodes. Delimiters are marks of punctuation, spaces, and the non-printing line-feed and carriage-return characters.

descriptors. Parts of a variable appearing in the master score which are enclosed within the variable's reserved delimiters and which define it.

digital-to-analog converter. A device which transforms numbers into voltages varying in step with them.

direct synthesis. The process by which a digital computer calculates the instantaneous amplitudes of the sound wave from information supplied by the user.

DT1. Abbreviation for DECtape drive number 1.

event. The smallest unit of a score which can describe a sound with respect to time, frequency, and
amplitude—a note.

**extender.** A variable which can assume the value of an item or member. It is used to extend the influence of an item or member. It has one term, which is the number of the item or member whose value is to be inserted in place of it. It is delimited by square brackets inside of which are placed parentheses or angle brackets depending on whether the term is the number of an item or a member.

**file.** A structure for storing information on random-access storage devices, such as DECTapes.

**fix.** To suspend the selection process for an item or a group so that in subsequent versions the item or group values remain unchanged.

**fixed-point arithmetic.** Computer arithmetic in which the location of the decimal point remains fixed. Its location must be managed by the programmer.

**FORTRAN.** A procedure-oriented language used for numerical applications. It is an acronym for FORMula TRANslator. It is similar in appearance to algebraic notation.

**free.** To allow an item or group to enter again the selection process so that in subsequent versions the item and group values will vary according to the current mode and probability of selection.

**grade.** See weight.

**group.** A set of a specific number of variables, known as members, whose values fall within a specific range and which are arranged in the master score in ascending order.

**group area.** Region of memory for storing information about groups.

**hardware.** Collective name for electronic equipment.

**herz.** A measurement of frequency corresponding to cycles per second.

**high-level language.** See procedure-oriented language.

**hz.** See herz.
indirect synthesis. The process by which a digital computer is used to generate control signals for sound generating equipment external to it.

information theory. A mathematical model which "deals with the likelihood of accurate transmission of messages subject to transmission failure, distortion, and noise" (Graf, 1972, 285).

instruction. (1) A computer word that is stored in memory and that is encoded in a certain way to be read by the CPU. It directs the CPU to perform a specific operation. (2) A unit of a score which describes an operation related to the production of sound.

instrument. A specification of a class of sounds obtained by symbolically patching together unit generators.

internal representation. The form into which MASC translates the current version so that it can be read by the sound synthesis subprogram.

interrupt. A signal generated by either hardware or software which tells the CPU to begin executing instructions at a new memory location. When the procedure related to the interrupt is completed, control is returned to the original program.

item. A variable having a specific number of possible values equally spaced within a specific range. Its reserved delimiters are parentheses which enclose three terms. The first two terms give the bounds of the range; the third, the number of equally spaced values available within the range.

item area Region of memory for storing information concerning items.

list. To print at the terminal.

load module. The final form of the binary program, prepared for being read into memory.

machine language. The binary instruction code recognized by a particular CPU.

macro. See macro-instruction.

macro-instruction. An instruction which causes a series
of instructions to be executed which have previously been given a macro designation.

Markoff chain. "The special case of a stochastic process in which the probabilities depend on the previous events" (Shannon and Weaver, 1949, 102).

Markoff process. See Markoff chain.

MASCH. See Master Score Handler.

Master Score Handler. The program running in Phase Three which allows the composer to explore the various possibilities inherent in the master score.

member. One of the components of a group.

memory. That part of a computer which stores both the program and the data which is processed during program execution.

minicomputer. A general purpose computer with small physical size and limited memory. Large central computers differ from minicomputers in that they have larger memories and additional hardware and software for time sharing and batch processing.

mnemonics. Short word-like symbols used in assembly language to represent machine operations, such as MOV (move) and CLR (clear). These can be remembered by the programmer much more easily than the corresponding binary machine language into which the assembler translates them.

mode of variation. One of four ways in which MASCH makes selections from available choices. Two pertain to choices of items: ITEM FREE and ITEM SERIES; two pertain to choices of sequences: SEQUENCE FREE and SEQUENCE SERIES.

modifier. A descriptor which alters the value of a variable after that value has been selected from the set of possible values as defined by its terms.

monitor. That part of the systems software which allows the user to communicate with the computer via the terminal and which directs system operations.

monitor-oriented structure. A mode of communication between user and computer in which the computer
accepts commands in many possible orders and issues a minimum of prompting messages; the user has a maximum amount of opportunity for intervention in the direction of program execution.

**note.** Information concerning fundamental frequency, amplitude, and time needed by a direct synthesis program to direct the playing of instruments.

**numeric.** Pertaining to the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, and -.

**object program.** The machine language program which results from the translation of an assembly language or procedure-oriented language by an assembler or compiler.

**octave/pitch number.** A method of specifying frequency from a scale with twenty-four nearly equally spaced pitches per octave.

**o/p number.** See octave/pitch number.

**opcode.** That part of an instruction which indicates the kind of operation to be specified.

**operand.** That part of an instruction which yields a quantity needed by the opcode in order to specify the operation.

**operation code.** See opcode.

**page.** An image of the sequence area which is stored on DT1.

**peripheral.** Device connected to a computer for purposes of receiving or transmitting data.

**Phase One.** The first thing a composer does in any compositional activity. The process by which the composer comes to decide on the general shape of the composition— all those considerations that precede specific quantitative decisions at the note level.

**Phase Three.** The exploration of the master score under control of MASCH. This program allows the composer to experiment with his concept through a process in which he evaluates versions of the master score that are presented to him and by which he directs their manner of presentation. By means of this procedure
the composer arrives at either of two goals: (1) the best possible version of a master score or (2) a master score and associated tables which are capable of generating a number of acceptable versions.

**Phase Two.** The translating of the composer's concept into a form that both he and the computer can manipulate. This phase consists of two activities: (1) making the translation to master score format and (2) entering the master score onto the computer's storage device.

**primary member.** The first member of a group to appear in a master score and the member that defines the characteristics of the group. It is delimited by angle brackets which enclose three terms. The first term indicates the number of members in the group. The next two indicate the range.

**probability table.** Part of the information stored in the item area for each item. It can be listed with the PRINT PROB command.

**procedure-oriented language.** Computer programming language in which expressions correspond to procedures and only indirectly to machine language instructions and are independent of any particular kind of computer.

**program.** The series of instructions decoded and executed by the CPU.

**random-access storage device.** A storage device which allows information to be accessed in any order.

**real time.** "Having to do with the actual time during which physical events take place" (Graf, 1972, 479), as opposed to the internal representation of time used by a computer for calculating parameters for events which it will control.

**regenerate.** To generate a version from a sequence which has been previously stored by using one of the SEQUENCE modes.

**sample.** A number representing the instantaneous amplitude of a sound wave.

**score.** A string organized into substrings, called instructions, which defines an ordered arrangement of sounds in a form readable by both composer and
computer.

**secondary member.** A member of a group other than the primary member. It is indicated in the master score by a series of asterisks terminated with a semicolon.

**seed.** An integer which establishes an initial state for the pseudo-random number generator. It can be specified with the RAND command. If no RAND command is given at the beginning of a session, a default seed is used.

**sequence.** A table which contains the choice numbers for items and the values for groups which were selected for a particular version. Sequences and versions have corresponding numbers; e.g., the choices for Version 1 are contained in Sequence 1.

**sequence area.** Region of memory for storing sequences.

**session.** An unbroken period of time spent at a terminal in communication with a computer.

**single-precision arithmetic.** Computer arithmetic in which values to be manipulated are stored in single computer words. Some computers allow increased precision by providing for manipulation of values stored in pairs of words. This is called double-precision arithmetic.

**software.** Programs which control computer operation.

**sound synthesis subprogram.** That part of MASCH which reads the internal representation of the current version prepared for it and uses that information to generate its audio representation.

**source program.** The program which is directly written by the programmer.

**square brackets [ ].** Reserved delimiters for extenders.

**stochastic process.** "A system which produces a sequence of symbols according to certain probabilities" (Shannon and Weaver, 1949, 102).

**string.** An ordered group of symbols taken from the set of characters available on typewriter keyboards, including the letters of the alphabet, the numbers, and the marks of punctuation.
**subprogram.** A segment of a program which performs one function.

**subroutine.** A subprogram which is 'called from more than one part of the program.

**swap.** To exchange a segment of memory with a memory image stored on a high-speed random-access storage device. This technique makes it possible for more than one program segment to use the same memory space. This feature is valuable on small systems with limited memory.

**systems software.** A set of programs which provide services of a general nature for all users. Systems software includes the monitor and the various utility programs.

**term.** Part of a variable that alone or in conjunction with other terms defines the set of possible values of the variable.

**time sharing.** A term used in electronic data processing to refer to the execution of more than one job by a computer simultaneously.

**tutorial structure.** A mode of communication between user and computer in which the computer prompts the user for most responses. Flexibility of direction by the user is reduced. The user is assumed to be unfamiliar with the system.

**unit generator.** A subroutine which simulates the behavior of a module of an electronic music synthesizer.

**utility programs.** A set of programs, part of the systems software, which performs frequently needed functions, such as passing data between peripherals and creating and editing source programs.

**value.** A quantity assigned to a choice.

**variable.** An operand whose value is changeable during Phase Three. It is indicated by reserved delimiters which enclose its descriptors.

**version.** A score similar to its parent master score except that each variable is represented by one value selected from the set of its possible values. A
listing of a version shows all descriptors of variables replaced by the values selected for them. A version, if error free, is playable. A master score is not.

weight. An integer with an absolute value less than 15 which is used in EVAL and SET commands to change the probability of selecting choices of items and of selecting sequences.
Appendix B

TECHNICAL DESCRIPTION OF PROGRAM

The following information has been included to fill in some of the technical details of MASCH that it was not appropriate to discuss in the body of the dissertation but that may be of interest or of value to those seeing an application of their own for some of the ideas presented here.

The method of allocating memory space and the design of the internal tables are discussed. A table of frequencies available to the sound synthesis subprogram is included. The significance of the various bits in the MASCH status word (WORD) is also presented. After these topics are taken up briefly, the rest of this appendix is devoted to the procedural description of the subroutines. This description enables the reader to follow the logic of the program without having to learn PDP-11 assembly language. The language used for the description is an approximation to COBOL (Common Business-Oriented Language), a language used extensively in business applications. COBOL is so similar in appearance to ordinary English that a COBOL program can be understood with little special training.
The approximation of COBOL used here is designed to give an enhanced logical structure to English prose. This language has the advantage over flow-diagrams that names do not have to fit into small boxes on the page. They can be as long as required to make their meaning clear. The conventions used in the language are the following:

1. Symbolic names used in the assembly language program and some special words, such as ITEM and FORCE, are in upper case letters.

2. The only descriptive names that are not indented are the paragraph names. A paragraph name that is not also a symbolic name in the assembly language program has its first letter capitalized.

3. Verbs changing the flow of execution can only reference paragraph names. These verbs are go to, perform, and call. Go to accomplishes a simple transfer of control. Perform requests that all the instructions in a paragraph or range of paragraphs are to be executed. After the perform is completed, control is passed to the instruction after the perform. Call is similar to perform but requests instead that an entire subroutine be executed.

4. Sentences beginning with If are executed only if the condition in the if-clause is true. An if-clause can be followed by an else-clause, which is executed only if the condition is false.

5. Arithmetic statements leave their result in the last data-name to appear in the statement; i.e., "Add X to Y" (result in Y); "Add X to Y giving Z" (X and Y are unaltered, result in Z).

6. Comments are enclosed in parentheses.
Memory Map

As can be seen from Example 44 (p. 232), the system software uses space at both the very top (high addresses) and very bottom (low addresses) of the available 24K words of memory. The user program has available the space within the two sets of double lines. MASCHE is loaded beginning at 1000 (octal)—the location of MON. When MASCHE is executed, one of the first things it does is request workspace up to 100000 (octal). When a master score is first scanned, tables are created for the variables beginning at the high end of the workspace and working down (note the direction of the arrows). After the space for items and groups has been reserved, sequences are stored, also from the top down. At the same time, MASCHE is writing the internal representation of the current version directly above the sound synthesis subprogram in an upward direction. During the process of version generation, several checks are made to insure that the sequence area does not encroach upon the internal representation area. If this does occur, the NO MORE ROOM message is sent to the composer.

Structure of Internal Tables

Example 45 (p. 233) contains details of the structure of the internal tables for items, groups, and sequences.
RT-11 System Software

- Item Area
- Group Area
- Sequence Area

Internal Representation of Current Version

Example 44. Map of memory usage.
Example 45. Details of structure of internal tables for (a) items, (b) groups, and (c) sequences.
In the case of items and groups, the delimiter words are used to locate the beginning of a table entry. The numbers in parentheses indicate the values to which the words are initialized. Entries in a sequence table are positive if they are item choice numbers and negative if they are member values. When a choice word in an item or a grade word in a sequence is graded, it is shifted left. This has the effect of doubling the value for each bit it is shifted. These same words are made negative when their choice or sequence is selected. This difference of sign indicates to the program which choices have been selected. The program must know this during SERIES modes.

**Table of Available Frequencies**

Example 46 (p. 235) contains a table of frequencies available to the sound synthesis subprogram as well as a listing of the FORTRAN program that produced it. The first column of the table contains the table-increment (INC); the second column, the interrupt-duration (c); and the third column, the frequency in herz (below, p. 258).
Example 46. Table of frequencies available to the sound synthesis subprogram.

<table>
<thead>
<tr>
<th>INC</th>
<th>C</th>
<th>Hertz</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>100</td>
<td>15.625</td>
</tr>
<tr>
<td>2</td>
<td>99</td>
<td>15.783</td>
</tr>
<tr>
<td>2</td>
<td>98</td>
<td>15.944</td>
</tr>
<tr>
<td>2</td>
<td>97</td>
<td>16.108</td>
</tr>
<tr>
<td>2</td>
<td>96</td>
<td>16.276</td>
</tr>
<tr>
<td>2</td>
<td>95</td>
<td>16.447</td>
</tr>
<tr>
<td>2</td>
<td>94</td>
<td>16.622</td>
</tr>
<tr>
<td>2</td>
<td>93</td>
<td>16.801</td>
</tr>
<tr>
<td>2</td>
<td>92</td>
<td>16.984</td>
</tr>
<tr>
<td>2</td>
<td>91</td>
<td>17.170</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>17.361</td>
</tr>
<tr>
<td>2</td>
<td>89</td>
<td>17.556</td>
</tr>
<tr>
<td>2</td>
<td>88</td>
<td>17.756</td>
</tr>
<tr>
<td>2</td>
<td>87</td>
<td>17.960</td>
</tr>
<tr>
<td>2</td>
<td>86</td>
<td>18.169</td>
</tr>
<tr>
<td>2</td>
<td>85</td>
<td>18.382</td>
</tr>
<tr>
<td>2</td>
<td>84</td>
<td>18.601</td>
</tr>
<tr>
<td>2</td>
<td>83</td>
<td>18.825</td>
</tr>
<tr>
<td>2</td>
<td>82</td>
<td>19.055</td>
</tr>
<tr>
<td>2</td>
<td>81</td>
<td>19.290</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>19.531</td>
</tr>
<tr>
<td>2</td>
<td>79</td>
<td>19.778</td>
</tr>
<tr>
<td>2</td>
<td>78</td>
<td>20.032</td>
</tr>
<tr>
<td>2</td>
<td>77</td>
<td>20.292</td>
</tr>
<tr>
<td>2</td>
<td>76</td>
<td>20.559</td>
</tr>
<tr>
<td>2</td>
<td>75</td>
<td>20.833</td>
</tr>
<tr>
<td>2</td>
<td>74</td>
<td>21.115</td>
</tr>
<tr>
<td>2</td>
<td>73</td>
<td>21.404</td>
</tr>
<tr>
<td>2</td>
<td>72</td>
<td>21.701</td>
</tr>
<tr>
<td>2</td>
<td>71</td>
<td>22.007</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
<td>22.321</td>
</tr>
<tr>
<td>2</td>
<td>69</td>
<td>22.645</td>
</tr>
<tr>
<td>2</td>
<td>68</td>
<td>22.978</td>
</tr>
<tr>
<td>2</td>
<td>67</td>
<td>23.321</td>
</tr>
<tr>
<td>2</td>
<td>66</td>
<td>23.674</td>
</tr>
<tr>
<td>2</td>
<td>65</td>
<td>24.038</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
<td>24.414</td>
</tr>
<tr>
<td>2</td>
<td>63</td>
<td>24.802</td>
</tr>
<tr>
<td>2</td>
<td>62</td>
<td>25.202</td>
</tr>
<tr>
<td>2</td>
<td>61</td>
<td>25.615</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>26.042</td>
</tr>
<tr>
<td>2</td>
<td>59</td>
<td>26.483</td>
</tr>
<tr>
<td>2</td>
<td>58</td>
<td>26.940</td>
</tr>
<tr>
<td>2</td>
<td>57</td>
<td>27.412</td>
</tr>
<tr>
<td>2</td>
<td>56</td>
<td>27.902</td>
</tr>
<tr>
<td>2</td>
<td>55</td>
<td>28.409</td>
</tr>
<tr>
<td>2</td>
<td>54</td>
<td>28.935</td>
</tr>
</tbody>
</table>
Example 46 (continued).

<table>
<thead>
<tr>
<th>INC</th>
<th>C</th>
<th>HERZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>53</td>
<td>29,481</td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>30,048</td>
</tr>
<tr>
<td>2</td>
<td>51</td>
<td>30,637</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>31,250</td>
</tr>
<tr>
<td>2</td>
<td>49</td>
<td>31,888</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
<td>32,552</td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td>33,245</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>33,967</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>34,722</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>35,511</td>
</tr>
<tr>
<td>2</td>
<td>43</td>
<td>36,337</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>37,202</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>38,110</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>39,063</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>40,064</td>
</tr>
<tr>
<td>2</td>
<td>38</td>
<td>41,118</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>42,230</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>43,403</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>44,643</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>45,956</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>47,348</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>48,828</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>50,403</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>52,083</td>
</tr>
<tr>
<td>2</td>
<td>29</td>
<td>53,879</td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td>55,804</td>
</tr>
<tr>
<td>2</td>
<td>27</td>
<td>57,870</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td>60,096</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>62,500</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>62,500</td>
</tr>
<tr>
<td>4</td>
<td>49</td>
<td>63,775</td>
</tr>
<tr>
<td>4</td>
<td>48</td>
<td>65,104</td>
</tr>
<tr>
<td>4</td>
<td>47</td>
<td>66,489</td>
</tr>
<tr>
<td>4</td>
<td>46</td>
<td>67,935</td>
</tr>
<tr>
<td>4</td>
<td>45</td>
<td>69,444</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>71,023</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
<td>72,674</td>
</tr>
<tr>
<td>4</td>
<td>42</td>
<td>74,405</td>
</tr>
<tr>
<td>4</td>
<td>41</td>
<td>76,219</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>78,125</td>
</tr>
<tr>
<td>4</td>
<td>39</td>
<td>80,128</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
<td>82,337</td>
</tr>
<tr>
<td>4</td>
<td>37</td>
<td>84,459</td>
</tr>
<tr>
<td>4</td>
<td>36</td>
<td>86,806</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>89,286</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>91,910</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>94,697</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>97,656</td>
</tr>
</tbody>
</table>
Example 46 (continued).

<table>
<thead>
<tr>
<th>INC</th>
<th>C</th>
<th>HERZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>31</td>
<td>100.806</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>104.167</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
<td>107.759</td>
</tr>
<tr>
<td>4</td>
<td>28</td>
<td>111.607</td>
</tr>
<tr>
<td>4</td>
<td>27</td>
<td>115.741</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
<td>120.192</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>125.000</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>125.000</td>
</tr>
<tr>
<td>8</td>
<td>49</td>
<td>127.551</td>
</tr>
<tr>
<td>8</td>
<td>48</td>
<td>130.208</td>
</tr>
<tr>
<td>8</td>
<td>47</td>
<td>132.979</td>
</tr>
<tr>
<td>8</td>
<td>46</td>
<td>135.870</td>
</tr>
<tr>
<td>8</td>
<td>45</td>
<td>138.889</td>
</tr>
<tr>
<td>8</td>
<td>44</td>
<td>142.045</td>
</tr>
<tr>
<td>8</td>
<td>43</td>
<td>145.349</td>
</tr>
<tr>
<td>8</td>
<td>42</td>
<td>148.310</td>
</tr>
<tr>
<td>8</td>
<td>41</td>
<td>152.439</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>156.250</td>
</tr>
<tr>
<td>8</td>
<td>39</td>
<td>160.256</td>
</tr>
<tr>
<td>8</td>
<td>38</td>
<td>164.474</td>
</tr>
<tr>
<td>8</td>
<td>37</td>
<td>168.919</td>
</tr>
<tr>
<td>8</td>
<td>36</td>
<td>173.611</td>
</tr>
<tr>
<td>8</td>
<td>35</td>
<td>178.571</td>
</tr>
<tr>
<td>8</td>
<td>34</td>
<td>183.824</td>
</tr>
<tr>
<td>8</td>
<td>33</td>
<td>189.394</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>195.313</td>
</tr>
<tr>
<td>8</td>
<td>31</td>
<td>201.613</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>208.333</td>
</tr>
<tr>
<td>8</td>
<td>29</td>
<td>215.517</td>
</tr>
<tr>
<td>8</td>
<td>28</td>
<td>223.214</td>
</tr>
<tr>
<td>8</td>
<td>27</td>
<td>231.481</td>
</tr>
<tr>
<td>8</td>
<td>26</td>
<td>240.385</td>
</tr>
<tr>
<td>8</td>
<td>25</td>
<td>250.000</td>
</tr>
<tr>
<td>16</td>
<td>50</td>
<td>250.000</td>
</tr>
<tr>
<td>16</td>
<td>49</td>
<td>255.102</td>
</tr>
<tr>
<td>16</td>
<td>48</td>
<td>260.417</td>
</tr>
<tr>
<td>16</td>
<td>47</td>
<td>265.957</td>
</tr>
<tr>
<td>16</td>
<td>46</td>
<td>271.739</td>
</tr>
<tr>
<td>16</td>
<td>45</td>
<td>277.779</td>
</tr>
<tr>
<td>16</td>
<td>44</td>
<td>284.091</td>
</tr>
<tr>
<td>16</td>
<td>43</td>
<td>290.698</td>
</tr>
<tr>
<td>16</td>
<td>42</td>
<td>297.619</td>
</tr>
<tr>
<td>16</td>
<td>41</td>
<td>304.878</td>
</tr>
<tr>
<td>16</td>
<td>40</td>
<td>312.500</td>
</tr>
<tr>
<td>16</td>
<td>39</td>
<td>320.513</td>
</tr>
</tbody>
</table>
Example 46 (continued).

<table>
<thead>
<tr>
<th>INC</th>
<th>C</th>
<th>HERZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>38</td>
<td>328.947</td>
</tr>
<tr>
<td>16</td>
<td>37</td>
<td>337.838</td>
</tr>
<tr>
<td>16</td>
<td>36</td>
<td>347.222</td>
</tr>
<tr>
<td>16</td>
<td>35</td>
<td>357.143</td>
</tr>
<tr>
<td>16</td>
<td>34</td>
<td>367.647</td>
</tr>
<tr>
<td>16</td>
<td>33</td>
<td>378.788</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>390.625</td>
</tr>
<tr>
<td>16</td>
<td>31</td>
<td>403.226</td>
</tr>
<tr>
<td>16</td>
<td>30</td>
<td>416.667</td>
</tr>
<tr>
<td>16</td>
<td>29</td>
<td>431.034</td>
</tr>
<tr>
<td>16</td>
<td>28</td>
<td>446.428</td>
</tr>
<tr>
<td>16</td>
<td>27</td>
<td>462.963</td>
</tr>
<tr>
<td>16</td>
<td>26</td>
<td>480.769</td>
</tr>
<tr>
<td>16</td>
<td>25</td>
<td>500.000</td>
</tr>
<tr>
<td>32</td>
<td>50</td>
<td>500.000</td>
</tr>
<tr>
<td>32</td>
<td>49</td>
<td>510.204</td>
</tr>
<tr>
<td>32</td>
<td>48</td>
<td>520.833</td>
</tr>
<tr>
<td>32</td>
<td>47</td>
<td>531.915</td>
</tr>
<tr>
<td>32</td>
<td>46</td>
<td>543.478</td>
</tr>
<tr>
<td>32</td>
<td>45</td>
<td>555.555</td>
</tr>
<tr>
<td>32</td>
<td>44</td>
<td>568.182</td>
</tr>
<tr>
<td>32</td>
<td>43</td>
<td>581.395</td>
</tr>
<tr>
<td>32</td>
<td>42</td>
<td>595.238</td>
</tr>
<tr>
<td>32</td>
<td>41</td>
<td>609.756</td>
</tr>
<tr>
<td>32</td>
<td>40</td>
<td>625.000</td>
</tr>
<tr>
<td>32</td>
<td>39</td>
<td>641.026</td>
</tr>
<tr>
<td>32</td>
<td>38</td>
<td>657.895</td>
</tr>
<tr>
<td>32</td>
<td>37</td>
<td>675.676</td>
</tr>
<tr>
<td>32</td>
<td>36</td>
<td>694.444</td>
</tr>
<tr>
<td>32</td>
<td>35</td>
<td>714.286</td>
</tr>
<tr>
<td>32</td>
<td>34</td>
<td>735.294</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>757.576</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>781.250</td>
</tr>
<tr>
<td>32</td>
<td>31</td>
<td>806.451</td>
</tr>
<tr>
<td>32</td>
<td>30</td>
<td>833.333</td>
</tr>
<tr>
<td>32</td>
<td>29</td>
<td>862.069</td>
</tr>
<tr>
<td>32</td>
<td>28</td>
<td>892.857</td>
</tr>
<tr>
<td>32</td>
<td>27</td>
<td>925.926</td>
</tr>
<tr>
<td>32</td>
<td>26</td>
<td>961.538</td>
</tr>
<tr>
<td>32</td>
<td>25</td>
<td>1000.000</td>
</tr>
<tr>
<td>64</td>
<td>50</td>
<td>1000.000</td>
</tr>
<tr>
<td>64</td>
<td>49</td>
<td>1020.408</td>
</tr>
<tr>
<td>64</td>
<td>48</td>
<td>1041.667</td>
</tr>
<tr>
<td>64</td>
<td>47</td>
<td>1063.830</td>
</tr>
<tr>
<td>64</td>
<td>46</td>
<td>1086.956</td>
</tr>
</tbody>
</table>
Example 46 (continued).

<table>
<thead>
<tr>
<th>INC</th>
<th>C</th>
<th>HERZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>45</td>
<td>1111.111</td>
</tr>
<tr>
<td>64</td>
<td>44</td>
<td>1136.364</td>
</tr>
<tr>
<td>64</td>
<td>43</td>
<td>1162.791</td>
</tr>
<tr>
<td>64</td>
<td>42</td>
<td>1190.476</td>
</tr>
<tr>
<td>64</td>
<td>41</td>
<td>1219.512</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>1250.000</td>
</tr>
<tr>
<td>64</td>
<td>39</td>
<td>1282.051</td>
</tr>
<tr>
<td>64</td>
<td>38</td>
<td>1315.789</td>
</tr>
<tr>
<td>64</td>
<td>37</td>
<td>1351.351</td>
</tr>
<tr>
<td>64</td>
<td>36</td>
<td>1388.889</td>
</tr>
<tr>
<td>64</td>
<td>35</td>
<td>1428.571</td>
</tr>
<tr>
<td>64</td>
<td>34</td>
<td>1470.588</td>
</tr>
<tr>
<td>64</td>
<td>33</td>
<td>1515.151</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>1562.500</td>
</tr>
<tr>
<td>64</td>
<td>31</td>
<td>1612.903</td>
</tr>
<tr>
<td>64</td>
<td>30</td>
<td>1666.667</td>
</tr>
<tr>
<td>64</td>
<td>29</td>
<td>1724.138</td>
</tr>
<tr>
<td>64</td>
<td>28</td>
<td>1785.714</td>
</tr>
<tr>
<td>64</td>
<td>27</td>
<td>1851.852</td>
</tr>
<tr>
<td>64</td>
<td>26</td>
<td>1923.077</td>
</tr>
<tr>
<td>64</td>
<td>25</td>
<td>2000.000</td>
</tr>
</tbody>
</table>

C HERZ: PROGRAM TO CALCULATE FREQUENCIES AVAILABLE
C TO SOUND SYNTHESIS SUBPROGRAM.
C
C FORMULA: F = 781.25 * I/IC
C WHERE F = FREQUENCY IN HERZ
C I = TABLE-INCREMEN'T (INC)
C IC = INTERRUPT-DURATION (C)
C
IC = 100
DO 100 IE = 1,6
   !2%IE
10   F = 781.25 * I/IC
   WRITE (6,20) I,IC,F
20   FORMAT (1H,F10.2,F7.5,F7.5,F11.5,1H,F9.3)
   IC = IC - 1
   IF (IC .GE. 61) GO TO 10
IC = 50
100  CONTINUE
   STOP
END
The *MASCH* Status Word: *WORD*

*WORD* is a status word used by various subroutines of *MASCH* to pass certain information. Below is a list of the significance of the bits:

Example 47. *MASCH* status word: *WORD*.

**Bit**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Group-bit; when set score contains groups.</td>
</tr>
<tr>
<td>1</td>
<td>Item-bit; when set, score contains items.</td>
</tr>
<tr>
<td>2</td>
<td>ITEM-SEQUENCE-BIT; 0 = Item mode; 1 = SEQUENCE mode.</td>
</tr>
<tr>
<td>3</td>
<td>FREE-SERIES-bit; 0 = a FREE mode; 1 = a SERIES mode.</td>
</tr>
<tr>
<td>4</td>
<td>TIME-double-read-bit; set when TIM is scanning.</td>
</tr>
<tr>
<td>5</td>
<td>SAVE-SCORE-bit; set when SAVE SCORE command decoded.</td>
</tr>
<tr>
<td>6</td>
<td>Set when SCRxxx.SEQ is open (see SAVE SEQ).</td>
</tr>
<tr>
<td>7</td>
<td>PRINT-SCORE-bit; set when PRINT SCORE is decoded.</td>
</tr>
<tr>
<td>8</td>
<td>not used</td>
</tr>
<tr>
<td>9</td>
<td>Bits 9 and 10 are read together;</td>
</tr>
<tr>
<td>10</td>
<td>1 = FORCE; 2 = HOLD; 3 = RELEASE</td>
</tr>
<tr>
<td>11</td>
<td>FORCE-TIME-bit or SAVE-UNSAVE-bit; 1 = SAVE; 0 = UNS.</td>
</tr>
<tr>
<td>12</td>
<td>When set, bit 11 is to be SAVE-UNSAVE-bit.</td>
</tr>
<tr>
<td>13</td>
<td>PRINT-bit; 1 = PRINT</td>
</tr>
<tr>
<td>14</td>
<td>not used</td>
</tr>
<tr>
<td>15</td>
<td>Play-enable-bit; 1 = score playable.</td>
</tr>
</tbody>
</table>

Bits 9 to 13 are set and cleared by *MON* when processing multiple-word command strings. See *MON* (below, p. 241) for more information.
Procedural Description of Subroutines

The procedural description of subroutines that is given below is written in the language discussed above (p. 229). The subroutines are grouped according to function into the following categories:

1. Subroutines for Control of Session
2. Subroutines for Version Generation
3. Subroutines for Version Manipulation
4. Subroutines of the Sound Synthesis Subprogram

The categories are not mutually exclusive. Some subroutines belong to more than one, such as NUMBER, which is used by other subroutines in the second and third categories. The order of subroutines parallels as much as possible the order of subroutines in the program listings (see Appendix C).

Subroutines for Control of Session

MON.

(When NASCH is first loaded, control is passed to this part of the program. This is the monitor routine. It accepts, decodes, and begins execution of instructions from the composer, which the composer types in at the terminal.)

Initialize.

Disable line-clock.

(This is the clock within the computer which keeps track of time of day. It is separate from the real-time clock used by DACNV. It is disabled here because it interferes with the sound synthesis subprogram.)

Clear Play-enable-bit of WORD (prohibits version from
being played. This will be set during version
generation if the version is playable).

Ready.
Print *[] (Signal to composer that MASCH is waiting
for a command).
Wait for command string.

Search-loop.
Move command string to character-buffer (CHBUF).
Pack first three letters of command-word into a
register.
Search command-list for a match.
If match, go to subroutine whose address is the next
word in the list,
Else write *??????* and go to Ready.
(Commands with multiple words use Search-loop more
than once. The first word is matched; the subroutine
sets some bits in WORD (above, p. 240) and returns to
Search-loop to process next command-word. When all
command-words are processed, WORD is read to determine
exact nature of command. A jump is then made to the
proper subroutine.)
If a command dealing with variables is requested and
if neither item-bit nor group-bit is set in WORD,
Write *?? ILL CMND ??* and go to Ready.

EXIT.
(Called by MON when EXIT command is decoded. Returns
control to computer system monitor.)
Close any open channels (making any new DEC-tape files
generated during this session permanent).
Print seed (NUM in RAND).
Save final value of all registers (debugging aid).
Return control to system monitor.

Subroutines for Version Generation

LOAD.
(Called by MON when LOAD command is decoded. Sets up
file name for score.)
Get score-number.
Call R50NUM to convert score-number to RAD50 format
(The identification used by the system to locate
the score files on DT1 is stored in four
contiguous words in memory called a device block.
The information in the device block is encoded in
a special format called RAD50 which allows three
characters to be packed into one 16-bit word. For
score files the first word in the device block is "DT1", the second is "SCR", the third is the score-number input by the LOAD command, which is converted to RAD50 format at this point. The fourth word is blank for all master scores. For any versions saved by the SAVE SCORE command, the fourth word will be three digits corresponding to the sequence number of the version.

Move converted-score-number to third word of device block.

If the next character input is a period,
   Get version-number,
   Call R50NUM to convert version-number,
   Move converted-version-number to fourth word of device block.

Go to LDSCR.

R50NUM.
   (Subroutine for converting a three-digit number to RAD50 format for inclusion in device block.)
   Move 1 to place.
   Perform Loop 3 times.
   Result is sum.
   RETurn.

Loop.
   Remove ASCII mask from digit.
   If unmasked-character is not numeric,
      Print "?RAD50?" and return to system monitor.
      Add RAD50 mask (36, octal).
      Multiply number by place giving product.
      Add product to sum.
      Multiply 50 (octal) times place.

LDSCR.
   (Jumped to from LOAD. Does initial processing of master score.)
   Request workspace extending to 100000 (octal) and move address of upper limit of workspace to HICORE.
   Point SCANNER to beginning of character-buffer.
   Close score-file if it is open.
   Lookup score file given in device block (DT1 SCR sss vvv).
   Clear WORD except for PRINT-SCORE-bit.
   Read score-file scanning for items.
   If score-file has items, set item-bit in WORD.
   Make internal table entry for each item consisting of:
      increment (distance between values)
      offset (low end of range)
FORCE-word (if zero, item is free to vary; if non-zero, the number contained in it is the choice number at which the item is fixed).

One word for each choice, each set to one delimiter-word, set to zero.

Read score-file, scanning for groups.
If score-file has groups, set group-bit in WORD.
Make internal table entry for each group consisting of:
delimiter-word, set to zero
FORCE-word set to zero
One word for each member, set to -1
Move zero to a final delimiter word.
Move proper address to boundary words (BGNTIM, BGNSEQ).
If item-bit or group-bit is set,
Print 'EVAL MODE' and go to Ready of MON.
Go to PRESCH (If a score has no variables, proceed immediately to version generation, since a score without variables can only be played).

PRESCH.
(Called by MON upon decoding of all mode selection commands. Also called by LDSCR for scores without variables. It initializes HASCH for the scanning of the master score for version generation.)
Initialize pointers used to create internal representation of version.
Point Scanner to beginning of character-buffer.
Initialize pointer for PRINT-SCORE-routine.
Read first block of score.
If ITEM-SEQUENCE-bit of WORD indicates ITEM mode,
Call SEQFND to get last-sequence-stored,
Initialize grade-word of new sequence to 1,
Go to Next-task.
(SEQUENCE mode)
If a FORCE SEQ is in effect,
Point to selected sequence,
Mark selected-sequence (make grade negative)
Go to Next-task.
Make random selection of sequence according to the probability of selection of each sequence.
If FREE-SERIES-bit of WORD indicates FREE,
Mark chosen sequence,
Go to Next-task.
(SERIES)
If sequence is previously chosen (grade is negative),
Search for sequence not previously chosen,
If all sequences previously chosen,
Select any sequence;
If sequence is not previously chosen or if a sequence
has been selected according to the above procedure, mark choice and go to Next-task.

Next-task.

Print 'SEQUENCE' number-of-selected-sequence.
Set Play-enable-bit.
Initialize pointers to variables.
Go to SCAN.

SCAN.

(Reads master score in order to generate internal representation of current version. Scans and decodes instructions and, in turn, calls subroutines to do the actual conversion to the format of the internal representation)

Loop-1.

If Scanner is outside bounds of character-buffer, 
Print 'READ PAST END OF SCORE'
Go to Ready of MON.
Pack first three letters of instruction-word into register.
Search instruction-list for match to contents of register.
If a match, call subroutine whose address is contained in the next word in instruction-list, Else, go to Error-routine.
If Scanner is beyond first block of character-buffer, 
If SAVE-SCORE-bit of WORD is set, perform SAVE-SCORE-routine;
If PRINT-SCORE-bit is set, print first block of character-buffer, reset Scanner, and read next block of score-file;
Go to Loop-1.

SAVE-SCORE-routine.

Examine first block of character-buffer, replacing all opening-parentheses, opening-angle-brackets, opening-square-brackets, and asterisks by spaces (This removes all indications of variables; at this point all descriptors have been replaced by their selected values; consequently, this block is now a version ready to be written onto DT1 and saved).

Write first block of character-buffer onto DT1.
If last block of score-file has just been written, 
Close file;
Clear SAVE-SCORE-bit.
Reset Scanner.
Return.

Error-routine.

Clear Play-enable-bit of WORD.
Print line of question-marks, one for each character in current line, counting backwards from point at which error was detected to the last line-feed-character.
Print current line (This will cause the end of the line of question-marks to point to the offending instruction).
Go to Loop-1 (continue scanning).

NUMBER.
(This subroutine is called for almost all conversions of numeric character strings to binary numbers. It is called both by command subroutines and by instruction subroutines. When called to return a value for an operand that is a variable, it accesses the internal tables in order to determine the value and then, not only passes the value to the calling subroutine, but also inserts the value into the character buffer in place of the descriptors that defined it.)
Move scanner past alphabetic characters.
If next character is an angle bracket,
If ITEK-SEQ-bit of WORD indicates ITEM mode,
Then find table containing information for this group;
If error, print 'TIME BND ERR' and go to Ready of WIN;
If group is fixed, find sequence designated in FORCE-word, move member values from sequence into corresponding words in group area, move value for primary member into current-sequence-record; go to Insert-group-value;
Else (free) read number-of-members, lower-bounds, upper-bounds (three numbers within angle brackets);
generate one random number for each member and store in group area for this group; sort numbers moved into group area in ascending order; move value of primary member into current-sequence-record; go to Insert-group-value;
Else (SEQUENCE mode), get value from selected sequence; go to Insert-group-value.
If next-character is an asterisk,
If ITEK-SEQ-bit indicates ITEM mode,
Then, increment pointer to group area, move value from group area into current-sequence-record, go to
Insert-group-value;
Else (SEQUENCE mode), get value from selected sequence, go to Insert-group-value;
If next-character is an opening-parenthesis, Get next item and retrieve increment and offset;
If ITEM-SEQ-bit indicates ITEM mode, Then, if FORCE-word indicates item is fixed, find and mark choice; move choice to current-sequence-record; go to Insert-item-value;
Else (item is free), make random selection of choice depending on probability of selection of each choice;
If FREE-SERIES-bit indicates FREE, move value to current-sequence-record;
Else (SERIES), if choice is not previously chosen, mark it; else, find first available unchosen choice; or if all choices have been chosen, make all choices again available and mark choice; move value to current sequence record; go to Insert-item-value;
Else (ITEM-SEQ-bit indicates SEQ mode), get value from selected sequence, go to Insert-item-value;
If next-character is opening-square-bracket, If ITEM-SEQ-bit indicates ITEM mode, Then, find current sequence and go to Square-bracket;
Else (SEQUENCE mode), call SEQFND with CHOISQ (contains sequence number) to find current sequence; Go to Square-bracket;
If next-character is alphabetic, Print "NO ARG" and go to Error-routine of SCAN.
If next-character is a line-feed, go to Finale.
If next-character is numeric, Call DECBIN to convert number to binary; Go to Finale.
Insert-group-value.
Call ARITH to check modification of value by modifiers.
Call BINDEC to convert returned value to numeric characters.
Call STRCH to insert characters into character-buffer. Go to Finale.
Insert-item-value.
Multiply value times increment giving value.
Add offset to value.
Call ARITH to check for modification of value by
Call BINDEC to convert returned value to numeric characters.
Call STORCH to insert characters into character-buffer, including a slash if the value is an o/p number.
Go to Finale.

Square-bracket.
If next-character-plus-1 is an opening-angle-bracket, get following number; find corresponding member; go to Insert-group-value;
If next-character-plus-1 is an opening-parenthesis, get following number, find corresponding item; retrieve increment and offset; go to Insert-item-value;
Print 'SQ BRCT ERR' and go to Error-routine of SCAN.

Finale.
If SEQUENCE-area is encroaching on area for internal representation of current version, Print 'NO MORE ROOM' and go to Ready of MON.
Return.

PIN.
(Called from SCAN when PINE instruction is decoded.)
Call NUMBER to get operand (time of last event on a channel).
Call STORE to move address of PINE (in sound synthesis subprogram) to both frequency and amplitude areas.

EXI.
(Called by SCAN when EXIT instruction is decoded.)
If PRINT-SCORE-bit is set, print first block of character-buffer (finish printing version).
Save contents of all registers (debugging aid).
Go to Ready of MON.

SECT.
(Called from SCAN when SECTION instruction is decoded.)
Call NUMBER to get past operand.
Return.
(The operand of SECTION is used only by the routine processing the previous TIME instruction; see TIM.)

COMMENT.
(Called by SCAN when COMMENT instruction is decoded.)
Advance scanner to first character after next line-feed character.
Return.

CHA.
(Called by SCAN when CHANNEL instruction is decoded.)
Call NUMBER to get operand.
If operand is 1,
Set up pointers to beginning of frequency and amplitude areas (of internal representation) for Channel 1.
Return.
If operand is 2,
Set up pointers to beginning of frequency and amplitude areas for Channel 2.
Return.
Go to Error-routine of SCAN (If this point is reached, the operand is neither 1 nor 2 and is therefore not allowed.).

FUN.
(Called by SCAN when FUNCTION instruction is decoded.)
Call NUMBER to get index.
Use index to select from entries in a table of addresses of functions (LIST).
Call STORE to store address of function in frequency area (of internal representation).

(At this time, the table of addresses of functions (LIST) contains 12 entries, 11 of which are empty. Only the first entry, corresponding to FUNCTION 0, contains a real address, the address of the sine function (TABLE). If a new function is to be added, it must correspond to the following requirements:

1. The table should contain 64 entries.
2. The function should be symmetrical.
3. The numbers should be 16-bit binary and signed.
4. The function should use the entire 16-bit range. (The entries in the function table are used as fractions between 1 and -1 which are multiplied times a maximum-amplitude.
Once a new table is created, it must be stored on DEC-tape as a PDP-11 MACRO source listing from which the assembler can generate an object file that can be linked by the linker to the rest of MASCH. The name of the function table must be a GLOBL. It can then be
referenced by the FUNCTION instruction according to the order of its entry in the table; i.e., the first entry is 0, the second entry is 1, and so on.

QT.
(Called by frequency subroutines involved in generating internal representation of current version; converts o/p numbers to hz.)
Call NUMBER to get first operand.
If character after first operand is not a slash,
return (not an o/p number).
Call NUMBER to get second operand (number after slash).
Compute frequency-in-herz = 2**o * 16.5 * p * g
where o = octave number (first operand)
        p = pitch number (second operand)
        g = 24th root of 2
Return.

FNDTM.
(Subroutine used to locate beginning of specified group table.)
Point pointer to beginning of group area.
Perform Find-group-proc n times, where n = group number.
Return.
Find-group-proc.
Find next group-delimiter-word.
If positive number encountered,
Print *POS # IN TIM*
Go to Ready of MON.
If in sequence area,
Set error condition and return.
If next-word is not positive or zero (FORCE-word),
Print 'NEG NUM IN TIM FORCE' Go to Ready of MON.

NXTTN.
(Subroutine used to locate next secondary member.)
Move group-area-pointer to next location.
If in sequence area, set error condition and return.
If positive number, print *POS # IN TIM* and go to Ready of MON.
If zero (delimiter of group), set error condition and return.
Return.
FRCTM.
(Called by MON when FORCE TIME command is decoded.)
Call NUMBER to get group-number.
Call FNDTM to locate group.
Call NUMBER to get sequence-number (whose corresponding group values are to be used).
Move sequence-number to FORCE-word-of-group.
Go to Ready of MON.

HLDTM.
(Called by MON when HOLD TIME command is decoded.)
Determine number-of-current-sequence using SEQFM.
If no argument, go to All.
Call NUMBER to get group-number.
Call FNDTM to locate group.
Move number-of-current-sequence to FORCE-word-of-group.
Return.

All.
Using FNDTM locate every group in master score and move number-of-current-sequence to FORCE-word-of-group of each group.
Return.

RELTN.
(Called by MON when RELEASE TIME command is decoded.)
Call NUMBER to get group-number.
If no argument, go to All.
Call FNDTM to locate group.
Clear FORCE-word-of-group.
Return.

All.
Using FNDTM locate every group in master score and clear FORCE-word-of-group for each group.
Return.

ARITH.
(Called by NUMBER during version generation to process modifications of values by modifiers.)
Save current position of Scanner.
Loop-1.
If next character is A,
   Call NUMBER to get operand,
   Add operand to value
   Go to Loop-1.
If next-character is S,
   Call NUMBER to get operand,
   Subtract operand from value,
Go to Loop-1.
If next-character is H,
Call NUMBER to get operand,
Multiply operand by value giving value,
Go to Loop-1.
If next-character is D,
Call NUMBER to get operand,
Divide operand into value,
Go to Loop-1.
If next-character is a closing parenthesis, closing angle bracket, or semicolon,
Restore Scanner to old position,
Return.
Go to Loop-1.

STORCH.
(Called by subroutines involved in version generation to insert values in character-buffer in place of descriptors which generated them. ASCII numeric characters are passed to the subroutine by the calling subroutine.)
Move characters from argument-list into character-buffer.
If final reserved delimiter of variable is encountered before all characters are moved,
Print 'NO ROOM IN CH BUF'
Go to Error-routine of SCAN.
Move blanks into remaining locations of variable.
If no final reserved delimiter is encountered within 20 characters,
Print 'NO DELIMITER' and go to Error-routine of SCAN.
Return.

STORE.
(Called by subroutines involved in version generation to write instructions into proper locations in area containing internal representation of current version. Arguments are passed to the subroutine from the calling subroutines. These arguments include the instructions to be written, the number of data words to be written, and an indication of which area is to be written into—the frequency area or the amplitude area.)
If score is about to be written into the sequence area,
Print 'NO MORE ROOM' and go to Error-routine of SCAN.
If data words to be written exceed space in current data block,
Move data-pointer to next data block.

If instruction to be written is a frequency instruction,

If frequency instruction to be moved into sequence area,
   Print 'NO MORE ROOM' and go to Error-routine of SCAN;
If frequency instruction to be written exceeds space in current frequency block, move JUMP instruction into end of current frequency block (tells sound synthesis subprogram the location of the next frequency instruction), move frequency pointer to next frequency block.
Move address of data to frequency block (data used by instruction).
Increment pointer and move instruction to frequency block.
Move data words to data block.
Return.
(If this point is reached, instruction is an amplitude instruction.)

If score is about to be written into the sequence area, Print 'NO MORE ROOM' and go to Error-routine of SCAN.
If amplitude instruction to be written exceeds space in current amplitude block, move JUMP instruction into end of current amplitude block
Move amplitude pointer to next amplitude block.
Move address of data to amplitude block.
Increment pointer and move instruction to amplitude block.
Move data words to data block.
Return.

TIM.
(Subroutine called by SCAN when TIME instruction is decoded.)
Call NUMBER to get start-time.

Search.
If next instruction is a TIME or FINE instruction,
   Call NUMBER to get end-time.
   Subtract start-time from end-time giving duration of current event.
   Return.
If next instruction is a SECTION instruction,
   Call NUMBER to get reset-to-zero-time,
   Subtract reset-to-zero-time from duration giving (new) start-time.
Go to Search.

DECBIN.

(Utility subroutine for converting decimal numeric character strings to binary representation.)
Move 0 to sum.
Move 1 to place.
Loop-1.
Unmask digit.
Move digit to product.
Multiply place times product.
Add product to sum.
Point to next digit.
If next digit is a minus sign,
   Multiply -1 times sum,
   Return.
If next digit is a plus sign, return.
Multiply 10 times place.
Go to Loop-1.

RAND.

(Subroutine for generating a random number within a specified range; range must be positive.)
Do random operation to determine oddness or evenness, adding either 0 or 1 to seed.
Increment seed (to eliminate multiplication by zero).
Multiply constant times seed giving result.
If result is negative, multiply result by -1.
Move result to seed.
Divide range into result.
The remainder is the random number.
Return.

BINDEC.

(Utility subroutine for converting a binary number to ASCII numeric characters.)
If argument is negative, move minus-sign to sign-location;
   Else, move space to sign-location.
Perform Loop 5 times.
Locate minus-sign or space directly before high order digit.
Return.
Loop.
Divide 10 into argument.
Mask remainder.
Store masked-remainder in next-digit-position.
RIS.
(Called by SCAN when RISE instruction is decoded.)
Call STORE to store CLRAX instruction (start envelope at 0 au).
Call NUMBER to get duration-of-rise.
Call NUMBER to get new-level.
Move 1 to amplitude-increment.
Move new-level to number-of-executions.
Go to Continue of PAL.

HOL.
(Called by SCAN when HOLD instruction is decoded.)
Call NUMBER to get duration.
Call STORE to store LVLIN instruction including amplitude-executions (equal to duration),
duration-between-calls-for-amplitude (equal to 1000), and level (equal to current-level).
Return.

DECAY.
(Called by SCAN when DECAY instruction is decoded.)
Call NUMBER to get duration.
Move -1 to amplitude-increment (direction is always negative).
Move current-level to amplitude-executions.
Go to Continue of PAL.

FAL.
(Called by SCAN when FALL instruction is decoded.)
Call NUMBER to get duration.
Call NUMBER to get new-level.
Move new-level to amplitude-executions.
Move 1 to amplitude-increment.
If new-level is less than current-level,
Move -1 to amplitude-increment.
Continue.
Multiply 1000 times duration giving new-duration.
Divide new-duration by amplitude-executions giving duration-between-calls-for-amplitude (CPA).
Loop-1.
If duration-between-calls-for-amplitude is less than 500,
Multiply 2 times
duration-between-calls-for-amplitude,
Multiply 2 times amplitude-increment,
Divide 2 into amplitude-executions,
If amplitude-executions is less than or equal to 1,
Move 1 to amplitude-executions,
Move new-level to amplitude-increment,
Multiply 1000 times duration giving
duration-between-calls-for-amplitude.
Go to Loop-1.
Call STORE to store ENVIN instruction, including
amplitude-executions, duration-between-calls
for-amplitude, and amplitude-increment.
Multiply amplitude-executions by amplitude-increment
giving current-level.
Subtract duration from duration-of-event.
Return.

CRE.
(Called by SCAN when CRESCENDO instruction is decoded.)
Perform FAL thru Loop-1.
Call STORE to store ENVIN instruction with
amplitude-executions equal to duration-of-event,
with duration-between-calls-for-amplitude equal to
1000, and with amplitude-increment equal to 0.
Return.

LEV.
(Called by SCAN when LEVEL instruction is decoded).
Call NUMBER to get level.
Call STORE to store LVLIN instruction in amplitude
area with amplitude-executions equal to
duration-of-event, with duration-between-
calls-for-amplitude equal to 1000, and with level.
Move level to current-level.
Return.

AMI.
(Called by SCAN when AMI instruction is decoded.)
Call NUMBER to get current-level.
Call NUMBER to get modulating-frequency.
Call NUMBER to get maximum-deviation.
Move 1 to AM-increment.
Multiply 2 times maximum-deviation.
Divide AM-increment into maximum-deviation giving
steps.
Multiply modulating-frequency by steps, giving result.
Loop.
If result is larger than 16 bits,
Add 1 to AM-increment,
Divide AM-increment into maximum-deviation giving
steps,
Multiply modulating-frequency by steps giving
result,
Go to Loop.

Divide result into 50,000 giving duration-between-calls-for-AM.

If duration-between-calls-for-AM is less than 100,
Add 1 to AM-increment,
Divide AM-increment into maximum-deviation giving steps.
Multiply modulating-frequency by steps, giving result,
Go to Loop.

Call STORE to store AMIN instruction with duration-between-calls-for-AM, steps, and AM-increment.
Divide 2 into maximum-deviation.
Subtract maximum-deviation from current-level (adjust level to accommodate AMIN instruction).

Return.

AMO.
(Called by SCAN when AMO instruction is decoded.)
Call STORE to store ABOUT instruction in frequency area.

Return.

ENVLP.
(Called by SCAN when ENVLP instruction is decoded).
Call NUMBER to get envelope-number.
Move envelope-number to high-limit.
Move pointer-to-amplitude-area to the location whose address is given in the envelope-vector-table, using envelope-number as an index into envelope-vector-table.

Return.

SYN.
(Called by SCAN when SYNCH instruction is decoded).
Call NUMBER to get envelope-number.
If envelope-number is greater than high-limit of ENVLP,
Print "ENVLP NOT ASSIGNED",
Go to Error-routine of SCAN.
Call STORE to store SYNCH instruction in frequency area, with address contained in the entry in the envelope vector table indexed by envelope-number.

Return.

FRE.
(Called by SCAN when FREQ instruction is decoded).
Call QT to get frequency.
If FMGO is set, call ALTFMI and return.

Call FRECON to convert frequency to table-increment and interrupt-duration.

Call STORE to store INFREQ instruction in frequency area with frequency-executions equal to current-duration, with duration-between-calls-for-frequency equal to 1000, with table-increment and with interrupt-duration.

Return.

FRECON.

(Called by frequency-processing subroutines to convert frequency in herz to table-increment (INC) and interrupt-duration (C). Table-increment is used to set the octave of the frequency. It is a power of 2. It is added to the address of the pointer to the function table before calculating each amplitude sample. Doubling the table-increment increases the frequency by 2. Dividing the table-increment by 2 divides the frequency by 2, and so on. Interrupt-duration (C1 and C2 in the assembly language program) is the value that is moved to the counter register of the real-time clock. The value in the counter register is decremented once every 10 microseconds. When the counter reaches zero, an interrupt is generated and control is given to the d-to-a conversion routine, DACNV. Interrupt-duration determines the precise frequency within the octave set by the table-increment. The interrupt-duration is inversely proportional to the frequency.)

Divide frequency into 50,000 giving interrupt-duration.

Move 64 to table-increment.

Loop.

If interrupt-duration is greater than maximum-interrupt-duration,

If table-increment is greater than 2,

Divide 2 into interrupt-duration,

Divide 2 into table-increment,

Go to Loop.

If interrupt-duration is less than or equal to zero,

Print 'BAD FREQ—FRECON',

Go to Error-routine of SCAN.

Return.

FM1.

(Called by SCAN when FM1N instruction is decoded.)

Call QT to get center-frequency.

Call QT to get modulating-frequency.
Call QT to get maximum-deviation.
ALTPM1. (Alternate entry to FM1 from FREQ when FMGO is set)
Move 1 to FM-increment.
Add maximum-deviation to center-frequency giving new-frequency.
Call FRECON with new-frequency to get increment-from and interrupt-duration-from.
Subtract maximum-deviation from center-frequency giving new-frequency.
Call FRECON with new-frequency to get increment-to and interrupt-duration-to.
Adjust increment-to and interrupt-duration-to by multiplying by 2 until increment-to is equal to increment-from.
If increment-to is greater than 64 and is not equal to increment-from,
Print "FM ERR" and go to Error-routine of SCAN.
FM-loop.
Determine the number of steps between frequency-to and frequency-from and move result to steps.
Multiply modulating-frequency by steps giving product.
If product is larger than 16 bits,
Add 1 to FM-increment and go to FM-loop.
Divide 50,000 by product giving duration-between-calls-for-FM.
If division-by-zero,
Clear FMGO (If either maximum-deviation or modulating-frequency is zero, there can be no FM. In that case, the FM1 instruction becomes a FREQ instruction).
Call STORE to store FMOUT instruction in frequency area,
Call STORE to store INFREQ instruction in frequency area with current-duration, 1000, increment-from, and interrupt-duration-from.
Return.
If duration-between-calls-for-FM is less than 100,
Add 1 to FM-increment and go to FM-loop. (100 is an arbitrary number chosen to provide a limit on the speed of FM. If the modulating frequency is too high, the increased amount of processing will not allow the sound synthesis subprogram to calculate values fast enough).
Set FMGO.
Call STORE to store FMIN instruction in frequency area with duration-between-calls-for-FM, steps, and FM-increment.
Call STORE to store INFREQ instruction (for center frequency) in frequency area with
current-duration, 1000, increment-from, and interrupt-duration-from.

Return.

**FMO.**

(Called by SCAN when PMOUT instruction is decoded.)

Call STORE to store PMOUT instruction in frequency area.

Clear FMGO.

Advance Scanner to next non-alphabetic character.

Return.

**GLI.**

(Called by SCAN when GLISS instruction is decoded.)

Call NUMBER to get duration.

Call NUMBER to get destination-frequency.

If destination-frequency is less than current-frequency,

Move -1 to frequency-increment,

Else, move 1 to frequency-increment.

If FMGO is set, go to FMGO-set.

Call FRECON with destination-frequency to get increment-to and interrupt-duration-to.

Call FRECON with current-frequency to get increment-from and interrupt-duration-from.

Try.

Determine the number of steps between increment-to/interrupt-duration-to and increment-from/interrupt-duration-from and move result to steps.

If steps is less than or equal to zero (no gliss)

Call STORE to store INFREQ instruction in frequency area with current-duration, 1000, increment-to, and interrupt-duration-to.

Move destination-frequency to current-frequency.

Return.

If duration is greater than or equal to current-duration, (gliss is longer than event)

Move current-duration to duration,

Subtract 1 from duration,

If duration is not greater than zero (no gliss and a small or non-existent event),

Call STORE to store INFREQ instruction in frequency area with current-duration, 1000, increment-to, and interrupt-duration-to.

Return.

Multiply 1000 times duration giving product.
Divide steps into product giving
duration-between-interrupts.
If duration-between-interrupts is larger than 16 bits,
Print "GLISS TOO SLOW"
Go to Error-routine of SCAN.
If duration-between-interrupts is less than 500,
If increment is less than zero, add -1 to
increment,
Else, add 1 to increment;
Go to Try.
Call STORE to store GLISIN instruction in frequency
area with steps, duration-between-interrupts, and
increment.
Subtract duration from event-duration giving rest.
Call STORE to store GLISIN instruction in frequency
area with rest, 1000, 0 (increment is equal to
zero; once destination-frequency is attained, hold
it for rest of event).

(At this point the GLISS instruction encoded is tested
by actually being executed. Resulting frequency is
converted to herz and stored in current-frequency).
Return.
FHGO-set.
Move destination-frequency to current-frequency.
Call ALTFMI.
Return.

Subroutines for Version Manipulation

EVALIT/SETPRB.
(EVALIT is called by MON when the EVAL command is
decoded and the mode is an ITEM mode; SETPRB is called
by MON when the SET command is decoded.)
Next.
Call NUMBER to get item-number.
If no argument, return.
Call FINDIT to find item.
If error, print 'FINDIT BND ERR' and return.
Search choice-table of item to determine lowest-grade.
Reduction.
If lowest-grade is greater than 1,
Divide all grades by 2,
Go to Reduction.
(In the choice-table of each item, each choice is
represented by one 16-bit word. When the table is
created, each word has a 1 in it. When a choice is
given a positive grade, the value of the table entry
is shifted left, or multiplied by 2, once for every unit in the grade. That is, if a choice is given a grade of 5, the value of the table entry for that choice is doubled 5 times. When a choice is given a negative grade, the table entries of all choices except the one graded are doubled n times, where n is the grade. This shifting can only be done 15 times since there are only 16 bits in each word.

Consequently, the program checks to see that the lowest grade in a table is 1 before it proceeds with any grading. If the lowest grade is not a 1, all grades are proportionally reduced until the lowest grade is a 1.

If current command is EVAL,
Call NXTFLI to find choice-number (get current choice from current sequence);
Else, (if current command is SETPRB) call NUMBER to get choice-number (the choice number is in the command string).
Call FINDIT to locate choice-number within item.
Call NUMBER to get grade.
If no argument, return.
If grade is greater than zero, perform Grading-process on choice-number grade times.
If grade is equal to zero, go to Next.
If grade is less than zero, perform Grading-process on all choices except choice-number grade times.

(When a random selection is made from among the available choices, the grades for each choice are added together, and a random number is generated in the range of the total. If there are many choices or if there are many high grades, the total may exceed the capacity of the 16-bit word. If this occurs, the results are unpredictable. Therefore it is necessary to test for the possible occurrence of this overflow-condition and to reduce the grades in order to avoid it.)

Overflow.
If the sum of all grades in the item is greater than \((2^{15}) - 1\), divide 2 into all grades except those whose value is 1;
Go to Overflow.

Grading-process.
if grade is 40000 (octal),
Print 'EVAL LIMIT'
Else, multiply 2 times grade.
FRCIT.
(Called by MON when FORCE ITEM command is decoded.)
Call NUMBER to get item-number.
If no argument, return.
Call FINDIT to get item.
If error, print error message and return.
Call NUMBER to get choice-number.
If no argument, print 'NO ARG' and return.
Move choice-number into FORCE-word of item.
Go to FRCIT (Get next item-number).

HLDIT.
(Called by MON when HOLD ITEM command is decoded.)
Find last sequence.
Call NUMBER to get item-number.
If no argument, perform Hold-routine thru Again once for every item; when done, return.
Hold-routine.
Call FINDIT with item-number to get item.
Call NXTELI to get choice-number from current sequence.
Move choice-number to FORCE-word of item.
Again.
Call NUMBER to get item-number.
If no argument, return.

NXTELI.
(Utility subroutine to find next item entry in a sequence; distinguishes item entries from member entries.)
Move sequence pointer to next entry.
If entry is less than zero, go to NXTELI. (all item entries are positive, while all member entries are negative.)
If out of sequence area,
Print 'NXTEL BND ERR' and return.
Return.

RELIT.
(Called by MON when RELEASE ITEM is decoded.)
Call NUMBER to get item-number.
If no argument, perform Release-routine thru Next once for every item.
Release-routine.
Call FINDIT to get item.
If error, print error message and return.
Clear FORCE-word of item.

Next.
Call NUMBER to get item-number.
If no argument, return.
Go to Release-routine.

FINDIT.
(Utility subroutine to locate item table in item area.)
Move HICORE to pointer (set pointer to top of work-space).
If item-argument is less than or equal to zero,
Print 'FINDIT BAD DATA' and return.
Call NXTIT item-argument times (Pointer should now be pointing to FORCE-word of selected item).
If choice-argument is zero, go to Clean-up.
Perform Find-choice choice-argument times.

Clean-up.
If pointer is in group area, set error condition and return.
If pointer is pointing at word equal to zero (probably an item boundary word),
Set error condition and return.
Return.

Find-choice.
Move pointer to next word.
If word is equal to zero, go to Clean-up.
If word is in group area, go to Clean-up.

NXTIT.
(Utility subroutine for locating next item in item area).
Move pointer to next word.
If word is not equal to zero, go to NXTIT. (A zero word indicates an item boundary).
Move pointer down three words (skip FORCE-word, increment, offset).
If pointer is into group area, set error condition and return.
Return.

EVALSQ.
(Called by MON when EVAL command is decoded in a sequence mode).
Using SEQPND locate all sequence grades.
Reduce.
If lowest sequence grade is greater than 1,
Divide 2 into all sequence grades;
If grade is less than 1, move 1 to grade;
Go to Reduce.
Call SEQPND to find current-sequence.
If error, print 'SEQPND BND ERR' and return.
Call NUMBER to get grade.
If no argument, print 'NO GRADE' and return.
If grade is greater than zero, perform Grading-process grade times.
If grade is less than zero, perform Grading-process grade times on all sequences except the current-sequence.
Return.

Grading-process.
If grade is greater than or equal to 40000 (octal),
Print 'EVAL LIMIT',
Else, multiply 2 times grade.

SEQFND.
(utility subroutine to locate a sequence in sequence area.)
If sequence-number-arg is zero,
Print 'BAD DATA TO SEQPND' and return.
Multiply sequence-number-arg times seq-length giving result.
Multiply 2 times sequence-number-arg.
Add sequence-number-arg to result.
Subtract result from BGNSEQ (upper sequence area boundary) giving sequence-location.
If sequence-location is outside of sequence area, set error condition and return.
If sequence-location contains zero,
Print '0 IN SEQ GRADE' and return.
Return.

PRCSEQ.
(called by mon when FORCE SEQ command is decoded.)
Call NUMBER to get sequence-number.
Move sequence-number to CHOISQ (location from which PRESCN picks up sequence-number).
Go to PRESCN (generate that version)

LAST.
(called by mon when LAST command is decoded.)
If ITEM-SEQUENCE-bit indicates ITEM mode,
Find last sequence generated,
Print 'SEQUENCE' sequence-number,
Print listing of sequence.

If ITEM-SEQUENCE-bit indicates SEQUENCE mode,
Print 'SEQUENCE' CHOISQ (contains number of
current sequence when program is in SEQUENCE
mode),
Call SEQFND to find sequence with CHOISQ.
Print listing of sequence.
Return.

RECORD.
Called by MON when PRINT SEQUENCE command is decoded.
Perform Print-all once for every sequence thus far
generated.
Return.

Print-all.
Call SEQFND to find sequence.
Print 'SEQUENCE' sequence-number.
Print listing of sequence.

SVPRB.
Called by MON when SAVE PROB command is decoded.
Set up device block and call ENTER to open file on DT1
(DT1 SCR sss PRB).
Call WRITE to write item area into file (area bounded
by HICORE and BGNTIM).
Close file.
Return.

UNSPRB.
Called by MON when UNSAVE PROB command is decoded.
Set up device block (DT1 SCR sss PRB).
Call LOOKUP to access file.
Call READW to read file into item area (area bounded
by HICORE and BGNTIM).
Close file.
Return.

SAVSCR.
Called by MON when SAVE SCORE command is decoded.
Call NUMBER to get sequence-number.
If no argument, print 'NO ARG' and return.
Move sequence-number to CHICQ.
Set SAVE-SCORE-bit.
Set up device block.
Call BINDEC and R50NUM to convert sequence-number to
RAD50 format suitable for inclusion in device
block.
Move sequence-number-in-RAD50-format to 4th word of
device block (DT1 SCR sss vvv).
Call ENTER to open file on DT1.
Go to PRESCN (which will check SAVE-SCORE-bit after
each block of the master score is read, and,
finding it set, will strip off variable indicators
and call WRITE to transfer the version to DT1.).

PROB.
(Called by MON when PRINT PROB command is decoded.)
Call NUMBER to get item-number.
If no argument, perform Print-routine for every item
in score.
Print-routine.
Call FINDIT with item-number to get item.
If error, print *FINDIT CHOICE ERR* and return.
Print *ITEM* item-number.
Print *CHOICE* choice-number
Compute prob = choice-grade / sum-of-grades
Print prob.
End.
Return.

LOOKUP.
ENTER.
READW.
WRITE.
(These four subroutines are adaptations of the system
macros of the same or similar names. See the DT-ll
System Reference Manual (Digital Equipment
Corporation, 1974) for more information.)

SAVESEQ/UNSEQ.
(Called by MON when SAVE SEQ and UNSAVE SEQ commands
are decoded. In the assembly language listing these
two names are different entry points into the same
subroutine.)
Set up device block with SEQ in fourth word
(DT1 SCR XXX SEQ).
If one sequence has not yet been generated,
Print '*'?? ILL CMD ??*' and return.
Find end of area holding internal representation of
version and call it score-limit.
Find number-of-words and number-of-blocks in sequence
area (area is bounded by score-limit and BGNSEQ).
Move number-of-blocks to block-inc.
If SAVSEQ was called,
  Call ENTER to open file,
  If error, print 'ENT ERR' and return;
Move boundary-of-sequence area (SEQ) to first word
  in sequence area (so that the boundary is
  stored with the file);
Call WRITE to write sequence area into file on DT1
  with (the arguments) score-limit, word-count,
  starting-block (initialized during LOADING
  procedure);
Add block-inc to starting-block (to tell WRITE
  where on DT1 to write next page of sequence
  area),
  Print 'ERASE?';
  Return.
If UNSSEQ was called,
  Call LOOKUP to locate file on DT1;
  If error, print 'LOOKUP ERR' and return;
  Call NUMBER to get page-number;
  If no argument, move zero to page-number;
  Compute starting-block = block-inc * page-number.
  Call READ to read file into sequence area with
  (arguments) score-limit, word-count,
  starting-block;
  Move contents of first word of sequence area to
  sequence-bound-pointer (SEQ);
  Return.

Subroutines of the Sound Synthesis Subprogram

PLAY.
(Called by MON when PLAY command is decoded.)
If play-enable-bit of WORD is clear,
  Print "?? ILL CMD ??" and go to Ready of MON.
  Go to INIT.

CSET.
(Called by MON when CSET command is decoded. Sets
minimum interrupt duration. For more information, see
below, p. 269.)
Call NUMBER to get minimum-interrupt-duration.
Move minimum-interrupt-duration to FMLESS, LESS,
FMINUS+4, and GMINUS+4 (locations in sound
synthesis subprogram).
Multiply 2 times minimum-interrupt-duration giving
maximum-interrupt-duration.
Move maximum-interrupt-duration to FRECON+20, MORE,
and GMORE (locations in sound synthesis
subprogram).

INIT.
(Initializes sound synthesis subprogram.)
Initialize all special storage locations.
Clear alternate register set.
Set DAC for random mode.
Move address of DACHV to clock-interrupt-vector.
Clear error-indicator.
Set real-time-clock to tick = 10 microseconds.
Put first value in clock-register.
Start real-time-clock.
Go to QMNGR.

QMNGR.
(The function of this routine is to "manage the
queue"; i.e., to feed instructions to the DAC via
DACHV, in the proper order. Instructions from the
current version load values into the register of the
real-time-clock. The values, called
interrupt-durations, are expressed in terms of
clock-ticks, where one clock-tick equals ten
microseconds. The clock decrements the
interrupt-duration in the clock-register once for each
clock-tick. When the clock counts down to zero, it
generates an interrupt which transfers control to
DACHV. DACHV then reads one instruction from the
queue and uses that information to give a new value to
either or both of the channels of the DAC and to reset
the clock with a new interrupt-duration. The queue is
fed instructions from QMNGR. QMNGR must determine the
precise timing and the order in which the two channels
of the DAC need to be updated.)
Initialize.

Next.
If g-pointer is at end of queue, reset to beginning of
queue.
Add 1 to g-count (g-count is decremented by DACHV).
If g-count is less than 4,
Print "ERROR 1" and go to FINE (DACHV has
overtaken QMNGR).
If switch-register is equal to zero, go to FINE
(switches on front of computer; setting them to
zero is one way to stop the playing of a score before it would normally terminate.

Wait.

If g-count is greater than 80, go to Wait (Values are being calculated faster than they are being used. Wait for g-count to drop below 80).

If interrupt-duration-1 (C1—interrupt-duration for channel 1) is less than interrupt-duration-2 (C2) by a value greater than LIMIT (a value, currently 3 which is used to determine whether interrupt-durations for one or both channels should be sent with one instruction. If two interrupt-durations are less than 3 units apart, they must be sent in one instruction due to the length of time required to generate an interrupt—latency),

Move the following instruction to queue:

interrupt-duration-1
ONE (address within DACNV to which control is passed)
amplitude-1 (amplitude-value for channel 1, passed to DAC).

Call GETI to get next instruction.

If interrupt-duration-2 is less than interrupt-duration-1 by a value greater than LIMIT, Move the following instruction to queue:

interrupt-duration-2
TWO
amplitude-2

Call GETII to get next instruction.

If interrupt-duration-1 is equal to interrupt-duration-2,

Move to queue the following instruction:

interrupt-duration-1
THREE
amplitude-1
amplitude-2

Call GETI to get next instruction for Channel 1, Call GETII to get next instruction for Channel 2.

If interrupt-duration-1 is less than interrupt-duration-2 by a value less than or equal to LIMIT,

Move the following instruction to queue:

interrupt-duration-1
FOUR
amplitude-1
variable-delay (separating the d-to-a conversion of two channels)
amplitude-2

Call GETI to get next instruction for Channel 1,
Call GETII to get next instruction for Channel 2.

If interrupt-duration-2 is less than interrupt-duration-1 by a value less than or equal to LIMIT,
Move the following instruction to queue:
  interrupt-duration-2
  FIVE
  amplitude-2
  variable-delay
  amplitude-1
Call GETI to get next instruction for Channel 1.
Call GETII to get next instruction for Channel 2.

GETI.

(GETII is exactly the same as GETI except that it is for the other channel. GETI and GETII calculate individual sample (amplitude) values from the information supplied by the time-line-interpreter routines—TLIA, TLIF, etc.—which access the internal representation of the current version.)

Change register sets (The sound synthesis subroutine uses both sets of general-purpose registers available on the PDP-11 in order to optimize the running time.)

Decrement frequency-samples (SAMP1—timer for frequency instructions).
If frequency-samples is less than or equal to zero,
  Call TLIF for next instruction.
Decrement amplitude-samples (SAMAI—timer for amplitude instructions).
If amplitude-samples is less than or equal to zero,
  Call TLIA (time line interpreter for amplitude)
  for next instruction.
If new-increment (NINC1) is not equal to increment
  (INC1—used for incrementing table-pointer through function table),
  Move new-increment to increment,
  Adjust table boundaries according to new increment.
If relocate (REL1) is not equal to new-relocate
  (NREL1) (i.e., if the function has been changed;
  relocate contains the address of the beginning of the function table),
  Point table-pointer to new function-table,
  Adjust table boundaries.
Add increment to table-pointer.
If maximum-amplitude-1 (AMAX1) is less than zero,
  Clear amplitude-1,
  Go to FMAX-check (If AMAX1 goes negative, it is due to one of the enveloping amplitude
instructions, DECAY or CRESC, going past 0 au.
Silence is expected, but if AMAX1 is allowed
to go negative, there will be a phase shift
rather than silence).
Multiply the table entry pointed to by table-pointer
by maximum-amplitude-1 giving amplitude-1.

FM-check.
If FMGO is set,
Decrement PM-samples (SAMPFM),
If PM-samples is less than or equal to zero,
Call FM for next instruction (part of time
line interpreter).
If AMGO is set,
Decrement AM-samples (SAMAHM),
If AM-samples is less than or equal to zero,
Call AM for next instruction.
If table-pointer is beyond end-of-table,
Move beginning-of-table to table-pointer.
Change register sets.
Return.

TLIF.
(Called from GETI and GETII for instructions which it
has decoded from internal representation of current
version.)
Decrement frequency-executions.
If frequency-executions is less than or equal to zero,
Fetch next instruction from frequency area of
internal representation (The instruction
consists of two consecutive words, the first
of which is the address of the data for the
instruction--this is put into a pointer--the
second of which is the address of a subroutine
within the time-line interpreter to which
control is transferred).
Else, go to procedure the address of which is in
reenter-for-frequency (REP).

(When TLIF is called from GETI or GETII, the address
of GETI or GETII is contained in register 0. TLIF
then uses the address in Register 0 in index-mode
instructions to refer to the contents of addresses
which cluster around GETI and GETII. In this way both
GETI and GETII can use the same TLIF, and yet the data
associated with each channel can be kept separate.)

INPFREQ.
(A reference to this subroutine is given by a FREQ
instruction.
Move FREQ to reenter-for-frequency
Move data from data area (of internal representation)
to destinations:
frequency-executions
duration-between-calls-for-frequency
new-increment
interrupt-duration

FREQ.
Divide interrupt-duration into
duration-between-calls-for-frequency giving
frequency-samples.
Return.

GLISIN.
(A reference to this subroutine is given by a GLISS
instruction.)
Move GLISS to reenter-for-frequency.
Move data from data area to destinations:
frequency-executions
duration-between-calls-for-frequency
frequency-increment (INCF)
If FMGO is set,
Move 2000 to upper-limit-for-interrupt-duration,
Else, move GMORE to
upper-limit-for-interrupt-duration.

GLISS.
Add frequency-increment to interrupt-duration.
Loop-1.
If frequency-increment is less than zero,
If interrupt-duration is less than
minimum-interrupt-duration,
If new-increment is 64, print 'ERROR 3' (Freq
too high) and go to PINE,
Multiply 2 times new-increment and
interrupt-duration,
Divide 4 into FM-increment,
Go to Loop-1.

Loop-2.
If frequency-increment is greater than or equal to
zero,
If increment-duration is greater than
maximum-interrupt-duration,
If new-increment is greater than or equal to
4, divide 2 into new-increment and
interrupt-duration,
Go to Loop-2;
Divide interrupt-duration into
duration-between-calls-for-frequency giving
frequency-samples.
Return.

FMIN.
(A reference to this subroutine is given by an FMIN instruction.)
Set FMGO.
Move data from data area to destinations:
  duration-between-calls-for-FM
  FM-count, FM-range (both have same value)
  FM-increment (INCPM)
Fetch next instruction.

FM.
(Called from GETI and GETII if FMGO is set)
Loop-1.
Add FM-increment to interrupt-duration.
If interrupt-duration is less than
  minimum-interrupt-duration,
  If new-increment is greater than or equal to 64,
    Print 'ERROR 3' (freq too high) and go to PINE,
    Multiply by 2 new-increment and interrupt-duration,
    Go to Loop-1.
  If interrupt-duration is greater than or equal to 2000,
    Divide by 2 increment-duration and
    new-increment.
  Go to Loop-1.
Decrement FM-count (number of frequency steps
  modulated in one direction).
If FM-count is less than or equal to zero,
  Move FM-range to FM-count,
  Multiply FM-count by -1 (reversing direction).
Divide interrupt-duration into
duration-between-calls-for-FM giving FM-samples.
Return.

PHOUT.
Clear FMGO.
Fetch next instruction.

TLIA.
(Called from GETI and GETII for instructions which it
has decoded from internal representation of version.)
Decrement amplitude-executions.
If amplitude-executions is less than or equal to zero,
  Fetch next instruction from amplitude area of
  internal representation (Very similar to
Else, go to subroutine the address of which is in reenter-for-amplitude (REA).

LVLIN.
(A reference to this subroutine is given by a LEVEL instruction.)
Move LEVEL to reenter-for-amplitude.
Move data from data area to destinations:
  amplitude-executions
  duration-between-calls-for-amplitude (CPA)
  maximum-amplitude

LEVEL.
Divide interrupt-duration into
duration-between-calls-for-amplitude giving
amplitude-samples.
Return.

ENVIN.
(A reference to this subroutine is given by one of the amplitude instructions: RISE, FALL, HOLD, DECAY, CRESCENDO.)
Move ENV to reenter-for-amplitude.
Move data from data area to destinations:
  amplitude-executions (EXA)
  duration-between-calls-for-amplitude
  amplitude-increment

ENV.
Add amplitude-increment to maximum-amplitude.
Divide interrupt-duration into
duration-between-calls-for-amplitude giving
amplitude-samples.
Return.

AMIN.
Set AMGO.
Move data from data area to destinations:
  duration-between-calls-for-AM
  AM-count, AM-range (same value)
  AM-increment (INCAM)
Fetch next instruction.

AM.
(Called from GETI and GETII if AMGO is set)
Add AM-increment to AMAXI
Decrement AM-count.
If AM-count is less than or equal to zero,
Move AM-range to AM-count,
Multiply AM-increment by \(-1\) (reverse direction).

Divide interrupt-duration into
duration-between-calls-for-AM giving AM-samples.
Return.

AMOUT.
Clear AMGO.
Fetch next instruction.

SYNCH.
(A reference to this subroutine is given by a SYNCH instruction.)
Move address in data word to
instruction-pointer-for-amplitude (C-IPA). (The
next time TLIA is called, instructions will be
fetched from this address).
Fetch next instruction.

FUNCT.
(A reference to this subroutine is given by a FUNCT
instruction.)
Move address in data word to new-relocate (NREL) (This
word now contains the starting address of the new
function table.)
Fetch next instruction.

JUMP.
(A reference to this subroutine is placed in the
internal representation by STORE in order to tell TLIA
or TLIF to begin fetching instructions from a new
block).
Move address of next instruction to
instruction-pointer.
Fetch next instruction.

CLRMAX.
(A reference to this subroutine is placed in the
internal representation by a RISE instruction in order
to clear AMAX1 before incrementing it—in order to
begin from 0 au).
Clear maximum-amplitude.
Fetch next instruction.
DACNV

(This subroutine does the actual digital-to-analog conversion. It is called by the interrupt generated by the real-time clock. When called, it reads an instruction from the queue. Instructions are placed there by QMNGR. The subroutine is composed of five smaller routines, only one of which is executed at each interrupt.

Decrement q-count. (q-count is checked by QMNGR to determine how fast DACNV is picking up instructions from the queue.)

If there has been a second interrupt before the first one was serviced,
Print 'ERROR 2' and go to FINE.

If next data word is ONE,
Select Channel 1,
Move next data word (amplitude-1) to conversion-buffer.

If next data word is TWO,
Select Channel 2,
Move next data word (amplitude-2) to conversion-buffer.

If next data word is THREE,
Select Channel 1,
Move next data word (amplitude-1) to conversion-buffer,
Select Channel 2,
Move next data word (amplitude-2) to conversion-buffer.

If next data word is FOUR,
Select Channel 1,
Move next data word (amplitude-1) to conversion-buffer,
Perform Delay-loop next-data-word times (variable delay),
Select Channel 2,
Move next data word (amplitude-2) to conversion-buffer.

If next data word is FIVE,
Select Channel 2,
Move next data word (amplitude-2) to conversion-buffer,
Perform Delay-loop next-data-word times,
Select Channel 1,
Move next data word (amplitude-1) to conversion-buffer.

If pointer is greater than or equal to end-queue (ENDQ),
Move begin-queue (BEGINQ) to pointer (reset pointer).
Move next data word to clock-register. (Put new interrupt-duration into clock. This word was actually written by QMNGR as the first word of the instruction after the instruction containing the previous data words. The next time the clock counts down to zero, it will be reset to this value.)

Return from interrupt.

Delay-loop.

Go to Delay-loop.

FINE.

(Performs clean-up operations when sound synthesis subroutine is finished playing a score.)

Disable real-time-clock.

Save contents of all registers (for debugging).

If error condition exists, print error message.

Go to MON.
Appendix C

PROGRAM LISTINGS
TITLE NDN S.127/S 84 R.C. SMITH
GLOBAL NDN, DBL, INIT, PRESCN, CLEAR, EVALIT, NRD, CHRT, OUT
GLOBAL LGSM, EVRSE, REG, FRCT, FGROE, FRCTN, GD, RND
GLOBAL HLDIT, HLDTH, RELIT, RELTH, CHROG, SYMT, SHYSER
GLOBAL SYMCH, UNBGR, UNSEF, PNOF, RECROF, LNT, RGNM, STACK
GLOBAL NUMER, SRCFL, SENG, SFX, SFRFND, TCHDG, TCHRL, SFRBR
GLOBAL PARE, SHESS, SITRES, REG, ERCT, EROGER, FRTK, SW
GLOBAL MEMR, KLDTM, RETIT, RETBO, CMCISG, SWF, GMEVSEG
GLOBAL CMVCE, CMOREB, OEMGER, CNET, RECORD, EBO, EOMOM, ETOB, ERO
GLOBAL ROOD'ER, OETRRO
GLOBAL EOEOO, EOEO, EMIMO, CMIMO, EREC, MORE, GREUS, CMORE
GLOBAL TTVIM, CMEOOE, VEM, VEM

CD: CLR #177754G /DISABLE LINE CLOCK
BIC #1000D0, RND /DISABLE PLAY BIT
MON: NOV SP, CP
STACK: D
MOV #READY, R0
ENT 351 /PRINT [ ]
MN1: NOV #3, R3
CLR R4
LP1: ENT 348
BGS -2
BIC #177700, RO
AGH #5, R4
B15 RO, R4
SB R3, LP1
NOV #127, R2 /BEGIN SEARCH FOR MATCH
LP2: CHP (R2), R4
BR DSPCH
IST (R2)
BLR ER
ADD #4, R2
BR LP2
DSPCH: JMN #2(R2) /EXECUTE COMMAND
ER: NOV #GM, R6
ENT 351
JMF PC, CLEAR
BR MON
CLEAR: NOV #CHOUT+224, SC
CLR1: ENT 348
BGS -2
NOYO RO, (SC)+
CHN RO, #12
BNE CLR1
NOV #CHOUT+224, SC
FRS PC
EXIT: JSM PC, CLEAR
CLOSE #4 /USER CHANNEL
CLOSE #1 /USER CHANNEL
NOV NUM, R6 /PRINT FINAL RND NUM
MSG: 550 /TROUBLE?

END 60
ITEM: 0

ITEM NUMBER: USED IN NUMBER.

DIS \*100000.00=0, 000000.00 \* GET PLAY ENABLE BIT

JMP Scn

PR10: MOV C01050, R0 \* 38

DNC PR12

CLR R2
CLR R0

PR15: INCK R0

JSR PC: SERFD

DVC PR3

DST \*4

NEG R1

ADD R1, R2

PR16: MOV R2, (C3)

JSR PC: KROM

INC (C3)

CLR R0
CLR R0

PR17: INC R0

JSR PC: SERFD

DVC \*4

MUL

MOV \*(R3), R1

DST \*4

NEG R1

ADD R1, R2

CMP R2, \*(SP)

BLY PR17

ADD R2, \*(SP)

DIT \*(R3), R0 \* FREE OR SERIES

BLY PR10

TST \*(R3)

BLY PR19

PR18: NEG \*(R3) \* MARK 'CHOICE'

MOV R0, \*(R3), R0 \* FREE

JSR PC: OUT

MOV R3, SERMOD

BR PR11

PR19: INC \* CONTINUE SEARCH (SERIES)

JSR PC: SERFD

DVC PR20

TST \*(R3)

BLY PR19

BR PR10

PR20: CLR R0

PR21: INC R0

JSR PC: SERFD

DVC PR22

TST \*(R3)

BLY PR21

BR PR19

PR22: CLR R0

PR23: INC R0

JSR PC: SERFD

DVC PR40

NEG \*(R3)

BR PR23

PR12: JSR PC: SERFD

DVC \*4

MUL

BR PR19

READ: TST \*(PC)

DCLK: 0 \* BLOCK COUNT
ERRNO:

SCAN:
CLR R4
MOY #3, CNT
JCN PC, GETLET

LD1:
CHF 5C, (PC)+

DNR:
BEQ UNDERR
MOV (C0), R2
DIC #177760, R2
ASH R5, R4
DISC R2, R4
DEC (PC)+

CNT:
DGT LP1
MOY 5C, (PC)+


LP3:
CHF (CR2), R4
VER DSPCH
TST (R2)
ERR
ADD #4, R2
DR LP2

DSPCH:
ADD R2, R2
JCN PC, CR2
CHF 5C, #CHBUF+812
DIC 100000, NOED
DR LP3

ERR:
DIC #100000, NOED

DIR CLEAR NOT PLAYABLE

MOY #15, R6
ENT 341

SKIP LINE

MGK 2

LGK R5, R6
ENT 341


MGK 2

MOY 5C, (SP)

LGK SAVE 5C

LP5:
MOY #77, R8

THRESH ERROR WITH ?????????

CRF 341

DCS 2

CHF (CC), #12

PRINT UP TO BEGINNING OF LINE

DNC LP3

INC 5C

MOY #15, R6

CRF 341


MGK 2

MOY #12, R6

ENT 341


MGK 2

MOY (CC), R8

PRINT LINE WITH ERROR

ENT 341

DCS 2

CHF R6, #12

DNC LP4

MOY (SP)+, 5C

PUT SCANNER WHERE IT HAS AN ERROR
ASGN4: MOV -(R4), R1
DEC .R4
NEG R1
ADD R1, R8
TST (R4)
BLE ASGN4
MOV R8, -(SP)
JSR PC, RAND
MOV (SP)+, R2

ASGN31: MOV STR, R4
CLR R3
MOV # 1, R8
ASGN5: MOV -(R4), R1
DEC .R4
ADD R1, R8
INC R3
CMP R8, R6
DLT ASGN5
BIT #16, RORD
FREE OR SERIES?
DER ASGN30
TST ASGN30
DEC (R4)
DEC .R6
JMP ASGN46
ASGN30: SUB #0, R6
MOV R3, R5
ASGN50: DEC R5
MOV INC.R3, CAL CNUM
MOV OPOET, R5
JMP PC, ARITH
CHECK FOR ARITH MANIPULATION
ASGN21: CMPD (R8)+, @PAR
DER ASGN20
CMPD -(R8), #SLASH
BER ASGN21
DR ASGN21
ASGN20: MOV R3, -(SP)
JSR PC, DIVDEC
MOV #PARC.R8
JSR PC, STREH
MOV R5, R8
JMP R5PC
ASGN41: MOV (PC)+, R3
SRMHD: TST -(R3)
BLL SHER
CMP SELMHD.SER
BLO SHDR
MOV R3, SERMHD
UPDATE SRMHD
MOV (R3)+, R3
SHER: .PRINT $9SMHS
JMP NON
SHDR: .PRINT $9SMHS
JMP NON
SA1: BIT #4, RORD
BNE SRCB
CLR R0
LOOK FOR PRESENT SEQUENCE
JCR PC, SRCND
JR SRC SRCB
PROCEED WHEN LOWER THAN NEW
DEC R0
JCR PC, SRCND
INDROY AFTER OVERSHOOT
5821:  CMPD  (SC)+ #ANGLOR
DEB  583
CHPD -1<SC>, #PRNTNS
DEC  584
PRINT #680RMS
JMP  ERR
JMP  584

583:  JSR  PC, NUMBER  ; (<>), TIME; GET TIME NUMBER

584:  JSR  PC, NTTLET
DEC  46

587:  JND  NDER
SDD  RS, SDD
DBR  #500, WROD  ; CLEAR DOUBLE READ INDICATOR
MOV  R3, R4  ; MAKE R4 THE POINTER

589:  CMPD  -<SC>, #588R  ; MOVE SC BACK TO OVERPRINT
DNE  589
JMP  588

588:  JSR  PC, NUMBER  ; ITEM
MOV  RO, -<SP>

589:  JSR  PC, NTTLET
DEC  587
SDD  RO, SDD
CLR  -<SP>
JSR  PC, FINDIT
MOV  4<RS>, INC
MOV  2<RS>, #5LT
MOV  <R3>, R3

590:  CMPD  -<SC>, #588R
DNE  588
INC  SC

591:  MOV  CHOILD, RO
JSR  PC, SERIND
BR  592

592:  MOV  ENGINED, RO
JSR  394

594:  JMP  ERR
BR  591

595:  CLR  R2
MOV  #24, R2
MOV  RO, -<SP>
JSR  PC, DINDIC
ADD  #10, SP
MOV  <GP>, <SC>+1
MOV  #SASH, <SC>+1
MOV  RS, -<SP>
JSR  PC, DINDIC
MOV  #PARCL, RB
JSR  PC, STRCH

598:  CMPD  -<SC>, #SASH
DNE  598
MOV  RO, R3

599:  JMP  R57NC

580:  INC  R3
TST  -<RS>
BCR  599
BLT  598
JMP  599

583:  CLR  R3
MOV  #57NC, R3
TST  -<RS>
BCR  599
BLT  598
JMP  599
TITLE SETTHN 19/20/73 24 R.E. SALLFIELD
.
GLOBL TDF; TDFER; FINDH; NHTTH; MODR; RSCNF; NORD
GLOBL NGSNS; D9THM; D9SNS; MOD; NOH3; TROTH; RHTTH
GLOBL THTNH; NUMBER; HAND; SEG; SEPRNG; NATELT; NDLER
.
GLOBL TDF; TDFER; FINDH; D9THM; D9SNS; NORT; TROTH; RHTTH
.
GLOBL NRTTH
.
HEAL; PRTNT; REGDEF
.
SCHCL=73
.
DEC=95
.
R=104
.
S=123
.
H=115
.
D=104
.
FORCL=51
.
ANGCL=76
.
REGDEF
.
RFTH1: DBT .H4; HORD "ITEM OR SEQUENCE"
.
DER .+G
.
JMP NGSNS
.
INC (PC)+
.
TIMNO: 0
.
MOV TIMNO, RD
.
JCR PC; FINDH
.
DVC .+G
.
AGS00: JMP TDFER
.
TDF (R4) +FORCE
.
DER .+G
.
JMP NGSNS
.
MOV SC; (PC)+
.
SAVE2: 0
.
75B BEFORE NUMBER IS CALLED
.
JCR PC; NUMBER
.
MOV RD; (PC)+
.
INTHUM: 0
.
JCR PC; NHTTH
.
INC RD
.
MOV RD; (R4)
.
NEG (R4)
.
MOV RD; (PC)+
.
FR: 0
.
BIT #29; HORD
.
DNC AGS00
.
JCR PC; NUMBER
.
INC RD
.
MOV RD; (PC)+
.
TB: 0
.
MOV RD; R2
.
MOV R4; (PC)+
.
SAVE1: 0
.
SUB FR; R2
.
MOV INTHUM; R1
.
SUB R2; R1
.
SET4: MOV R2; -168
.
JCR PC; HAND
.
ADD FR; (SP)
.
JCR PC; NHTTH
.
DCC .+G
.
AGS004: JMP TO
.
MOV (SP)+; (R4)
.
NEG (R4)
.
SEC R1; SET4
.
JCR PC; NHTTH
.
DSE AGS00
.
MOV TO; (R4)
ADD3: JSR PC, NUMBER
  ADD RD, RS

SUB3: JSR PC, NUMBER
  SUB RD, RS

MUL3: JSR PC, NUMBER
  MUL RS, RS

DIV3: JSR PC, NUMBER
  DIV RS, RS
  ROL R2
  CMP RS, R3
  BGT .+4
  INC R2
  MOV R2, R3

PROC: MOV R2, SC

RTS PC

FTBNG: .ASC12 "TIME END ERR"
TDNG: .ASC12 "TIME DELIMIT"
MNTH: .ASC12 "POS # IN TIM"
MNGS: .ASC12 "REG # IN TIME FORC"

ARG00: .ASC12 "NO ARG0"

END
TT:=DECCIN, MAC

.TITLE DECCIN 5/27/76 1976 R.E. CARNFELD
.GLOBAL DECCIN, NARG, pointer, NUM
.HGLOBAL .RESULT

POINT=41
SUM=40
PROC=43
PLACE=45
TAB=40
MINUS=55

.REDEL

SAVE: .WORD

BEGIN: MOV   POINT,-(SP)   !FOR CONTENTS OF POINT
       MOV   SUM,-(SP)     !SAVE REGISTERS
       MOV   PROC,-(SP)
       MOV   PLACE,-(SP)
       MOV   #0, SUM
       MOV   #0, POINT
       ADD   #18, POINT
       MOV   #1, PLACE
       LOOP: MOV   (PLACE), PROC
             MOV   SUM, PROC
             DLT   ERROR
             CMP   PROC, #2
             DHT   ERROR
             HLT   ?PLACE, PROC
             ADD   PROC, SUM
             ADD   #2, POINT
             CMP   (POINT), #55
             CMP   (POINT), #55
             CMP   (POINT), #55
             CMP   (POINT), #55
             CMP   (POINT), #55
             CMP   (POINT), #55
             CMP   (POINT), #55
             CMP   #18, PLACE
             BR   LOOP
             MIN: MOV   SUM
             PLUS: MOV   SUM, (POINT)
             MOV   #2, POINT
             MOV   (SP), (POINT)
             MOV   POINT, SUM
             MOV   (SP), (POINT)
             MOV   (SP), (POINT)
             MOV   (SP), (POINT)
             MOV   (SP), (POINT)
             MOV   SAVE, SP
             RTS   PC
       ERROR: HLT
       RAND: MOV   R0,-(SP)
              MOV   R1,-(SP)
       NUM: MOV   (PC)+1, R0
ASH #8. RO /USE BIT 7 TO RANDOMLY CHANGE ODDNESS:
   SEVENNES.

BCH #177777, R8 /THICK OFF ALL BUT LOW ORDER BIT

ADD R0, R0

INC R8 /ELIMINATE MUL BY 0

MUL (CP)+, R0 /MUL BY K

R7: 77777

TST R1 /MAKE SURE R1 IS POSITIVE

BCH . R4

BCH R1, R1 /SAVE LOW RESULT

CLR R8 /TAKE LOW RESULT

DIV (CP)+, R0 /DIV POLY/UPLIM

MOV R1, (CP)+ /PUT REM ON STACK

MOV (SP)+, R1 /RESTORE REG

MOV (SP)+, R8

RTS PC

*****************************************************************************

BINDEL: SUB #18, SP

MOV 18, (SP), (SP)

MOV R0, -(SP)

MOV R1, -(SP)

MOV R2, -(SP)

MOV R3, -(SP)

MOV R4, -(SP)

MOV (SP)+, R0

MOV 5P, R4

ADD #24, R4

MOV 22, (SP), R2 /GET ARG

MOV #FRAME, 12, (SP)

TST R2

BCH TTV1

MOV #MULS, 12, (SP)

NEG R2

TTV1: MOV #5, R1

TTV2: MOV R0, R2

CLR R2

DIV #18, R2

DIS (CP), R3

MOV R2, -(R4)

SUB R1, TTV2

DIS #200, (SP)

TTV3: CMP (R4)+, #60

DNE TTV4

MOV -(R3), -(R4) /ADVANCE SIGN

MOV #FRAME, -(R4)

DR TTV3

TTV4: DIS #200, 22, (SP) /REMOVE DISTINGUISHING MASK

MOV (SP)+, R3 /RESTORE REG

MOV (SP)+, R3

MOV (SP)+, R1

MOV (SP)+, R8

RTS PC

END
<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1</td>
<td>Data 2</td>
<td>Data 3</td>
<td>Data 4</td>
</tr>
<tr>
<td>Data 5</td>
<td>Data 6</td>
<td>Data 7</td>
<td>Data 8</td>
</tr>
<tr>
<td>Data 9</td>
<td>Data 10</td>
<td>Data 11</td>
<td>Data 12</td>
</tr>
<tr>
<td>Data 13</td>
<td>Data 14</td>
<td>Data 15</td>
<td>Data 16</td>
</tr>
</tbody>
</table>

**Notes:**
- This table represents data collected over the course of an experiment.
- Each column corresponds to a different variable or measurement.
- The data is organized in rows, with each row representing a different observation or trial.
L: MOV RO, IND+2
    MOV #VEC, A3
    CLR -<SP>
    MOV #SYSCALL,<SP>
    IND: MOV B<SP>, -<SP>
    MOV #1,-<SP>
    JSR PC, STORE
    RTS PC
    NOTASS: MOV #RD, RO
    EMT 3UL
    JMP ERR
    NA: ASCI2 <12> <12>,/ENULP & NOT ASSIGNED/
    .EVEN
    .END
```
TITLE FRECON 11/24/75 R. E. SOMFELD
GLOBL FRECON, TH1, THO, THN, PHOUT, MONITOR
GLOBL FRECON, TH1, THN, GOUT, GT, GOUTG, FWCOL, NINCPR
GLOBL CRR, HEP, ALTTH1, DELTH1
CALL PRINT
R0=R0
R1=R1
R2=R2
R3=R3
R4=R4
CC=CC
CP=CP
PC=PC

DUR: B
H2: B

FRE: JSP  PC, GT
MOV  R0, H2  ; GET FRE
TST  FWCOL  ; IF FN, GO TO GH
DEQ  FR2
JSR  PC, ALTH1
RTS  PC

FRE: CLR  -(SP)  ; IF NO FN, PROCEED WITH FRE
MOV  #INFREQ, -(SP)  ; I
MOV  #25600, -(SP)  ; J
MOV  H2, R0
JSR  PC, FRECON
MOV  R0, -(SP)
MOV  R4, -(SP)
MOV  #4, -(SP)  ; JNUM
JSR  PC, STORE
RTS  PC

FRECON: MOV  R0, R2  ; FREQUENCY CONVERSION
CLR  R0  ; RETURNS NINC IN R0, C IN R1
MOV  #25600, R1
ASNC  #1, R0  ; 25600
DIV  R0, R3
ASR  R2  ; REM . GT. DIV/2?
JLT  14
INC  R0  ; IF YES, INC REMAINDER
MOV  #64, R4  ; INITIALIZE NINC
LPI: CMP  R0, #50  ; =====FRECON100, =====
BGT  DN
FR1: MOV  R0, R1  ; E
BLE  DF  ; ERROR
BCE  R4, R8  ; NINC
BLE  DF  ; ERROR
RTS  PC

DN: CMP  R4, #2
BLC  FR1
ASR  R0  ; C
BCE  R4, THINC
DN  LPI

DF: PRINT "BAD FRE CON FRECON"
```

EVER

ENTRY POINT FOR CALL FROM CLI

GIVE BOTH THE SAME NINC

IF PRODUCT IS LARGER THAN 16 DITS,
TRY AGAIN WITH LARGER INCH
```
X:

MOV D80, (SP)
MOV CR, (SP)
MOV #4, (SP)
JSR PC, STORE
RTS PC

TH1: CLR (SP) ; IF DELFT=0, NO TH

MOV #THOUT, (SP)
CLR TM001
CLR (SP)

MOV #1, (SP)
BR TH1

; ----------------------------------------------- ;
TH2: CLR (SP)

MOV #THOUT, (SP)
CLR (SP)

MOV #1, (SP)
JSR PC, STORE
CLR TM001

TH3: CMP (SC), #121 ; POINT SC TO NEXT NON LETTER
DEE TM02
R15 PC

TH0: CMP (SC), #132
DEE TH02
R16 PC

; ----------------------------------------------- ;
HDBF: CLR r4 ; STEPS

MOV (SP), R2; R2 NOW CONTAINS OLD PC
MOV 4(R2), R9 ; INCRE
MOV 8(R2), R12 ; INCRE

BR HFLP2 ; FOR OVER INCREMENT THE 1ST TIME TO ALLOW
; FOR ZERO STEPS

LP2: ADD 12, (R2), R1 ; INCRE, CR

HFLP1: CMP R1, R2 ; LESS

LTY L5

LP3: CMP R1, 2(R2) ; THIRD

GTY L7

CONT: INC 12, (R2) ; INCRE(M)

DLY L6

BTR BRT ; INCREMENT SHOULD NOT BE ZERO

CMP R8, 0, (R2) ; INCRE(

BLE M1

BR LP2

L5: CMP R8, #64.

BLE HIGH

ALU R9

BR L4

HOR: CMP R8, #2

BLE COPY

ASR R8

ASR R1

BR LP3

NEG: CMP R8, 8, (R2) ; INCRE(TO

DLY M2

BR LP2

M1: CMP R1, 16, (R2)

DLY M3

BR LP2

M2: CMP R1, 16, (R2)

DLY M3

BR LP2

M3: ADD #14, (SP)

MOV R4, R8

R15 PC
```
CLR R1

EVAL3: MOV -(R4), R8
DEC EVAL1 ; DONE
BGT +4
NEG R8 ; NEG IF POS
ADD R0, R1 ; ADD
BVC EVAL3

EVAL4: MOV -(R4), R6
NEG R4 ; OVERFLOW HAS OCCURRED
BGT +4
NEG R6
CMP R5 ; AND R5: \!IF NOT L, DIV BY 2
DLE EVAL4
ASR (R4)
BR EVAL4

SP3: MOV CNS, (SP)
VCR PC, NUMBER
BVC EV9
MOV R0, (SP)
DR SP4

EV9: HALT

EV9: MOV SAVE, R4

EV10: ADD -(R4)
BRC EV10
MOV SAVE, R4
BR EV2

EV7: MOV R4, R2 ; NO

EV7: MOV SAVE, R4

EV7: TST -(R4)
NEG EVAL2
CMP R4, R2
BGE EV7B

EVA: MOV CNS, R0

EVA: PRINT #LIMIT

EVA: PRINT #NUMBER
RTS PC

EVA: MOV CNS, (R43)

EVA: PRINT #ERROR

EVA: PRINT #ERROR

EVA: PRINT #ERROR

EVA: PRINT #ERROR

EVA: PRINT #ERROR

EVA: PRINT #ERROR
.TITLE STRCH 5/27/76 2330 R.E. BANFIELD

.GLOBAL STRCH ERR, NORD
.GLOBAL STORE 334H +, KMSG
.GLOBAL IN 3H, DATA, IALIM, ILIM, DLIM, Hire, ERR, SER
.GLOBAL TIN, NUMBER, DUM, GETLET
.REDF PI, PRINT, REGDFT

HIRE: 0
DATA: 0  
;POINTER TO DATA
IN:  0  
;POINTER TO AMPLITUDE INSTRUCTIONS
DLIM: 0  
;LIMIT OF DATA POINTER
IALIM: 0  
;LIMIT OF IN
ILIM: 0  
;LIMIT OF IF
SPACE=48
5C=48
STCH: MOV R1,<SP>  ;SAVE REG 1

MOV R4,<SP>
MOV R6,R4
ADD R6,R4
MOV R6,R1  ;COUNT G CHARACTER
ST1: CMP (R4)+, R5,SPACE
JBR ST2
CMPO (5C)+, R6  ;COMPARE CURRENT CHAR WITH DELIMITER
JBR NR
ST2: MOV R1,ST1
MOV R26,R1  ;ATTEMPT TO FIND DELIMITER WITHIN 120 CHARACTERS
ST3: CMPO (5C)+, R0
JBR ST4
ST4: MOV R50,-1,(5C)
ST5: MOV R50,R0
JMP ERR
JMP ERR
ST6: MOV (SP)+, R4
MOV (SP)+, R4
ADD R4,R4
ADD R5,R0
ADD R5,R0
R5S PC
NMSG: ASCIZ "NO DELIMITER"
NMSG: ASCIZ "NO ROOM IN CH DUM"

;*******************************************************
;STORE: MOV R2,(SP),R1  ;NUMBER OF DATA WORDS; DNUM
;       MOV R4,R0
;ADD R4,R0
;       JBR ADDR OF TOP OF STACK
;ADL R0
;ADD R5,R0
;MOV R0,R0,(PC)+
;*******************************************************
;TOP: 0
MOV R4,R0  ;DNUM NUMBER OF DATA WORDS
ADL R0
CMP R0,SECR
DNTS OVRER
CMPO R0,DLIM
DNTS DI
TST (R2)  ;FREE OR AMPL?
IN:  MOV  DATA: B1N
ADD  $2, IN
MOV  <-<R2>, B1N
ADD  $2, IN
LOOP: MOV  <-<R2>, BDATA
ADD  $2, DATA
SNO  #1, LOOP
OUT: MOV  <-<R2>, BTOP
MOV  TOP, SP
MOV  PC

FREE: MOV  IT, R0
ADD  #0, R0
CMH  R0, SFR
CMH  R0, IFLIN
CMH  "FL1
IF2: MOV  DATA, B1F
ADD  #2, IF
MOV  <-<R2>, BIF
ADD  #2, IF
OUT2: MOV  <-<R2>, BDATA
ADD  #2, DATA
SNO  #1, LOOP
BR  OUT
D1: ADD  #400, DLIN
MOV  DLIN, DATA
ADD  #200, DLIN
DR  PG
IN1: ADD  #400, INLIN
MOV  IN, R3
INLIN: (R3)+
MOV  #JUMP, (R3)+
MOV  INLIN, IN
ADD  #200, INLIN
DR  IN2
IF1: ADD  #400, IFLIN
MOV  IF, R3
IFLIN: (R3)+
MOV  #JUMP, (R3)+
MOV  IFLIN, IF
DR  IF2
, OVRKR: .PRINT #RRMSG
JMP  ERR

TIM:  DIC  #20, NDIC  J CLEAR DOUBLE READ BIT
MOV  PC, NUMBER
MOV  R0, DUR
JSR  PC, GETLET
MOV  SC, (PC)+
SAVE:  0
LOOK:  CLR  R4
MOV  #3, R4
LIT:  MOVD  (SC)+, R2
MOV  SC, (PC)+  ; DEBUGGING AID
0
JSR PC:SECOND

CPU RS,R2 IS THIS THE ONE?

HEY RS, RS

GET .RS

CPU RS, #40000

LRT EV530

HEY RL, GHS

JSR PC: EV5HT

DR EV57

EV530: ASL BS3

DR EV57

SECOND:HEY RI, (CP)

HEY RI, (CP)

HEY RI, (CP)

HEY RI, RI

HEY RI, RS

HEY (CP)+, R4

HEY (CP)+, R1

CPU RS, SGR

BLE SE1

DER SE2

EC 3

RT5 PC

SE4: GEV

RT5 PC

SE2: .PRINT #Z10000

SE4: RDP RS, DP

JMP RON

SE3: .PRINT #666

.SE4

;##############################################################################

FACED: JSR PC: CLEARK

JSR PC, NUMBER

MOV RS, GCHOI50

JMP FRESCH

;##############################################################################

STOPS: ASCI1 "SECOND AND ERR"

NOMS: ASCI1 "NO GRADE"

SIGNOS: ASCI1 "B IN 9 GRADE"

CT: ASCI1 "SECOND"

EVEN...
...
UNSPRD: MOV DDL#4, DVLK#4
J8R PC, CLEAR
MOV PM, DVLK#4
MOV D2, -(SP)
MOV D8VLK, -(SP)
J8R PC, LOOKUP
MOV D2, -(SP)
MOV DUNIIL, -(SP)
MOV HIC4AL, KO
SUB (SP), KO
CLR KO
MOV KO, -(SP)
CLR -(SP)
J8R PC, READN
CLOSE KO
J8P NON

SHY86: J8R PC, CLEAR
MOV KS, HORD
DIS #44, HORD  MUST BE IN SER MODE
MOV D8VLK#4, DVLK#4
MOV KO, -(SP)
J8R PC, DIODEC
MOV (CP), RO:
MOV (CP), (SP)
J8R PC, READN
CLR BLOCK
J8P PRESCN
NOARG: .PRINT ERRARG
J8P NON

PROD: J8R PC, CLEAR
MOV PM, NUMBER
DVC PRINT
PR12: INC KO
J8R PC, PRINT
PRINT: MOV KO, KO  PRINT PROG OF SINGLE ITEM
J8R PC, PRINT
J8P NON
PRODIT: MOV KO, -(SP)  SUBROUTINE FOR PRINT OF ONE ITEM
CLR -(SP)
J8R PC, PRINT
DVC PRINT  IN ALL MODE: DONE
DVC 46
J8P PCER  PRINT ERROR
MOV KO, (PC)+-
SAVE: O
.PRINT #ITEM
MOV KO, KO
J8R PC, OUT
.PRINT #ITEM
CLR KO  CLEAR REG FOR TOTAL DIVIDEND
PRG: MOV (R4), KO
... · DMM2

: 169650: RAD50 "GEN"
: DAVELS: CLR R4 1R4 INDICATES IF SAVE OR UNSAVE
: OR DB
: UN5550: NOV #1, R4
: DB: NOV BUL14, DVLK14 "SET UP DEVICE BLOCK"
: NOV SEVERG5, DVLK14
: JKR FC, CLEAR
: NOV #2, R8 1MSG 2 SEQUENCE BEEN GEN TO SET BOUNDS?
: JKR FC, DERN7
: DVC SV1

SV1: NOV #DATA, R2 1GET UPPER BOUND OF SCORE
: CRL R6
: D51: CMP (R2)+, R6
: BLD BS2
: D52: NOV -(R2), R6
: SCD R1, D52
: NOV R6, (PC)+
: SCRNT: NOV BONSER, R1 1COMPUTE # OF HARD AND BLOCKS
: SCR SCRNT, R1
: INR R1 1TO INCLUDE BONSER
: NOV R2, (PC)+
: ING: CLR R6
: DIV BS6, R6
: TST R1 1IF REMINDER, INC # OF BLOCKS
: DCS SV2
: INC R8
: D52: NOV R8, (PC)+

DLY: TST R4 1SAVE OR UNSAVE
: DLT UN51
: DCR SV4

SAV1: ENTER #R4, #REVULK, R6
: DCC SAV2
: TSTI REFRESH 715 CHAN OPEN
: DCR SAV2, YES
: PRINT REFRESH OTHER ERROR
: JMP NOH
: NOV SCD, BONSER 1STORE LOG OF SEQ
: NOV #4+(5P)
: NOV SCRNT, -(5P)
: NOV NOCTN, -(5P)
: NOV (PC)+(5P)
: STDLK: 8 1STARTING BLOCK
: JSR FC, WRITE
: ADD DLYING, STDLK 1UPDATE STARTING BLOCK
: PRINT ERRMSG 1REMINDER TO ERASE
: JMP NOH

UN51: .LOOKUP SERRH, #4, REVULK
: DCC UN52
: TSTD SERRH5 1CHAN OPEN?
: DCR UN52
: PRINT SERRH5, OTHER ERROR
: JMP NOH

UN52: JSA FC, MUNDER 1GET SEGMENT 
: DVC UN53 1IF NO ARG, ASSUME 0
: CLR R6
TITLE INIT 5/20/76 2050 R.E. SANDFELD

GLOBAL INIT, DEB1NO, ENDO, RACHV, FINE, GANN, ERROR
GLOBAL GANNI, GANN2, GANN3, GET1, GET2, GET3
GLOBAL GELL, KELL, NON
GLOBAL C1IE1, C1IE2, C1IE3, C1IE4
PC=NO
ST=NO
R5=NO
R4=NO
R3=NO
R2=NO
R1=NO
R0=NO

CLVCD=184

DSTAT=164000 16/R STATUS

CLSTAT=172540 16/CLOCK STATUS
CLSET=172542 16/CLOCK SET

POINT: TEXT
TEXT: NOETI
ERROR: NORD

SAVE: EBXM 13.

INIT: CLR GANN1
CLR GANN2
CLR GANN3
CLR GANN4
MOV #21, R2 ; CLEAR ALL

LP2: CLR ~<R0>
SOD R1, LP2
MOV #21, R1
MOV #GET1, R0

LP3: CLR ~<R0>
SOD R1, LP3
CLR GET1-84. ; FMO
CLR GET1+84. ; FAMUH
CLR GET1+60. ; FAMUH
CLR GET1+48. ; FAMUH
CLR GET1+24. ; FAMUH
CLR GET1. ; FMO
CLR GET1+12. ; FAMUH
MOV RACHV1274, C1IE1
MOV RACHV1276, C1IE2
MOV RACHV300, C1IE4
MOV RACHV300, C1IE5
MOV RACHV300, C1IE6
MOV RACHV300, C1IE7
MOV RACHV300, C1IE8
DIS #40800, #177776
CLR R2
CLR R3
CLR R4
CLR R5
CLR KELL
CLR KELL2
DIC #40800, #177776

MOV #10, UDDSTAT ; SET D/R STATUS FOR RANDOM MODE
MOV #DACHY,RECLOC
MOV #300,#ELCE0+12 ;SET CLOCK INT STATUS,ORG
MOV #2000,#ELCE0 ;SET TIME BEFORE 1ST INT

CLR ERROR ;INIT ERROR INDICATOR
MOV #TEXT,POINT ;INIT CHARACTER BUFFER
DIS #111,#ELSTAT ;SET CLOCK STATUS,START COUNT
JMP ENMSG

FINE: DIC #111,#ELSTAT ;DISABLE CLOCK
MOV RO,SAVE
MOV R1,SAVE+2
MOV R2,SAVE+4
MOV R3,SAVE+6
MOV R4,SAVE+8
MOV R5,SAVE+10.
MOV SP,SAVE+12.
DIS #4800,#177776
MOV RO,SAVE+14.
MOV R1,SAVE+16.
MOV R2,SAVE+18.
MOV R3,SAVE+20.
MOV R4,SAVE+22.
MOV R5,SAVE+24.
DIC #4800,#177776

CMP ERROR, #8 ;CHECK FOR ERROR
BGT ENMSG

JMP NON ;EXIT
ENMSG: MOV R0,#40,ERROR+4 ;SPACE
ADD #20,ERROR ;ASCII MASK
LOOP: MOV R0,2#POINT,RO ;ERRORS:

1:CHNR OVERTAKEN
2:CHNY NESTING INT
3:ATTEMPT TO GEN FTER ABOV LIM

INC POINT
INF 344
DEC 2
CMPD 2#POINT,#0
DNE LOOP
JMP NON

END INIT
.TITLE ERROR G/28/TS 52 R.C. SALFIELD

.END

.GOBAL ERROR,GETI,GETI1,ERROR

.GOBAL ONE,THREE,FOUR,FIVE,CHDIR

.MK

.PROG

.CM

.PC

.LIMIT=2

.DWTE 15,12

.END

.BEGIN:

.END:

.ERROR:

.MOV $BEGINING,0

.MOV $BEGINING,0

.MOV $THREE,(R)+

.MOV $START1

.MOV $START2

.MOV $C1

.MOV $C2

.CLR PNT

.CLx R7

.CLX DM4+4

.NEXT:

.CMP 0,$ENDR

.DLX ER7

.INC R7

.CMP R7,0

.DLX DM1

.TST $177972

.DEX FIN

.LOOP:

.CMP R7,0

.DEX R

.ERROR:

.CMP $START1,$START2

.DEX RND

.SUB $START1,$START2

.CMP $START2,$LIMIT

.DEX RND

.MOV $1,$START1

.CMP $START2,$START1

.DEX RND

.MOV $1,$START1,(R)+

.DR GNC12

.ERROR:

.MOV $START2,$R9

.MOV $1,$C1

.MOV (PC)+4,(Q)+

.AI:

.JSR PC,GETI

.DR NEXT

.SLDX:

.MOV $1,ERROR
JMP NEXT

CH1: MOV C1, START1
MOV C2, START2
CMP START1, START2, THREE
DLY
MOV START2, (0)+
OUT

CH17: MOV START1, (0)+
MOV RTHEE, (0)+
MOV R1, (0)+
MOV R2, (0)+

JMP PC, GET1
JMP PC, GET11

END BHHGR
SUB KELL, END1
SUB KELL, TOPTNT1
MOY KELL, REL1
ADD KELL, DEGIN1
ADD KELL, ENDS1
JMP G111

BEGIN1: 0
END1: 0

END GETYI
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>REL2, REL2</td>
</tr>
<tr>
<td>ADD</td>
<td>REL2, BEGIN</td>
</tr>
<tr>
<td>ADD</td>
<td>REL2, T0PT2</td>
</tr>
<tr>
<td>JMP</td>
<td>GIII11</td>
</tr>
</tbody>
</table>

BEGIN: 0
END2: 0
.END GETII
<table>
<thead>
<tr>
<th>TITLE</th>
<th>TABLE 472/75</th>
<th>R.E. SMITHFIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN</td>
<td>TABLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HORD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8135133, 143535, 284533, 38374, 56132, 69335, 85453, 854533</td>
<td></td>
</tr>
<tr>
<td></td>
<td>95365, 61331, 68135, 95335, 733185, 75535, 755353, 7553537</td>
<td></td>
</tr>
<tr>
<td></td>
<td>58777, 778837, 78778, 787782, 787782, 787782, 88135, 982341</td>
<td></td>
</tr>
<tr>
<td></td>
<td>55335, 89453, 43433, 29332, 38374, 284535, 284535, 2845353</td>
<td></td>
</tr>
<tr>
<td></td>
<td>17464, 16544, 151356, 147484, 145556, 135442</td>
<td></td>
</tr>
<tr>
<td></td>
<td>125375, 122552, 110557, 111262, 167456, 104675</td>
<td></td>
</tr>
<tr>
<td></td>
<td>102616, 181165, 108249, 106681, 168249, 181665</td>
<td></td>
</tr>
<tr>
<td></td>
<td>102616, 184675, 107456, 116262, 116437, 126652, 127375</td>
<td></td>
</tr>
<tr>
<td></td>
<td>134342, 141666, 147484, 159556, 183445, 171674</td>
<td></td>
</tr>
<tr>
<td></td>
<td>END</td>
<td>TABLE</td>
</tr>
<tr>
<td>R1:</td>
<td>CMP</td>
<td>28. (R2), #64</td>
</tr>
<tr>
<td>CEC</td>
<td>GNC</td>
<td>GNICH</td>
</tr>
<tr>
<td>AGL</td>
<td>28. (R2)</td>
<td>INC</td>
</tr>
<tr>
<td>AGR</td>
<td>-10. (R2)</td>
<td>INC</td>
</tr>
<tr>
<td>DSR</td>
<td>R2</td>
<td></td>
</tr>
<tr>
<td>RSR</td>
<td>-20. (R2)</td>
<td></td>
</tr>
<tr>
<td>DR:</td>
<td>GHHH</td>
<td></td>
</tr>
<tr>
<td>H2:</td>
<td>INC</td>
<td>-20. (R2)</td>
</tr>
</tbody>
</table>

**GHHH:**
- MOV: #3, ERROR
- DR: GCONT

**GHN5IN:**
- MOV: R2, R2
- MOV: R2, R2
- MOV: #GL155, 28. (R2)
- MOV: (R1)+, 14. (R2)
- MOV: (R1)+, 10. (R2)
- MOV: (R1)+, 28. (R2)
- TST 54. (R2) TST5
- BLE GL4
- MOV: GNCRC, GPLUS44
- DR: GL4

**GHN5IN:**
- MOV: #8000, GPLUS44
- DR: GL4

**GHN5H:**
- MOV: #3, ERROR
- CMP: FINC

**GHHH:**
- MOV: R2, -5P
- MOV: R0, R2

**THINUS:**
- ADD: -20. (R2), 18. (R2)
- CMP: -10. (R2), #25
- DLY: FG
- GPLUS: CMP: -10. (R2), #2000

**GCONT:**
- MOV: -32. (R2)
- INC
- DSR: FN

**T4:**
- MOV: -34. (R2), R1
- DIV: -10. (R2), R0
- MOV: R0, GO. (R2)
- MOV: (SP)+, R2

**TG:**
- CMP: 28. (R2), #64
- DCE: THINUS

**THINUS:**
- CMP: 28. (R2), #64
- INC
- CMP: 28. (R2), #64
- DCE: THINUS


```
354

|   TITLE: 3/17/75 14 N.C. SHAPLES |
|   END:   |
|   EXIT:  |
|   ENLP | "03401:010 |
|   ENLP |

CLODL PROCY, ONE, TWO, THREE, FOUR, FIVE, BEGIN |
PC=X7 |
DH=KC |
CHAN=165002 |
BUF=164004 |
CLOCK=172542 |
CON=42 |
RA=41 |
RB=40 |

PROCY: DEC RENT ; "THERE HAS BEEN AN INTERRUPT."
SET B8172540 ; ERROR IN NESTING INT?
BLT NEXT ; IF NEST2, YES.

NEXT: MOY #2, ERROR
FINP |
DNE: CLR #CHAN |
MOY (R0)+. B8DUF |
CMY R0, BEND2 ; END?
DMIS D1 ; IF YES, BRANCH.
MOY (R0)+. B8CLOCK |

D1: MOY #BEGIN, RO |
MOY (R0)+. B8CLOCK |

THO: CLR #CHAN |
MOY (R0)+. B8DUF |
INC #CHAN |
MOY (R0)+. B8DUF |
CMY R0, BEND2 |
DMIS D1 |
MOY (R0)+. B8CLOCK |

THI: CLR #CHAN |
MOY (R0)+. B8DUF |
INC #CHAN |
MOY (R0)+. B8DUF |
CMY R0, BEND2 |
DMIS D1 |
MOY (R0)+. B8CLOCK |

FOUR: CLR #CHAN |
MOY (R0)+. B8DUF |
MOY (R0)+. B8DUF |
V: DEC (PC)+ |
D4: |

GOT 1 |
INC #CHAN |
MOY (R0)+. B8DUF |
CMY R0, BEND2 |
DMIS D1 |
MOY (R0)+. B8CLOCK |
RTI |

FIVE: MOY (R0)+. DS |
CLR #CHAN |
```
LIST OF REFERENCES


_________ (1973b). Toward a musical intelligence system: OBSERVER. Numus-West, no. 4, pp. 11-16.


Nelson, Gary (1973). Comments given to a meeting of the
Ohio Theory-Composition Teachers.


Smith, Leland (1973). *SCORE: a musician's approach to*


