An Integrated Computer Music Software System

DONALD BYRD
Wrubel Computing Center
Indiana University
Bloomington, IN. 47401

INTRODUCTION

I have long felt that what computer musicians (if that is an acceptable term) need is not a group of stand alone programs – however good – for analysis, music printing, sound synthesis, etc., but rather an integrated system in which each component feeds and/or accepts output from others. The seed of this idea appears in my 1974 article, "A System for Music Printing by Computer" [1]. This article is mostly a discussion of an early version of my SMUT music printing program, but also talks briefly about two interface programs (JANUS and SMIRK) which made it convenient to prepare SMUT data with a composing program or though the MUSTRAN music input language.

More recently, Raymond Erickson has described related work, the DARMS Project [2], which has very ambitious goals in music encoding, printing, and analysis capabilities. However, only a small portion is actually implemented (to my knowledge) and it neglects computer-assisted composition and sound synthesis completely.

My purpose here is to describe what I call (for lack of a better name) the "Indiana University Computer Music System" (henceforth IUCMS). This system has been assembled at Indiana University over the past eight years from programs written there and elsewhere, and currently runs on a CDC 6000 or 7600 or CYBER. While a great improvement on what I had to show in 1974, it is certainly far from ideal. However, it has some capabilities in every area suggested by Erickson or myself and, overall, is the most powerful computer music software system I know of. It also has two characteristics that should enhance its usefulness to people who have different hardware:

1. Almost all of the programs are written in a widely available higher-level language. Most are in FORTRAN, and several are pretty much ANSI (X3.9, 1966) standard FORTRAN. This should ease conversion to other computing systems.
2. Unusual peripheral equipment required is minimal. Of course, music printing programs require digital plotters, which are not available everywhere. However, SMUT (and JANUS, which has a plotting option) is organized and coded in such a way that converting it from Calcomp mechanical plotter to graphic CRT to electrostatic plotter has been almost trivial, and conversion to phototypesetter should not be much harder. In contrast, some music printing programs are highly device-dependent.

All of the programs in the IUCMS are batch-oriented. To a considerable extent, this is implied by the above two points, because interactive or real-time programs useful in music tend to require unusual hardware (e.g. graphics terminal, organ keyboard, digital synthesizer).

So much for the hard sell. Figure 1 graphically depicts the relationship among the 14 programs (or sets of programs, in several cases) in the IUCMS. Names in large capitals are major programs, in small capitals, interfaces. Arrows indicate possible data flow. (The "organ keyboard and tablet programs" are not really part of the IUCMS. See below.) Table 1 gives a brief description and bibliography for each program. Thus, for example, data keypunched in the appropriate language – MUSTRAN – can be digested by the MUSTRAN II translator. Translator output can be sent to analysis programs directly or (through the appropriate interface program) to SMUT for printing in conventional notation or to MUSIC5 for sound synthesis. With the aid of a powerful operating system like CDC's KRONOS, the entire operation can be done in one job.

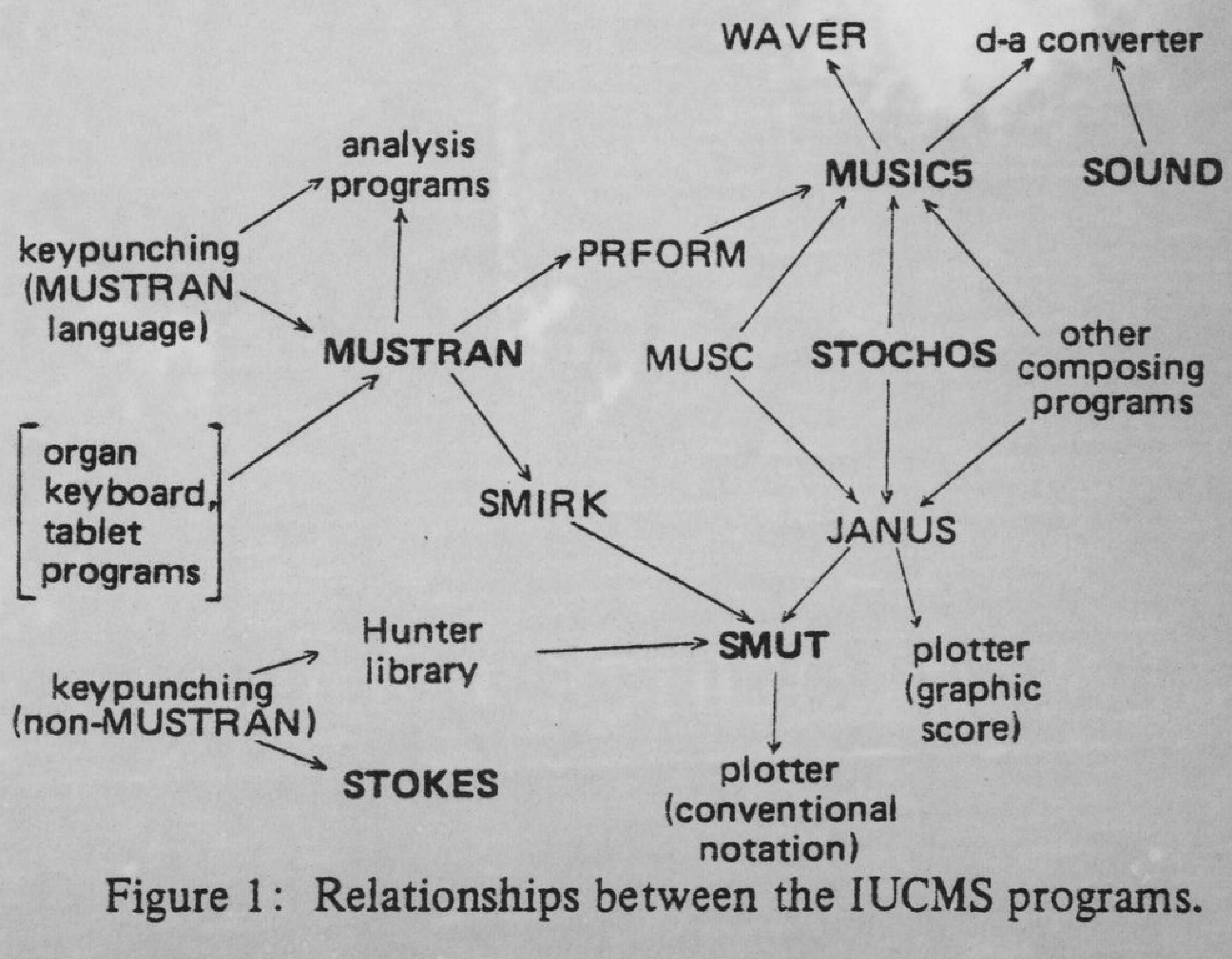


Figure 1: Relationships between the IUCMS programs.

THE PROGRAMS

I would now like to discuss most of the programs in more detail. Full bibliographic references are in Table 1.

STOCHOS is basically the "free stochastic music" program discussed and listed in Xenakis' book. I've modified it slightly to output MUSIC V-type note cards.

JANUS accepts as input MUSIC V note cards giving, for each note, the attack time and duration in seconds (or beats), the number of the instrument to play, pitch, dynamic level, and optional information on glissandi or other parameters. JANUS then finds a good approximation in conventional music notation for the rhythm of each beat and outputs its SMUT equivalent plus an optional graphic score. It also performs utility operations such as substituting character strings for numeric codes for dynamics, and suppressing redundant dynamics (the user has control over what, if anything, is considered redundant). Since virtually all composing programs have on hand the data needed by JANUS, minor modifications to most would allow them to produce standard musical notation output.

Table 1: INDIANA UNIVERSITY COMPUTER MUSIC SYSTEM
March 1977

	Name	Language	Notes	Description
1.	STOCHOS	FORTRAN	C	Xenakis' composing program.
2.	JANUS 2.2	FORTRAN(A)	C	General composing program-to-SMUT interface + does graphic scores
3.	SMUT 2.0	FORTRAN(A)	C	Polyphonic music printing program by Donald Byrd
4.	MUSTRAN II	FORTRAN(A) + assembler	C,W	Music input language translator by Jerome Wenker
5.	MUSTRAN lib.	FORTRAN(A) + assembler	C,W	Five analysis and two utility programs by Jerome Wenker
6.	MUSIC5	FORTRAN + assembler	C,M	Sound synthesis program by M.V. Mathews et al.
7.	WAVER 1.1	FORTRAN + assembler	C	Printer plotting utility for sound synthesis programs
8.	SMIRK 2.0	FORTRAN(A)	C	MUSTRAN-to-SMUT interface
9.	Gross lib.	SNOBOL4	C	3 translation and 5 analysis programs by Dorothy Gross
10.	MUSC 1.3A	ALGOL-60	C	12-tone composing program by Donald Byrd
11.	PRFORM	FORTRAN	C	MUSTRAN-to-MUSIC5 interface
12.	STOKES	SNOBOL4		Set theoretic analysis program by Charles Stokes
13.	Hunter lib.	FORTRAN	C	4 white mensural notation programs by Bockett Hunter
14.	SOUND	PASCAL + assembler	G	Sound synthesis language compiler by George Cohn

Notes:

C A CDC 6000-7000-CYBER version is available from me.

G A CDC 6000-7000-CYBER version is available from me only to users of these machines. Others should contact George Cohn, Wrubel Computing Center, Indiana University, Bloomington, IN 47401.

M A standard version is available from M.V. Mathews, Bell Telephone Laboratories, Murray Hill, NJ.

W Versions for several machines are available from Jerome Wenker, 1998 Pacific Ave., Unit 105, San Francisco, CA 94109.

Under 'language,' (A) means that FORTRAN is virtually ANSI standard (3,9).

My versions of most programs could be run on other machines with fairly minor changes. No program contains more than 100 or so assembler statements, except MUSIC5, which contains about 1000, and SOUND, which contains about 2000.

SMUT is, as mentioned before, my music printing program. It has recently undergone two major revisions, of which the first greatly impoved the quality of its output and the second gave it the capability of notating scores of any number of voices as long as there is only one voice per staff. Two examples, drawn on a 200 point per inch Versatec electrostatic plotter, are included here (Figs. 2, 3). Unlike any other music printing program I am familiar with, SMUT was originally intended as an output package for use with music composing programs. Since the music was to be generated and transcribed with no manual intervention, SMUT is more fully automated than most such programs. For example, it can decide reasonably well what notes to beam together, where to draw the beams, and when to switch between, say alto and treble clefs in a viola part. It also has an unusual "rhythm decomposition" feature that will renotate a 16th rest followed by a whole note in 4/4 time as 16th rest, dotted eighth note, quarter note, half note, barline, 16th note – the notes all being tied together. These capabilities can be independently disabled if desired.

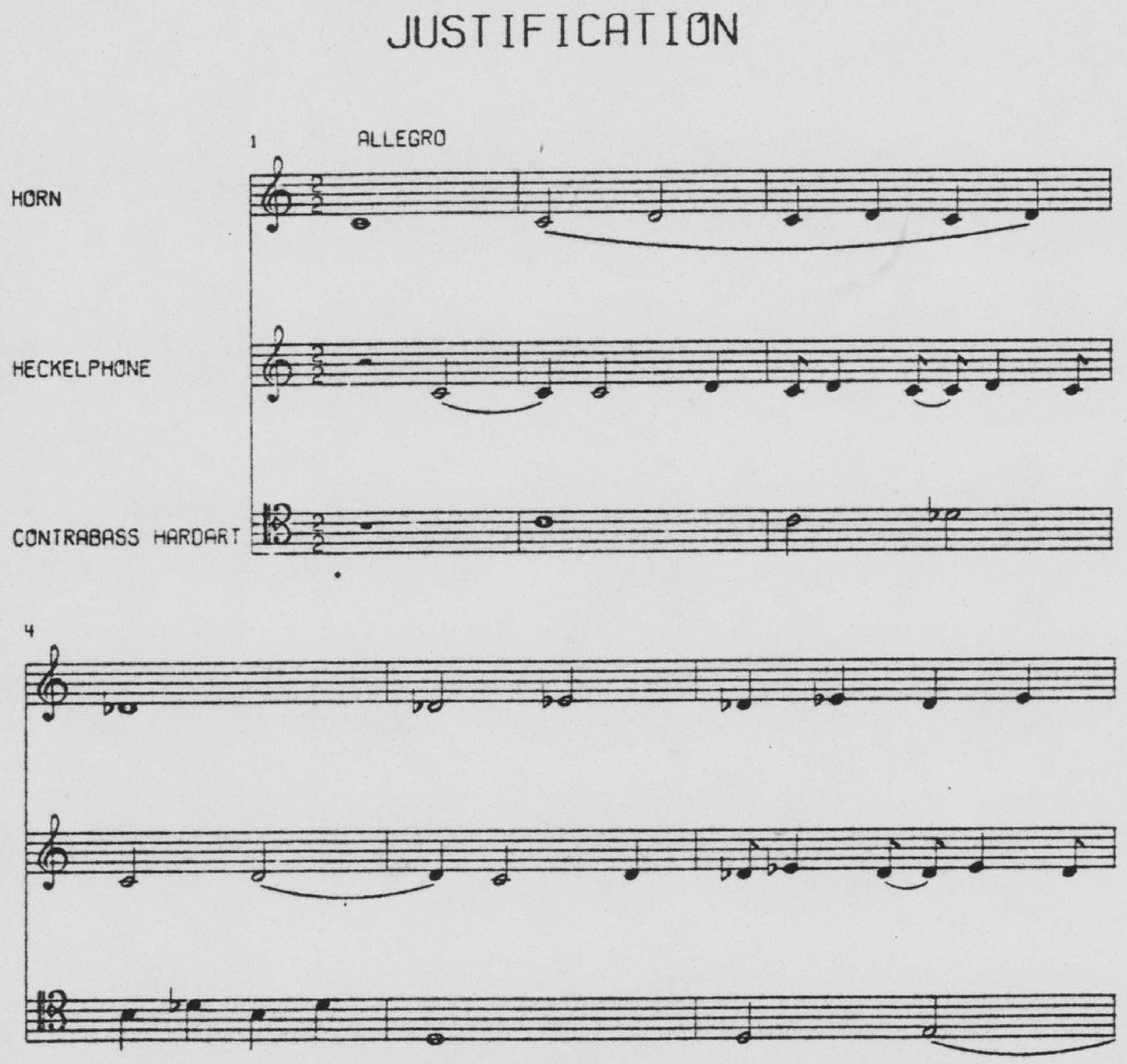


Figure 2.

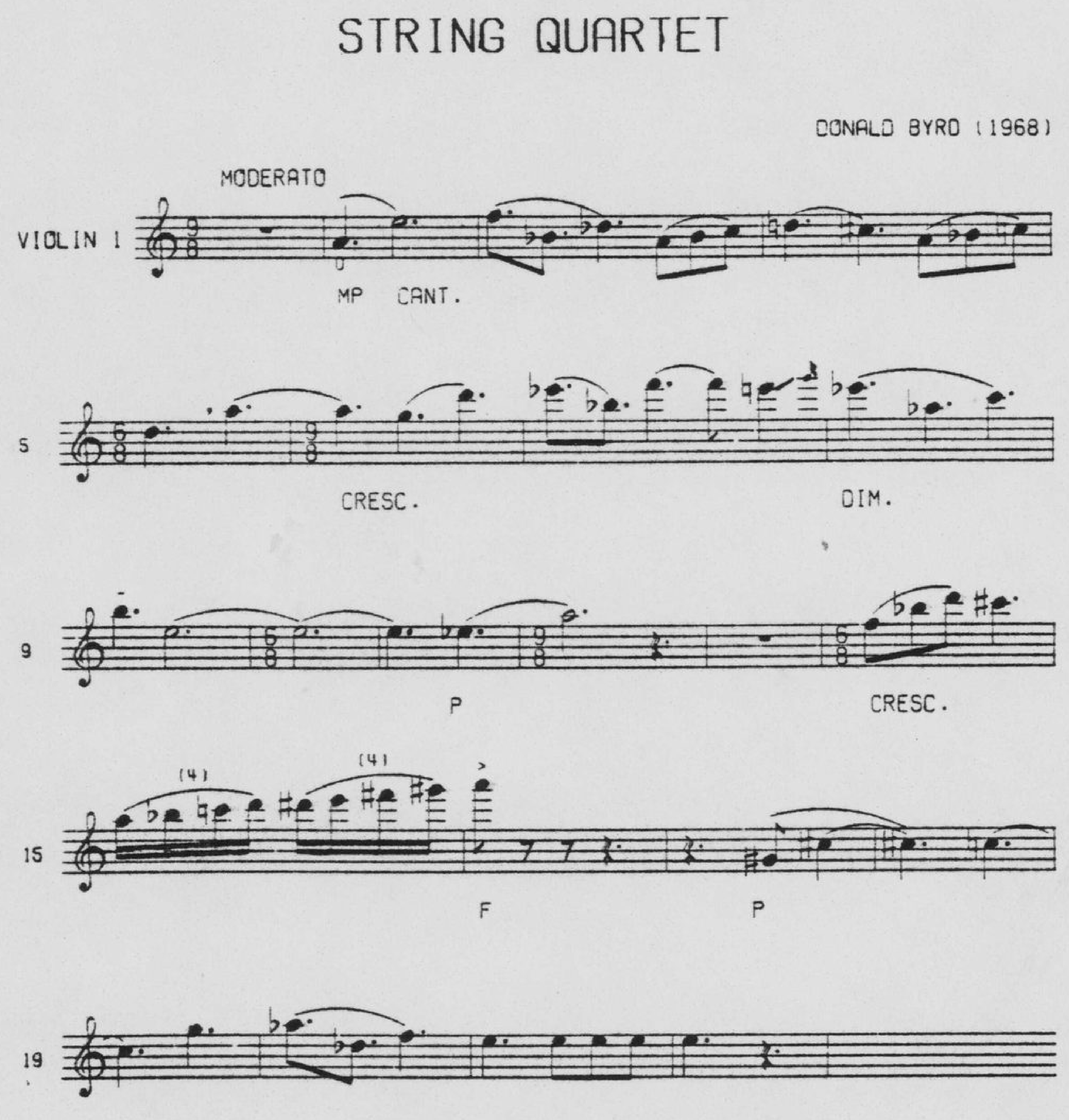
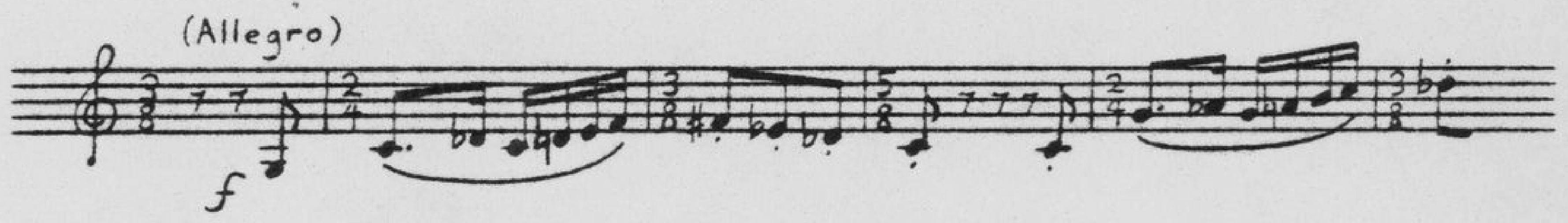


Figure 3.

MUSTRAN II is Jerome Wenker's translator for the latest version of his MUSTRAN alphanumeric music encoding language. The only other encoding language that is anywhere near as well-developed is the much more widely known DARMS code of Bauer-Mengelberg and others. By way of comparison, Erickson [2] says that "DARMS [the encoding language, not the complete system] may be said to approach more than any other system the ideals of completeness, objectivity, and encoder-directness." On completeness, he's probably correct, though MUSTRAN II doesn't fall far short overall and has some capabilities (e.g., ethno-musicological notation) that appear to be lacking in DARMS. In objectivity I consider them equivalent, and as for "encoder-directedness" – by which he apparently means ease of use – MUSTRAN is clearly superior. The simplicity of representation of pitch and rhythm information is of overhwelming importance in a music input language. DARMS' "2Q" for a quarter-note F in the bottom space of the treble staff is "4F" in MUSTRAN. In any case, the MUSTRAN II translator is somewhat large and unwieldy but works quite well. Fig. 4 contains a melody by Bartok and its MUSTRAN equivalent.



```
GS, 3=8, '(ALLEGRO)', 8R, 8R, WFW, 8G-, /, 2=4, 8C.K1, 16$D, 16C, 16ND,
16E, 16FK1, /, 3=8, 8*FV3, 8$EV3, 8$DV3, /, 5=8, 8CV3, 8R, 8R, 8R, 8CV3, /,
2=4, 8G.K2, 16$A, 16G, 16NA, 16B, 16C+K2, /, 3=8, 8$D+V3, ...
```

Figure 4. MUSTRAN code for a Bartok theme

A number of "library subroutines" come with MUSTRAN that make it rather convenient to write FORTRAN programs that access the information output by the translator itself.

The MUSTRAN library contains several analysis programs written by Wenker. These are pretty simple-minded and, I think, more likely to be useful for ethnomusicologists than those interested in "art" music.

Everybody – well, a lot of people – are familiar with MUSIC V or one of its relatives (MUSIC4BF, MUSIC7, MUSIC65, MUSIC360, etc.) These are all descended from Bell Telephone Labs' MUSIC IV and V, the classic sound synthesis programs. Everyone has their own version, and IUCMS is no exception. This one has some nice enhancements which however are poorly documented and are implemented in COMPASS, the CDC 6000 assembly language.

WAVER is a utility originally written by George Cohn that does printer plots of the samples generated by a sound synthesis program for any specified time period or periods. Simple, but of obvious value to anyone who has ever carried a digital tape across campus from the computer center to the music school only to discover that the tape contains 10 minutes of silence.

The "Gross library" contains some rather sophisticated analysis programs. The input language is MUSTRAN. There is a linear grouping program that has such options as type of rhythmic and metric calculation (disregard rhythmic positions, calculate rhythmic positions, or calculate metric positions); parameter in music to be grouped (pitch, rhythm, articulation, dynamics, or chords or sets); whether to scan for inversions and/or retrogrades of patterns; how much more important patterns in the top voice are (if at all); and so on. There is a similar program for vertical grouping, a thematic analysis program, a harmonic and set analysis program (yes, it considers resolution), and a summarizing program. The thing that strikes me looking at these programs is that the author obviously thought about musical considerations when designing her programs. Refreshing.

The fundamental assumption of the MUSC composing program is that the overall structure of each voice of a composition can be described by a line segment function, called a "contour function". This function, or this function inverted, controls each of pitch, rhythm and dynamics as a function of time for its voice. The controlling mechanism is different for each parameter. In pitch, the contour function determines the register – i.e., octave position – of each note within the instrument's range, while the 12-tone technique is used to choose exact pitches. Thus the melodic line should have coherence at both micro (note to note) and macro (entire composition) levels. For rhythm, the user supplies any number of rhythm patterns arranged in order of increasing or decreasing average note duration. Then the contour function is quantized to choose one of these patterns. A simpler method is applied to choice of dynamics. The limitations of these methods can be overcome somewhat by using the program's ability to add an arbitrary amount of randomness to the control of each parameter. For example, the "randomness of octave choice" parameter might be set to zero – octave choices totally determined by the contour function – while randomness of rhythm pattern choice is .1 (some random disturbance of contour function) and randomness of dynamics is .75 (mostly random). Note, however, that each instrument's part is composed completely independently of the others. Thus, any resemblance of results to traditional, or any other, harmonic practice is purely coincidental. The program is set up to call a user-written procedure if desired to revise its decisions.

Finally, George Cohn's SOUND is a compiler and run-time library for one of the most powerful sound synthesis languages around. For comparison, the MUSIC V instruction set is a very weak programming language in the usual sense. There are procedures, yes (termed "instruments" and called with the "NOT" instruction), but without parameters. There are no variables, expressions, or control constructs ("IF"s, loops, etc.), and the only I/O is that to the sample file that results from "OUT" instructions in instruments. (MUSIC V does have "hooks" for user-written FORTRAN subroutines (PLFs, PLSs, and CONVT), which greatly mitigates the situation.) On the other hand, SOUND, like some other recent sound synthesis languages, has all of these features built in. Actually, though, calling it a sound synthesis language is somewhat of a misnomer. Cohn has written of it [3] :

> "SOUND is a procedure oriented language designed for writing digital sound synthesis programs. Where most languages of this type attempt to incorporate music related concepts, SOUND allows the user to develop and realize any kind of conceptual framework. The implementation of SOUND for the CDC 6600 includes a compiler which generates executable code for event generators (composition routines) and sample generators (instruments). The ability to define and use macros makes it possible to have program modules that look like functions but generate inline code. This is important in sample generation routines where the code will be executed for hundreds of thousands of iterations.

A SOUND program that employs FM is given in Fig. 5.

SOUND is implemented in CDC 6000 assembly language and in PASCAL, a very elegant general-purpose language recently developed and not yet available everywhere. So it is, unfortunately, not easily portable. Non-CDC users should contact Cohn, as mentioned in Table 1.

Composition: This is where the records are generated which drive the event scheduler in the synthesis environment. Code generated by the compiler comes from the composition driver, which is the main body of code in the user's program, and from actions, which are subprograms that can be used as functions to return values or as compositional procedures to issue any number of event records.

Synthesis: This is where the samples get generated. A sound driver is loaded in from the library, which reads up the file of event records presumably sorted by start time. Event records cause the driver to invoke SAMGENs (sample generators) and COMGENs (utility generators). SAMGENs have duration, which must be supplied as the first parameter. A SAMGEN's loop is implicit, not explicit. COMGENs have no duration associated with them, they generate no samples, and therefore have no assumed loop.

If an action is called from the composition environment, it is invoked immediately as a function or procedure subprogram. If a SAMGEN or COMGEN is called from the composition environment, an event record is issued containing the current action time, the synthesis module number, a SAMGEN/COMGEN flag, and all the parameters.

An action cannot be called from the synthesis environment, obviously, because actions are elements of the composition environment which is already spent by that time. SAMGENs may not be called from within the synthesis environment because they must be scheduled by the sound synthesis driver. If a COMGEN is called from the synthesis environment, it is invoked immediately as a function or procedure subprogram.

Figure 5: SOUND program to define and play an FM instrument

```
***   ALLOCATE VARIABLES FOR COMPOSITION AND SYNTHESIS OVERLAYS.   ***

COMPOSITION SR,FRFACT,X;
SYNTHESIS SINE(512),I,SIG,SIG2;


***   DECLARE 2 OUTPUT CHANNELS, A AND B.   ***

CHANNEL A,B;


***   DEFINE MACRO TO EMULATE MUSIC5 TYPE OSC UNIT GENERATOR    ***

MACRO OSC(AMP,FREQ,OUT,FN,PH):
        OUT := (AMP)*FN[PH];
        PH := (PH+FREQ+51200) MOD 512 ENDM;


***   TRYIT IS AN FM INSTRUMENT WITH EXPONENTIAL ENVELOPES
      ON ITS AMPLITUDE AND STRENGTH OF FM.
***

SAMGEN TRYIT(DUR,AMP,CF,MF,MI,MIF,LOC,AMPF,SCR1,SCR2):
   BEGIN
      OSC(MI,MF,SIG,SINE,SCR1);
      OSC(AMP,SIG+CF,SIG2,SINE,SCR2);
      SIG := LOC*SIG2;
      A := A+SIG;
      B := B+SIG2-SIG;
      AMP := AMP*AMPF;
      MI := MI*MIF;
   END;


***   GIVEN AN INITIAL VALUE L1, A FINAL VALUE L2, AND A DURATION
      DUR, EXPO CALCULATES A MULTIPLIER TO CUMULATIVELY GENERATE
      AN EXPONENTIAL CURVE FROM L1 TO L2.
***

ACTION EXPO(L1,L2,DUR):
   EXPO := (L2/L1)^(1/(DUR*SR));


***   TRY IS USED TO ISSUE AN EVENT RECORD FOR TRYIT WITHOUT BOGGING
     THE USER DOWN IN THE INTERNAL FORMS OF TRYIT'S PARAMETERS.
      TRY'S PARAMETERS ARE ASSUMED TO BE IN THE FOLLOWING UNITS:
            DUR         SECONDS
            AMP         RELATIVE VALUE
            CF          CPS
            MF          CPS
            MI1         FRACTION OF CF
            MI2         FRACTION OF CF
            LOC         FRACTION REPRESENTING PROPORTION OF
                        SIGNAL ON CHANNEL A.
***

ACTION TRY(DUR,AMP,CF,MF,MI1,MI2,LOC):
   BEGIN
      CF := CF*FRFACT;
      MF := MF*FRFACT;
      MI1 := MI1*CF;
      MI2 := MI2*CF;
      TRYIT(DUR,AMP,CF,MF,MI1,EXPO(MI1,MI2,DUR),LOC,EXPO(AMP,AMP/512,DUR));
   END;

***   SRATE IS A LIBRARY FUNCTION THAT RETURNS THE SAMPLING RATE.
      SIN4 IS A LIBRARY FUNCTION THAT RETURNS THE SINE OF ITS ARGUMENT.
      HALT IS A LIBRARY ROUTINE THAT TERMINATES SYNTHESIS.
***

ACTION SRATE: EXTERNAL;
COMGEN SIN4(RAD): EXTERNAL;
COMGEN HALT: EXTERNAL;


***   FILLSINE INITIALIZES THE ARRAY SINE TO A STORED SINE FUNCTION   ***

COMGEN FILLSINE:
   REPEAT FOR I := 0 TO 511:
      SINE[I] := SIN4(I*6.2832/512);

***   BEGIN COMPOSITION DRIVER CODE   ***

BEGIN
   SR := SRATE;
   FRFACT := 512/SR;
   FILLSINE;
   AT 5, BEGIN
      TRY(.5,1,500,1400,.8,.2,0);
      AT 1 TRY(.5,2,100,600,.9,.3,.5);
      AT 2 TRY(2,1,1000,5000,.6,.1,1);
      X := 1;
      AT 5 REPEAT WHILE X < 15:
         BEGIN
            AT X TRY(.3*X,1,130*X,130*(16-X),2,.5,1/X);
            X := 1.2*X;
         END;
      AT 1.5*X, HALT;
   END;
END;
/
```

CONCLUSIONS

I won't argue the superiority of an integrated system such as the IUCMS over a set of programs that cannot communicate with each other; I think the advantages of the former are quite obvious. In fact, the IUCMS can justly be criticized for not going far enough. Some perfectly reasonable paths do not exist – for example, composing program output to be used as anlysis program input. Some desirable capabilities (not mentioned by Erickson or my 1974 article) are not included; see the long wish list recently given by Peters [4], which includes numerous interactive facilities. And, not surprisingly, every individual program in the IUCMS has its shortcomings. Regardless, I feel the IUCMS has considerable power now, and all of its programs are available to anyone who wishes to use and/or improve them. (Note however that several are copyrighted and permission is only granted for nonprofit use. Notices are in the source code of these programs.)

As mentioned before, the "organ keyboard and tablet programs" of Figure 1 are not properly part of the IUCMS. They run on the Indiana University Physics Department's Xerox Sigma 5 and generate MUSTRAN data from "unusual peripheral equipment" preferable in many ways to keypunching. They, and a music-oriented text editor, are still being developed under the supervision of Gary Wittlich; most of the programming has been done by Rosalee Nerheim, Virginia Leiter, and myself.

ACKNOWLEDGEMENTS

It is now my pleasant task to thank a number of individuals. For a number of years the Wrubel Computing Center of Indiana University was home to an informal computer music group consisting of George Cohn, Gary Levenberg, Bruce Rogers, and myself. I can only say that, without the participation of each of these people, the IUCMS would never have reached the state it is in today. I am also grateful to the Wrubel Computing Center itself, a truly unique institution, and to several of my other colleagues there, in particular Jim Hettmer, for many helpful discussions (not to mention debugging assistance). Several current and former faculty members of the I.U. School of Music and their students have provided assistance, encouragement, or nuisance value; I refer especially to Dennis McGreer, Gary Wittlich and Iannis Xenakis. Lastly, I owe some of the major ideas in MUSC to Gary Nelson.

REFERENCES

[1] Donald Byrd, "A System for Music Printing by Computer", *Computers and the Humanities*, Vol 8, No 3 (May 1974), p. 161.

[2] Raymond Erickson, "The DARMS Project: A Status Report", *Computers and the Humanities*, Vol 9, No 6 (November 1975) p. 291.

[3] George Cohn, " SOUND Language", read at the First International Conference on Computer Music, MIT, October 1976.

[4] G. David Peters, "The Complete Computer-Based Music System: A Teaching System – A Musician's Tool", in *Proceedings* of ADCIS 1977 Winter Conference, University of Delaware, February 1977.

BIBLIOGRAPHY BY PROGRAM NUMBER

1. Iannis Xenakis, *Formalized Music* (Bloomington: Indiana University Press, 1971). See also Bruce Rogers, "A User's Manual for the Stochastic Music Program" (unpublished paper, 1972; available from ~~Bruce Rogers, 20 S. Maple, Mount Prospect, IL, 60056).~~
2. User's writeup available from me.
3. An early, monophonic version is described in Donald Byrd, "A System for Music Printing by Computer", *Computers and the Humanities* 8, 3 (1974). Also, a user's guide is available from me.
4. Jerome Wenker, "A Computer Oriented Music Notation including Ethnomusicological Symbols" in Barry Brook, ed., *Musicology and the Computer* (New York: CUNY Press, 1970). Extensions in MUSTRAN II are described in "MUSTRAN II – A Foundation for Computational Musicology" in John Mitchell, ed., *Computers in the Humanities* (Edinburgh University Press, 1974).
5. Very briefly described in "Music Input Languages", available from me.
6. M.V. Mathews et al., *The Technology of Computer Music* (Cambridge: MIT Press, 1969).
7. No documentation exists other than the source listing, but WAVER is very simple to use.
8. Briefly described in "Music Input Languages", available from me.
9. Dorothy Gross, *A Set of Computer Programs to Aid in Music Analysis* (dissertation, Indiana University School of Music, 1975).
10. User's guide available from me.
11. No documentation available.
12. Use explained in comments in the source program.
13. Use explained in comments in the source program.
14. Documentation available from George Cohn, Wrubel Computing Center, Indiana University.