MIT/LCS/TM-172

SOME NEW METHODS OF MUSIC SYNTHESIS

William Gerhard Paseman

August 1980

Some New Methods of Music Synthesis

by

William Gerhard Paseman

Massachusetts Institute of Technology
Laboratory for Computer Science

Cambridge Massachusetts 02139

Some New Methods of Music Synthesis

by

William Gerhard Paseman

Submitted to the Department of Electrical Engineering and Computer Science

on August 15, 1980 in partial fulfillment of the requirements for

the Degree of Master of Science

Abstract

There are two distinct sections to this thesis.

The first section discusses music composition, shows why it is a useful domain for Artificial Intelligence research and presents a set of "Design Rules" that facilitate research in the field of tonal music composition.

It begins with a short chapter presenting a subset of music theory. This chapter assumes no prior knowledge of the subject, it completely defines all terms used in the thesis, and is geared particularly toward those unfamiliar with music, those unwilling to learn standard music notation and those interested in Artificial Intelligence research.

Next, (using the terms defined in the thesis), a context sensitive generative grammar for producing pitch progressions in the major mode is introduced. It is seen that the grammar can be made context free by switching between two interpretations of the input string. A mechanism for switching from one interpretation to another when parsing sentences generated from this grammar is described. It is shown that a model of music composition, perception and improvisation fits within the framework of the grammar. This multiple view model and switching mechanism can be interpreted as a primitive "frame".

The second section describes some of the problems and issues encountered while designing the initial hardware for the Music Aided Cognition Project at M.I.T. All of the developed hardware permits computer control, performance and recording of music in real time.

The first chapter in this section discusses a machine called the Inexpensive Synthesizer/Recorder. It capable of synthesizing 14 square wave voices, each voice having a range of 7 octaves, with each octave having 12 bits of frequency control. Its purpose is to allow the user to record key depression times, key release times and key impact velocities when playing a keyboard piece. Its primary constraint was low cost, allowing many copies to be made. Its microprocessor interface allows it to be easily controlled by many different means, including home computers. The complete schematics for the synthesizer and the controller are provided as an appendix.

The next chapter discusses an oscillator which synthesizes sound using 32 sine or 8 FM waveforms. The machine can be easily expanded to produce 256 sine voices and 64 (or more) FM voices. All sine waveforms in both types of synthesis are weighted with two independent coefficients. Microprogrammable firmware allows one to produce sound by a limited number of methods other than sine summation or FM synthesis.

Name and Title of Thesis Supervisor:

Stephen A. Ward,
Associate Professor of Electrical Engineering and Computer Science

Key Words and Phrases:

Artificial Intelligence, Music Composition, Real Time, Music Synthesis

Acknowledgements

I would first like to thank Steve Ward for allowing me to work in the Real Time Systems Laboratory. His support was invaluble, as was the time he devoted to debugging my hardware, programming the synthesizer control software and providing the initial log-synthesizer idea.

I would also like to thank the members of the laboratory, particularly Jon Sieber and Al Mok for helping me in my work at crucial times through the course of the project.

Clark Baker, Rae Mclellen, David Godeau, Mark Johnson, Jeff Arnold and Tom Teixeira patiently (and repeatedly) helped me attack many of the problems that I encountered when working with some of the tools available at the Laboratory.

I would like to thank some people recently concerned with the project. Matt BenDaniel helped me find a mistake in the controller board. David Fry recently produced some computer composed stochastic music in another section of this university.

The conversations I had with members of the M.I.T. community, particularly Jon Sieber, Marvin Minsky, John Amuedo, Al Mok and David Levitt were too few and too short. These conversations are the reason I came here and are what I will value the most when I leave.

Although this thesis is not written in grammatically correct, smoothly flowing English, it isn't because the readers didn't try. I would like to thank them; Jon Sieber, Al Mok and Jeff Arnold particularly.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract number N00014-75-C-0661.

Dedicated to *Das Glasperlenspiel*

CONTENTS

[illegible]

1. Chapter One: Introduction

1.1 Motivation for Artificial Intelligence Research in Music

This work's basic goal is to show that A.I. research can be furthered through the study of music.

Why should A.I. investigators care about studying music? One reason lies in the area of *knowledge representation* research.

The problem of knowledge representation plays a key role in most A.I. questions. Those who have studied the problems of knowledge representation generally focus their attention on one of several broad areas of application. A prominent area of application is "Real World" knowledge representation.

However, the world is a big place full of diverse things used in diverse ways. In developing a formal system of representation for everything in it, one quickly runs into big problems. There are usually one of three reactions to these problems. The first is to "patch" the system. This means that the "well defined" system is not as well defined as it was. The second is to say--- "Well, no system can respond to every problem, but this one responds to most". These two reactions lead to a third approach, one of using multiple representation systems, each viewing the world from a different perspective. In all cases, the system developer essentially admits that the problems are very difficult to get one's arms around and that the domain is too complex for one system to handle.

Therefore, it has proved worthwhile to search for problem domains smaller than the real world in which to do A.I. research. Many workers have done this by studying problems arising in a less complex subset of the real world. Indeed, the A.I. literature abounds with "Blocks World" problems. This is a valid approach, but it is apparently quite hard to leave the "Blocks World" once you have entered it. That is, the way one extends a developed system back to the "Real World" is not apparent. One reason for this is that there is no real metric to determine what subsets of real world representation problems follow one another. If real world representation problems were ordered historically, then A.I. would be easier, but we have had the same way of viewing things for many millennia. The recorded philosophy of our views shows development, but the views themselves have no available record of development (that we can find).

However there is simpler, but not trivial, problem domain having a relatively long recorded history of development. It is music composition.

In this field, the problems are well ordered (historically) in terms of difficulty, well documented, and relatively complete. (Problems in music composition are partially ordered in time, if you will). Therefore, in doing A.I.-Music research, one can start at music composition's primitive beginnings and work forward in time, developing theories about how music was structured, how it was composed and how it was perceived. As naive theories are proposed and discarded to make way for newer (and hopefully less naive) theories, one should get a better and better idea of the correct way to model the problems involved (and more importantly, the wrong way to model the problems).

This is the basic idea. One approach to it would be to apply some of the simpler A.I. paradigms at the beginning of the music composition "time line" and increase the complexity of the applied paradigms until the analysis-synthesis problems for the studied works at that time were "solved". One then would increment "time" and repeat the procedure as far up the time line as possible.

Note that this idea can be used building models which are "derivatives" of microworlds. Instead of constructing a series of microworlds, each representing an element of some time progression in music history, one constructs some absolute model initially and specifies each successive model as being the old model, plus a set of differences.

Although things such as "cultural factors" would be an important difficulty here, this method could potentially yield quick returns to the A.I. community. Simple bugs existing in standard paradigms might be more quickly uncovered and clarified by using the "simpler" problem space.

As the avowed purpose of this project is to help solve "Real World" A.I. issues, several immediate, and valid, objections to this choice of domain can be raised. The first is "Perhaps the path from the 'Blocks World' to the 'Real World' is difficult, but the path from the 'Music World' to the 'Real World' is probably non-existent." This may be true, but it misses part of the point. If A.I. researchers are able to completely work out the problems in any domain, the insights gained from the experience will help them in any other domain that they care to attack. The area of music is simple enough and well documented enough to allow A.I. researchers to achieve considerable success in a relatively short period of time.

Conceivably, there is another benefit that can arise from this choice of knowledge representation domain. That is in A.I. system testing. One idea of an appropriate test for an integrated Knowledge Representation and Control system would be something like a Turing test. For example, in the systems proposed here, one would ask the system to compose a piece of music in the style of some period, artist or group. In a linguistic knowledge representation system, this would be roughly equivalent to asking the computer to say something reasonably intelligent about a particular topic in the way that a given person would. This idea is admittedly a little naive, however such tests administered to music generation programs show promise of being simple without being simplistic.

This idea has implications in the way a music system is built. Since any given person "enjoys" or "interprets" music differently than any other given person, having a machine that composes "good" music can not be the goal here. Rather, the goal is to produce a machine that can produce music in the style of certain composers. Therefore, a typical computer routine would be called "Bach Counterpoint" as opposed to "Counterpoint".

This illustrates yet another benefit. Master composers developed methods of directing the interest (and sometimes even the emotions) of their audiences simply by controlling the sounds that they heard. Records of how they developed particular works before they were publicly performed is often available to researchers. The results of such "autograph" studies were used to formulate many of the ideas reported here.

If the idea of spontaneous generation of classical music by computer programs fails, it will still have shown something if it produces a system by which layman to the musical world are able to generate pieces of music relatively easily and these works are similar to the works of masters. Still, to show something, the model of music composition developed must be intuitively or instinctively understandable. It won't do to have the work of the layman be limited to operations like selecting probability kernels or note occurrence matrices.

In summary, the problems encountered in musical analysis and synthesis are identical to many of the important problems encountered by cognitive scientists in all fields. Music differs from most such fields in that the material studied has a well recorded history of development. Intuitively, this recorded history seems to order the problems that a researcher can attack. This intuition alone provides strong motivation for research in the field if the proper tools are available. In this thesis, I provide some of the necessary tools.

1.2 How to Lure the Right People into Doing Music Research

Enticing A.I. researchers into the study of music is difficult. Most of the people that should be working on the problems in this field "know nothing about music" and won't even consider studying it. Therefore, one goal of this work is to provide tools which make the study of music more attractive and tractable to them. The question is then, what tools should be made available.

A similar problem has been worked on recently by Mead and Conway[1].

Mead and Conway are interested in the design of very large scale integrated (VLSI) circuits. Traditionally, work in this field required a knowledge of semiconductor devices, which required a knowledge of solid state physics, which in turn required a knowledge of calculus and classical physics.

Mead and Conway virtually eliminated the need for all of this background knowledge by compressing it into a simple two page set of design rules. In addition, they facilitated the actual chip design process by introducing a simple language that completely specifies all the parameters needed by the companies who fabricate the VLSI device. Learning these rules and using various interfaces to this language allowed specialists in the fields of algorithms, digital logic, topological complexity and systems architecture to directly participate in solving VLSI related problems (as well as problems in their own field). In short, by removing the stifling detail, Mead and Conway induced a lot of valuable talent to participate in the field.

Some may say that the design rules are not a "compression" of the information needed to design chips, but rather a gross oversimplification of key design concepts. This may be true, but it has not hindered people ignorant of device physics from advancing the state of the art in the VLSI field. This result is important and it is why their approach should be transferred to the A.I.-music field.

This thesis can be viewed as a progress report on the attempt to duplicate the pedagogical approach (and its consequences) of Mead and Conway. The function of the hardware section is to describe progress on the music project equivalent of Mead and Conway's layout language.

The music theory section attempts to parallel Mead and Conway's design rule idea by specifying several sets of composition rules for the various (extremely) limited musical examples.

The basic model of "music" central to this work is shown in the figure labeled "A Model of Music". This figure is a block diagram illustrating the transformations between four different representations of music. The right hand side of the diagram is concerned with some aspects of music synthesis and the left hand side equivalently with music analysis. In an ideal system we would extract or add parameters such as "Performance Practice" and "Sound Objects" as we transform the music from one representation into another. User interaction would occur through editors at each level.

The delineation of music shown in the figure is central both to the hardware development in this project and to this work's ideas about music composition, representation and cognition.

This diagram explicitly indicates an interest in both music analysis and synthesis. It would certainly be easier to simply indicate an interest in analysis. Those interested in having the computer compose music are strongly opposed by many different factions for a variety of reasons. However, attempting to do both analysis and synthesis results in the discovery of problems (and the solutions to problems) in music analysis that would not arise as quickly from a study of music analysis alone.

As an aside, the ideas represented in the diagram are admittedly a product of the times. The concept of using procedures to map data from one representation to another wouldn't have occured if computer science hadn't provided the parallel. The tendency of the scientific community to use modeling mechanisms that are currently "fashionable" is annoying. Past examples of "fad" modelling mechanisms are fluid mechanical models of the human nervous system and models of high level thought processes based on simple control loops. Now, modeling thought using computer science concepts is in fashion. Unfotunately, this paper can only follow suit.

To test the theories and results produced at any level in the diagram, support at the other levels must be provided. This support must lie in both the software and hardware domains. The majority of the initial hardware support should occur at the Performance Schedule and Acoustical Signal levels. One approach to the hardware support configuration would be to associate a system bus with each of these levels.

The figure labeled "The Music Project Bus Scheme" shows one such bus scheme. The "Notated Score" and "Score Kernel" section of the "Music Model" figure is embodied in Lisp programs in a List Processor, the "Performance Schedule" is centered around the Micro-bus and Nu-bus, the "Acoustical Signal" appears on the S-bus. The appropriate place to start work on this or any similar system is with the Inexpensive Synthesizer-Recorder and/or the Digital Signal Processor. The hardware section of this thesis discusses these two aspects of the system.

1.3 Hardware Goals and Constraints

The most stringent constraint placed on all our music producing systems is that they produce music in real time. There are several reasons for having real time (as opposed to non-real time) music synthesis. One is that real time synthesis will allow users to compose music in an interactive environment. This means that they can compose more quickly and with less frustration than non-real time systems allow. Another reason is that the size difference between the data representations at the Performance Schedule and the Acoustical Signal levels is large. If the "Synthesis" link joining them is fast, not as much data storage on the Acoustical Signal level will be required as would be otherwise.

Since each of our Computer Music producing systems will be centered about a computer system of some sort, it is important to have an idea of the current relationship between hardware, software and price in electronic sound synthesis. One way to do this is to order the currently available sound synthesis features by desirability.

One possible list is shown below.

basic square wave monophonic scale generation in any desired octave
polyphony
envelope shaping
wave shaping
multiple tonalities
frequency modulation and other non-linear techniques

In producing any of the above list's features in the Inexpensive Synthesizer or the Digital Signal Processor, it is desirable to let the relatively cheap processing power of microprocessors take the brunt of the signal processing load. This means that the final instrument will be software as opposed to hardware intensive.

Slow processing speed is the biggest problem encountered in using "code" to generate the above features with current microprocessor instruction cycle times. This is where a suitable compromise between the price, software and hardware aspects of the design is important. It means that microprocessors must be used more for control than pure synthesis purposes.

The inexpensive systems only possess the first few items on this list since their primary purpose is to act as cheap scratchpad devices. Also, the final inexpensive synthesizer is relatively applications independent, that is, little interfacing is required to attach the machine to either a computer or any other control device (such as a keyboard). This was an important constraint in the final instrument.

1.4 A Microprocessor Applications Learning Example

The circuitry developed and built for this project over the past 9 months has made heavy use of microprocessors. Microprocessors limited the scope of the project in many ways. One signal processing algorithm that shows how badly microprocessors can perform is shown below.

High Resolution Digital to Analog converters are an expensive piece of equipment. Perhaps they can be done away with entirely by using a microprocessor implementation of an analog modulation technique called Pulse-Duration Modulation (PDM). The advantage of using PDM for encoding is that one can recover the original signal by simply feeding the encoding through a low pass filter. This method would eliminate the need for a DAC and a track on the tail end of a Digital Synthesizer.

PDM is certainly a viable encoding mechanism in the analog domain. It works by encoding the analog signal's instantaneous amplitude in the pulse width of each sample pulse. One algorithm for this method of encoding is shown in the figure labeled "Pulse Duration Modulation". Here, a sawtooth is added to the sample value and the result is compared to a threshold value. If the result is above the threshold, the serial output sends a "1". If the value is below the threshold, the serial output sends a "0".

Replacing the DAC using a simple microprocessor algorithm based on the figure seems reasonable. The algorithm first inputs the value to be converted. Next, it outputs a "1" and initializes a comparison counter (which can be assumed to be the length of the value to be converted) to all "0"'s. It increments the counter until it matches the input value and then drops the output to "0". Finally, it waits until the sample period ends and starts over again.

Let's calculate what type of instruction cycle time is needed to execute this algorithm. We will choose a sampling frequency of 10 Khz, and let the amplitude take on 256 discrete values. This is certainly a minimal system. The synthesizable waveforms are bandlimited to signals having a frequency content of less than 5,000 hz, and the signal to noise ratio is only 52 dB. Still, with a worst case input (consisting of the minimum amplitude), this means that we need to perform the algorithm at a frequency of (10 Khz)(256) = 2.56 Mhz! This is about the frequency at which the clock of most microprocessors operate. Therefore, the idea is clearly unsuitable for microprocessor implementation. Why is this so? There are two reasons.

A microprocessor has an unacceptably high overhead when it has to execute a trivial subroutine. In the above example, the trivial subroutine is in the inner loop of the algorithm, so the overhead occurs all the time. A DAC with a PDM output would have its ramp and compare functions implemented in hardware.

Even more importantly, most DACs operate on the input binary word using a parallel conversion algorithm. Increasing the length of the input word to a DAC requires a linear increase in hardware (up to a point) to keep the conversion algorithm a constant time operation. Increasing the word length of the input word to the processor algorithm requires an exponential decrease in processing time for a fixed sampling frequency. This exponential time versus linear hardware tradeoff hurts the microprocessor considerably.

As this example shows, current microprocessor implementation of some hardware analog (and digital) functions is still very limited in what it can do.

2. Chapter Two: Background

2.1 Prior Research in Computer Music Composition

All programs that attempt to compose in different styles (jazz, classical, modern) use parameters obtained by analysis on some level.

In studying methods of music analysis, one sees that most operate by music (or score) decomposition. It is the methods of decomposition and the decomposition's atomic units which serve to differentiate one analysis proceedure from another.

Some of the earliest attempts at computer composition tried to view music serially[4]. They decomposed music from left to right and composed by a generate and test proceedure. Notes were randomly generated, and allowed to pass into the output file if they didn't violate local constraints. Such methods produced reasonable music locally, but the overall structure was not coherent. Attempts were made to get around this problem. One method involved dividing a composition into sections, and having strict specific rules for each section. This provided further "filtering" for the probabalistic note generation proceedure and did not let as much "noise" through. This produced somewhat better compositions, but they were usually locked into one very strict style.

Rader[7] has developed a method for composing simple rounds using a set of composition rules which are monitored by set of meta-rules. The long term structure in this work was therefore a function of the round "style".

Recently, methods of musical analysis based on work done by Heinrich Schenker[9] have been proposed and extended by Kassler[2], Smoliar[13], Lehrdahl-Jackendoff[5] and others. Kassler and Smoliar framed their ideas in terms of computer programs. Unfortunately, detailed results of Kassler's latest research in this area are not available[3]. Smoliar has translated some of Schenker's work into a series of Lisp functions. A small manual describing the functions is available[13].

The initial gross structure of the analysis method proposed here was developed without any knowledge of the above authors' work. There are similarities however. As the gross structure is described below, the similarities and differences will be described as they arise.

2.2 A Model of Music Perception

Suppose, as one listened to a piece of music, one analysed it by several different procedures simultaneously. Suppose further that these analyses could not nicely describe the music in terms of their primitive operators throughout the length of the entire piece, that is, that there were a few "rough spots" present in each procedure's analysis. What are some of the details one would notice if these analyses were laid out, measure by measure, one beneath the other?

One is that the rough spots in the different forms of analysis would occur in different places in the music. It seems clear that an ultimate representation of (well composed) music at the perceptual level should be free of these rough spots. To get rid of them, one can take advantage of the multiple analyses by tying them together into one global representation of the musical piece. This can be done by switching from one analysis to another before hitting these rough spots. This process can be thought of as periodically switching viewpoints of a piece of music as the piece progresses. An analogy that might be made here is one of walking along side a cylindrical crystal prism. Inside the prism is the score. As we proceed down the prism's length, we look through the facet that currently gives us the best view of the music. (This analogy is reminiscent of those used in Minsky's Frames[21] and Sussman's Slices[23].)

As these representations of musical knowledge are viewed, problems could arise as one proceeded down the crystal column. If one arbitrarily switched viewpoints, one could switch from the end of one representation into the middle of another representation. This is a problem because it would seem best to switch from the end of one representation to the beginning of another. We will make switching at a boundary a constraint in the system of viewpoint analysis that produces our global representation.

Suppose there were cases where there are no "beginnings of representations" that the analyst can switch to, in these cases, the analyst will have to "back up" on some earlier analysis in order to find the most appropriate place to switch. This would correspond to confusion on the listener's part.

Sometimes this procedure may not suffice. If that's the case, then a new lexical viewpoint may have to be formed which is a distortion of a more standard lexical viewpoint. This distortion could be viewed as "an imposition of additional constraints on the syntactic structure of an earlier lexical analysis". Realistically, this is nothing more than altering the original conception of reality in order to add functionality to the model.

As an aside, one recalls that one of the key problems in the Frame paradigm is one of control. How does one know what frame to pull in next without having the search tree grow combinatorially? Assuming that this model is valid, investigating how good composers lead people into switching their "viewpoints" without confusion as a score progresses could lend some insight into this problem.

It has been implicitly stated that each of these analyses is a complete description of the piece. For example, one analysis may be melodic, another harmonic, depending on how the listener is viewing the piece at the time, but each completely specifies all the information that the listener needs to know at that moment. Can each of these viewpoints be further subdivided? It is assumed here that the answer is yes. It is further assumed that each of these "atomic analyses" are only partial analyses, that a complete description of the piece can only be obtained by combining two or more of the partial analyses. The idea of many partial analyses is central to the work of Lehrdahl-Jackendoff and other disciples of Schenker.

2.3 Methods of Analysis

One type of "atomic analysis" will be specified here. It is Tonal Analysis, which is transcription of a part of Schenker's work into a context sensitive grammer and a set of meta rules.

Some view musical composition as a tangle of interrelated constraints. It is suggested here that the tangle be attacked by grabbing hold of several parts in the tangle and pulling them apart. Naturally, they will not come totally apart, there will be connections between them. (Picking an analysis proceedure that, in general, minimizes the total number of these interconnections may be a good heuristic for determining whether a given "n" part description is better than another "n" part description.) It is felt that focusing the total analysis around these "almost-hierarchies" will give a more coherent picture of musical structure than simply specifying the constraints connecting them together.

3. Chapter Three: Some Music Theory

3.1 Of Strings, Boards and Pegs

Take a board. At either end of the board, mount a peg. String a wire between the two pegs. Pluck the wire. A sound will result. This sound is due to the wire vibrating at its *first harmonic frequency.* We will give this sound another name, we will call it the wire's *tonic pitch.* (*Pitch* and *frequency* are terms that will be used interchangeably.)

Clamp the wire to the board at its midpoint. With the wire clamped to the board, pluck one side of it. A sound different from the first will result. This second sound, produced by the wire vibrating at twice the frequency of the first sound, is called the wire's *second harmonic frequency.* Sounds whose frequencies differ by one factor of two are said to be an *octave apart in pitch.* Sounds whose frequencies differ by only factors of two are said to belong to the same *pitch class.* Therefore the first sound's frequency and the second sound's frequency belong to the same pitch class (that of the tonic) and are an octave apart.

Reclamp the wire to the board one third of the way from one peg to the other. With the wire clamped to the board, pluck the shorter side of the wire. A sound different from the first two will result. It is the result of the wire vibrating at three times its original frequency. This is called the wire's third harmonic frequency. The third harmonic frequency is given another name. It (and each of its pitch class equivalents) is called the wire's *dominant pitch.*

In general, the nth harmonic of the wire is produced by clamping the wire down one nth of the distance from one peg to the other and plucking the shorter side. A frequency n times that of the tonic will then result.

The wire's fourth harmonic frequency is one octave above its second harmonic frequency. Therefore, the wire's fourth, second and first harmonic frequencies belong to the same pitch class.

The wire's fifth harmonic frequency (and each of its pitch class equivalents) is given a special name, it is called the wire's *mediant pitch.*

The wire's sixth harmonic frequency is the pitch one octave above the dominant. The eighth harmonic frequency is an octave above the fourth harmonic frequency. The seventh and ninth produce new pitches.

Now that some of the relationships between the first nine harmonics of a vibrating wire have been made clear, an interesting observation can be made after introducing four new terms. These terms are octave membership, scale, pentatonic scale and system of intonation.

3.2 Scales and Systems of Intonation

If the frequencies of a set of pitches are divided or multiplied by two until they lie between the pitch designated as the tonic and the pitch an octave above the tonic, then these pitches have been made *members of the same octave.* If these pitches are then sorted by ascending order of frequency, they form a *scale.* If these operations are performed to the first, third, fifth, seventh, and nineth harmonics of the above example, the *pentatonic scale* is produced. (In the pentatonic scale, the dominant and the mediant have frequencies 3/2 and 5/4 that of the tonic.)

In order to perform most music, one must choose a *system of intonation*, that is, a fixed set of pitches, in which to play it. Separate cultures around the world have independently invented and used systems of intonation based on the pentatonic scale for many millennia. The ancient Scottish, Chinese, African, American Indian, East Indian, Central American, South American, Australian, Finnish and Balanese cultures, just to name a few, used variations of the pitches forming the pentatonic scale in their music. However, *no* culture uses the pentatonic scale in its pure form. The pentatonic pitch varied in their systems of intonation is usually the one based on the seventh harmonic. This means that although the series of frequencies found in the integer harmonics of vibrating objects are related to the pentatonic scale (and all other scales), they are not the only influence that produced them.

Documenting all the influences that led to the system of intonation in popular use today is beyond the scope of this thesis. The current result of this tonal evolution is a system of intonation called the *equal* *tempered* system. The equal tempered system is based on a scale of 12 pitches, each pitch's frequency separated from that of its neighbor by a factor of the twelfth root of two. This twelve pitch scale is called the *chromatic scale.* The pitches in this scale are called members of the *chromatic collection.* Members of the scale, spanning two octaves and ordered by frequency, are sometimes designated by the symbols:

C C# D D# E F F# G G# A A# B C' C#' D' D#' E' F' F#' G' G#' A' A#' B'

The number of quotes (') indicates which octave each pitch class belongs to. The twelfth root of two factor between notes is called a *semitone.* The term "semitone" is used with words denoting distance. For example: "The distance between any two adjacent scale members, such as C# and D, is one semitone."

3.3 The Diatonic Collection

Seven members of the chromatic scale have a fundamental role in many styles of western music. These styles of music emerged before the time of Bach and continued in strength till time of Wagner. They are also present in popular music today. These seven members of the chromatic collection form a group called the *diatonic collection.* The diatonic collection can be constructed using the first, third, fifth, sixth, eighth, tenth, and twelfth members of the chromatic scale (relative to the tonic). If a piece of music uses a diatonic collection chosen in this way, it is said to be written in a *major key.* A scale formed from these members is called the *major scale.* If a piece of music is written in G major, the composer chose G as the piece's tonic and the associated major scale would be:

G A B C' D' E' F#'

The semitone interval between these members of the chromatic collection form the pattern 2,2,1,2,2,2,1. If, relative to the tonic, the music produced is based on the semitone intervals 2,1,2,2,1,2,2 , then the piece is in a *minor key.* Suppose a performer played a falling diatonic scale running over five octaves . If one came in after the performance started and left before it finished, one could not determine if the scale had been played in a major or minor key (unless it was indicated by the performer "stressing" certain notes as he played). The reason is that one would not know what the tonic pitch of the scale was. A section of that five octave pattern based on intervals is shown below:

. . .1,2,2,2,1,2,2,1,2,2,2,1,2,2,1,2,2,2,1 . . .

Therfore, one can conclude that the concept of a tonic pitch is important in diatonic music.

Note that nothing has been said yet about absolute pitch. As all the frequencies in the well tempered system of intonation differ from one another by a fixed ratio, one need only specify one pitch in order to specify all of them. "Standard" pitchs have been specified several times in the last several hundred years. The tendency of each new standard is to increase the frequency of the system used up to that point. The result is that the current standard is almost one semitone higher than the standard used in Beethoven's time. This also means that Beethoven's works written in the key of C major are now performed in C# major.

3.4 Notes on Note Notation

Standard music notation employs a floating point unary representation for encoding pitch(i.e. Clefs and notes on a staff). This method is advantageous for performance and some types of analysis. It will not be used here.

The method used here is geared toward representing single voice music written totally in a major key. It designates pitch names with the numbers 1,2,3,4,5,6 and 7. Each number represents a member of the diatonic collection sorted by pitch into the major scale. Mapping this representation to sound is easy. If the key of C major were chosen to be the major key, 1, 2, 3, 4, 5, 6, 7 and 1' would map to C, D, E, F, G, A, B and C' respectively.

Close relatives to the tonic, dominant and mediant of the pentatonic scale are present in the major scale. The relative to the tonic is labeled "1", the dominant's relative is labeled "5", and the mediant's relative is labeled "3". Remember that the members of the major scale are separated by the semitone pattern 2,2,1,2,2,2,1 , and that one semitone is the twelfth root of two. This means that the ratio of the major dominant's frequency to the tonic is 2**(7/12) = 1.4983 (Recall that the pentatonic scale's value was 3/2 = 1.5). And the ratio of the major mediant's frequency to the tonic is 2**(4/12) = 1.2599 (Recall that the pentatonic scale's value was $5/4 = 1.25$). These values are so close that the distinction "major dominant" or "pentatonic mediant" will no longer be made. Just the terms tonic, mediant and dominant (equivalent to 1, 3 and 5 in our notation) will be used.

The numbers in our notation will also be called *scale degrees.* The octave of the scale degree will be notated using single quotes ('). For example; 2' is the scale degree located between the mediant and the tonic in the second octave.

3.5 Intervals

This work is interested in composing and modeling the perception of pitch progressions in major keys. As this is the case, it is first necessary to determine the relationships between the pitches and groups of pitches which form these progressions, such as melodies. One immediate observation is that the vast majority of melodies have no frequency jumps spanning more that an octave. (This is due in part to the limitations of the instrument used to perform most melodies; the human vocal tract.) Therefore, in examining the local relationships between two pitches in a melody, one could construct a table with a two octave span of the diatonic collection on each axis. The intersection of each axis entry could be used to record information about the experiment performed.

In one such experiment, it was determined how "unstable" any two sucessive pitches sounded when played one after another. This instability could result from one of two feelings, either that the pitches formed a progression, and the progression was incomplete, or that the two pitches simply sounded as though they didn't belong together.

The figure labeled "Intervals Between Members of the Diatonic Collection Forming the Major Scale" contains the condensed results of this experiment. In the figure, the diatonic members forming the major scale were separated by their semitone distances along a line. Then, all line segments with endpoints lying on a diatonic member were drawn and sorted by semitone length. As previously discussed, the seven diatonic members are represented by the numbers 1 through 7. Their pitch class equivalents an octave higher are represented by the numbers 1' through 7'.

The results of the experiment were that the groups tagged with black squares: m2, M2, a4, d5, m7 and M7, were felt to be unstable. The groups not tagged: u1, m3, M3, p4, p5, m6, M6 and p8 were felt to be stable. It was found that in isolation, all members of each group were perceived as being equivalent. As can be seen, there are only 14 groups. They are each designated by a letter-number pair. The number indicates how many different members of the diatonic scale are crossed by the segment (including its endpoints). The letters are the first letters of names given to the groups. It is not necessary to know the names to understand the work presented here, but they are presented as a matter of interest. The complete names in order from the top of the figure are *unison, minor second, Major second, minor third, Major third, perfect fourth, augmented fourth, diminished fifth, perfect fifth, minor sixth, Major sixth, minor seventh, Major seventh and perfect octave.* M2 and m2 are the minimum length intervals shown. The number of these intervals contained in each segment is recorded in the columns labeled "M2" and "m2".

In addition to the main result, this figure reflects two empirical observations that were made. First, the order in which the pitches are played does not matter. The succession 3 4 gave results equivalent to those obtained from 4 3. The second observation was the concept of octave equivalence of interval. This means that the progression 3 4 gave the same results as the progression 3' 4'.

These three empirical observations: limitation of octave jump, directional equivalence of interval jump and octave equivalence of interval mean that there are only 56 absolute intervals that can be used by melodies whose members consist *solely* of pitches belonging to the diatonic collection. (If non-diatonic members are allowed, the number of possible absolute intervals increases to 156 using these three observations.) All these 56 intervals, represented as line segments between members of the diatonic collection, are shown in the figure.

3.6 The Tonic, Dominant and Mediant

The title of this section lists the three pitches given special names in this chapter. This group of three pitches will be given a special name here, they will be called the *tonic triad.* This term can mean different things in different contexts, but here it will refer only to this group of three pitches. Note that all the intervals between members of this group: 1-3, 1-5, 3-1', 3-5, 5-1' and 5-3' are stable intervals.

Music that embodies the concept that these three tones are special is called *tonal* music. Theories that attempt to say things about such music are called *tonal music theories.*

This thesis presents a small tonal music theory. The main argument that will be presented is that diatonic pitch progressions produced in the major keys have a common mechanism associated with them. The key feature in this mechanism is that attention is constantly drawn toward and away from the three pitch classes that belong to the tonic triad. The ramifications of this theory will be presented in the following chapters.

3.7 Summary: Constraints is spelled with 'ai'

The purpose of this section was to introduce several terms. It had another purpose as well. That was to present some of the local physical constraints that exist in a major key melodic line. Assuming that the results of the presented experiment demonstrate "physical constraints" is dangerous. This assumption was not made lightly. It is based on the fact that the "experiment" describes many perceptions that have been recorded in hundreds of thousands of documents since the beginning of civilization.

But the results of the experiment are a function of time. Only a subset of the intervals now viewed as stable were considered stable 1,000 years ago. And only a subset of those stable a 1,000 years ago were considered stable 1,000 years before that. The perception of stability also varies from culture to culture, and sometimes from person to person. However, the same can be said about key concepts in the phonology, syntax and semantic structure of language. Here, as in all AI research, one must work with what material one has. The material presented in the figure is better than most.

Knowing the physical constraints in any problem is important. As much A.I. work has shown[22],[21],[23], when making a breakdown between two elements that are physically related to each other, it is best to make the breakdown along some percieved physical boundary. The initial boundary perceptions observed in this thesis are those that have been described here.

4. Chapter Four: A Tonal Grammar

Semper idem sed non eodem modo

4.1 Music

In contemporary Western Society, "Music" is a word used to denote a large number of different experiences. The catholic meaning given to the term is probably due more to people's inability (or lack of desire) to discriminate between perceived events than to anything else. The same can be said for the term "Music Theory". Therefore, since the usage here is very specific, a definition of "Music Theory" will be presented. Recall that no definition can be "wrong" (by definition!), however it can prove to be worthless. It is, of course, left to the reader to judge the merits of the definition himself.

A Music Theory is a formal system, some of whose objects are to be interpreted as auditory events.

There is nothing new or startling about the above definition. It is essentially a recapitulation of the "Music Model" figure with the added constraint that the indicated procedures and symbols satisfy the requirements of a formal system. It is stated here explicitly in order to still the majority of arguments made concerning ideas in this work. Once it is stated, the only real argument left is whether or not work based upon this viewpoint is worthwhile.

The music theory presented in this chapter is concerned with the tonal progressions of music written in the major mode. At the beginning, it totally ignores questions of pitch durartion and stress. It will report observations on pitch progressions from a number of different levels. Representing knowledge obtained from one set of these observations is the subject of the next section.

4.2 Grammars

There are a variety of ways to express knowledge in a formal system. On the level of music examined here, knowledge is expressed in terms of production rules. Production rules are used because on a local level, music consists of multiple, non-trivially different, independent states. This makes it feasible to write multiple, non-trivial, modular rules[18]. A benefit of this method of representation is that it draws a clean line between pieces of knowledge and the processes that use them. This allows this work to present these production rules as a set of "design rules" for a subset of music.

The work upon which these rules are based was written in the early part of this century by an Austrian music theoretician named Heinrich Schenker[11][9]. He and his modern day disciples expressed their ideas in what are essentially sets of context sensitive rules. Some of the rules can be found in Regener[8], Kassler[Kassler75] and Smoliar[13] as well as in Schenker. The subset of rules presented here are designed primarily for use in the upper line of a piece of multivoice music. These rules are expressed procedurally below.

All music has a main structure. This stucture is called the *fundamental descending line*. The fundamental descending line has one of three forms, they are:

3 2 1, 5 4 3 2 1, or 1' 7 6 5 4 3 2 1

Rules govern the addition of pitches to (and the insertion of pitches into) the fundamental descending line. These rules are called *triadic repetition, neighbor insertion, triadic insertion* and *step motion.*

The rule of triadic repetition states that one pitch may be replaced by two successive pitches if the pitch is a triadic member.

The rule of neighbor insertion states that any repeating triadic member may have one pitch one scale degree higher or one scale degree lower inserted between the repeating notes.

The rule of triadic insertion states that a triadic member may be inserted between any two notes if no unstable intervals are created.

The rule of step motion states that any two pitches may be joined by a complete ascending or descending series of intervening scale degrees or *steps.*

The figure entitled "Part of J. S. Bach's Two Part Invention Number Eight" shows a derivation of a well known pitch progression using these rules. The results of each derivational step is indicated in bold letters. It is incorrect to assume that this is a derivation of the piece. It is simply a convenient way to show some of the rules. Note that the triadic repetition must precede a neighbor insertion. An ordering not shown on this picture is that triadic insertion preceeds step motion. This derivation is defined as being a *synthetic* or *top down parse* representation of the piece. Another representation is shown at the bottom of the figure, this is a *synoptic* viewpoint of the piece, as no ordering on how the derivation was done is shown. A third representation would be an *analytic* representation or a *bottom up parse.*

An abbreviation of the rule names will be used in the figures. The descending fundamental line will be labeled F, neighbor notes will be labeled N, triadic insertion will be labeled I, triadic repetition will be labeled R, ascension and descension will be labeled A and D or S.

Each of the above rules expresses a facet of the idea presented in the last chapter. There, it was theorized that some classes of tonal music use methods to direct attention to and from the members of the tonic triad. The rules expressed above do this. Note that both the concept of step and neighbor involve intervals that were defined in the last chapter as being unstable. The system of rules guarantees that the unstable intervals will come to rest on a tonic triad member. The concept of repetition reenforces the presence of a triad member. The fundamental descending line connects members of the triad by step motion. Triadic insertion guarantees that the major sections of the progression are reinforced with members of the tonic triad. From the fundamental line to the most local structure, it is clear that the concept of stability, instability and resolution to the tonic triad are embedded deeply into the framework of the system.

4.3 Major I

These rules are all defined with known terms. It is easy to state them using a grammar in terms of pitch. This is done in the figures entitled "Major I".

This grammar, like all others, consists of four things: *terminals, non-terminals, productions* and the *sentential symbol.*

Here, S is the sentential symbol, the key symbol from which the tonal representation is derived. The production using the sentential symbol consists of the string of symbols following the sentential symbol. The arrow means "can be replaced by". The dark vertical bars located on the right hand side of the productions represent the "or" operation. The @ sign simply designates where the melody starts. This means that the descending fundamental line production can be read as:

S can be replaced by **@EDC** or **@GFEDC** or **@C'BAGFEDC.**

The lower case numbers are terminal symbols. They are what the non-terminal symbols (the letters in bold) must eventually resolve to. This means that the first triadic repetition rule can be read as: **C** can be replaced by two **C**'s or by the tonic.

The triadic insertion rule is essentially a grammatical implementation of the interval table presented in the last chapter. A shorthand notation is used to express this knowledge. The set membership symbol indictes that the designated non-terminal can resolve to any terminal in the brackets. The information could have been represented just using "-->" symbol, but this version was viewed as being more concise. Note that although this notation has the disadvantage of indicating the resolution of non-legal intervals(i.e. 4 4, 4 6 or 4 5' in the UV production for example), these intervals will never appear in the string due to the structure of the language.

In this grammar, the dark symbols in the ascension-descension productions are non-terminals. In the ascension-descension productions, a shorthand notation for the concept of octave equivalence is used. The quoted (') symbols designate that both sides of the indicated production are to be raised in pitch one octave.

For example, J' refers to the J production where all entries to the right of the arrow are raised one octave in pitch to 4', 5', 6' 7' and 1". Each of the rules are octave equivalent, that is, each side can be quoted or unquoted at will in order to produce a new rule. However one must quote both sides when doing this. 1 does not produce 1'1' for example.

In the neighbor insertion, ascension, descension and the triadic insertion productions, two non-terminals appear on the left hand side of the production. This means that the rules are *context sensitive.* If there is only one non-terminal on the left hand side of the production (and nothing else), as in the triadic repetition rule, the rule would be *context free.* The distinction between the two becomes apparent when constructing automatic procedures for the analysis of strings generated by these rules. Automatic analysis procedures or bottom up parsers are much easier to construct for context free procedures than for context sensitive ones.

Note that this grammar consists solely of *string lengthening rules.* This means that one can always determine whether or not a sentence is a member of the grammar by a very simple procedure. The procedure is to first note the length of the sentence in question. Next, view derivation as a "tree" with the sentential symbol as the root. Grow branches on this tree from the central root out by applying each of the possible rules whenever applicable. Grow the tree until all of the productions on the outer branches are the length of the input string. Finally, compare the input string to all the branches. If a match is found, then the string is in the language. If no string matches, it is not. This procedure is guaranteed to work, because it generates all the top down parses the length of the input. The procedure is guaranteed to terminate because all rules lengthen the string, none shorten it or leave it the same size.

The system looks powerful, but power of one sort must usually be traded off for power of another sort. Consider the musical example labeled "Twinkle, Twinkle Little Star". Note the repeated sixth, fourth and second degrees. The presented system is simply incapable of producing them, and hence, incapable of producing any composition containing them. This brings us to the next result.

The Major 1 system is consistent, that is, if the terminals of any given string produced in the system are interpreted as being the scale degrees of a major mode melody, then only tonal compositions are produced.

Completeness is defined as being the property that every "true" string in the system is producable from the axioms (the sentential production) and the rules of inference (the other productions). *The Major I system is incomplete.* That is, there are tonal compositions that cannot be produced using the grammar.

The Major I system is guaranteed to produce only tonal compostions, but it will not produce all of them. This means that the system is not powerful enough to justify the interpretation of being a total tonal progression grammar.

To get a better perspective on this, let's consider the general problem of grammatical inference[19]. Grammatical inference is the process of inferring the grammar of a language when only strings produced in the language are allowed to be used in the inference. In order to infer any grammar from music, one must have source material to work with. It is interesting to note that *the grammatical inference problem is unsolvable for most general grammars.* In order to infer a grammar from music, or to perform any equivalent operation, working purely from examples is insufficient. One must first have a definition of what is music and what isn't. Unfortunately, like "love" and "intelligence", "music" is a term where "definitions" lead to arguments, not enlightenment. One of the basic complaints made of Schenker in his own day was that he tried to state what music wasn't. It turned out that his definition fit the compositions that the major composers of his day were producing. Perhaps this is why his theories were not as respected in his own day as they are (by many) today. Since, in general, no one wishes to make normative statements about what is music and what isn't, then it must follow that *no grammatical system (or system that can be transformed into a grammatical system) will ever be able to completely "explain" music.* The point of this is not that music theory is a dead field. It is that "music" itself has such a broad interpretation, anything that purports to explain it is probably doomed to failure. The best that one can hope to do is theorize about certain aspects of it and see how big a chunk can be explained using as small a system as possible. This is done in the next section for the class of music shown here.

4.4 Major II

The Major I grammar seems a little unwieldy at points, particularly the last set of productions. This can be interpreted a number of ways. One is that grammars are not the best way to implement the ideas shown here. Maybe the procedural method presented at the beginning was "optimal".

But this would also violate the initial intuition, that local state can be characterized by simple rules. Perhaps these productions will seem simpler if viewed from another perspective. In the Major Melody I grammar, each tone was viewed as an object. If instead, the intervals between the tones are viewed as objects, then another grammar can be constructed. This is shown in the Major Melody IIa grammar. Note that some of the rules are now written using a different size font. The smaller font is used to express the concept of step. The unit step, designated as a 1 for a rising step and a -1 for a falling step, is the only terminal for the interval grammar. The other numbers are nonterminals expressing the step size between two notes.

Now we see something rather interesting. All the productions are defined in terms of step or pitch, that is, all except one. The triadic insertion rule is defined in terms of pitch and jump (a "jump" is a change of pitch larger than a step). This leads to the observation that, excepting the triadic insertion rule, *all the rules are context free when interpreted in the right domain.* It is proposed here that the rules be divided into three types, those that look at pitch, those that look at interval, and those that provide a switch between the two. The triadic insertion rule is of the last type. It operates by doing two things, inserting an interval greater than a step, and leading the interval to a tonic triad member. Since it operates in this switch role, the triadic insertion rule is apart from the others, it may be thought of as a *metarule.* It is not the only possible "switching" metarule.

The result of the context freeness of certain classes of music has not been obtained previously. It is similar to a claim currently made in the field of linguistics, namely that english syntax can be described by a context free grammar, as long as syntax is not used to account for regularities better attributed to semantics. The bit of semantics needed in this case is whether it is best to look at the notes, or to look between them.

The reader may be a little confused at this point. First, the rules were context sensitive, for a second they seemed context free, and now they are a combination of both. The confusion can be cleared up quite easily. A normal formalism consists of something else besides objects and rules, it also consists of an interpretation. The above result seems to point to the worth of determining a formalistic way of showing when to switch interpretations on the same object.

4.5 Why Two Grammars are Better than One

This multiple grammar model affects the number of sentences generatable by the system. Take the system consisting of parts A and B in the figure labeled "Multiple Grammars". Split **Ga** into two independent grammars; G1 and G2. Consider the resulting total system consisting of the grammars listed in parts A and C. Decomposing the melody in part E, it is seen that the sentences in the language are constructed of "parse segments". Only one of the two grammars generate the terminal symbols of each parse segment. The length of each segment is designated by the non-bold symbols starting with the letter "s".

Note that Grammar **Ga** can generate 8**N sentences of length N. Grammars G1 and G2 can only generate 4**N different melodies apiece. So, parsing the melody with two different grammars may alter the number of generatable melodies.

What is the exact relationship? Consider G Grammars, each containing X terminals (As diagrammed in part D). Let each grammar generate melodies with N total notes containing P parse segments of arbitrary length (the lengths of the segments need not be equal). Then there are (G**P)*(X**N) total melodies possible. For the example shown here, the ratio of the generatable strings in the Ga system to the generatable strings in the G1 and G2 systems is (8**N)/(2**P)*(4**N) = 2**(N-P). The point of this exercise is to show that by using different grammars to generate different sections of a melody, one can reduce the number of generatable melodies by an exponential factor. Note that this reduction will only occur if the grammars are selected so that P is significantly less than N.

Although the number of generatable melodies may be less for decomposed grammars than for undecomposed grammars, there is a problem with the above argument. If the number of grammars in the example had been 4 instead of 2 and the grammars' 4 terminals were still members of the undecomposed grammar's set, then the upper bound on generatable melodies would have been 16**N. This doesn't mean that decomposed grammars generate more melodies than the undecomposed grammar, indeed, the undecomposed grammar completely spans the space of all possible melodies, so that would be impossible. Rather, it means that one can obtain ambiguous parses. Since grammars G1 and G2 are completely independent (they have no terminals in common), ambiguous parses don't occur with the above two grammars.

Therefore, in order for the decomposition to reduce the number of generatable strings, the grammars should be reasonably independent, (and hopefully the ambiguous derivations have meaning).

4.6 A Theory of Music Perception - Major IIb

Major IIa has been slightly modified and is now presented as Major IIb. The neighbor note insertion in this grammar is interpreted as being a restatement of a tonic triad member via an adjacent scale degree, as opposed to being an interval production. Note that in addition to being context free, the grammar has been structured to produce *leftmost derivations.* This grammar can be said to present a partial model of music understanding because it allows the listener to parse the music as it is being heard.

As music is played, expectations are created by the appearance of certain notes and are fulfilled by others which create their own expectations. This grammar models these concepts quite well. Let's look at the two part invention number eight again. In hearing the initial 1, the listener knows that it is either a member of the fundamental descending line, the first part of a triadic repetition or a neighbor insertion. In hearing the second note, he knows that it must be the result of a triadic insertion. In hearing the third note, he knows that the first note was repeated, this means that the original 1C production resulting from the first has been evaluted completely and the expectation of a neighbor note resulting from it can be thrown away. However, the third note can have a neighbor or a repetition . . . and so on till the sixth note. Here, the switch production has been invoked, and the seventh note is a descending step. The interpretation changes and the D production in the descending step is activated along with the possibility of a neighbor note production. Another step occurs, this 1' is not going to be reenforced, so the BC' possibilty is not evaluated. Two more steps; will the 5 be reenforced? Yes!, and so on.

This is all fine and good, but, the above paragraph did not list all the possibilities that were valid at a given instance. It did not reflect the fact that Bach did the listener a favor in the first few notes by establishing where the tonic, mediant, and dominant were. It did not account for the possibility that a neighbor note to the first 1, might occur after the second 1 had been stated, and so on. Worse yet, there was nothing in the grammar to prevent termination of the descending step at the sixth scale degree (Note that Major I didn't have this problem!). All of these things could be implemented in the Major IIb grammar. None of them were. The reason is very simple. The grammar would have been much longer if this had been done. A grammar model can represent all aspects of musical structure, but it cannot represent them well. The aspects that it can represent well have been presented. In the work here, the grammar was transformed, but it was never (and probably will never be) expanded much beyond this level.

This argument answers a question raised when analyzing music by "layers". If we view tonal structure as being the elaboration of a basic concept through many levels, should the elaboration primitives be the same for each level? Perhaps for model of music cognition above the feature detector level, a convincing argument for the homogeniety of elaboration operators on all levels can be made. Here it can not.

What will the elaboration operators be? How is the above knowledge used? This is where the concept of *control procedure* and *meta-rule* comes in again. One example meta-rule is that a neighbor note association is made to the last appropriate triadic member heard. Another is that a descending step may not end on a non-triadic member unless the last step is used in a neighbor note configuration. What other meta-rules should be used here? Expectation of common cliches, "common sense", rhythmic cues, sound intensity and other methods can be used to decide what to expect, and when to throw away unevaluated non-terminals. Investigation into this is starting, but current results will not be reported here.

4.6.1 Confusion, Boredom and Interest

If this theory is true, what are the ramifications? One is that we can anticipate what boredom, confusion and interest would correspond to in the model.

A person can be bored or confused on many levels. If the rules are learnable on a basic level, then those who don't possess these production rules won't have a "deep understanding" of the Major II music they hear. They will hear it, but they won't be listening to it. They could be bored by the whole thing.

The idea of boredom also brings to mind the idea of interest. Major II is *ambigious.* That means that many productions can be derived in many different ways. An example is shown in a fugal theme in the figure labeled "Cherubini". It is not clear whether to view the 1 or 5 as repeating. Fulfilling expectations in a way that is logical, but not expected, can be construed as corresponding to a mechanism of interest.

The figure entitled "More of Bach's Two Part Invention Number 8" shows just that. Some of the concepts expressed here are seen working in the diagram. Bach opens the piece by establishing the key. Note the tonic repetition. Next he reenforces all members with a descending step and neighbor motion, The dark lines preceed areas where the music is given rhythmic stress. So the tonic is used to end the step, it is reenforced rhythmically and is used to launch the next section. All this is important because it is questionable where the music going at this point. Is 3 or 5 repeating? We see that 5 wins this round as it is stressed by neighbor motion.

4.7 A Theory of Music Improvisation

One can expand the Major II grammar in a left to right order "on the fly" if one remembers the unevaluated terminals.

Embracing the concepts presented thus far would seem to imply the following model of improvisation. The improviser must have three things with which to work. First, several sets of local rules, such as the set presented above. Second, a store which records the productions not yet been expanded.

Third, a set of pattern parameters, acquired as the improvisation proceeds, which helps decide which of the set of local rules to apply next. How large a repertoire of rules, coupled with how big a store, coupled with what (and how big) a set of pattern parameters are required to do "good" improvisation? Can a tie between understanding a sentence and improvising music be found? If so, then the work of Marcus [20] could be used in this problem to see if improvisation is LR(3).

4.8 The Minor Mode

To conclude this chapter, some productions for the minor mode are introduced. They basically constitute a set of exceptions to the rules previously discussed. These rules are listed in the figure entitled "Minor Additions". In these rules, the symbol " # " means that the frequency of the preceeding pitch should be raised one semitone.

In looking at the rules, it is easier to see why there are two separate sets of rules for step motion. Apparently, ascent is different than descent in the minor mode. The reason for this lies in the minor scale itself. Recall that the minor scale interval steps form the pattern 2,1,2,2,1,2,2 and the major scale forms the pattern 2,2,1,2,2,2,1. There exists a scale called the *melodic minor* which uses the minor pattern when descending the scale, but when ascending, it uses the pattern 2,1,2,2,2,2,1. The ascent uses the raised sixth and seventh scale degrees in order to make clear the direction of the step motion. A final example, Bach's Bouree, is shown derived.

4.9 Directions for Further Research

A broad set of metarules is needed to extend this model. Partial research has been done on this, but the results will not be reported here.

The grammatical inference methods of Kaiser[19] could be used to expand the grammar. In her paper, she parsed Morse code conversations using a limited class of grammatical productions. When a new production was encountered, a special grammatical extension procedure was invoked. One example extension that is possible in the present grammar is the inclusion of a *double neighbor* production:

E --> EDFE

However, even for a grammar that only produces monophonic tonal progressions, the present set of productions are very limited. Therefore it may be difficult to use this set of productions as a basis for such work. Perhaps a different "working set" of grammatical productions can be developed based on a set of different rules. One could then develop procedures for switching between one working set and another.

The addition of rhythm and multiple voices is also a logical next step for other researchers in this field.

However, the real direction should be obvious to each individual researcher as he or she realizes how music theory can benefit their own interests.

4.10 Conclusions

It has been shown that a class of music theoretical rules used in the study of composition can also account for musical perception and improvisation.

Unlike many papers, the purpose of this one is not to present a result. Its purpose is to prove a point. The point is that music is a valuble domain for AI research.

For example, recall the "frames" paradigm. Few microworld models can state a resonable mechanism for switching viewpoints. In showing that a jump larger that a step causes the expectation of some change, this paper comes close to having this capability. It would not have been possible were it not for the simple nature of the microworld model chosen.

This microworld system can be (and is being) developed further. In the course of that development, I suspect that this work will be quickly exposed as a wrong approach and that the ideas presented here will give way to the use of some other cognition modeling concept. But then, that's the whole idea. Something new will be learned, and the pieces of the previous work can be salvaged for the new model.

Marvin Minsky once said that writing "Perceptrons", the book he co-authored with Seymour Papert, may have been a mistake. That in the book, many of the simple questions about Perceptrons were answered. This deprived workers in the field of handholds when approaching the subject.

This thesis certainly doesn't have that problem. The work presented here does not consider any of the interesting subjects in music deeply, although it does lay the groundwork for studying some topics. I sincerely hope that readers will use this groundwork (or develop their own) as a tool to investigate the field.

5. Chapter Five: The Inexpensive Synthesizer System

The purpose of the inexpensive synthesizer is to provide an inexpensive tool that allows one to keep a record of any keyboard performance done while away from the mainframe computer. The system must be complete and portable. This rules out standard keyboard instruments, such as the piano. It must also interface easily to other equipment used in the project. These constraints dictate that the inexpensive synthesizer produce sound electronically.

The figure labeled "Music Synthesizer Architectures" illustrates some possible music synthesizer designs.

In design A, one records the audio signals of the actual performance and has the computer transcribe the recording into its Performance Schedule using "The Ear" link shown in the "Music Model" figure. The signal processing problems encountered when trying to extract the performance parameters from a given signal are immense[33] and preclude the use of this approach.

It is easier to interpose some mechanism that actually records what the performer is doing, as opposed to the sounds that he is producing. Design B illustrates this idea. The "processor" could be random logic wired to to perform the appropriate algorithm, or it could be a general purpose device. When recording parameters in this manner, one must remember that a good performer will compensate when playing a bad instrument in order to produce the sounds he wants. Therefore, when using the performance parmeters extracted by the processor, it is important to know what the performer actually heard as he played. As electronic synthesis allows a large degree of control over the actual sound production, there should be no disparity between the produced sounds and recorded performance parameters in an electronic synthesizer.

It would be nice if the processor could perform both the keyboard scanning algorithm and the synthesis routines as in design C. However, even if some of the commonly available 8 bit microprocessors were only used for the synthesis function, they would be limited to synthesizing four voices through an 8 bit D/A at an 8 khz sample rate[29]. (In such a system, the function of the processor is to provide the appropriate indices for a set of table memories.) This method is therefore quite memory intensive, and a microprocessor used in this architecture could certainly not perform the additional task of keyscanning.

In order to have a keyscanner and/or more voices, one could implement the system shown in design D. Here, the work of synthesis is divided up among many processors. However, if a microcomputer were used as the synthesizer processor, it would still be used only for table lookup; and as pointed out in the introduction, when a microprocessor is used solely to execute simple algorithms, the overhead becomes unacceptably high. Due to cost considerations, using microcomputers as processors in this architecture must be rejected.

It was therefore decided that design B was the best gross structure to use. Having decided to use design B, it was necessary to construct the keyboard, the processor-controller, and the synthesizer, as well as to implement the interface software. The next sections will discuss these aspects of the overall design.

5.1 The Synthesizer

A block diagram of the synthesizer is shown in the "Inexpensive Synthesizer" figures. In order to understand the operation of the device, we will start by considering how to best use a binary rate multiplier (labeled BRM in the "Frequency Generation Section" figure) for producing a given system of intonation. To do this, let's first review this and other concepts presented in the first chapters.

5.1.1 Producing Systems of Intonation using a Binary Rate Multiplier

In order to perform most western music, one must choose a *system of intonation*, that is, a fixed set of pitches, in which to play a given piece. The most common system of intonation used in western countries today is the *equal tempered system*. The equal tempered system of intonation consists of a set of 12 different *pitch classes*. The frequencies of the members of a given pitch class differ from one another by factors of two. If two pitches' frequencies differ by only one factor of two (and are hence members of the same pitch class), they are said to be an *octave* apart in pitch. If all the members of all the pitch classes are grouped together and sorted in order of frequency, then each group of 12 pitches from the pitch class C to the pitch class of the next higher frequency B is called an octave. Each pitch in this group is called a member of that octave. In the equal tempered system of intonation, the ratios of the adjacent frequencies in the octave is equal to the twelth root of 2. A part of the equal tempered system of intonation is shown below.

octave	pitch class	frequency(hz)	frequency/8372
9	C	8372	1.00000
8	B	7902	0.94387
8	A#	7458	0.89089
8	A	7040	0.84089
8	G#	6644	0.79370
8	G	6271	0.74915
8	F#	5919	0.70710
8	F	5587	0.66741
8	E	5274	0.62996
8	D#	4978	0.59460
8	D	4698	0.56123
8	C#	4434	0.52973
8	C	4186	0.50000

As shown in the last column, the scale can be thought of having a main note, the ninth octave C, from which all the other notes are obtained. Using this conception, one way of obtaining this series of pitches is to make a box that can multiply the frequency of the nineth octave C by the given fractions and produce the resulting frequencies as outputs. A BRM (binary rate multiplier) is just such a box. It has two inputs; a frequency f and a binary word x. It produces one output, a frequency of value $f*(x/n)$ where $0 =< x < n$. The value of n is a function of the specific hardware implementation, it is usually a power of 2. Both x and n are integers. The input frequency is used to clock a counter and the binary word controls a state decoder.

The pulses produced by the BRM are not of equal width. However, if the frequency multiplication is done at a high frequency and the resulting pulse train is divided down to audio frequency using a divider chain, the jitter is not noticible. The only other synthesizer which uses this technique of producing a high frequency signal that is divided down to remove jitter is the Northeastern Digital Synclavier Synthesizer[27]. They do not use a BRM however.

To produce notes in the equal tempered scale, a set of x's are needed. Suppose n is 4096. Then one method of generating the x inputs that are needed is with the equation

x=Round(4096 * desired frequency / top frequency).

This idea can be developed further. What is wanted is a series of frequencies that are related as in the table above. Therefore, the top frequency input to the BRM need not be the top frequency. In fact, the highest that a 12 input BRM can multiply by is 4095/4096, so it can't even pass the input frequency out. What is therefore needed is the "best" set of binary word inputs to the BRM. "Best" will mean the sum of the squares of the differences between the approximating and actual fractions are a minimum. The sets of numerators are obtained by noting that the

(desired note multiplier numerator/4096)/(top note multiplier numerator/4096) =

(desired note multiplier numerator)/(top note multiplier numerator) =

(desired note frequency)/(top note frequency)

Utilizing the fact that the frequency ratios between notes in any system of intonation is fixed, a program can search for the "best" set of numerators quite simply.

A table of "best" numerators for use in a binary rate multiplier generating several different systems of intonation is shown below. The ratios for these systems can be found in [17]. The equal tempered scale is listed as being "Diatonic" in the reference. This is reflected in the table.

Intonation	Notes/Octave	Top numerator	Sum of Squared Errors
Sub Infra Diatonic	5	3994	1.20E-9
Infra Diatonic	7	3880	6.75E-9
Diatonic	12	3902	1.85E-8
Supra Diatonic	19	3994	4.05E-8
Just	12	2912	0
Pythagorian/Chinese	12	4012	2.06E-8
Mean	21	4064	5.06E-8
Mercatorian	53	4092	1.83E-7
256 Diatonic	256	4071	1.15E-6

The above systems of intonation are quite interesting. As mentioned, the Diatonic scale is based on 12 notes in the octave, each adjacent pair differing from the other by a factor of 2**(1/12). The other diatonic scales, the Sub-Infra, the Infra and the Supra have their adjacent tones differing from one another by the fifth, seventh, and nineteenth roots of two. The Pythagorean scale is based on the frequency ratio 3/2 (a pure fifth). The Mercatorian scale produces a scale by the same algorithm as the Pythagorean scale, but it carries it to 52 iterations as opposed to 11. The Just scale is based on the intervals of the pure fifth and the pure third (which has a frequency ratio of 5/4). The Mean tone scale is based on a fifth that is slightly smaller than a pure fifth. The 256-Diatonic entry is included to show how fast the error grows in the diatonic scales using a 12 bit BRM.

If an 18 Mhz crystal oscillator is used to clock the BRM, and the BRM is producing the equal tempered scale using the 3902 value for the top octave numerator, then the American Standard Pitch equal tempered scale (based on a fourth octave A of 440 hertz) will be produced.

Although the BRM needs to be run at as high a frequency as possible to divide out its inherent jitter, the synthesizer multiplexes it into 16 oscillators in order to reduce chip count and cost. A consequence of this in the present design is that some timing constraints are violated. Two of the 16 multiplexed voices don't operate properly. It was deemed preferable to have 14 voices instead of 16 voices and more hardware, so the design was left in its present form.

The RAM in "Frequency Generation" figure contains the multiplexed frequency multiplication numerators. The write controller updates their contents on processor command.

It is assumed in this section that a 12 bit BRM is used to generate just one octave of these systems. The lower octaves would be obtained by dividing the produced frequencies by 2**m where m is the number of octaves below the top octave where the desired frequency lies. How this is accomplished in practice is described in the next section.

5.1.2 The Octave Shifter

As mentioned above, it was assumed that the BRM produced notes in just one octave. It is undesirable to produce notes over an eight octave frequency range using only 12 bits of total precision. The reason is that under ideal conditions, human listeners can distinguish a 1 part in 5300 frequency difference in the lower octaves (around 160 hz)[34]. Therefore, even 12 bits of frequency precision per octave are not sufficient for a high quality sound synthesizer. However, the remote synthesizer is not used under ideal conditions, so 12 bits of precision per octave were deemed sufficient (i.e. Glides produced by the remote synthesizer do not sound "grainy" at the lower octaves).

The question of how one should change octaves using a BRM still remains. The method used here takes advantage of the fact that shifting a binary number one position to the right in a fixed length binary word divides the number by two. If a 12 bit code with zero fill at the bottom is input to a BRM, a frequency f will be produced. If this 12 bit code is shifted down one position with zero fill at the top, it will produce the frequency $f/2$. Thus, an 18 bit wide BRM can produce any frequency specified to 12 bits over a 7 octave range.

If the frequency word is shifted one position more after it reaches the bottom, the least significant bit will be lost. So one can use a BRM to synthesize frequencies over an 8 octave range if one is willing to have only 11 bits of precision in the lowest octave.

The function of the "Floating to Fixed Converter" is to perform this shifting operation on the 12 bit frequency word. This hardware is equivalent to the aligning hardware used in some floating point machines. The shifting was done in hardware because the controller is an 8 bit machine and does 18 bit shifting poorly.

The major parts of the frequency generation section have now been completely described. In actual operation, the processor tells the synthesizer what pitch to play using the frequency select input, what octave to play it in using the octave select input, and which voice to play it from using the voice select input. After the data is set up, the processor sends a "Ready" signal to the synthesizer. The floating to fixed converter then causes the voice register section to reload the count-down register afresh each time the zero state is detected. This NOR gate also drives the *multiplexed frequency output* pad.

There are a number of reasons for not using this, or similar, VLSI designs to implement the remote synthesizer. One is cost. A mass production run for this chip would cost approximately $10,000 to produce 500 chips. It would yield 100 to 200 good chips at a cost of $50 to $100 each. The chip cost of the equivalent part of the discrete design is currently about $25. (Also, this project cannot currently utilize 100-200 ICs.)

Another reason is speed. An NMOS IC using the design methodology of this chip cannot, as of this writing, be clocked as fast as discrete TTL.

Finally, the IC design is not as expandable as the discrete design. For example, it is relatively easy to change the discrete design to strobe out a waveform memory even after the boards are built. It is currently impossible to "retrofit" an IC as signals needed from the chip's mechanism may not be available on the output pads.

5.1.4 Amplitude Modulation

The current remote synthesizer design has no amplitude modulation capability yet. It simply demultiplexes the "16 voice multiplexed output" shown in the "Frequency Generation" figure into 14 separate voices divides them down eight octaves and then adds them together through an analog summer. However, two different amplitude modulation schemes have been devised for this synthesizer.

The first method is shown in the figure labeled the "Rom Amplitude Modulation Section". Here, the output frequency produced by the BRM is used to clock a counter which accesses a 256 element waveform memory table. (This signal was simply divided down to a lower frequency square wave in the previous design.) Naturally, the output waveform from the table will occur at 1/256th of the frequency fed to the counter. Any waveform can be used, and there could be several different ones in the table. The output wave will have jitter at the higher harmonics, but this jitter will not be noticable in the audio range [27]. Since this synthesizer is not a fixed sample rate system, the table can be as short as 256 entries long and still perform better than a fixed point system table lookup scheme using interpolation on a 1K memory[27]. The reason for this is that a variable sample rate system hits every table entry on a table entry boundary as it increments through the memory.

The multiply operation indicated in the figure is obviously digital, although there are advantages to making it analog. For instance, by cascading three 8 bit multiplying DACs, one controlled by the wave table, one controlled by the amplitude register and one controlled by an envelope generator, one could obtain the equivalent of a 24 bit DAC. This method is used in the Northeastern Digital Synclavier. The real trick in doing this is to have a good set of normalization procedures to keep all of the DACs filled.

The final step in the figure has the adder mixing the voices together before they are passed out to the DAC.

A second method of amplitude modulation is illustrated in the figure labeled "Square Wave Amplitude Modulation Section". This method requires less hardware than the first, but is more limited. Instead of modulating the amplitude of an arbitrary waveform, it amplitude modulates the square waves output from the BRM. One can do this with AND gates instead of multipliers because square waves are dual valued functions. Multiplying square waves by a given value is equivalent to switching that value on and off at the frequency which the square wave is oscillating. The modulated square waves are summed and output as before.

5.2 The Keyboard

The keyboard was bought commercially. It has a 5 octave range (61 keys) and has two SPST switches per key. All SPST switches connect to a common bus. This gives the keyboard the capacity to be velocity sensitive. This option is not currently used, although if it were implemented, the measurement error could be very great. Sampling at several points during key depression would eliminate much of this error on a multiswitch keyboard, but this keyboard does not have that capability.

It is interesting to note that there are only three significant parameters that a keyboard performer aligns the 12 bit word and the write controller writes the aligned word into the appropriate voice. The 4 bit counter multiplexes the BRM by processing each voice through it and then changing the BRM's state with the carry output line. This produces the 16 multiplexed voices on the BRM's output.

5.1.3 An Aside; A VLSI Synthesizer

A variant of the another square wave sythesis algorithm was implemented in VLSI for an MIT 6.371 VLSI course term project. A block diagram of the project is shown in the figure labeled "A VLSI Frequency Synthesizer".

The project consists of eight (frequency) programmable oscillators which multiplex their outputs onto a single output line. This is similar to the above design, however here each oscillator's output frequency is determined by the relation fout=fclock/N*8 (1 < N < 2**20).

The design consists of three main parts.

The data is fed serially into the machine from the left. This is done by applying a bit of data to the DATA-INPUT pad and taking the DATA-BIT-RDY input high. The high going edge is detected using a discrete logic edge detector. The output of the edge detector is used to shift or hold the the serial input data in the *serial input register.* The fully loaded register contents consist of three bits of control information which point to the *voice register* to be loaded along with 20 bits of frequency data which specify the new contents of the indicated voice register. Serial as opposed to parallel loading was used to save pads and pad space.

After the serial input register is loaded, the DATA-WORD-RDY signal is raised. On recieving this signal, the *voice register controller* PLA waits until the proper voice register recirculates to the *input multiplexers* and then loads the the register with the new 20 bit frequency word. (The PLA normally allows the old voice register contents to recirculate.)

The parallel output of the voice register section feeds a *synchronous down counter* section which (almost) looks like a recirculating shift register. The eight down counters are realized using a two input xor gate to change the down counters' state and a 20 input NOR gate to detect the zeroth state. This NOR gate sends to many keyboard instruments. They are the key's velocity, the time of string impact and the time of string release. Therefore, things like key acceleration need not be recorded as a performance parameter.

During operation, the common bus is grounded and the switches are connected to a 64 input multiplexer. This allows the controller to randomly access each key. There is no debounce circuitry on the switches or the multiplexer. The debounce operation is performed in software.

5.3 The Controller

The controller is the device which glues all the parts together.

A block diagram of the controller is shown in the figure labeled "Control Board". There were three reasons for designing a controller board as opposed to using an off the shelf system. They were size, cost and parallel I/O capability. The processer is mounted on an 8 1/2" x 5" card. It is cheaper than any other commercially available card of equivalent power. As shown, the synthesizer and keyboard require four 8-bit ports of parallel input to control. More parallel I/O is needed if the system wants to send the parameters it records in parallel to some device.

The processor can act as a slave to a remote system, recieving note information, encoding it and playing it. However, its primary purpose is to act as a keyboard scanner, sending performance update information to the storage device and simultaneously playing the notes struck on the keyboard through the synthesizer. It can communicate with the storage device through a parallel or serial port. Only the serial port has been used for this function so far. Serial I/O allows one to record the performance parameters on a cassette tape and send them to a modem or directly to a mainframe machine.

Since the amplitude modulation section of the remote sythesizer has not been built yet, no exact claims can be made about how well the processor simultaneously updates the synthesizer, scans the keyboard and transmits performance parameters. However, the synthesizer described in the next chapter was controlled using this controller, and it was able to update four groups of three 12 bit amplitude envelopes while connected to an N-key rollover input device. The currently implemented device handles all of its tasks easily using the algorithm described in the next section.

5.4 The Software

The controller must perform three tasks. First, it must scan the keyboard and eliminate the key bounce and contact noise it encounters. Second, it must route the pressed keys to the correct oscillator voices. Third, it must send the performance parameters to the storage device.

There are two ways to implement the keyboard scanning algorithm. The first and most obvious is to use a dedicated piece of hardware. However, this would entail additional cost, as no small, cheap keyscanner is currently available. (Another problem is that none of the organ keyboards considered could be easily converted to have a key matrix output, which dedicated keyboard interface chips, such as the Intel 8279, require.) It was therefore decided to let the synthesizer controller also perform the keyscanning algorithm.

The routing algorithm scans M keys and assigns the played keys to N oscillators. It is not allowed to overwrite oscillators until they have finished playing a note. It is not allowed to assign the same played key to two oscillators. (i.e. the mapping between played keys and oscillators is one-to-one and onto.) This being the case, if more than N keys are hit, the algorithm will not allow the excess keys to be played.

The performance parameter transmission algorithm simply packs values in a queue and then unpacks and sends them when the processor is free.

All the algorithms are performed in linear time. The routing algorithm is reported here because all previous algorithms considered were O(MN) or required a doubly linked list or a stack.

The figure labeled "The Keyscan Algorithm" shows intermediate stages of the algorithm. It reflects the fact that there are two structures associated with it; a keyboard scan list (KSL) and an oscillator note list (ONL). The KSL indicates if a key is pushed down. Each member of th ONL represents an oscillator and points to the current key that that oscillator is playing. In the Figure, the letters beneath the keyboard indicate which keys being pressed. The black marks on the keys indicate which entries in the KSL are turned on. The black squares above the OSL are 1 bit tags used in the course of the algorithm.

The algorithm starts by scanning the ONL. Any "0" entry indicates that that oscillator is currently playing a rest, so "0" entries are skipped. If the ONL entry is not "0", it is marked and a check is made to see if the key it points to is still being pressed. If it is, the KSL entry designating that key is marked. If it is not, the oscillator register's value is put in a queue along with the key release time (this information is required by the performance parameter transmission algorithm) and the ONL entry is assigned the value "0". After the ONL is completely scanned, the free register pointer is then set to point at the first oscillator register.

Next the keyscan algorithm (KA) is called. It simply scans the keyboard from left to right and reports keys that are being pressed (it may also mistakenly report noise as a keypress). If the KA reports a key that the KSL has marked, we know that the register scan has just examined it, so we simply unmark that KSL entry and go on. If the KA reports a key that is not marked, we assign the note to the first unmarked (hence unassigned) oscillator register. This is done by advancing the free register pointer until it is pointing to an unmarked location and then filling that location. When the KA finishes scanning the keys or the free register pointer increments past the last ONL entry, then the allocation algorithm is done. The marked non-"0" OSL entries are keys that were pressed down and are still pressed down. The unmarked non-"0" entries are new keys to be played (or are noise). The marked "0" entries are keys that have been released. The unmarked "0" entries are free registers where the oscillator is playing a rest. Next performance parameter transmission algorithm outputs the queued values for 10 ms. This output loop provides the debounce time necessary to let the keys settle.

Finally, the oscillators are updated. This is done by scanning the ONL registers. If the entry is unmarked and the key it is pointing to is not pressed, then the register is deallocated. This case would occur in practice if there had been noise on the key input. Again, this is the reason for the 10 ms wait. If the unmarked register's key is being pressed, then the key is played and queued along with its "start time". If the contents of the register indicate a rest and the register is marked, then a rest is played and the register is unmarked and deallocated.

5.5 Conclusions

The system is adequate, reasonably portable and cheap. It can probably be packaged into the keyboard case, and this will be tried. The storage device is the bulkiest part. Until the technology evolves to the point where cheap portable mass storage is a reality, the machine will never be totally satisfactory as a portable music typewriter. Perhaps bubble memory will solve this problem someday.

Lack of amplitude modulation and the limitation of the waveform to square waves was viewed as being a problem by some, but not by others. Such a view is apparently dependent on what type of music synthesis hardware the person had worked on before. Amplitude Modulation hardware will probably be added to the design eventually.

The true purpose of this hardware is to provide a cheap device for music composition research. It serves this purpose well.

6. Chapter Six: Theme and Variations on a Digital Signal Processor

6.1 If God has His mathematics, let Him do it

In constructing the main digital signal processor two choices were made. The first was to decide what model(s) of signal generation the processor should be geared toward. A processor embracing many models of generation would be much more complex (and expensive) than a processor based on a small set of models. Secondly, the implementation mechanism was chosen.

The models considered can be broken down into two classes: models using non-linear signal generation techniques and models using linear signal generation techniques. Models using linear signal generation techniques differ from one another in the sets of orthonormal functions used to approximate a given waveform. The most common among these methods is sine summation synthesis, which is sometimes called Fourier synthesis. Another interesting set which is briefly discussed here is Walsh function synthesis.

Non-linear signal generation techniques are not as well understood as linear systems. Also, the set of functions that they can synthesize is usually not complete. That is, there exist periodic functions which cannot be synthesized by some non-linear methods. The advantage in using such methods is that they are usually very cheap computationally. One of the few methods in wide use is called FM synthesis. It synthesizes sounds by varying the parameters to the FM Equation (See entry III in the figure labeled "Synthesis Functions Used in Music"). Other models were considered but won't be discussed here.

The main idea behind constructing the digital signal processor was that it be sophisticated enough to fulfill most people's needs and yet not so sophisticated that people can't understand it. Therefore, models of sound production were chosen that were reasonably well known. These models and their implementations are discussed below.

6.2 A Design Based on Synthesis by Walsh Functions

One mechanism of sound production considered was Walsh function synthesis. Walsh functions are a set of orthonormal functions that take on the values -1 and 1 in the [0,1] interval. When plotted, they look like a set of square waves (Rademacher functions) with duty cycles that change through the period of the function[31]. Walsh functions contain square waves, and have better convergence properties. Example Walsh functions are shown in figure labeled "The First Eight Walsh Functions". As can be seen, Walsh functions are ordered by the number of zero crossings per unit interval that they possess. The ordering number is also called the function's *sequency.*

The bivalued amplitude of Walsh functions makes them particularly well suited for real time digital waveform synthesis. Recall that in sine summation synthesis, one must multiply each sine wave by some weighting value. Since the sine wave takes on continuous values in the unit interval, this requires that a true multiplication be done when performing the weighting. Multiplication is an expensive operation to do in real time. However, since a Walsh function only assumes two values, the weighting operation can only produce two values for any weighted input. These values are either the weight or its negation. So in practice, the multiplication is replaced with a "Complement/Not Complement" operation. This is an inexpensive operation to perform in real time.

What remains is to compute the 1 and -1 values for the Walsh functions in the unit interval. The following fact gives a simple method for doing this. *The rising edges produced by a binary rate multiplier with input* i *indicate where the zero crossings of the* i*th Walsh function occur in the unit interval.*

Therefore, if a BRM is followed by a "T" flipflop, the output of the flipflop produces the set of Walsh functions exactly (multiplied by either +1 or -1 depending on the initial state of the flipflop as it detects the first edge). When viewing the waveforms produced by such a circuit, one interprets the flipflop's "zero" state as representing -1 and the "one" state as representing +1.

In the Amplitude Modulation section of the last Chapter, the BRM was viewed as a method of producing jittery square waves. Here, (combined with the integrating flipflops) it is viewed as a cheap generator for Walsh functions. This alternate interpretation of the BRM makes it a much more powerful device than before.

Used as a Walsh function generator, the BRM can be thought of as a table memory with an extra input. This input chooses which Walsh function is currently being strobbed out of the memory. By combining a set of these functions, each appropriately weighted and accessed at the same frequency, it is possible to generate any wavefocm of that frequency.

A Walsh function synthesizer that would perform algorithm "I" shown in the "Synthesis Functions Used in Music" figure was designed on paper. A block diagram of the machine is shown in the "Walsh Function Synthesizer" figures. Note the similarity between these diagrams and the "Inexpensive Synthesizer" figures.

The reason for this similarity is that since a BRM can be used for producing Walsh functions, *the circuit produced for the inexpensive synthesizer can be used to generate the Walsh function derivatives in a Walsh function digital signal processor.*

The task of selecting an oscillating frequency for the BRM output functions in the "Walsh Waveform Generation Section" is done by using a second BRM (labeled BRM-2). The output frequency of BRM-2 is specifiable to 24 bits. If desired, a "Floating to Fixed Converter" could be used to feed the binary word input into BRM-2.

Unfortunately, the output of BRM-2 must be viewed as a jittery square wave. The 1/n counter would be used to remove some of the jitter before this signal was used to clock the Walsh Function Generator unit.

The jitter problem is critical to the design. Walsh function synthesis seems to trade off the information storage in amplitude for information storage in sequency. (An analogy useful in understanding this is to recall the difference between AM and FM encoding of information). Since the jitter would corrupt the duty cycle, it would severely affect the information stored in it. Jitter is removed by making the n in the

1/n divider very large. However, if n was made too large, then the BRM-2 method would produce a fundamental frequency of too low a value to be acceptable. In this case another method of frequency generation would have to be substituted.

After the Walsh function derivatives were produced, they would have to be integrated and "multiplied" by their coefficients. The method for doing this is shown in the "Coefficient Weighting and Amplitude Modulation Section" figure. Note the similarities between this figure and the "Square Wave Amplitude Modulation Section" figure. The "Two's Complementer" in the Walsh synthesizer could be as simple as a row of "Exclusive Or" gates if the error produced by not adding 1 in the two's complement algorithm is small. Next, the Walsh "harmonics" are summed and multiplied by an amplitude envelope.

Unfortunately, people are currently taught to view signals in terms of frequency as opposed to sequency. Therefore this digital signal processor design would be harder for potential users to use. Another disadvantage is that each block of Walsh functions is tied to a particular harmonic frequency. Also, if the synthesizer is not used to generate general purpose waveforms, but just to generate a small number of sine waves, then the Walsh design is inefficient. Finally, and most important, although the Walsh function synthesizer can do the weighting without resorting to multiplication, it cannot (as far as I can tell) resort to any trick to do the amplitude envelope multiplication. Standard multiplication techniques must be used here.

6.3 A Design Based on FM and Sine Summation Synthesis

The digital signal processor constructed for this project was based on ideas proposed by Snell [35] and Ward[36]. The data flow paths are shown in the "Log Synthesizer" figures. The numerals bounded on each side by an asterisk are the control points. The numerals labeling the slashed lines indicate the width of data paths. The heavy dark lines represent pipeline registers. The machine is horizontally microcoded and, as can be seen, fully pipelined. It is primarily designed to do sine summation synthesis using log table lookup to perform the required multiplication. All random access memory is interleaved. (Interleaved memory permits odd memory locations to be written as even locations are read and vice versa.) This is represented by pairs of boxes between pipeline registers. The feedback path allows the machine to do real time FM synthesis[30]. Other similar methods of synthesis can be implemented by using the tables to lookup other functions.

The data flow of the machine is quite simple. Ignoring the FM input for the present, we see that the "Phase Update" figure simply performs the assignment operation $A = A + B$ where A is the Phase Memory and B is the Phase Increment Memory. This loop updates the phase information for each voice. The quantity stored in the Phase Memory is viewed as being a two's complement number in the sine and FM algorithms. The Phase Memory contents are given as an argument to the $\log(\sin(x))$ lookup function (treatment of negative numbers will be discussed later). The result of this lookup is added to the logs of the Envelope and Coefficient quantities and the antilog of the result is taken. Since addition in the log domain is equivalent to multiplication in the non-log domain, the antilog operation produces the product of the Envelope term, the Coefficient term and the sine term. These can be summed in the digital mixer or added to the Phase Memory quantity via the feedback loop. If the latter option is chosen, then FM synthesis can be performed. Finally, the accumulated results can be clocked out through a DAC.

Multiplexed into 32 oscillators, the present implementation can synthesize the functions shown as II and III in the "Synthesis Functions Used in Music" figure. It could potentially synthesize functions IV and V. Using the 256 oscillator version, one could increase the upper index on the sums by a factor of 8. In the table, ai, bi, and ci represent weighting coefficients. Ai, Bi and Ci are the amplitude envelope multipliers. Note that they, unlike the frequency terms, do not have a time term associated with them. This is because the amplitude envelope is updated by the controlling processor, not the synthesizer.

An alternative to having the processor update every sample in the envelope waveform is to model the envelope as being a piecewise linear function. Then the processor could simply give the synthesizer "rate" and "limit" information which is fed to an interpolation algorithm in hardware. Although this method is clearly superior, it was not implemented in the prototype due to space considerations.

The present design has a 500 ns pipeline clock and implements 32 oscillators. This produces a sampling speed of 62 Khz. The speed of the table lookup memory causes the pipeline bottleneck. When parts (Mostek MK4802(P) 2K x 8 Static Rams with 55ns access time) become available, the table lookup memory can be replaced and the number of oscillators can be expanded to 256. This would produce a sampling speed of 39 Khz with a pipeline clock of 100 ns.

6.4 Multiplying Using Log Arithmetic

The most unusual part of the synthesizer algorithm is its method of multiplication.

Multiplication is important in sound synthesis. Algorithms described in the sound synthesis literature involve a variety of complex operations: functional composition, convolution, weighting, etc., but in actual implementations, all practical real time digital sound synthesis systems investigated could not perform their sound synthesis algorithms without a multiplication operation (*Unless* the synthesis was done totally with table lookup[29]). This multiplication has been done in the analog domain[27], with digital multipliers[35], or by interpolating table lookup[26]. (And it can be done with some functional limitations using the "complement/not complement" operation in the Walsh domain as explained above.)

There are a number of ways that multiplication could be performed in this design. One method uses simple table lookup. The sine table could output a 6 bit wide sign magnitude representation of the sin(x) function, the controlling processor limit the amplitude envelope to a total of 6 bits and the digital signal processor could use the combined 12 bits to address a multiplication table. The sign bits would be xored to supply the 13th bit to the table. The mutiplication table would also make the sign magnitude to two's complement conversion. (Note that the processor would not have two amplitude envelopes with this method.) The output would have a total of 14 bits and have an associated noise equal to that of the sine table, which is 44 dB [32]. Therefore, any method that we propose should have a signal to noise ratio of less than 44 dB in order to outperform this method. But wait, is this really an intellegent approach? Ignoring the two amplitude envelope question, we see that the amplitude ratio of the largest generatable signal to the smallest generatable signal (i.e. the dynamic range) is 64. This is insufficient for our needs. Two things desired of the synthesizer are low noise and a reasonable dynamic range, but we are willing to make tradeoffs between the two.

The compromise used here is fixed point log arithmetic. It does not give as low a noise value as some methods, but it does give a higher pipeline speed than any other method along with a reasonable dynamic range. In practice, the logarithms of the absolute value of the multiplier and multiplicand are added together and the result exponentiated. The parity of the result is the sign of the sine multiplier. The multiplicand is the sum of the coefficient and envelope logs. Both the coefficient and envelope are assumed to take on only non-negative values.

The log(sin(x)) table contains the first 180 degrees of the log(sin(x)) wave. The top 11 bits (not including the sign bit) of the Phase Memory are used to address the table. It is not necessary to take the two's complement of any negative number before it is input to the table because sin(-x) = -sin(180-x). Note that even storing a half cycle of the sine wave does not make optimal use of table space since all the information about a sine wave is contained in the first 90 degrees of its cycle, along with the knowledge that sin(x) = sin(180-x) and sin(-x) = -sin(x).

The procedure for filling the sine and antilog tables refect the compromises previously discussed. To facilitate describing these compromises, the actual values chosen for the table length and width will be stated. The log(sin(x) table is 2K deep and 12 bits wide. The antilog table is 16 bits wide and 8K deep, which means that it has 13 address bits. One bit is used for the parity conversion.

The hardware uses two's complement arithmetic, so the parity conversion section of the algorithm after the antilog requires an operation equivalent to a full add. The synthesizer has no special purpose hardware for this. It combines the parity and log operations in the same antilog table. The sign of the sine argument is used to switch from one table half to another when using the synthesizer for sine summation or FM synthesis.

The one bit taken for the parity operation leaves 12 bits of argument to the antilog table. If the range of the log(sin(x)) table fills 12 bits, then the log(sin(x)) wave sweeps the antilog's domain each cycle. The output of the antilog table produces a sine wave that has a 16 bit maximum amplitude. If any quantity is added to the log(sin(x)) wave, the input to the antilog table overflows, since the log(sin(x)) already sweeps the entire domain of the antilog table. If the log(sin(x) table output were to sweep over 11 bits, it would not overflow the antilog table so long as the log-amplitude value added to it was less than or equal to 4096. For a log-amplitude value of 0 the sine wave would have its minimum peak to peak output swing and for a log-amplitude value of 4096, it would have its maximum amplitude swing. From this we see that the amplitude range over which the sine wave can vary is dependent on its maximum value in the log domain. Let us call this value -MaxLog-. Then the relationship between MaxLog, the synthesizer's dynamic range and the tables' lengths and widths are of interest. They will be shown by the equations described in the next section.

6.5 Filling the Tables

If we call the maximum peak to peak amplitude that the sine wave assumes -MaxMag-, and the minimum peak to peak amplitude that the sine wave assumes -MinMag-, then it follows that the sine wave's dynamic range -Range- is described by

Range = MaxMag/MinMag

Let's define -MaxA- as the maximum value that a quantity can achieve in the log domain. Using the quantities defined thus far, we can define a scaling ratio that relates the range of values obtainable in the log domain to the range of values obtainable in the antilog domain. Let's call this value -LScale-, then

LScale = (MaxA - MaxLog)/log(Range)

The quantities defined thus far determine the values of the entries to be assigned in the antilog table. It turns out that

i th antilog entry = MinMag*exp[(i - MaxLog)/LScale]

We see that the extreme points produced by this equation give what is expected. When i assumes the value MaxLog, the exponent equals 1, and so MinMag is output. When i assumes the value MaxA, the numerator of LScale divides out, and the quantity log(Range) is exponentiated. This produces the value

Range, which is multiplied by MinMag to produce the value MaxMag.

If we define the quantity

Epsilon = 1/exp[MaxLog/LScale]

and

Norm = 3.14159/Maximum log(sin(x)) argument

then it turns out that

i th log(sin(x)) entry = LScale*log[sin(Norm*i)/Epsilon]

Norm is obviously a normalization constant that maps the logtable arguments into the range [0,3.14159]. Epsilon is another log domain normalization constant. We see that the boundary conditions are satisfied. If i takes on the value of the argument that maximizes the log(sin(x)) function, then sin(Norm*i) is 1. Epsilon is then inverted, which yields exp[MaxLog/LScale] and its log is taken, which gives MaxLog/LScale. MaxLog/LScale is multiplied by LScale, which yields MaxLog. The other boundary condition is where sine takes on its minimum value, which is zero. We are then obligated to take the log of zero, which brings us to our next section.

6.6 The Log of Zero or: You Can't Get There from Here

The Epsilon term in the above equations guarantee that most log(sin(x)) table entries will be greater than zero. But there are several entries near the zero crossing point that are not (About 52 entries at each edge of a 2K table with a MaxLog of 1023 and a Range of 2048). It was thought that these entries could simply be set to zero, but the resulting waveform's zero crossings were visibly distorted at all amplitudes, so the problem had to be corrected. To do this, first the log(sin(x)) entries at the tables edges were filled with their two's complement negative values calculated from the previously discussed equations. The log of zero was set to the maximum negative number representable in the table. (All numbers were sign extended.) Next, an exclusive or gate was wired to the sign bit of the log(sin(x)) table output, and to the carry output of the adder performing the final log domain addition. The exclusive or output was connected to the "clear" input of the pipeline register following the antilog table. This is the control input labeled *8* in the "Log Synthesizer Amplitude Modulation Section" figure.

The logic of this is clear. If there is no carry and the sign bit is zero, then this is a normal operation and the register should not be cleared. If there is a carry and the sign bit is zero, then there has been a positive overflow. It is assumed that the processor will not let this occur (unless it wishes to clip the signal). If there is no carry and the sign bit is one, then a positive number (the log-amplitude) was added to a negative number (the log(sin(x)) value) of greater absolute value and produced a negative result, so the register is cleared. Finally, if there was a carry and the sign bit is 1, then a positive number was added to a negative number of lesser absolute value and produced a positive result, so the register is not cleared.

As the system is shown, there is no way to completely shut a voice off. The reason is that when the oscillator is shut off, there is no control over what value is left in the Phase Memory, so some constant value is always being produced whenever that oscillator voice is processed by the log(sin(x)) table. The voice can be multiplied by a small amplitude, but never by a zero one. This means that the constant offset is always passed through. The end effect is that there could be many constant value offsets floating through the system, each the result of an oscillator stuck in some non-zero phase. The way this problem was solved was to use a special code in the Envelope Memory to designate "0". When the synthesizer sees that code, it clears the antilog output pipeline register *8* control for that voice.

6.7 How Not to Update Memory

As mentioned previously, all memory is interleaved. This not only increases the pipeline speed, it also permited a solution to a serious design flaw in the prototype. To understand the flaw, one must know that the synthesizer accesses oscillators sequentially, that is, in the sine summation synthesis algorithm, it first accesses the zeroth voice, then the first, then the second and so on. In the initial design, the Coefficient, Envelope, and Phase Increment memories are updated by the following algorithm. When they recieve an update value, they hold it until the location it is destined for is accessed by the synthesis algorithm and then they write it in. The problem with this method is that the wait time for *a single voice* could be as long as 32 clock cycles. If the controlling processor does not wait 32 clock cycles before sending another value, it stands the chance of overwriting data it had previously sent. When the controlling processor was updating all 32 voices one after another, it would take 32x32 = 1024 synthesizer clock cycles to update all the registers. This limits the sampling rate to 2 Mhz/1024 = 2 Khz in the prototype for a waveform where all voices are used and a guarantee of no overwrites is required. This problem is bad in the prototype, but it would be catastrophic in a 256 voice synthesizer. If a synthesizer of this design had to update 256 voices with a 100 ns pipeline clock and required no overwrites, then it would be bandlimited to envelopes with a frequency content of no greater than (10 Mhz/ (256*256))/2 = 75 hz.

However, in this design, all memory is interleaved. This allows one to construct hardware that makes the controlling processor wait a maximum of two cycles before updating memory. This is better than using a memory access algorithm that needs time proportional to the square of the memory length in order to update it.

6.8 Interpolating Table Lookup

Both the sine and antilog tables could have used interpolation to cut down on their lookup memory table sizes. However, interpolation requires both a multiplication and an addition to perform. The following argument demonstrates how much memory is saved using linear interpolation to lookup the value of a piecewise continuous function.

Interpolation is usually accomplished in hardware by using a multiplier, an adder and a slope or derivative table to calculate Interpolated Value = Lookup Value + Fraction * Slope.

The question arises, how much memory is saved by using interpolation in the table lookup of any given piecewise linear function? This is rather easily calculated. Since any given function might swing full scale, the most significant bit of the slope table must have the same weight as the most significant bit of the lookup table. Also, since we presumably want to preserve precision, the Fraction and hence the Slope values must be of the same length as the Lookup Value.

Long Term Power Spectrum studies using arbitrarily large precision[28] have indicated that if one uses the interpolation method with each table having 2**n entries, then it will take a table of 2**2n locations with a normal table lookup to get as good a signal. Together with the bit significance argument, which shows that we need two tables of the same length in order to interpolate, interpolation seems to save space by a factor of 2**(n-1) (if piecewise linearity can be assumed). This result was reported by [32] for sine functions but not for piecewise linear functions.

However, to do interpolation, it is necessary to perform a multiply and the author knows of no multiplier of 12 bit (or greater) precision that can (practically) operate in under 150ns.

6.9 The Curse of Fixed Point

Fixed point limits the algorithm in two crucial ways. The first is in the antilog table lookup operation. For many antilog table arguments, 10 bits of argument are compressed into 4 bits of result. (This points out the interesting fact that log table lookup can do a truncation on multiplication.) The sine wave is very distorted for small values, which is to be expected. The way to get around this would be to use floating point. Floating point gives one fixed signal to noise ratio across the entire dynamic range of the output. For a fixed word size, floating point does not give as good a signal to noise ratio for high amplitude output values as does fixed point, but it does produce a much better signal at low amplitudes. However, if the antilog operation looked up a value to be fed to a floating point DAC, the digital mixing would be more complex and there could be no feedback path from the mixer to the FM register.

The second place where fixed point limits the machine's operation is in the performance of the FM synthesis algorithm. The machine must choose between using a wide range of slightly varying modulation indices or a small range of greatly varying indices. This is due in part to the truncation occuring in the antilog table lookup.

6.10 Conclusions

The design is adequate, but it has several disadvantages.

The first and most important is that it is a hardware implementation of a specific algorithm as opposed to being a general signal processor. This is dramatically illustrated by the fact that the FM synthesis microcode can only produce a quarter the number of sine summation terms (8 versus 32). One would initially expect that the number of synthesizable FM terms would equal half the number of sine summation terms since FM uses twice as many sine terms as does sine summation. The processor has the memory capacity to synthesize more FM terms, it just has no fast data paths leading to that memory. (As more memory is multiplexed in, this ratio will approach 1/3.)

The second most important problem is that updating the processor amplitude envelope "by hand" limits the number of produceable sounds. Many sounds require that the envelope information be updated just as fast as the frequency information. This is clearly an impossible task in the present design for a conventional general purpose processor if the number of envelopes is 256. "Rate and limit" hardware is a necessary, if expensive, addition to the synthesizer if such high bandwidth envelopes are desired. Another solution to this problem would be to place these high frequency harmonics into the phase registers. But then the envelope would be harder to model.

The present design also has a large table lookup memory cost associated with it. However, this is more of a feature than a problem. The amount of table lookup memory can be dramatically reduced if one wishes to only use one set of optimized lookup functions.

It would have been nicer if the frequency word length had been four bits wider for some applications.

The use of log arithmetic was successful in this application. It produces a processor that can potentially outperform any other in its price class in terms of speed.

REFERENCES

1. Mead, C. and Conway L. *Introduction to VLSI Systems.* Addison- Wesley Publishing Co., (1980).

2. Kassler, M. "Proving Musical Theorems I: The Middleground of Heinrich Schenker's Theory of Tonality." The University of Sidney Basser Department of Computer Science Technical Report No. 103, August, 1975.

3. Kassler, M. Private Communication.

4. Hiller, L. A. "Computer Music." *Scientific American*, (December, 1959) pp. 109-120.

5. Lehrdahl, E. and Jackendoff, R. "Towards a Formal Theory of Tonal Music." *Journal of Music Theory*, 21, No. 1 (1977), pp. 111-172.

6. Moorer, J. A. "Music and Computer Composition." *Communications of the ACM*, 15, No. 2 (February, 1972), pp. 104-113.

7. Rader, G. M. "A Method of Composing Simple Traditional Music by Computer." *Communications of the ACM*, 17, No. 11 (November, 1974), pp. 631-302.

8. Regener, E. "Layered Music-Theoretic Systems." *Perspectives of New Music*, 6, No. 1 (1967), pp. 52-62.

9. Schenker, H. *Neue Musikalische Theorien und Phantasien, Vol III: Der Freie Satz.* Vienna: Universal Edition, 1956.

10. -------. *Free Composition (Der Freie Satz).* Translated by Ernst Oster. New York: Longman Incorporated, 1979.

11. --------. *Kontrapunkt I.* Vienna: Universal Edition, 1910.

12. --------. *Kontrapunkt II.* Vienna: Universal Edition, 1922.

13. Smoliar, S. W. "Schenker: A Computer Aid for Analysing Tonal Music." University of Pennsylvania Music Project Report No. 6, January, 1976.

14. Ulrich, John W. "The Analysis and Synthesis of Jazz by Computer." *Proceedings of the 5th International Joint Conference on Artificial Intellegence*, 2 (1977), pp. 865-872.

15. Westergaard, P. *An Introduction to Tonal Theory.* New York: W. W. Norton & Co. Inc., 1975.

16. Winograd, T. "Linguistics and the Computer Analysis of Tonal Harmony." *Journal of Music Theory,* 12, No. 1 (1969), pp. 2-49.

17. Yasser, J. *A Theory of Evolving Tonality.* New York: De Capo Press, 1975.

18. Davis, R. and King, J. "An Overview of Production Systems." Stanford Artificial Intelligence Laboratory Memo AIM-271, October, 1975.

19. Kaiser, Gail E. "Automatic Extension of an Augmented Transition Network Grammer for Morse Code Conversations." Massachusetts Institute of Technology Laboratory for Computer Science Technical Report MIT/LCS/TR-233, March, 1980.

20. Marcus, M. P. "An Overview of A Theory of Syntactic Recognition for Natural Language." Massachusetts Institute of Technology Artificial Intelligence Laboratory AI Memo No. 531, July, 1979.

21. Minsky, M. "A Framework for Representing Knowledge." Massachusetts Institute of Technology Artificial Intelligence Laboratory AI Memo 306, June, 1974.

22. Sacerdoti, E. D., "A Structure for Plans and Behavior." SRI Technical Note 109, 1975.

23. Sussman, Gerald J. "Slices, at the Boundary between Analysis and Synthesis." Massachusetts Institute of Technology Artificial Intelligence Laboratory AI Memo 433, July, 1977.

24. Bertoni, A., Haus G., Mauri G. and Torelli M. "A Mathematical Model for Analysing and Structuring Musical Texts." *Interface.* 7 (1978), pp. 31-43.

25. Paseman, W. G. "Some Applications of Compiler Concepts to Music." Massachusetts Institute of Technology Programming Language Processors Term Paper, December, 1978.

26. Alles, Harold G. "Music Synthesis Using Real Time Digital Techniques." *Proceedings of the IEEE,* 68, No. 4 (April 1980), pp. 436-449.

27. Alonzo, S. Personal Communication

28. Amuedo, J. Personal Communication

29. Chamberlin, H. "Advanced Real Time Music Synthesis Techniques." *Byte,* (April, 1980)

30. Chowning, J. M. "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation." *Journal of the Audio Engineering Society*, 21, No. 7 (September, 1973)

31. Harmuth, H. "*Transmission of Information by Orthogonal Functions.*" Springer Verlag, New York., (1972).

32. Moore, F. R. "Table Lookup Noise for Sinusoidal Digital Oscillators." *Computer Music Journal*, 1, No. 2 (April, 1977), pp. 26-30.

33. Moorer, James A. "On the Transcription of Musical Sound by Computer." *Computer Music Journal*, 1, No. 4 (November, 1977), pp. 32-38.

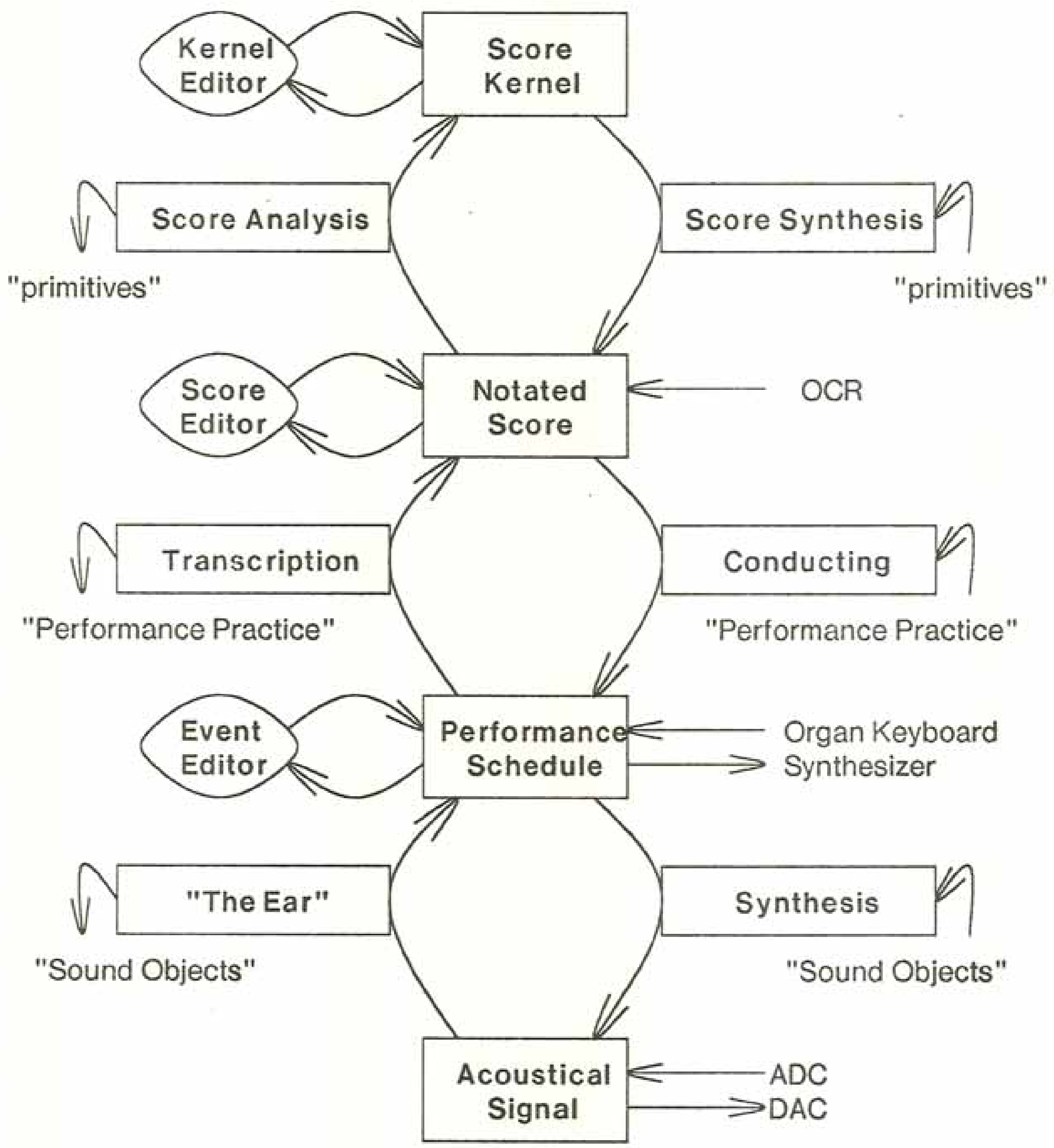
34. Rakowski, A. "Pitch Discrimination at the Threshold of Hearing." *Proceedings of the Seventh International Congress on Acoustics*, 3, Budapest, (1971) pp. 373-376.

35. Snell, J. "Design of a Digital Oscillator Which Will Generate 256 Low Distortion Sine Waves in Real Time." *Computer Music Journal*, 1, No. 2 (April, 1977), pp. 4-25.

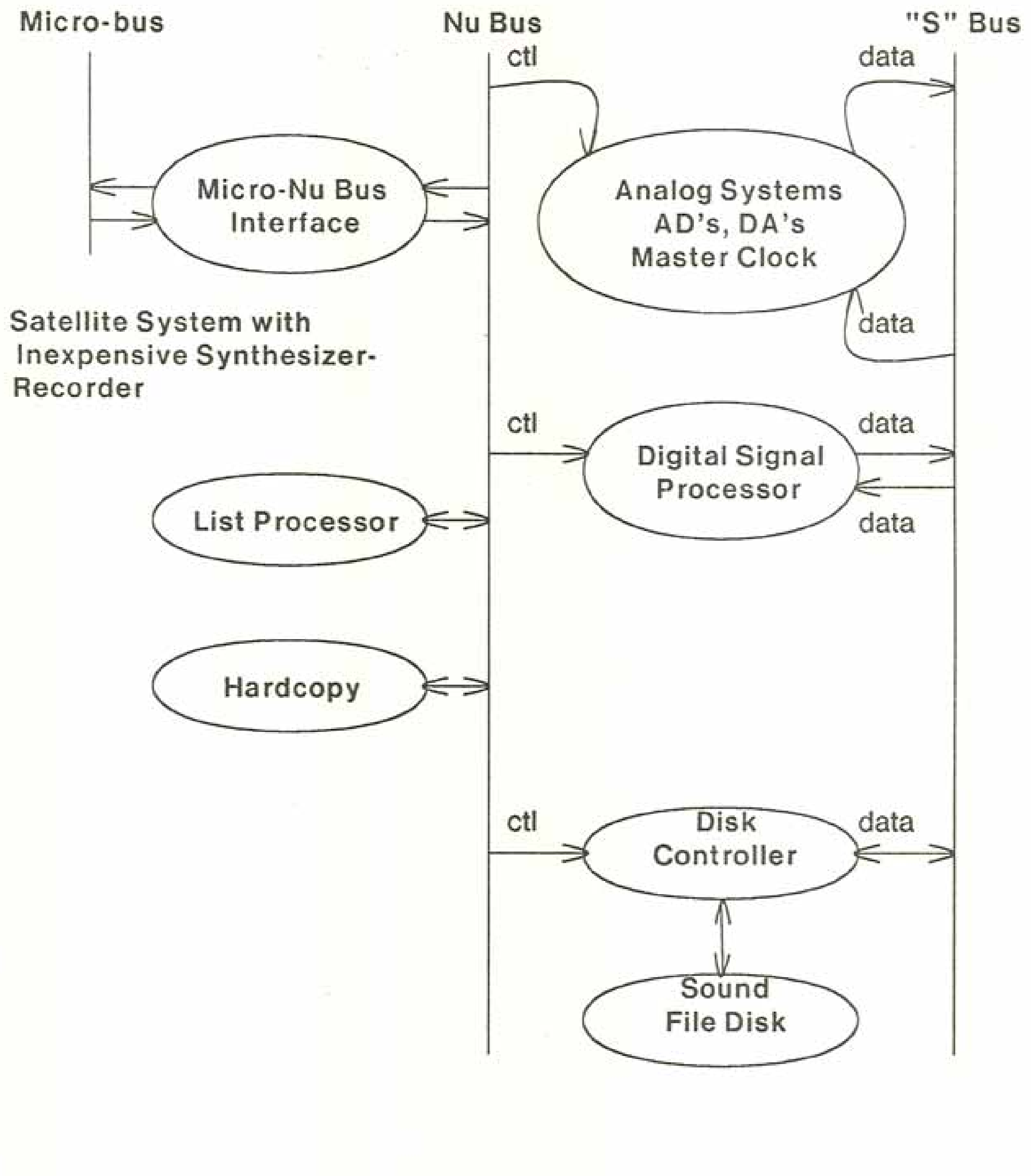
36. Ward, Stephen A. Personal Communication

FIGURES

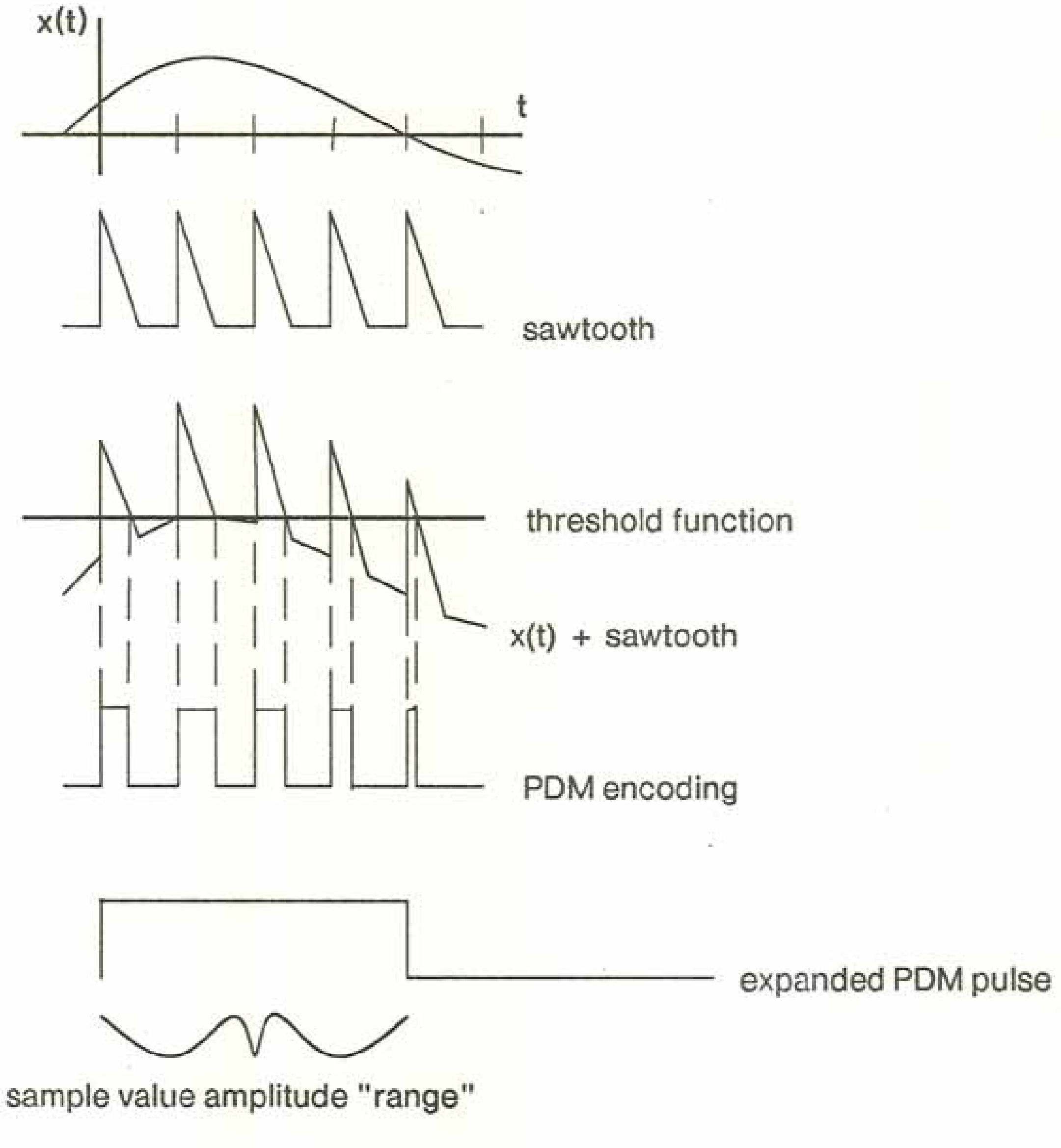
A Model of Music



The Music Project Bus Scheme



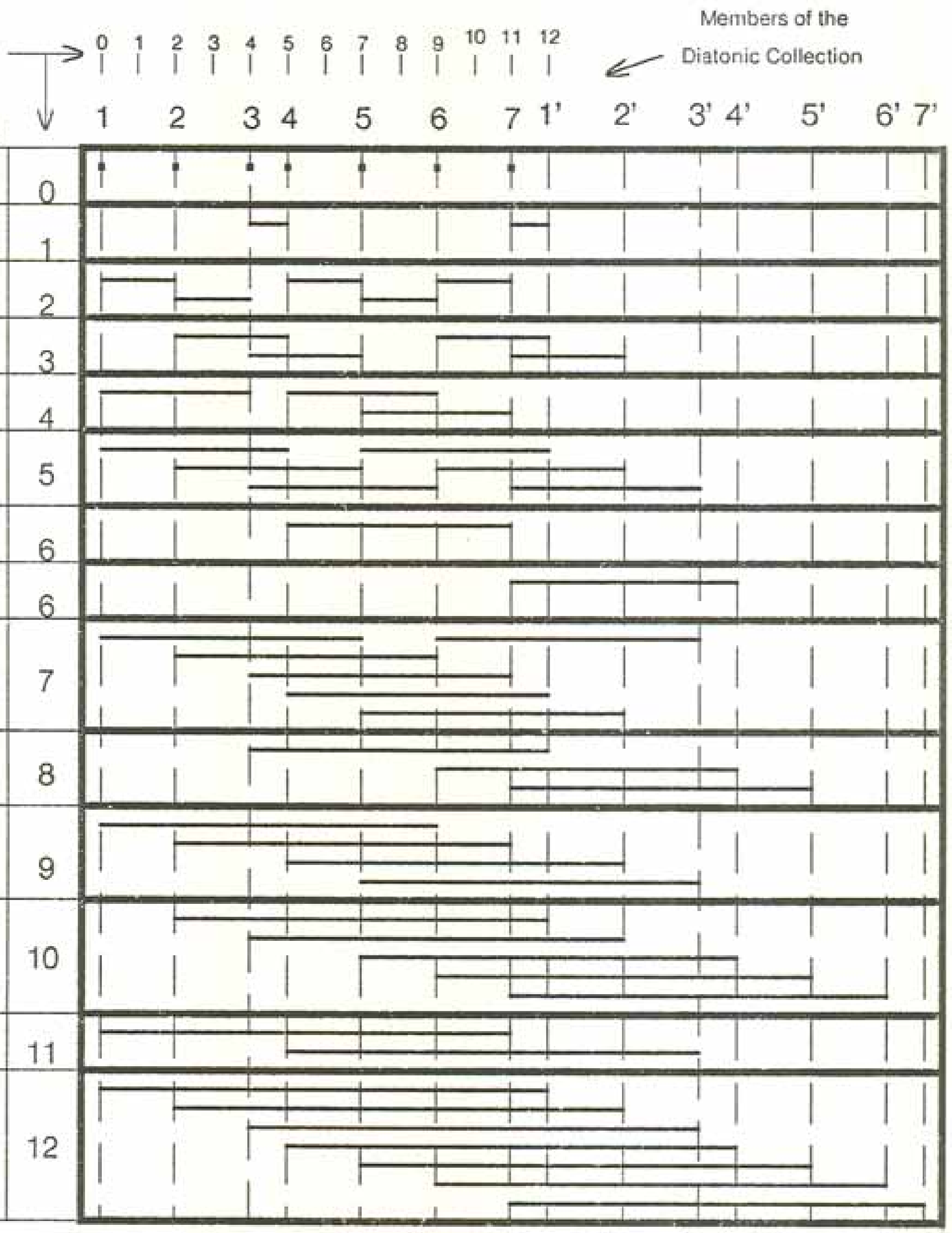
Pulse-Duration Modulation



Intervals Between the Members of the Diatonic Collection Forming the Major Scale

semitone distance

	M2	m2	
u1	0	0	0
m2 ▪	0	1	1
M2 ▪	1	0	2
m3	1	1	3
M3	2	0	4
p4	2	1	5
a4 ▪	3	0	6
d5 ▪	2	2	6
p5	3	1	7
m6	3	2	8
M6	4	1	9
m7 ▪	4	2	10
M7 ▪	5	1	11
p8	5	2	12



Part of J. S. Bach's Two Part Invention No. 8

F Fundamental Descending Line

1' 7 6 5 4 3 2 1

I Triadic Insertion

1 1' 7 6 5 4 3 2 1

R Triadic Repetition

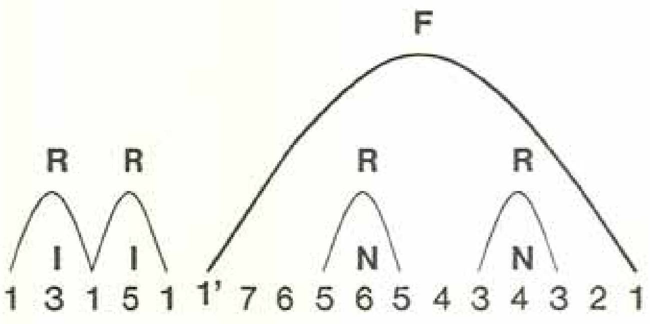
1 **1 1** 1' 7 6 5 **5** 4 3 **3** 2 1

N Neighbor Insertion

1 1 1 1' 7 6 5 **6** 5 4 3 **4** 3 2 1

I Triadic Insertion

1 **3** 1 **5** 1 1' 7 6 5 6 5 4 3 4 3 2 1



Major I

```
                                                                Fundamental
S --> @E D C | @G F E D C | @C' B A G F E D C                   Descending
                                                                   Line

C --> C C | 1                                                     Triadic
E --> E E | 3                                                    Repetition
G --> G G | 5


C'C' --> C'D'C' | C' B C'       D --> 2                           Neighbor
E E  --> E D E  | E F E         F --> 4                           Insertion
G G  --> G A G  | G F G         A --> 6
                                B --> 7

U V --> U 1' V     { 1,3,4,5,6,3',4',5',6',1" } ∈ V               Triadic
                                      { @, V } ∈ U                Insertion

W X --> W 3' X     { 3,5,6,7,1',5',6',7',1",3" } ∈ X
                                      { @, X } ∈ W

Y Z --> Y 5' Z   { 5,7,1',2',3',7',1",2",3",5" } ∈ Z
                                      { @, Z } ∈ Y
```

Major I
(Continued)

Ascension

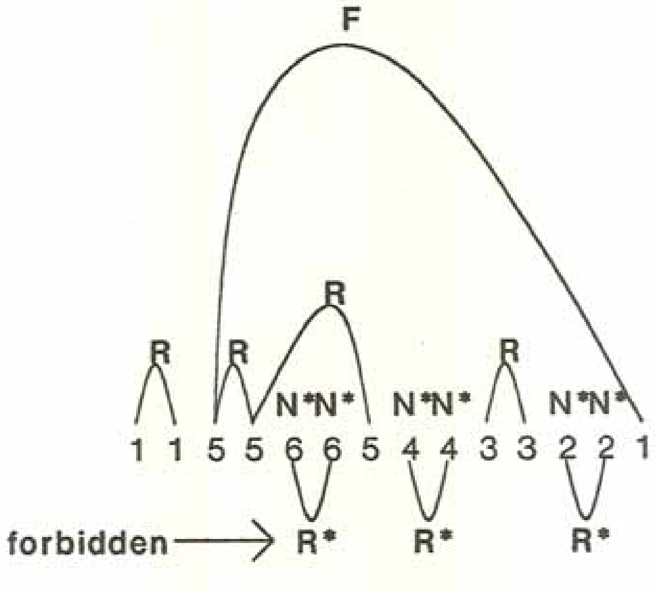
C	E	-->	C	D	E
C	J	-->	C	D	K
D	F	-->	D	E	F
D	K	-->	D	E	L
E	G	-->	E	F	G
E	L	-->	E	F	M'
F	A	-->	F	G	A
F	M'	-->	F	G	N'
G	B	-->	G	A	B
G	N'	-->	G	A	P'
A	C'	-->	A	B	C'
A	P'	-->	A	B	R'
B	D'	-->	B	C'	D'
B	R'	-->	B	C'	J'

{4 5 6 7 1' } $\in$	J
{5 6 7 1' 2' } $\in$	K
{6 7 1' 2' 3' } $\in$	L
{7 1' 2' 3' 4' } $\in$	M'
{1' 2' 3' 4' 5' } $\in$	N'
{2' 3' 4' 5' 6' } $\in$	P'
{3' 4' 5' 6' 7' } $\in$	R'
{4' 5' 6' 7' 1'' } $\in$	J'

Descension

B'	G'	-->	B'	A'	G'
B'	M'	-->	B'	A'	L
A'	F'	-->	A'	G'	F'
A'	L	-->	A'	G'	K
G'	E'	-->	G'	F'	E'
G'	K	-->	G'	F'	J
F'	D'	-->	F'	E'	D'
F'	J	-->	F'	E'	R
E'	C'	-->	E'	D'	C'
E'	R	-->	E'	D'	P
D'	B	-->	D'	C'	B
D'	P	-->	D'	C'	N
C'	A	-->	C'	B	A
C'	N	-->	C'	B	M

Twinkle, Twinkle Little Star



Major IIa

```
                                          Fundamental Descending Line
 S -->  @E D C | @G F E D C | @C' B A G F E D C
----------------------------------------------------------------------
 C -->  1 C | 1                D --> 2            Triadic
 E -->  3 E | 3                F --> 4            Repetition
 G -->  5 G | 5     B --> 7    A --> 6
----------------------------------------------------------------------
                                                  Neighbor
 0 -->  -1 1  | 1 -1                              Insertion
----------------------------------------------------------------------
 U V -->  U 1' V                                  Triadic
          { 1,3,4,5,6,3',4',5',6',1''  } ∈ V      Insertion
                             { @, V } ∈ U
 W X -->  W 3' X
          { 3,5,6,7,1',5',6',7',1'',3''  } ∈ X
                             { @, X } ∈ W
 Y Z -->  Y 5' Z
          { 5,7,1',2',3',7',1'',2'',3'',5''  } ∈ Z
                             { @, Z } ∈ Y
----------------------------------------------------------------------
 2 -->  1 1                                       Ascension
 3 -->  1 2
 4 -->  1 3
 5 -->  1 4
 7 -->  1 1 5
----------------------------------------------------------------------
 -2 -->  -1 -1                                    Descension
 -3 -->  -1 -2
 -4 -->  -1 -3
 -5 -->  -1 -4
 -7 -->  -1 -1 -5
```

Multiple Grammars

A

```
Composition --> Mevent
      Mevent --> Mevent*        | Note1    | Note2
```

B

```
Ga    Note1 = Note2 -->    C | E | G | B | D | F | A | C'
```

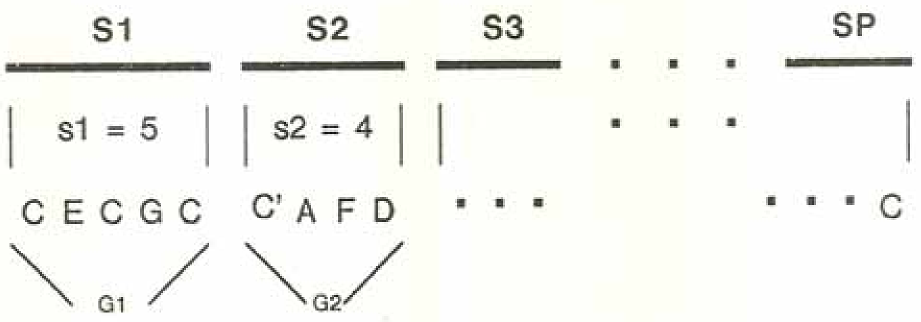
C

```
G1            Note1 -->    C | E | G | B

G2                          Note2 -->    D | F | A | C'
```

D

$$
\begin{array}{l|l|cccccccc}
 & & \multicolumn{8}{c}{\text{X terminals}} \\
 & \text{Grammar 1} & t11 & t12 & \cdot & \cdot & \cdot & \cdot & \cdot & t1X \\
 & \text{Grammar 2} & t21 & t22 & \cdot & \cdot & \cdot & \cdot & \cdot & t2X \\
\text{G Grammars} & \vdots & & \cdot & & & & & & \\
 & \vdots & & & & \cdot & & & & \\
 & \vdots & & & & & & & \cdot & \\
 & \text{Grammar G} & tG1 & tG2 & \cdot & \cdot & \cdot & \cdot & \cdot & tGX
\end{array}
$$

E



P Segments, each of length s, containing N total notes.

Major IIb

				Fundamental Descending Line
S -->	@E D C | @G F E D C | @C' B A G F E D C			
C -->	1 C | 1			Triadic Repetition
E -->	3 E | 3			
G -->	5 G | 5			
		D -->	2	Neighbor Insertion
C' -->	1' D'C' | 1' B C'	F -->	4	
E -->	3 D E | 3 F E	A -->	6	
G -->	5 A G | 5 F G	B -->	7	
U V -->	U 1' V			Triadic Insertion
	{ 1,3,4,5,6,3',4',5',6',1'' } ∈ V			
	{ @, V } ∈ U			
W X -->	W 3' X			
	{ 3,5,6,7,1',5',6',7',1'',3'' } ∈ X			
	{ @, X } ∈ W			
Y Z -->	Y 5' Z			
	{ 5,7,1',2',3',7',1'',2'',3'',5'' } ∈ Z			
	{ @, Z } ∈ Y			
A -->	1 | 1 K			Ascension
K -->	1 | 1 L			
L -->	1 | 1 M			
M -->	1 | 1 N			
N -->	1 | 1 P			
P -->	1 1			
D -->	-1 | -1 -K			Descension
-K -->	-1 | -1 -L			
-L -->	-1 | -1 -M			
-M -->	-1 | -1 -N			
-N -->	-1 | -1 -P			
-P -->	-1 -1			

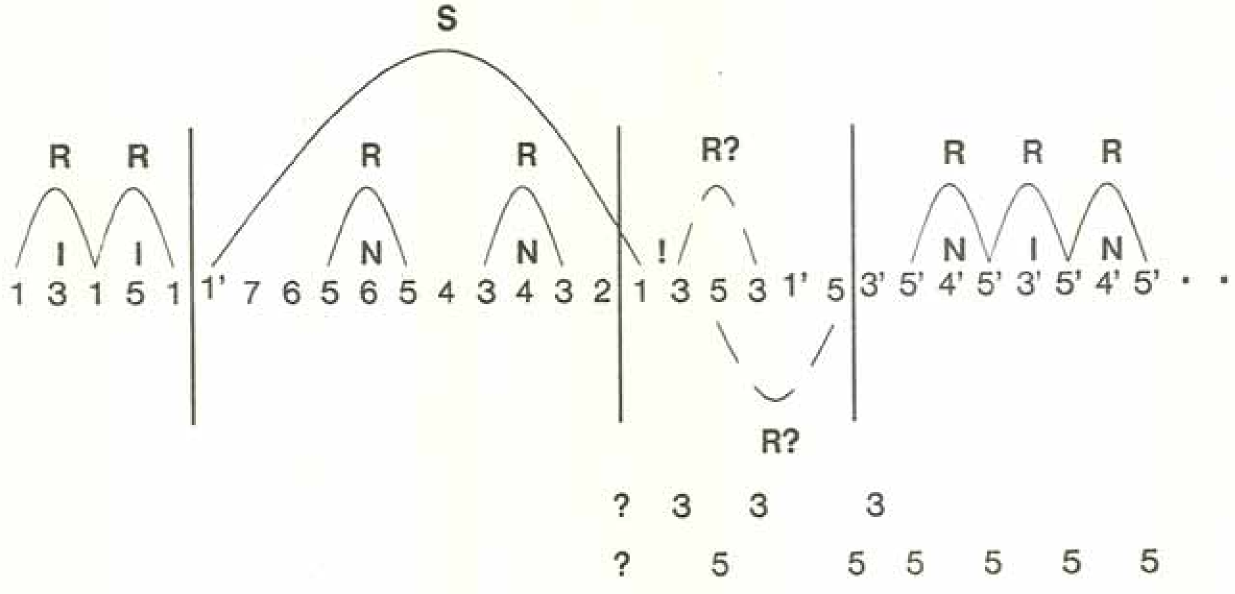
Cherubini

Alternate derivation shown in small font

```
F                               3'                                      2'  1'

I   5                           3'                                      2'  1'

R   5       5                   3'                                  3'  2'  1'

N   5   6   5                   3'                              2'  3'  2'  1'

                                        1   X
I   5   6   5           1'      3'          5                   2'  3'  2'  1'

                                        1   X   1
R   5   6   5           1'      3'          5               5   2'  3'  2'  1'

                                        1   5   1           5
I   5   6   5           1'      3'      1'  5   1'          5   2'  3'  2'  1'

A   5   6   5   6   7   1'  2'  3'      1'  5   1'          5   2'  3'  2'  1'

D   5   6   5   6   7   1'  2'  3'  2'  1'  5   1'  7   6   5   2'  3'  2'  1'

    5   6   5   6   7   1'  2'  3'  2'  1'  5   1'  7   6   5   2'  3'  2'  1'
```

More of

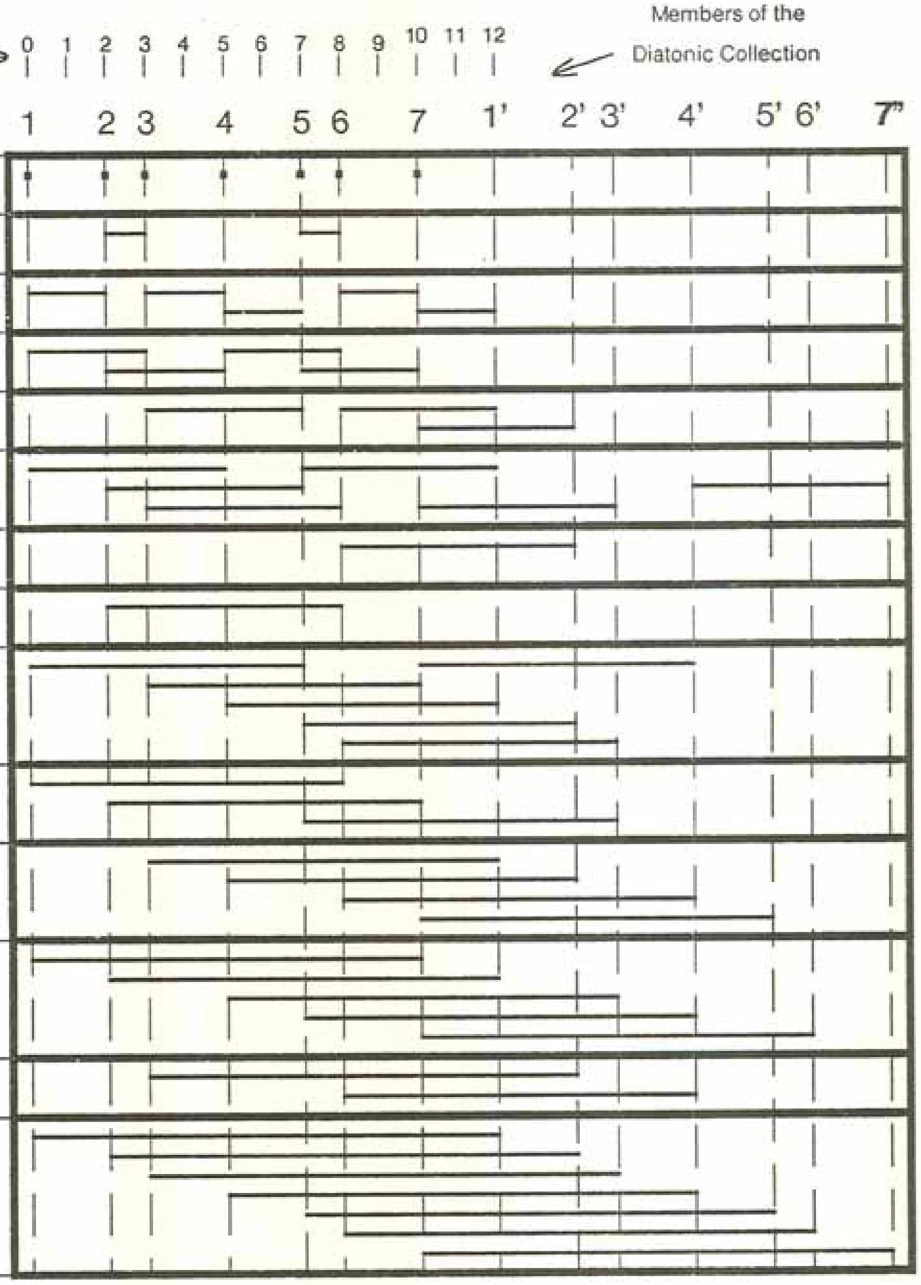
J. S. Bach's Two Part Invention # 8



Intervals Between the Members of the Diatonic Collection Forming the Minor Scale

semitone distance

	M2	m2	
u1	0	0	0
m2 ■	0	1	1
M2 ■	1	0	2
m3	1	1	3
M3	2	0	4
p4	2	1	5
a4 ■	3	0	6
d5 ■	2	2	6
p5	3	1	7
m6	3	2	8
M6	4	1	9
m7 ■	4	2	10
M7 ■	5	1	11
p8	5	2	12



Minor Additions

Neighbor Exception

~~1' 1' --> 1' 7 1'~~ 1' 1' --> 1'7#1'

Ascending Exception

~~5 1' --> 5 6 7 1'~~ 5 1' --> 56#7#1'

~~5 7 --> 5 6 7~~ 5 7 --> 56#7

Descending Exception

7#5 --> 7#6#5

Bach's Bouree

Fundamental Line

```
1'                          7 6 5 4 3                 2 1
```

Repetition

```
1'        1'   1'         1' 7 6 5 4 3    3          3 2 1
```

Neighbor Insertion

```
1'        1' 7 #1' 2'     1' 7 6 5 4 3 2 3           3 2 1
```

Triadic Insertion

```
1'   3'   1' 7# 1' 2' 5   1' 7 6 5 4 3 2 3      5    3 2 1
```

Ascension

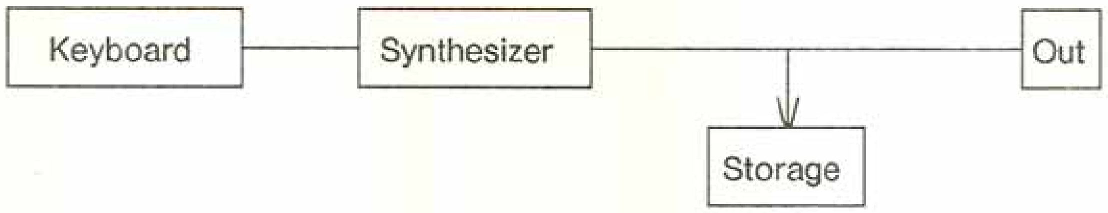
```
1' 2' 3'  1' 7# 1' 2' 5 6# 7 #1' 7 6 5 4 3 2 3 4 5   3 2 1
```

Descension

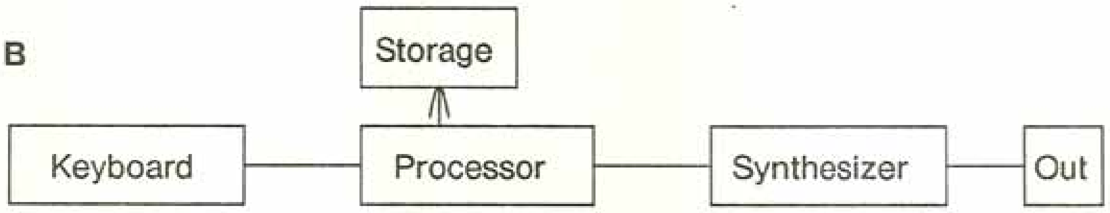
```
1' 2' 3' 2' 1' 7# 1' 2' 5 6#  7# 1' 7 6 5 4 3 2 3 4 5 4 3 2 1
```

A

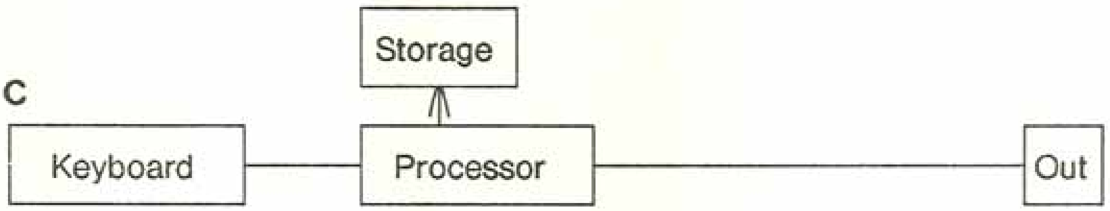
Music Synthesizer System Architectures



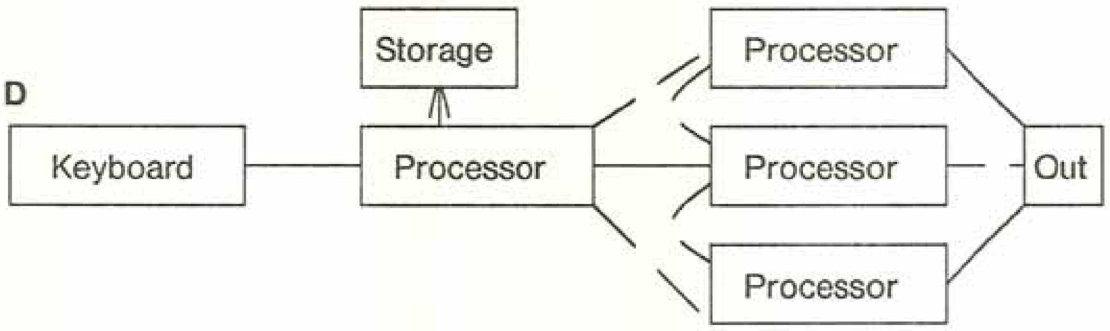
B

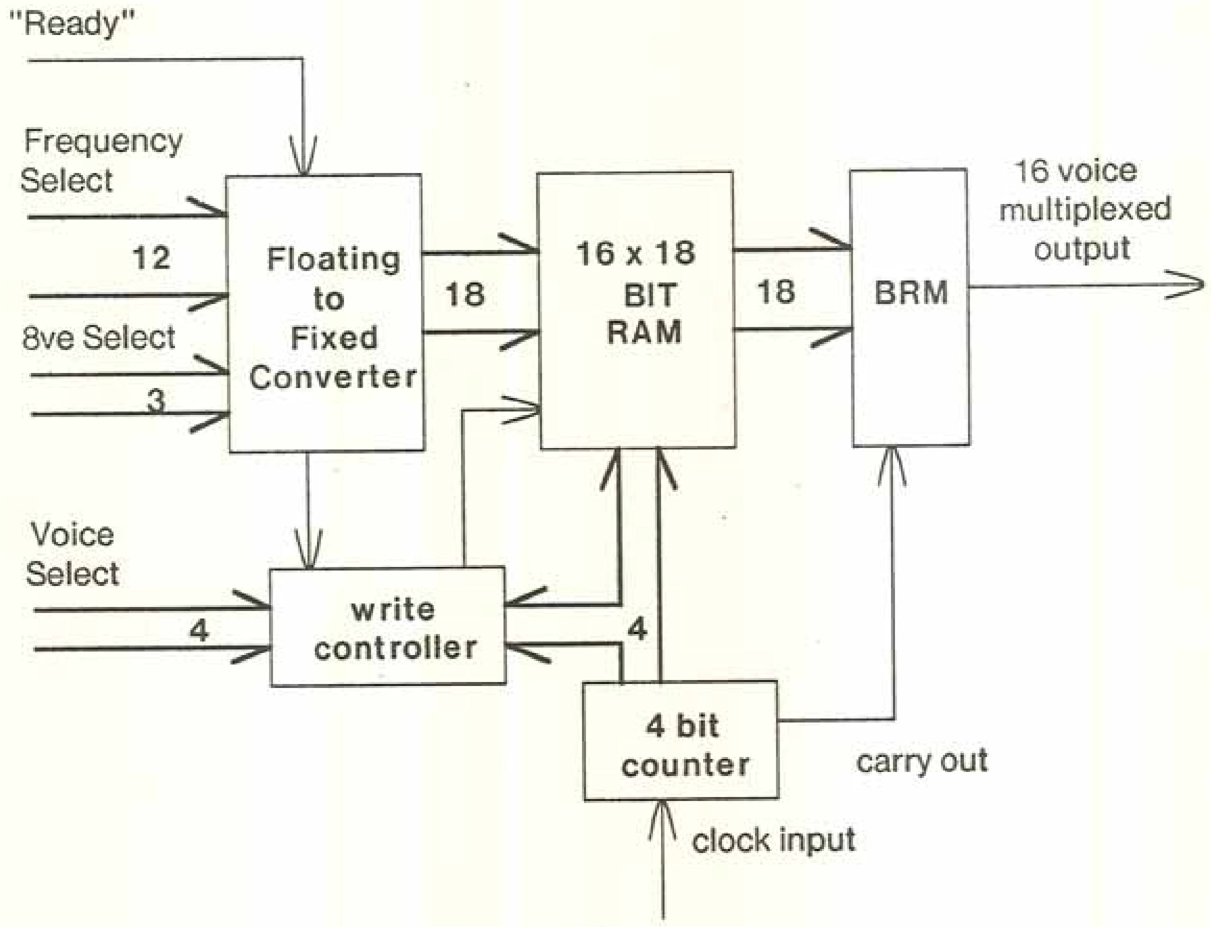


C

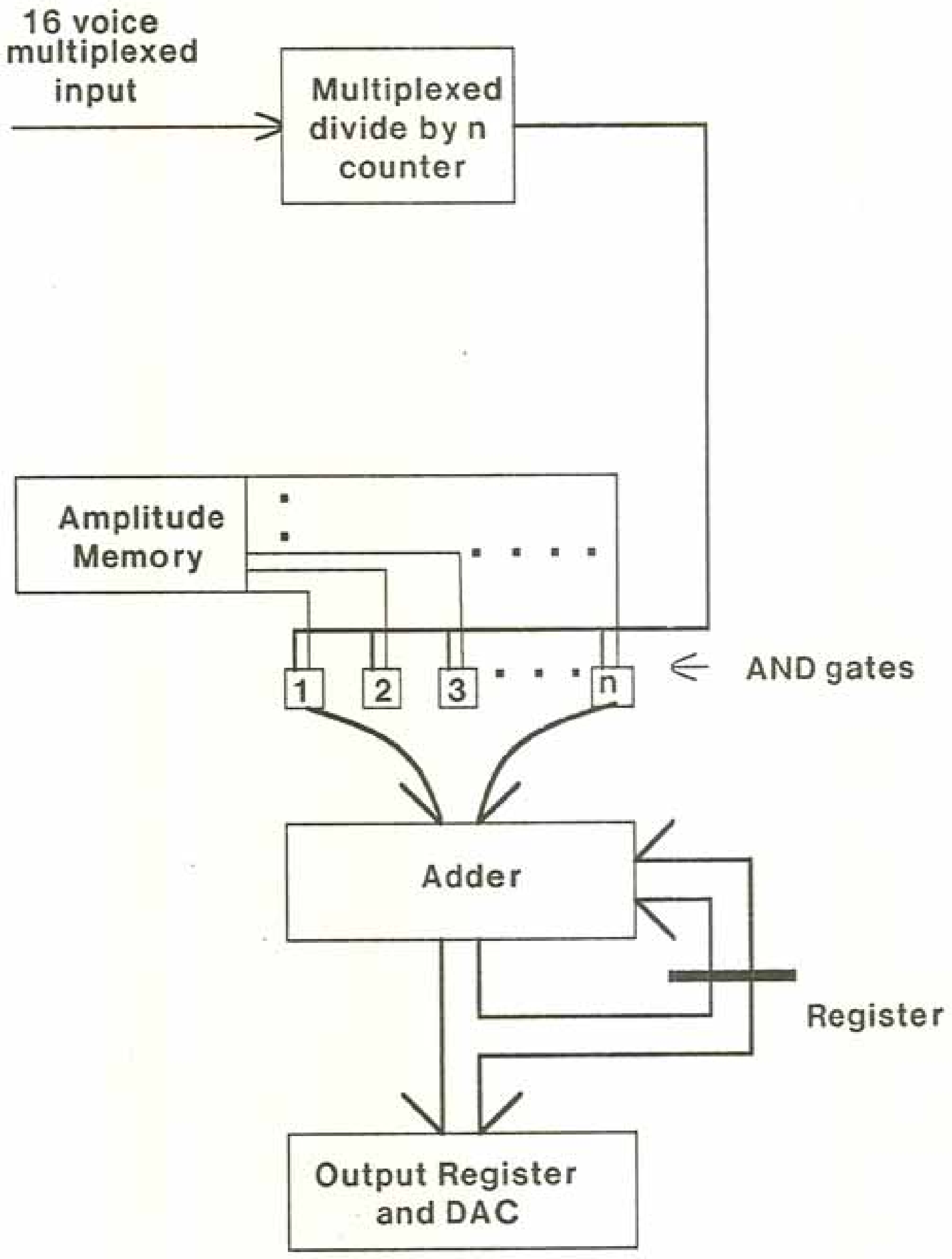


D

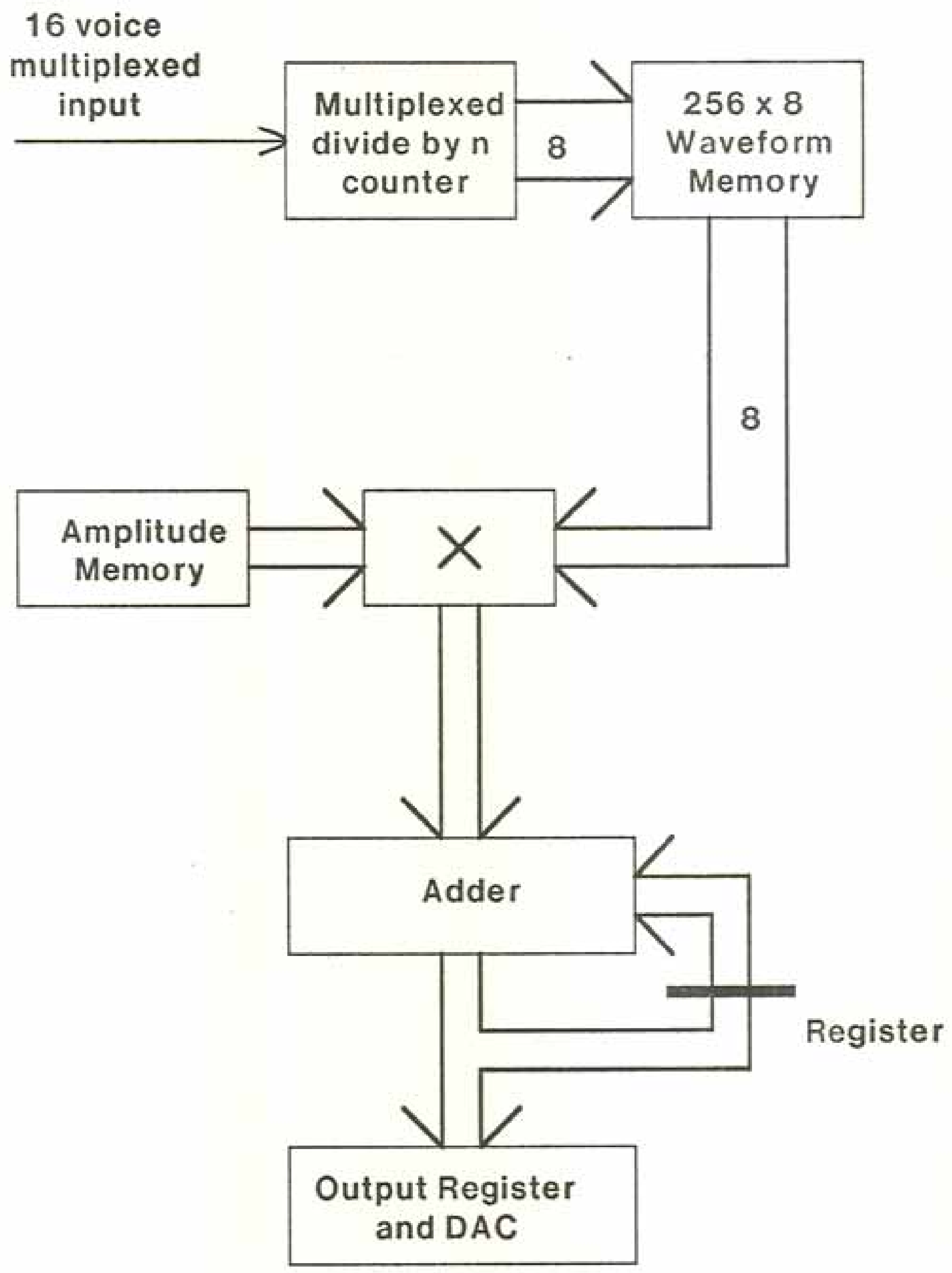




Inexpensive Synthesizer
Frequency Generation Section

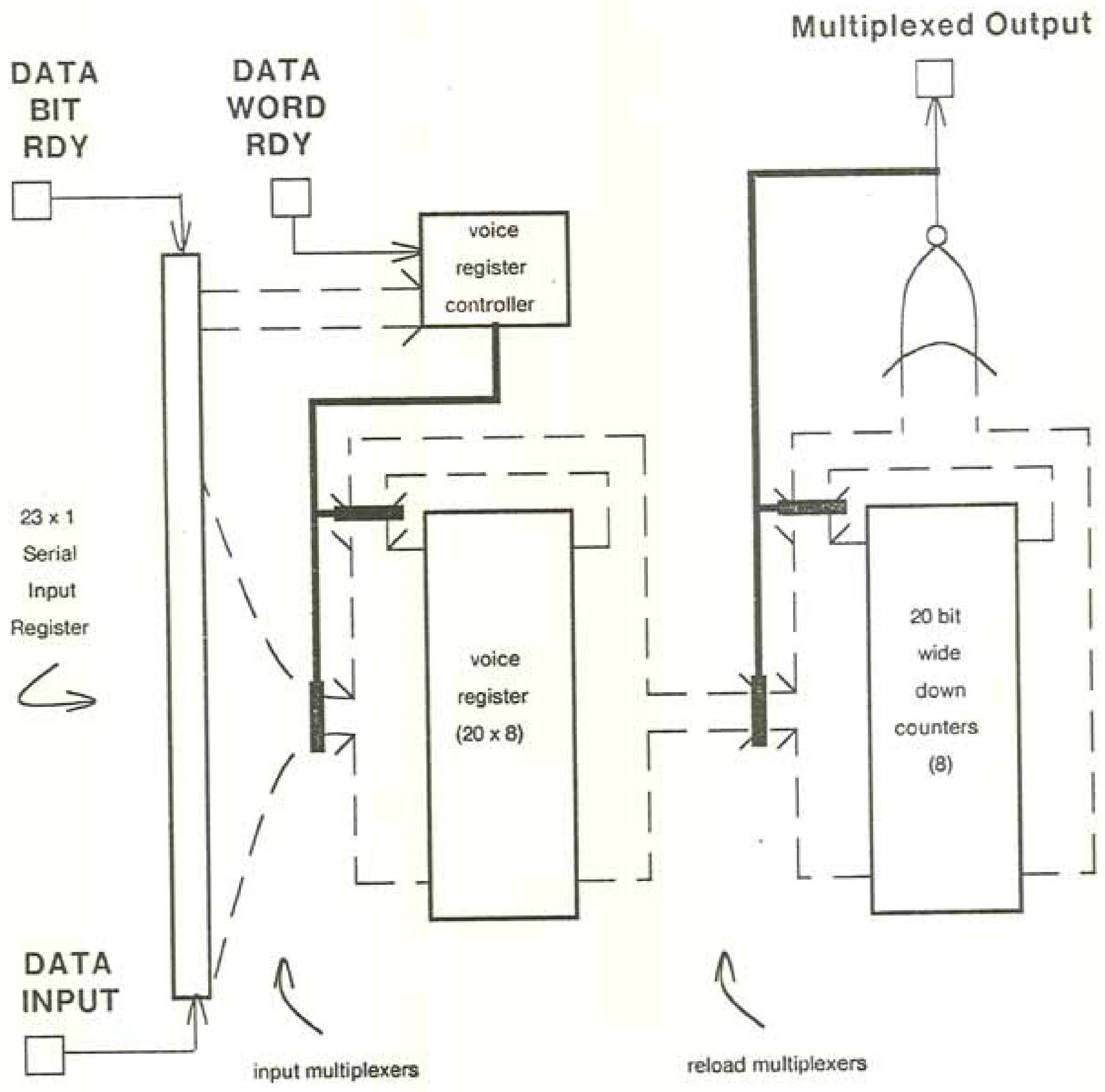


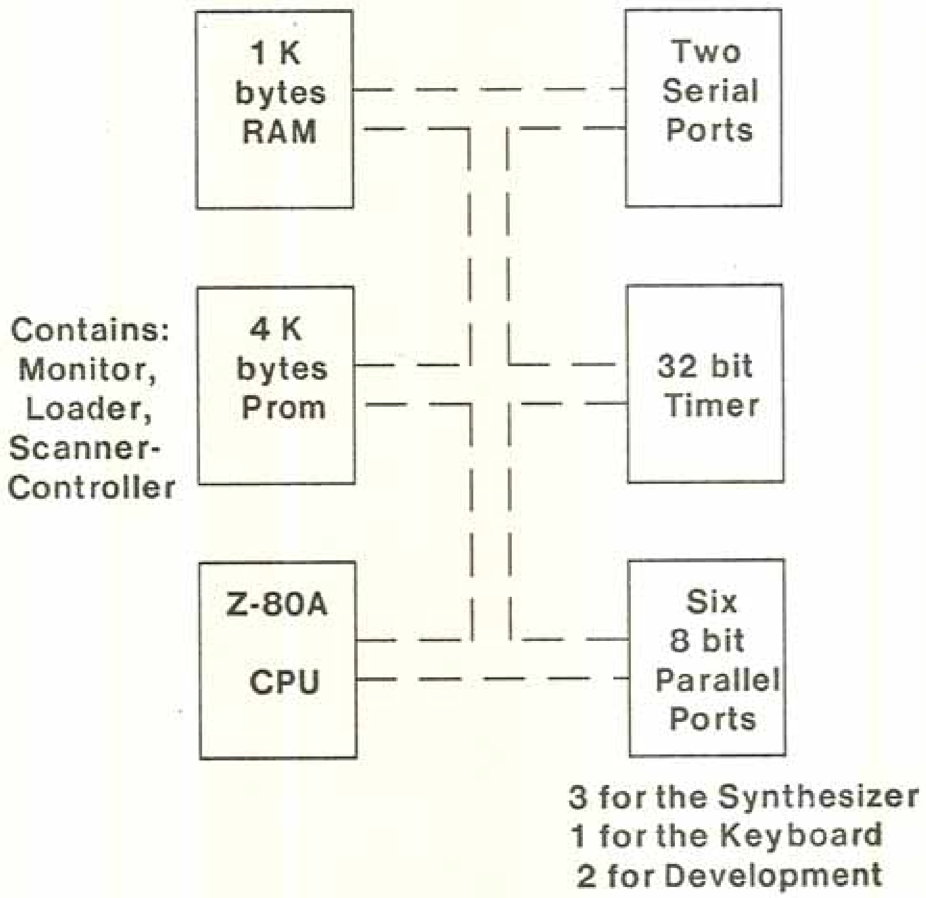
Inexpensive Synthesizer-Square Wave Amplitude Modulation Section



Inexpensive Synthesizer-Rom Amplitude Modulation Section

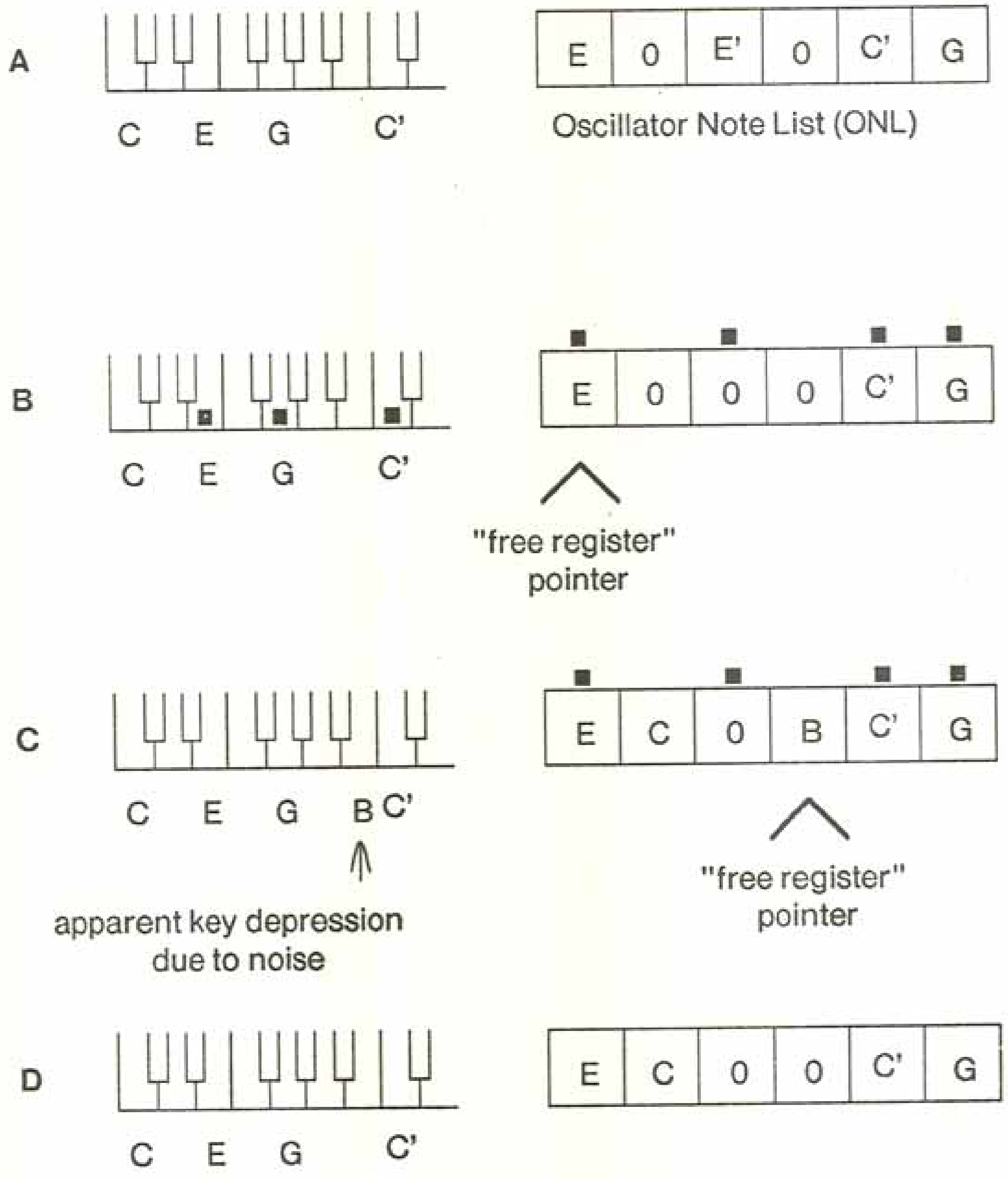
An Eight Voice VLSI Frequency Synthesizer



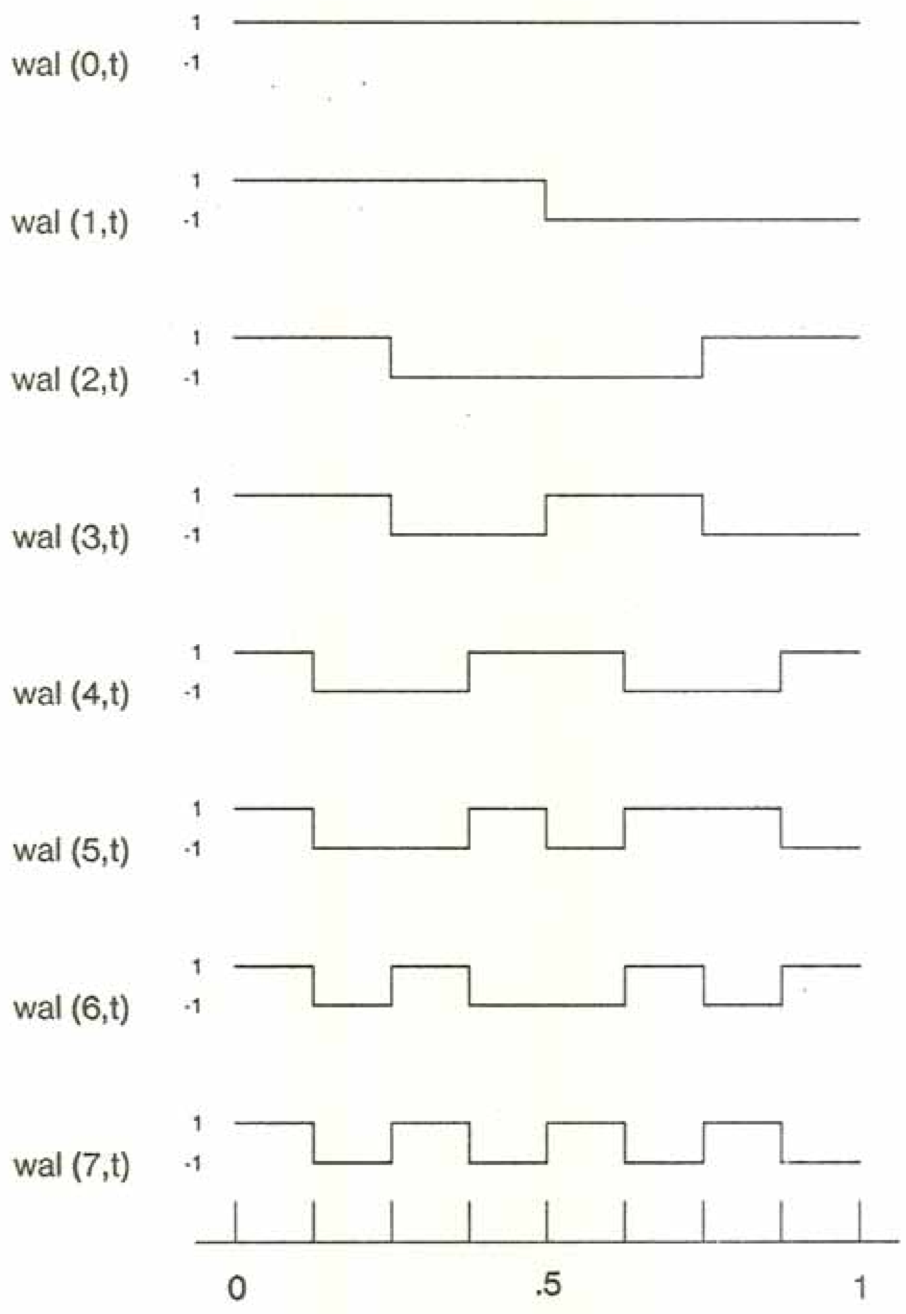


Inexpensive Synthesizer Control Board

The Keyscan Algorithm



The First Eight Walsh Functions



Synthesis Functions Used in Music

I

Walsh Synthesis

$$\sum_{i=1}^{16} ai \cdot \text{Wal}(bi,t)$$

II

Sine Summation Synthesis

$$\sum_{i=1}^{32} Ai \cdot ai \cdot \sin[(wi)(t)]$$

III

FM Synthesis

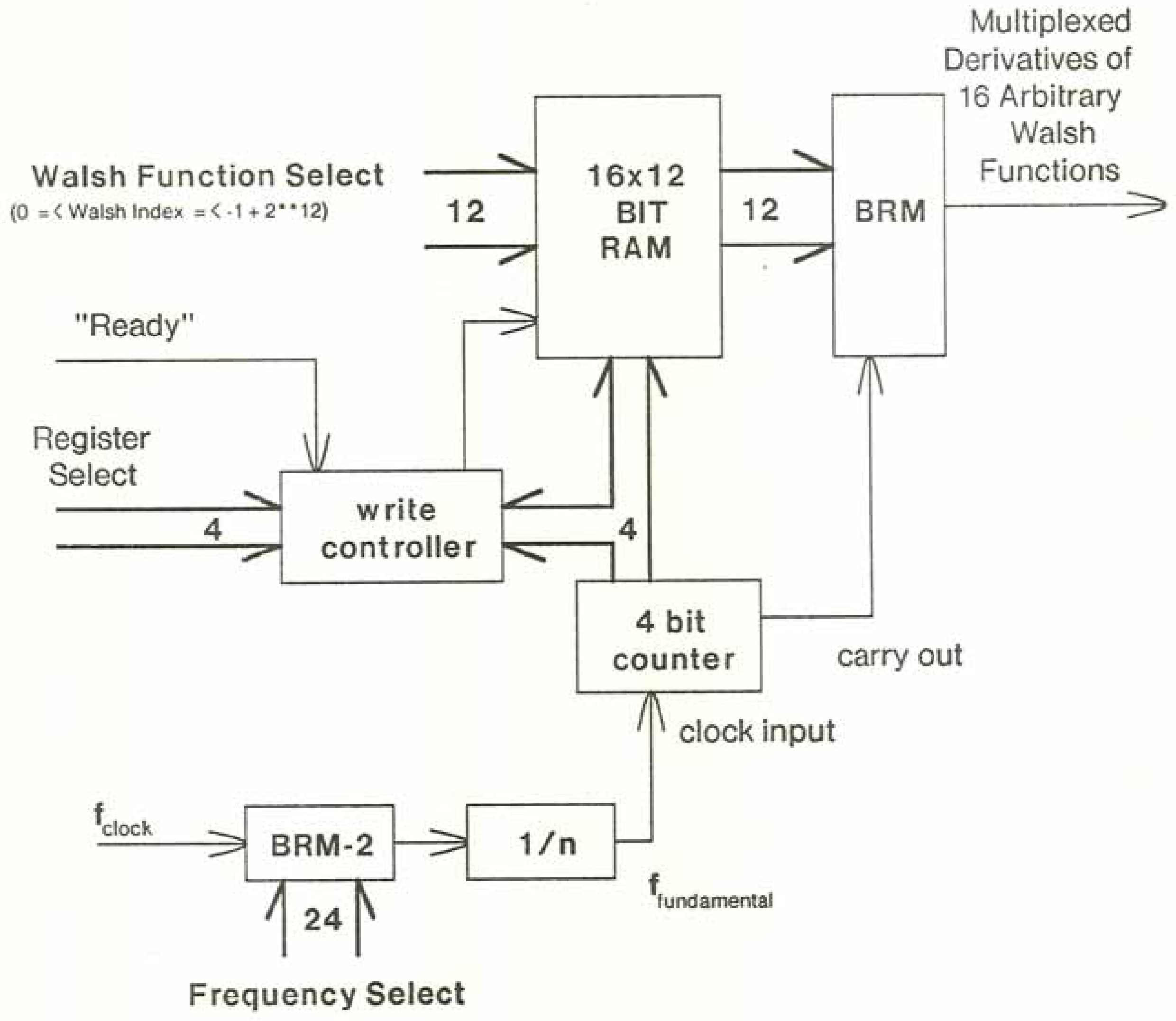
$$\sum_{i=1}^{8} Ai \cdot ai \cdot \sin[(wi)(t) + (Bi)(bi)\sin((vi)(t))]$$

IV

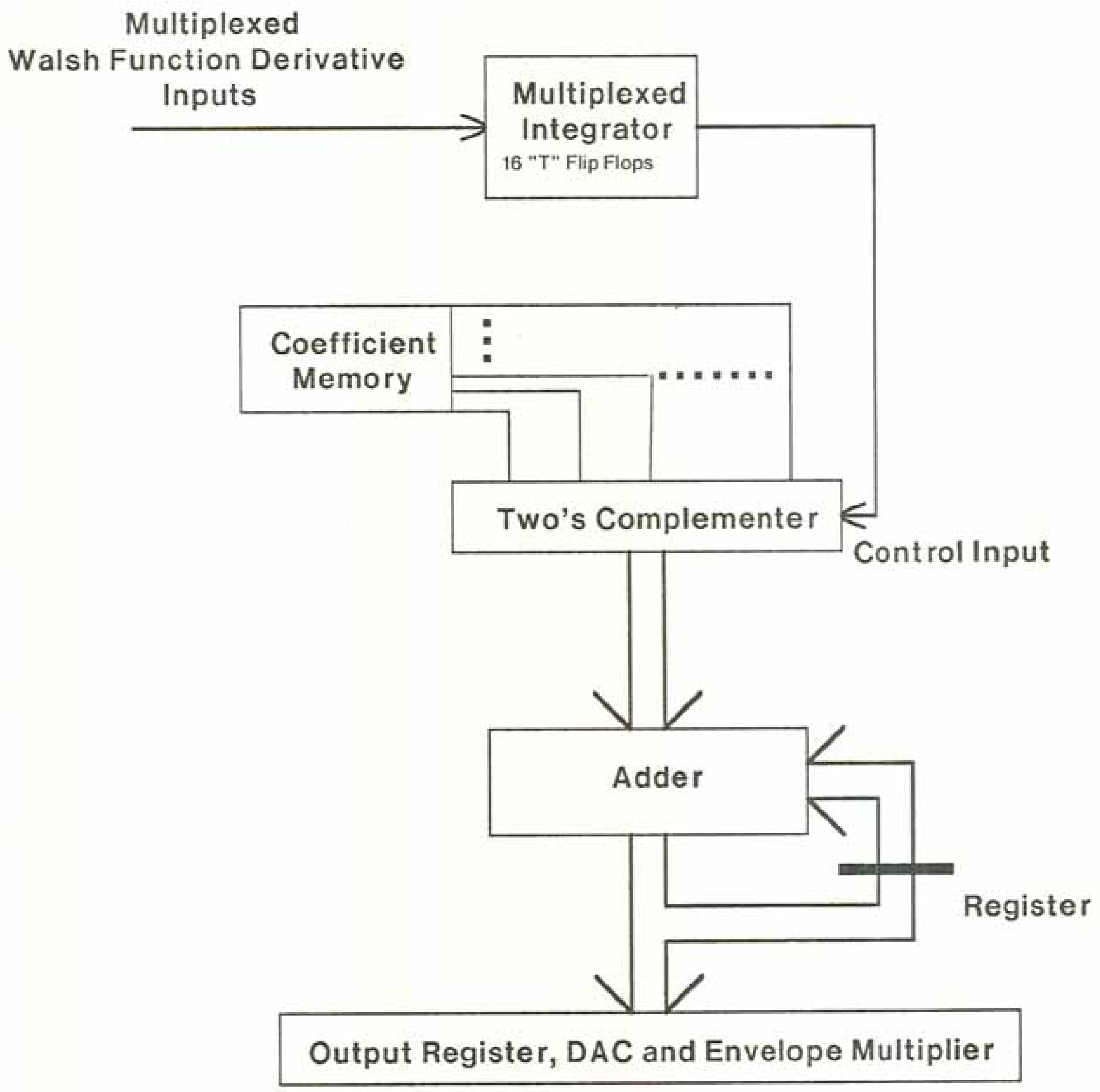
$$Ai \cdot ai \cdot \sin[(wi)(t) + (Bi)(bi)\sin((vi)(t)) + (Ci)(ci)\sin((ui)(t))]$$

V

$$Ai \cdot ai \cdot \sin[(wi)(t) + (Bi)(bi)\sin\{(vi)(t) + (Ci)(ci)\sin((ui)(t))\}]$$

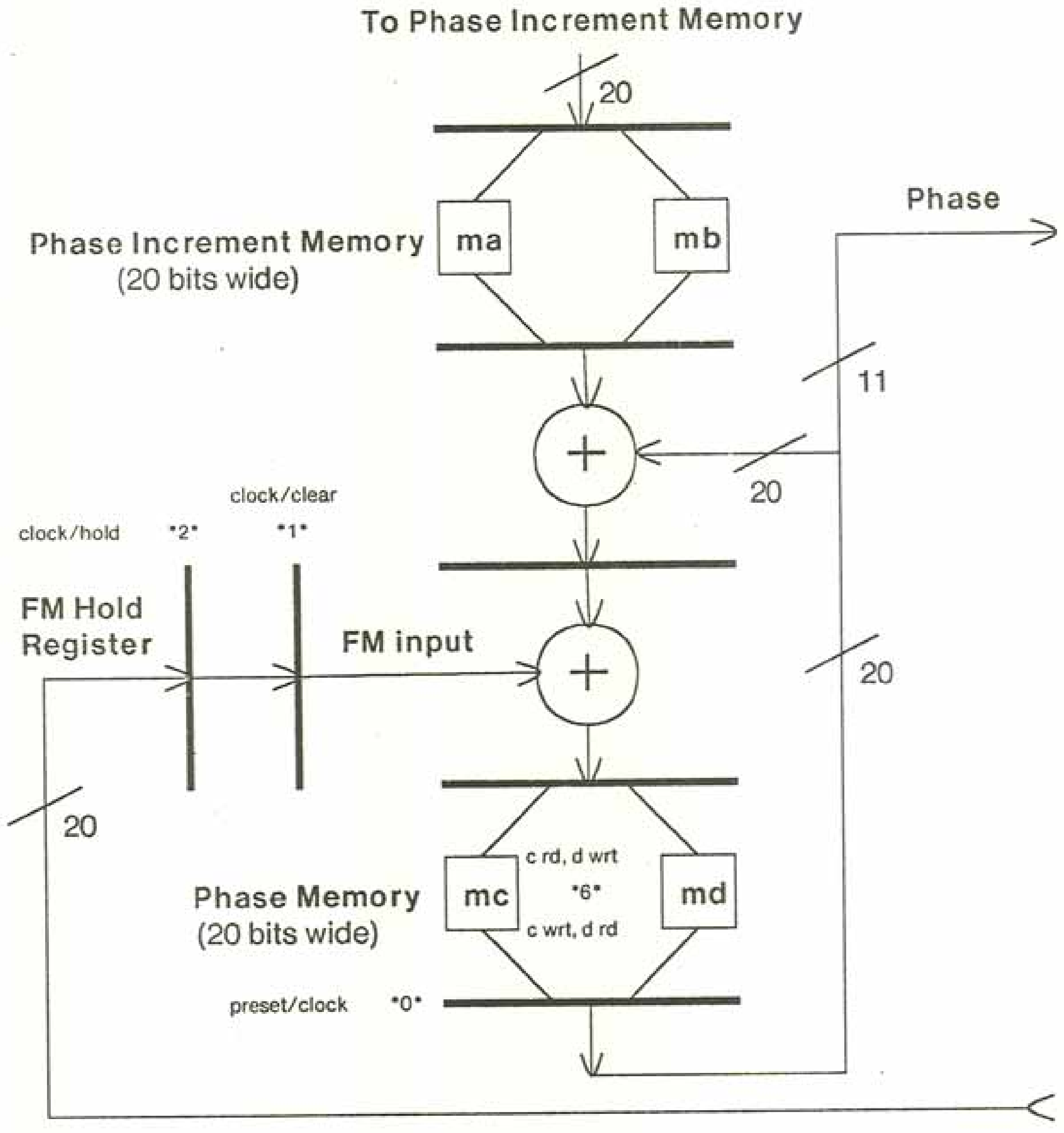


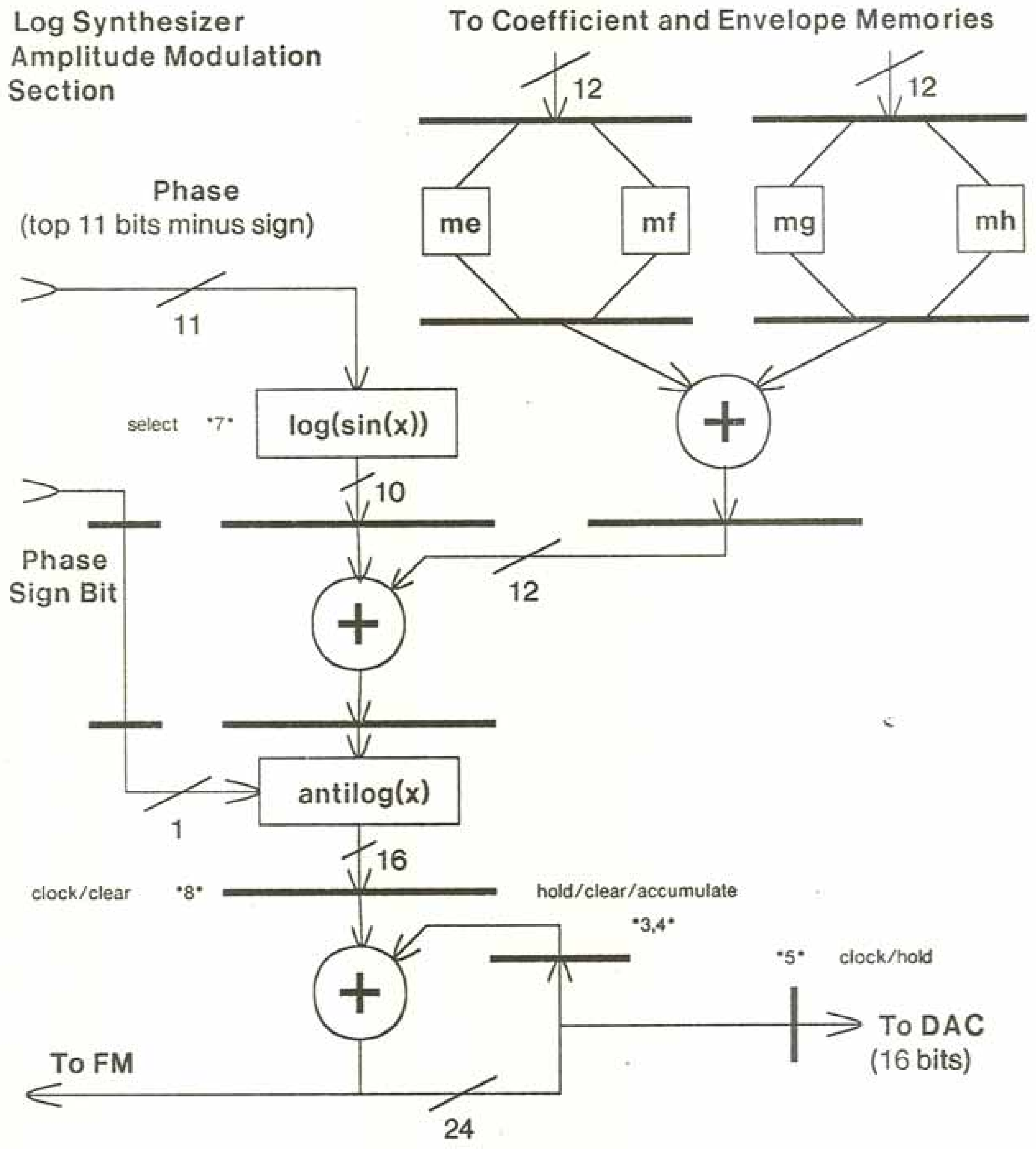
Walsh Function Synthesizer
Walsh Waveform Generation Section



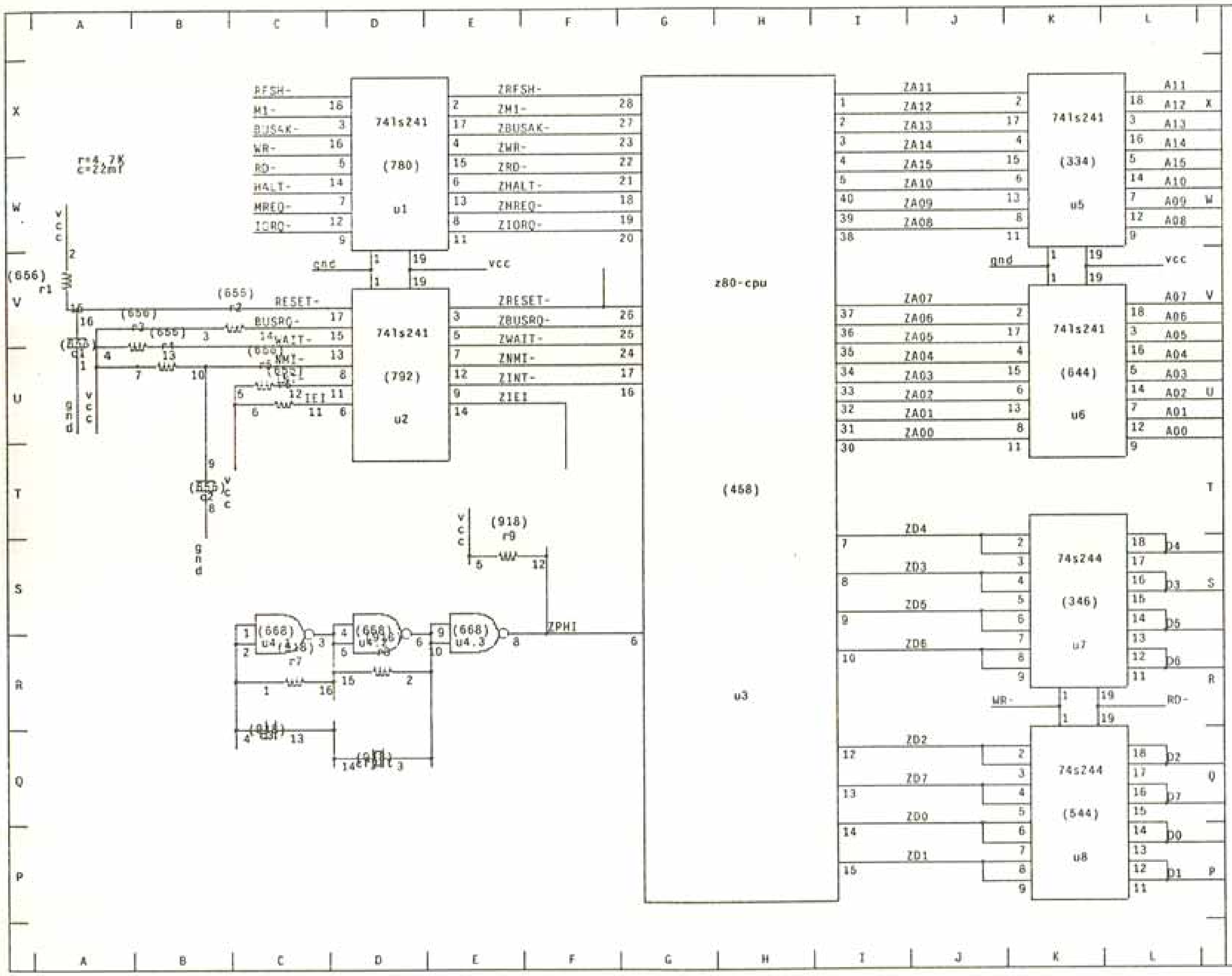
Walsh Function Synthesizer
Coefficient Weighting and Amplitude Modulation Section

Log Synthesizer-Phase Update Section



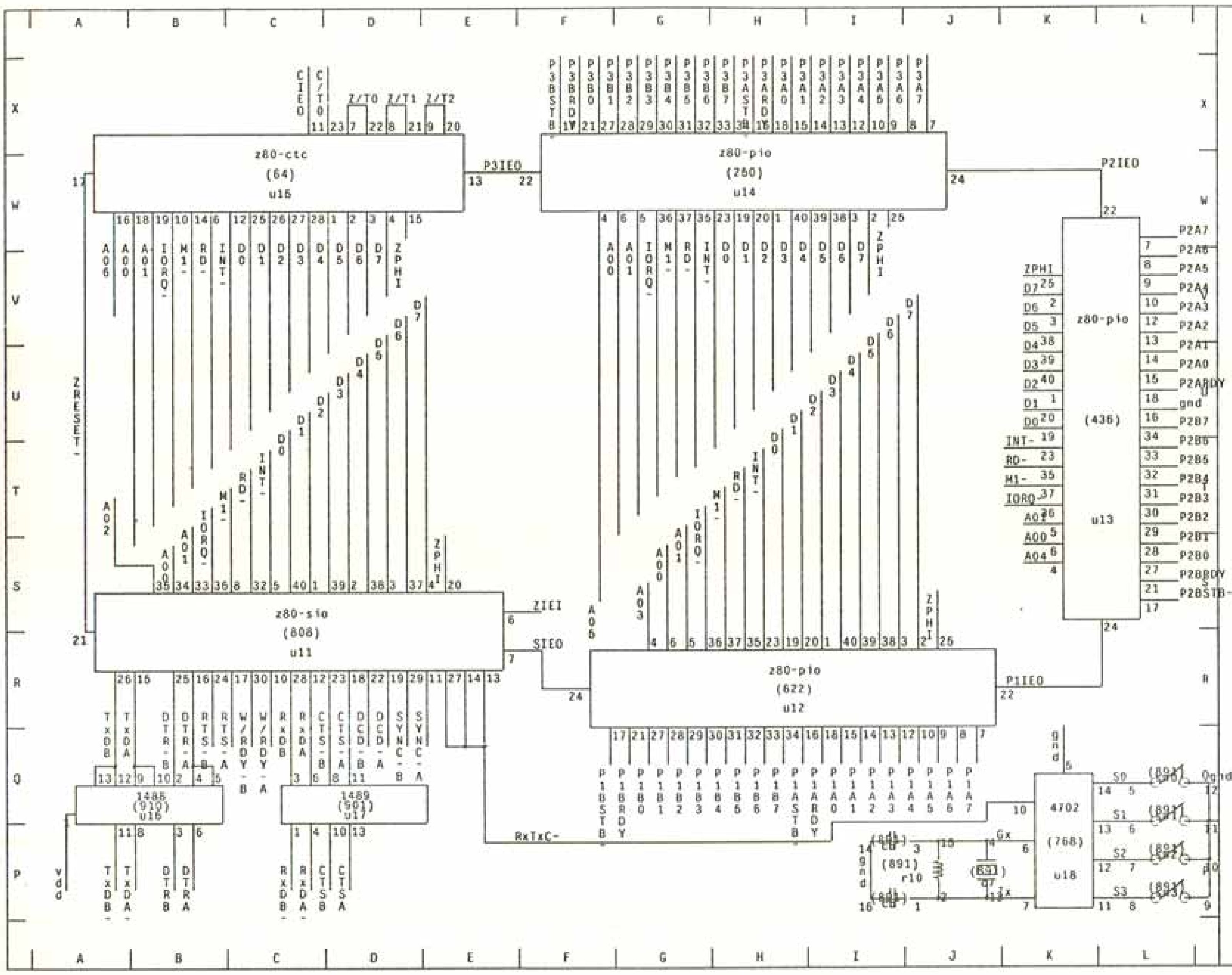

Log Synthesizer Amplitude Modulation Section
To Coefficient and Envelope Memories
12
12
me
mf
mg
mh
Phase (top 11 bits minus sign)
11
select *7*
log(sin(x))
10
12
Phase Sign Bit
antilog(x)
1
16
clock/clear *8*
hold/clear/accumulate
3,4
5 clock/hold
To DAC (16 bits)
To FM
24

APPENDIX



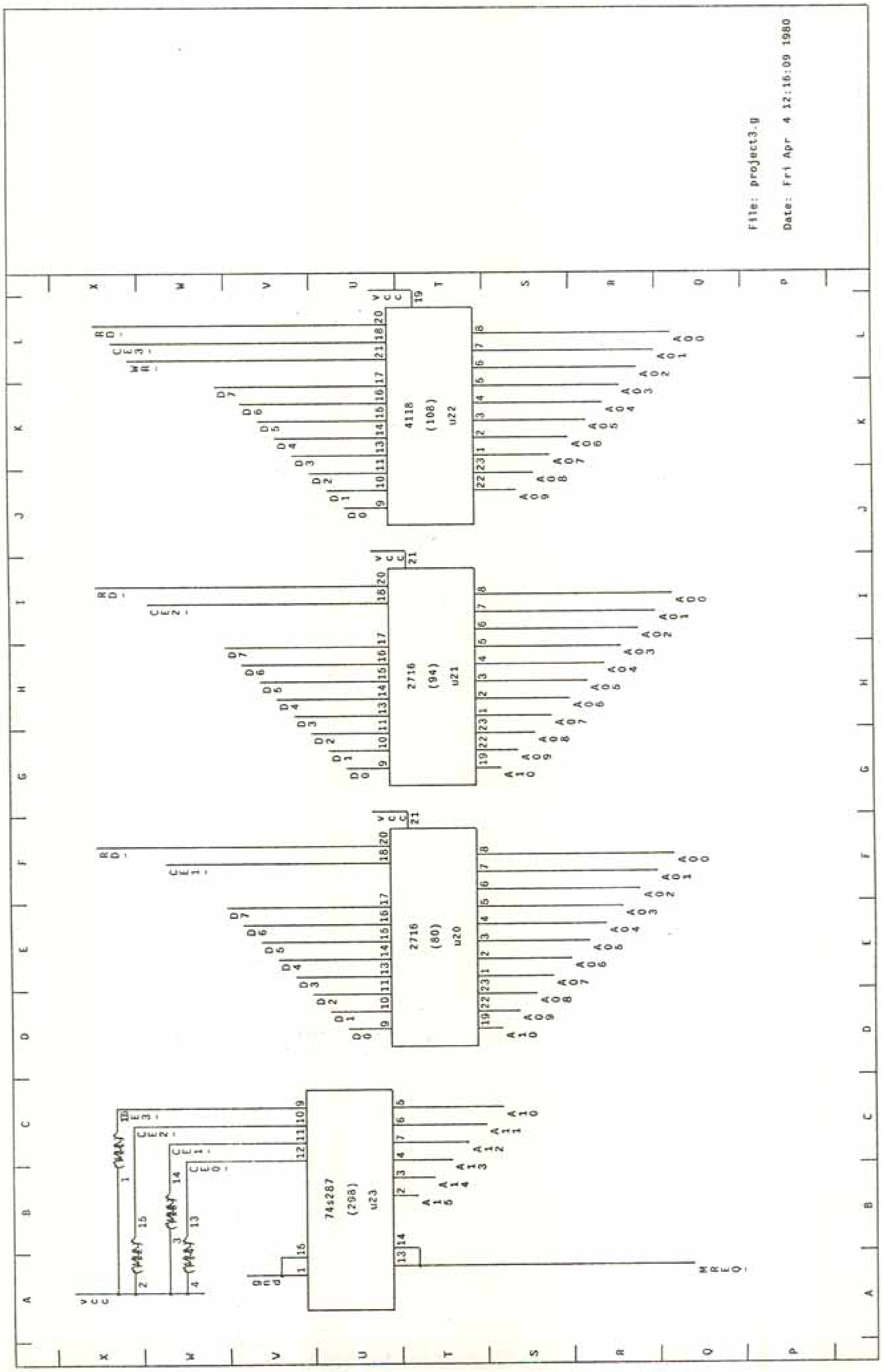
File: project1.g

Date: Fri May 16 13:25:07 1980



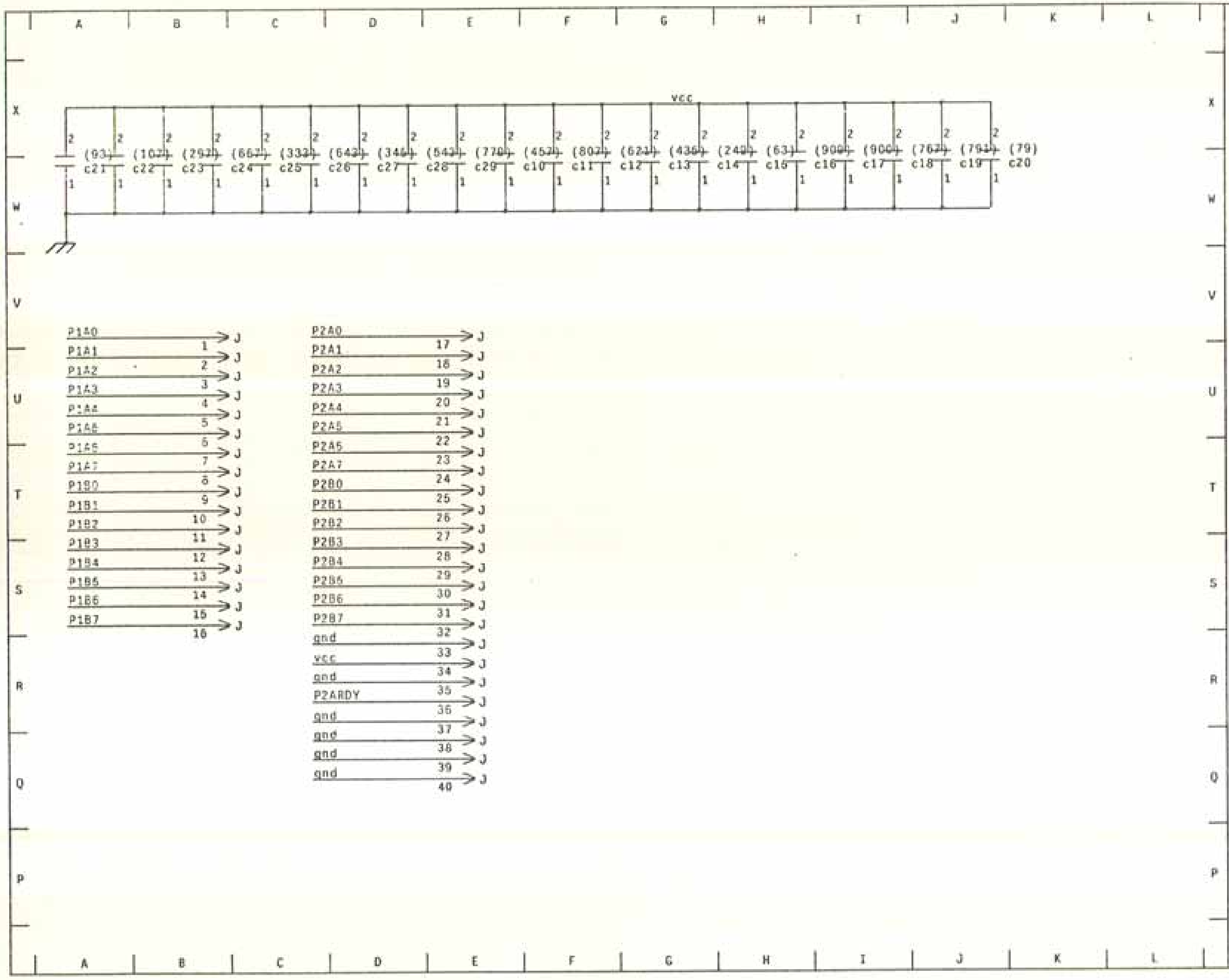
File: project2.g

Date: Thu Jun 19 00:19:20 1980



File: project3.g

Date: Fri Apr 4 12:16:09 1980

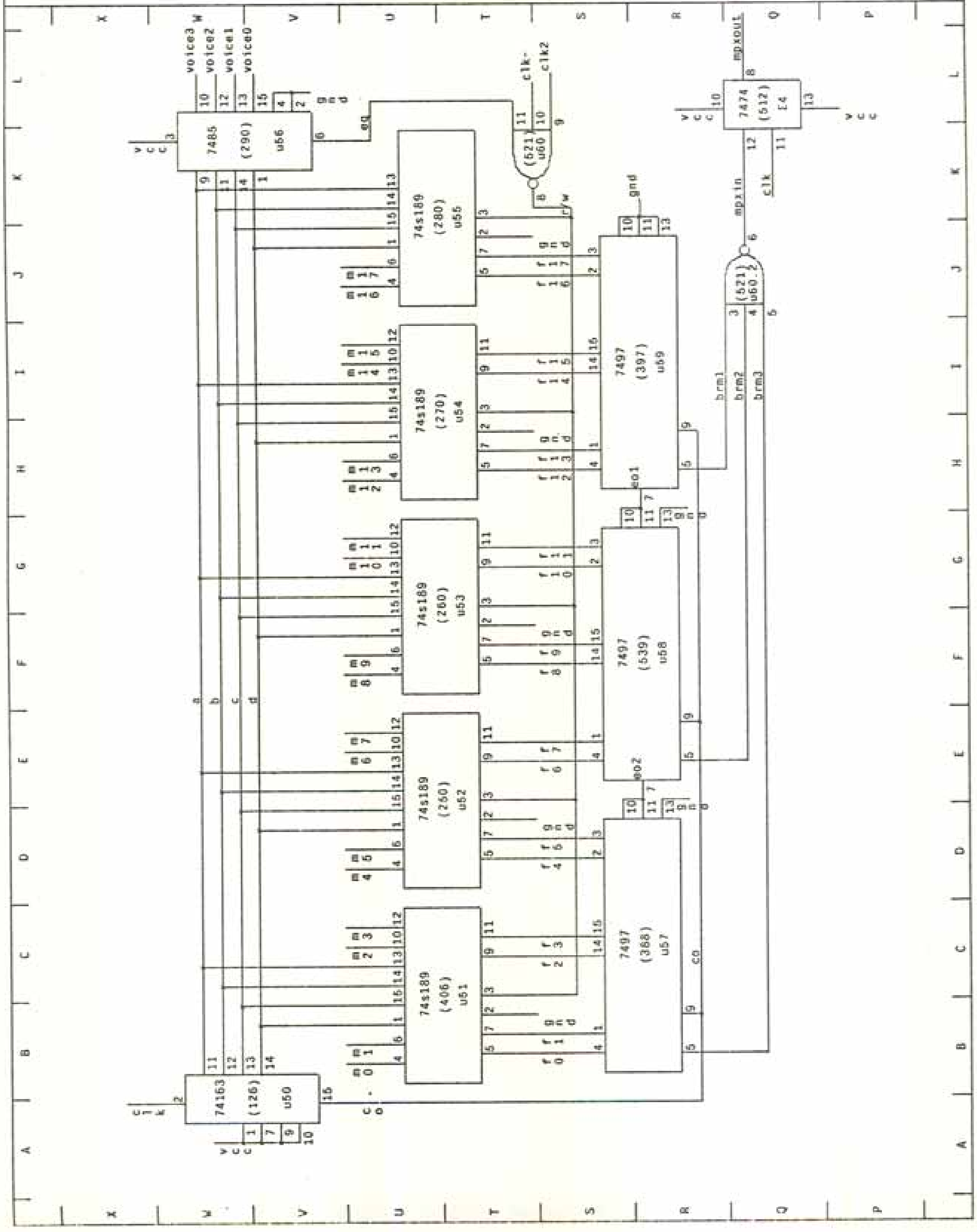


decoupling capacitors

vcc 0.1 uF

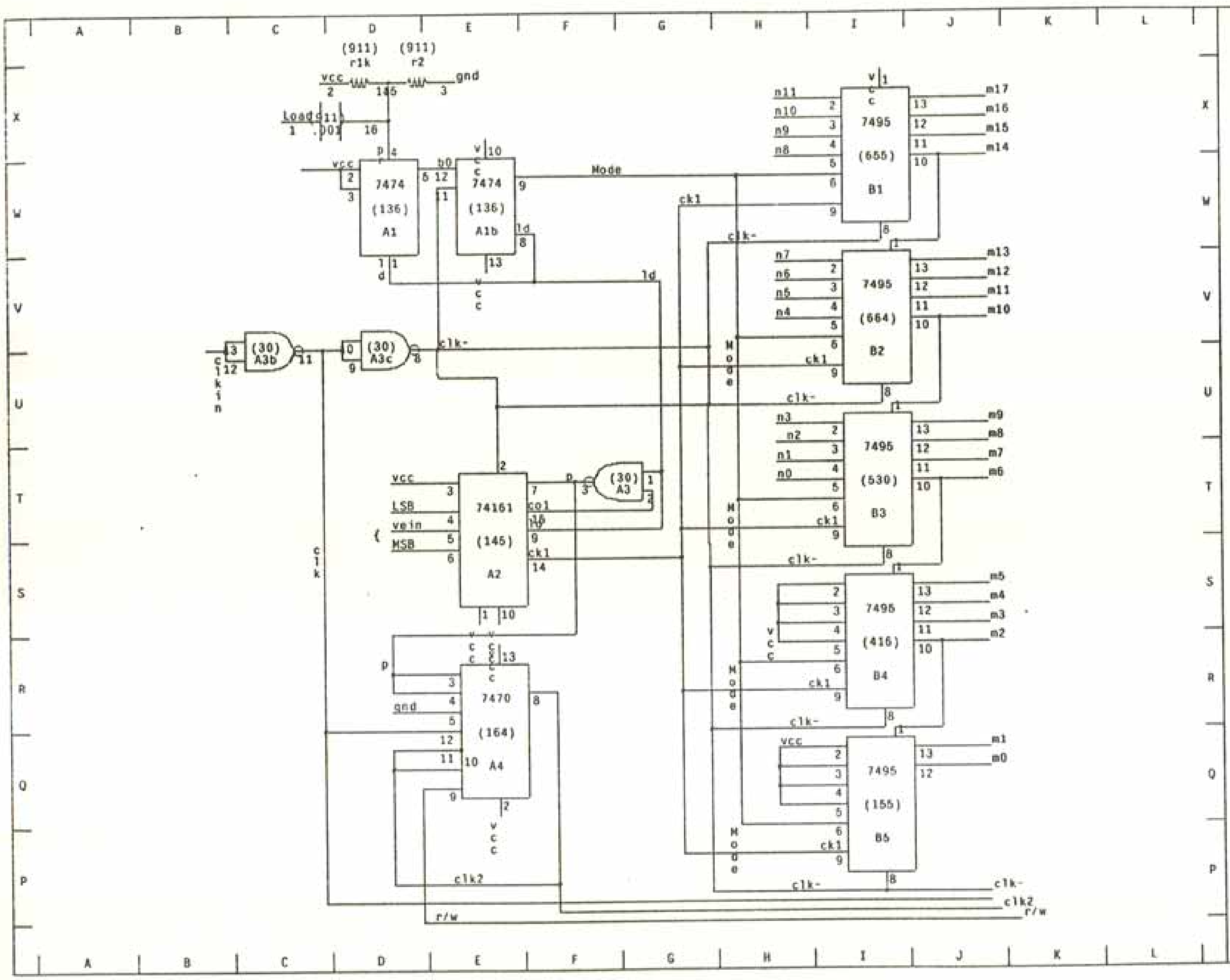
File: projectc.g

Date: Fri May 16 16:29:56 1980



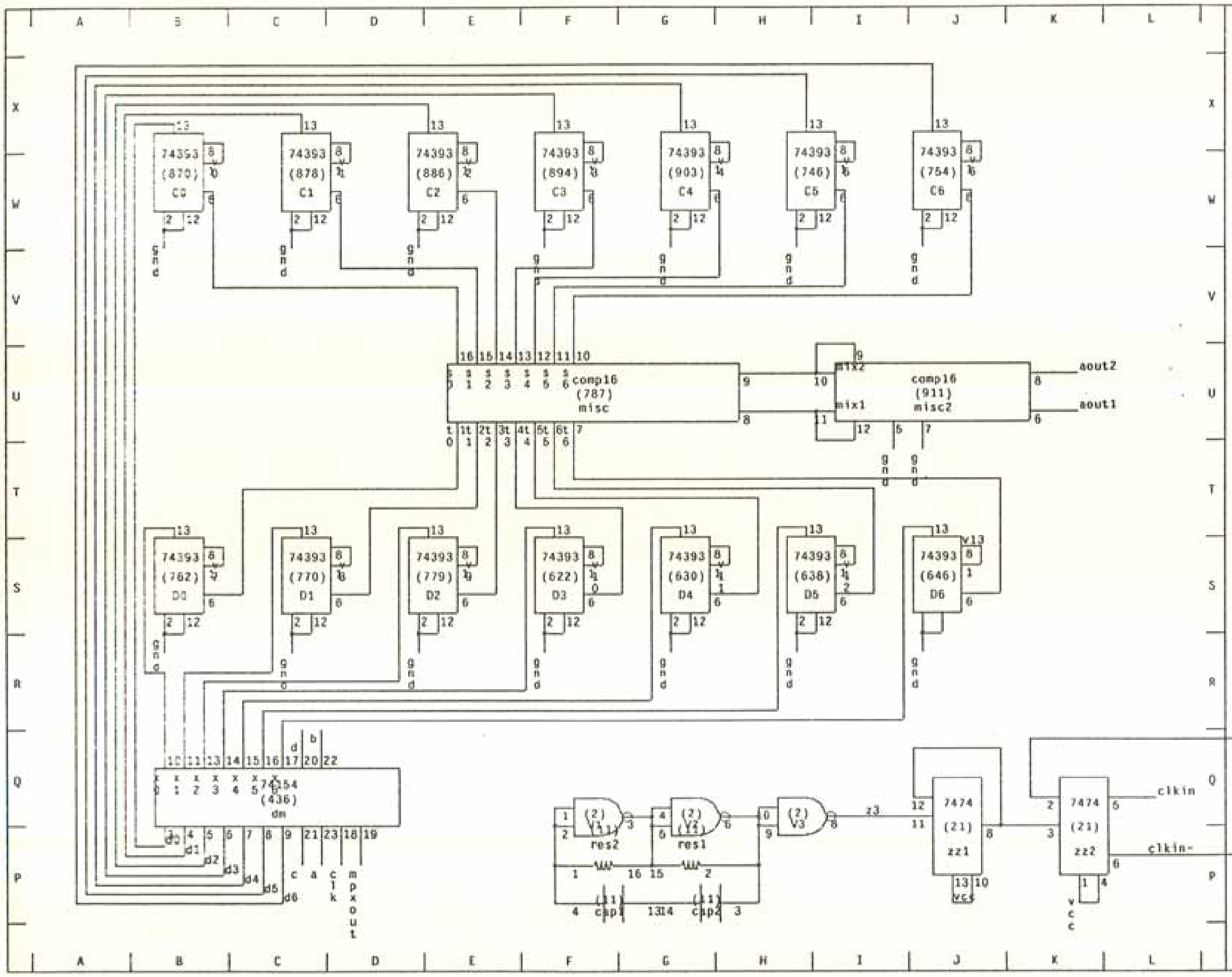
File: projects.g

Date: Mon Jun 30 16:12:48 1980



File: projects2.g

Date: Sun Jun 29 12:43:62 1980



File: projects3.g

Date: Sun Jun 29 12:43:58 1980