AN INTRODUCTION INTO THE USE OF WALSH FUNCTIONS IN A MICROPROCESSOR CONTROLLED MUSICAL WAVEFORM SYNTHESISER

ANDREW AMOS

UNIVERSITY OF NEW SOUTH WALES

SCHOOL OF ELECTRICAL ENGINEERING

1978

CONTENTS:

Abstract:

An electronic music waveform synthesiser is described in which the waveforms are generated digitally, by means of Walsh Functions, under the control of a microprocessor. Basic Walsh Function and Fast Walsh Fourier Transform theory is discussed and a program described for the latter. The suitability of the system is reviewed.

Introduction:

The main purpose of this Thesis is to investigate the use of walsh functions in the synthesis of periodic waveforms specifically applied to the area of musical instruments.

Many different schemes for a synthesiser capable of generating any waveform have been suggested. These include synthesis from a Fourier Series of sine functions, digital approximation, etc. So it was proposed to build a synthesiser using walsh functions so that this could be compared to the more conventional devices.

This involved the design and construction of a monophonic prototype, for control by a Motorola 6800 microprocessor and the writing of the necessary assembler language software.

A program which evaluates the coefficients for the walsh functions used was also developed to assist in the reproduction of the desired waveforms.

Walsh Functions - some theory.

'Walsh Functions' are a set of binary orthogonal functions first investigated by J.C.Walsh (1) in 1923. Their important characteristic is their binary nature which means that they are simple to generate and process using digital equipment.

It is a property of all sets of orthogonal functions that any periodic waveform can be generated by the adding together of the various functions in the set, when the correct weighting for each function is applied.

Many such sets occur, the most common being the system of sine and cosine functions. Block pulses are another example.

The actual coefficients applied to the different functions are, in the sine - cosine system, determined by applying a Fourier Transform to the desired waveform. In the case of walsh functions a Walsh Fourier or Hadamard Transform is applied instead. The only difference between these two transforms being the nature of the functions with which the periodic waveform is being described.

Thus, walsh functions are a very convenient set to use. The necessary theory is a more generalised form of normal sine - cosine system theory. Unlike sine - cosine systems, they can be generated and manipulated using the cheaper and more readily available digital computers. The first eight walsh functions are shown in Figure 1.1.

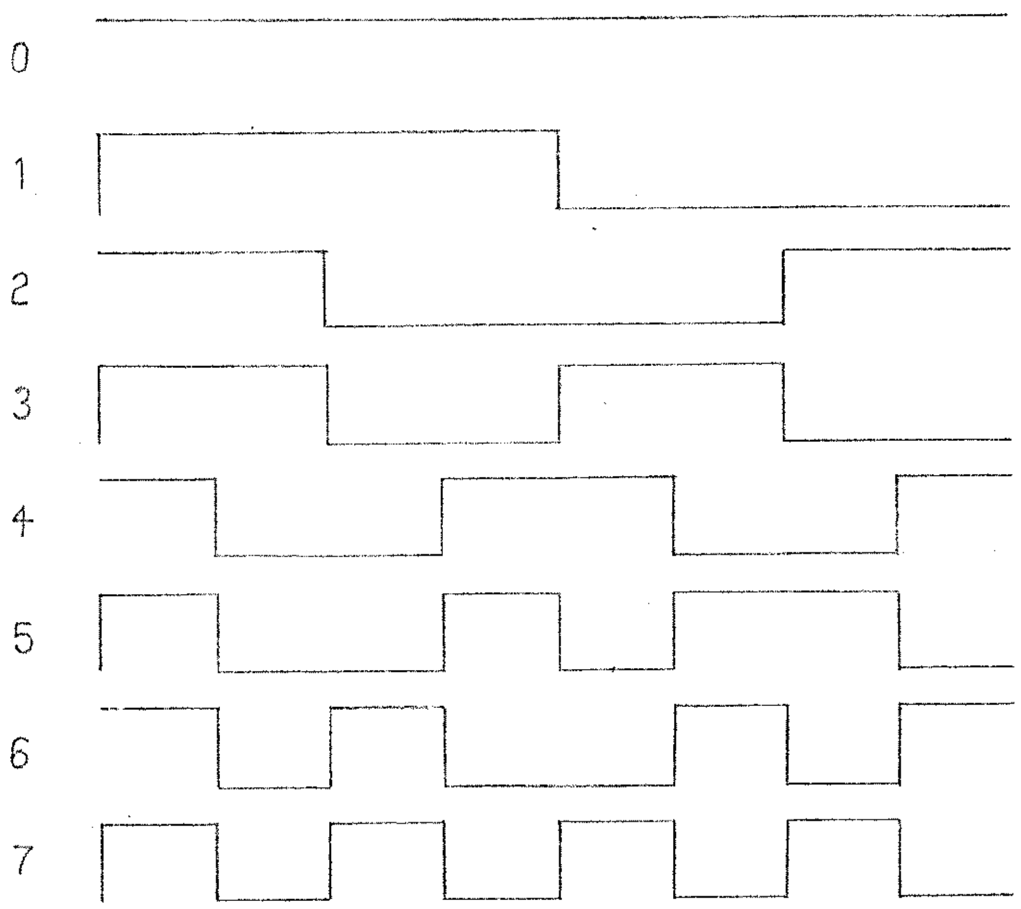


Fig 1.1 First 8 Walsh Functions.

Harmuth (2) uses a system for naming walsh functions which utilises their analogous nature to sine and cosine functions. In his system, the odd numbered functions are termed sal(1,t/T), sal(2,t/T), sal(3,t/T), ... and the even numbered are termed cal(l,t/T), cal(2,t/T), ... This is a highly logical terminology, unlike other systems, and thus is the one which I shall use.

This terminology is logical as the only difference between sal(1,t/T) and cal(1,t/T) is a phase shift, and the number specifying the function is equal to its normalised 'sequency', thus sal(2,t/T) has a normalised sequency of '1' and a base period of 'T'. It also clearly brings out the strong analogy to the sine and cosine functions.

Unlike sine functions, walsh functions need an extra parameter for their complete definition. A sine wave is specified by its magnitude, frequency and phase, whereas a walsh function is specified by its magnitude, sequency, period 'T', and delay. Walsh functions need the extra parameter, because most of the functions in any given set have the same base period.

Sequency is a generalised concept of frequency and is defined as half the average number of zero crossings per second of the function (½ zps), normalised sequency has the sense of half the number of zero crossings per base period.

The extra parameter could be very useful in communications applications of walsh functions, as an extra form of modulation is possible. The actual magnitudes of the walsh functions are either '+1'or'-1'.

Sa1(1,t), (2,t), (4,t), (B,t) ... are a subset of walsh functions called Rademacher functions which in fact, are just square waves of frequency t, 2t, 4t, ... It can be shown (3) that the complete set of walsh functions can be generated by the multiplying together, modulo 2, of the correct Rademacher functions. Modulo 2 arithmetic in logical terms is actually an exclusive oring of the functions.

The correct Rademacher functions to multi ply together can easily be found by reference to a reflected binary (gray) code where a '1' denotes 'multiply' and a '0' denotes 'leave out'. Fig. 1.2 shows how the first eight walsh functions may be generated in this way.

	R_3	R_2	R_1	
wal(0)	0	0	0	
wal(1), sal(1)	0	0	1	(R_1)
wal(2), cal(1)	0	1	1	
wal(3), sal(2)	0	1	0	(R_2)
wal(4), cal(2)	1	1	0	
wal(5), sal(3)	1	1	1	
wal(6), cal(3)	1	0	1	
wal(7), sal(4)	1	0	0	(R_3)

Fig 1.2 Function Generation.

Thus, to actually generate walsh functions digitally requires only the exclusive ORing of the correct Rademacher (square) waves.

With n square waves each an octave apart, it is possible to generate 2 to the power of n walsh functions. These are: wal(O) which is a constant function, 2n-1 sal terms and 2 n-2 cal terms. Fig. 1.3 shows the first sixteen sal functions, the cal terms being identical except for a phase shift.

Due to their ease of generation and their digital nature, walsh functions are easily used in computer controlled devices. The actual magnitudes represented by a binary representation of a walsh function are '+1' for the '1' state and '-1' for the '0' state. Thus, any situation where multiplication of a walsh function by a constant is involved (such as a Walsh Fourier coefficient), a '1' implies the passing of the constant unchanged, and a '0' the negating of the constant. This is very convenient where walsh functions are manipulated serially.

Many papers have been written on the subject of walsh functions, the best source of information being the Proceedings of the Symposiums on Walsh Functions held yearly in the USA since 1970.

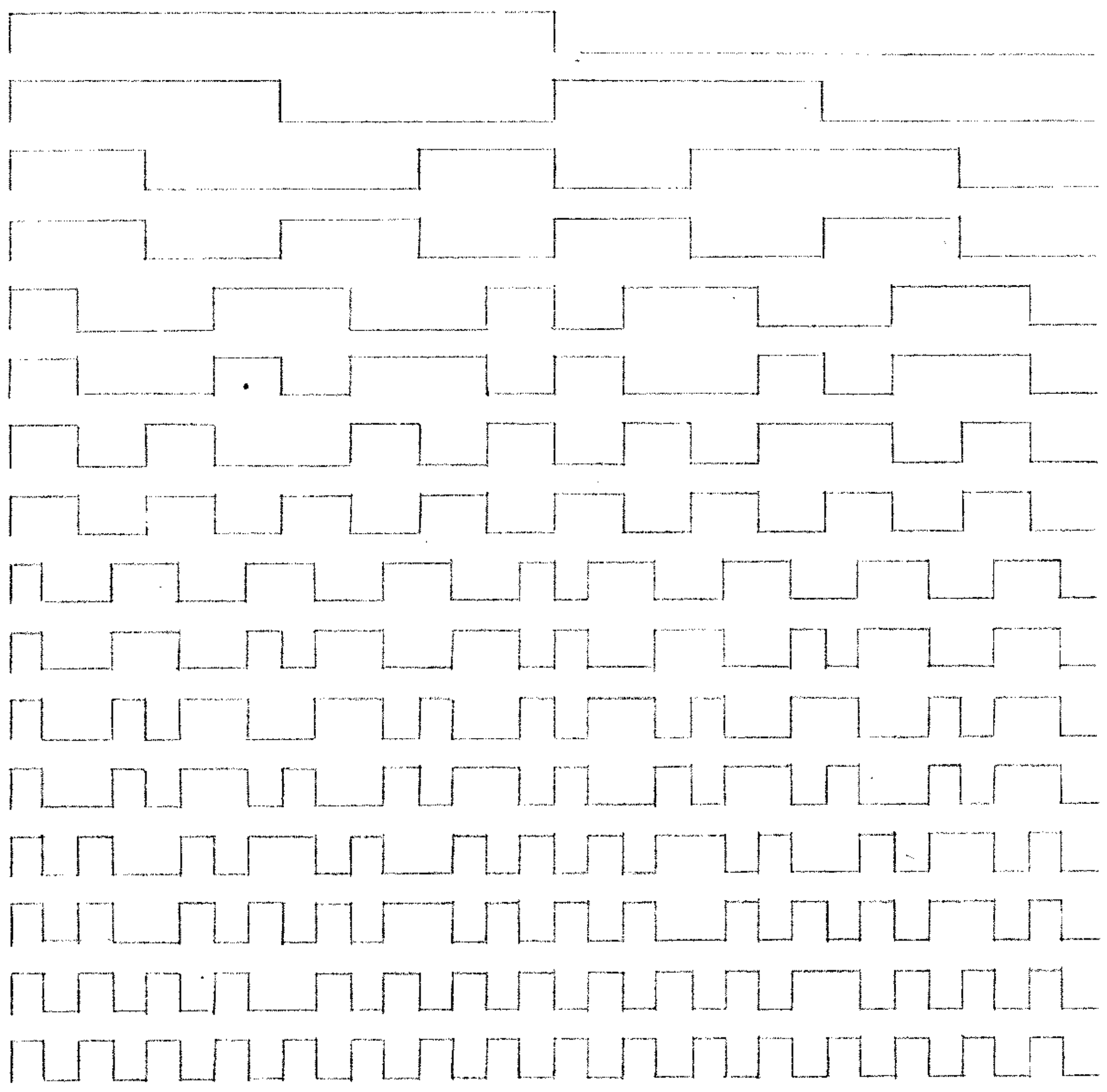


Fig 1.3 First 16 Sal Functions.

Circuit Description

General

The overall design object for the investigation was to produce a multiphonic synthesiser capable of having ten keys sounding simultaneously. It was to have a minimum frequency range of approximately 20 Hz to 2 kHz. This compares favourably to an A to A piano range of 27.5 Hz to 3520 Hz and a C to C organ keyboard of 65.4 Hz to 2093 Hz. The keyboard used had a switch matrix based on the C to B octave and this octave definition was used in the design.

It was decided to use 32 sal components in approximating the desired waveform as it was thought that this would probably result in sufficient accuracy.

With this system of 32 components, the largest frequency involved is that of sal (32), that is, 32 times the frequency of the desired waveform. For a top desired waveform frequency of 2 kHz this implies a top frequency of 64 kHz. For a completely general system, this gives a minimum period of 15.6µS to calculate the present state of each walsh function, to multiply it by its magnitude and sum all the results.

For ten notes playing simultaneously, 320 walsh functions would be involved. This is obviously far too many calculations for a microprocessor with a top clock period of 1µS.   It would even be impossible for a monophonic version to be implemented in this way.

For this reason separate hardware boards were designed for each note. These do the division necessary for obtaining the correct note frequency, the generation of the 32 walsh functions, the multiplication of these functions by their correct magnitudes and the summing of the 32 results.

The microprocessor is used to scan the keyboard matrix and thus decide which keys are down. It then either allocates notes to output channels, or clears output channels when a key is lifted. Different allocation algorithms can be implemented with software changes, but only one system was used in the prototype.

The system described has a possible range of eight octaves going from 16 Hz to 4186 Hz.

The keyboard can be situated anywhere in this range and only requires changes of software constants for a shift in apparent location.

The number of 'semitones' in an octave is not limited to the usual number of 13, but may be up to 16 in number.

With software changes more than one keyboard could be accommodated, as long as their total length remains within the eight octave limit. These keyboards could also be given different waveshapes. In fact, with sufficient programming, every note in the range could have a different sound.

Each of the 32 walsh functions is given an eight bit magnitude. This is encoded using 2's complement notation giving a possible range of ±127 (decimal). This notation is used as it greatly simplifies arithmetic operations.

For example, subtraction of a number only requires the addition of the inversion of that number, with a carry in.

When 32 eight bit magnitudes are summed, the maximum number possible is ±4095 (decimal), which requires a space of 13 bits. Due to the insignificance of the last bit, and the ready availability of a 12 bit digital to analog convertor, the least significant bit is dropped after this summation. This gives a range of ±2047 (decimal).

The actual frequency of the note is obtained by dividing down from a frequency of 16 MHz, this gives a maximum walsh function frequency of around 8 kHz.

A hardware timer was also incorporated to give switch debouncing using the system of quick scans with long delays. This system allows the microprocessor to be utilised to its maximum between each scan of the keyboard.

Two types of board were designed, the 'control' board, (one required) and a 'phonic' board (ten required). In the monophonic version actually constructed, only one of each type was built.

This system of separate boards allows the synthesiser to be gradually expanded to its full capability. Only a constant in the software needs to be changed for expansion up to the maximum of ten simultaneous notes.

The control board does all the address decoding and generation of control signals, as well as containing the timer and keyboard searching hardware.

Each phonic board generates the walsh functions and the note frequency. It also multiplies and sums the magnitudes of the walsh functions. This is all done using data supplied by the microprocessor.

An explanation of the microprocessor timing mechanism is required here. The Motorola 6800 uses a two phase, non overlapping clock signal. A number of other signals are also made available, Fig.2.1. The microprocessor only needs control of its address and data buses when DBE (data bus enable) is high. This is because this is the only time it makes valid read or write requests to its memory. The R/W (read/write) signal indicates whether data is to go to or from the microprocessor respectively. The VMA (valid memory access) signals that valid data transfer is to occur when DBE goes high. In the system used, the DBE input was tied to the phase 2 clock signal so that the control board hardware needs only the logical ANDing of the phase 2 signal and the DMA, as an enable for its address decoding circuitry.

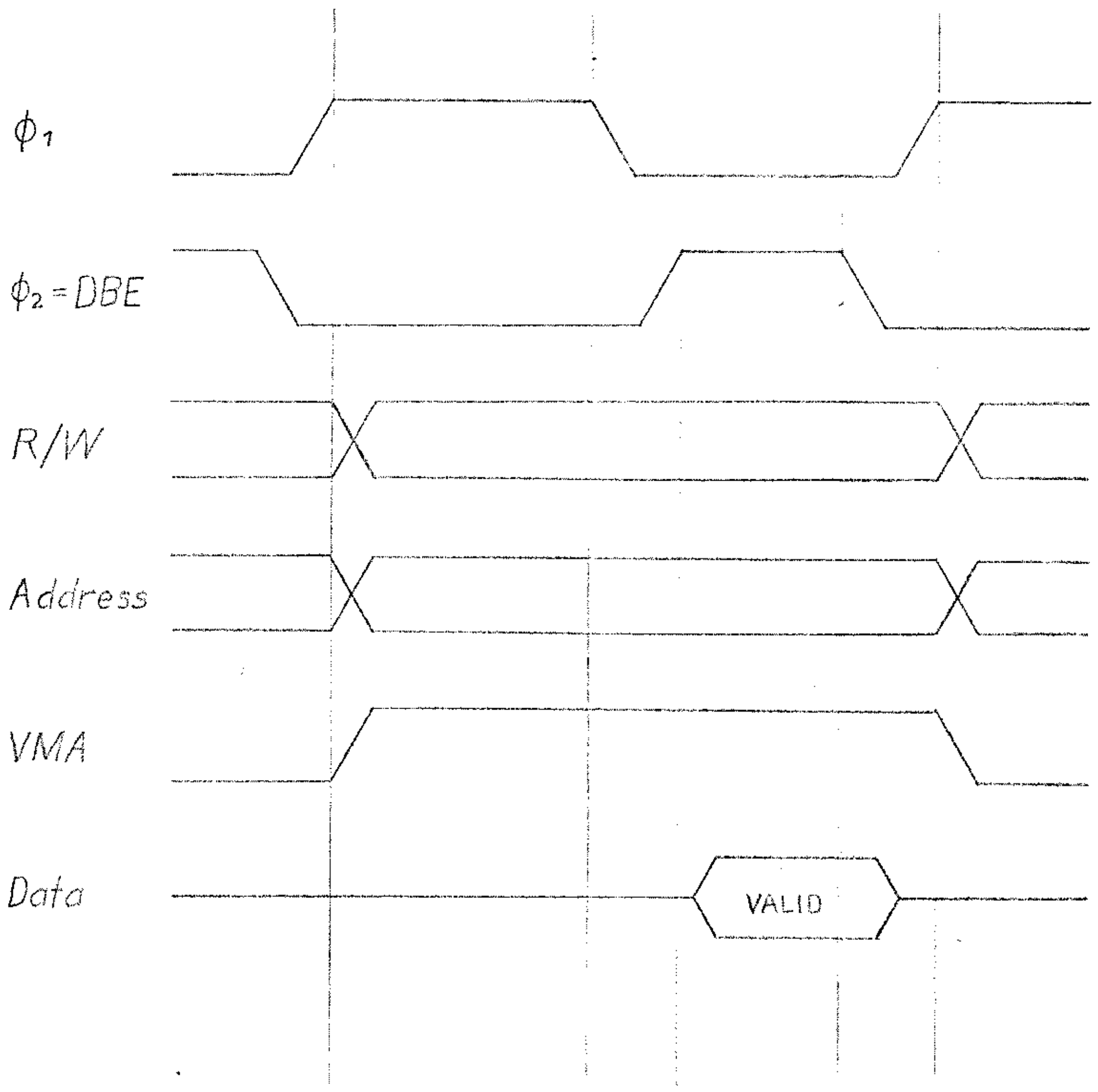


Fig 2.1 6800 Timing Signals

This dependence on the phase 2 clock (DBE) is uti1ised so that the function magnitude RAMs on each phonic board are only in the control of the microprocessor when phase 2 is high, and are controlled by the hardware when it is low.

The overall block diagram appears in Fig.2.2. There are five inter-connection buses used. The address bus of the microprocessor is used to address either the keyboard, or the frequency latches on each board, or the function magnitude RAM on each board.

The data bus is used either to send information or to receive it. The signals that may be received are the addressed key is up/down and the timer is set/clear.

The output bus makes the summed values from each board available to the control board for combination and digital to analog conversion.

This particular part of the circuit has not been developed to its full extent in the prototype and the output bus is connected straight into the input of the digital to analog converter. The actual circuitry required here depends on the requirements of the user, as envelope control may or may not be implemented.

The function select bus is either under the control of a counter or the lower five bits of the address bus, depending on the state of the phase 2 clock signal. It is used both for the addressing of the function magnitude RAM and for the selection of the appropriate function from the walsh function generator.

The control bus consists of:

1. An inverted phase 2 clock;
2. A 'count complete' signal, which means that all 32 functions have been summed and thus resets the latches of the phonic boards;
3. enables for the two 8 bit frequency divisor latches on each board;
4. chip select lines for the function magnitude RAMs, which are actually the read/write lines, as the RAMs are permanently selected.

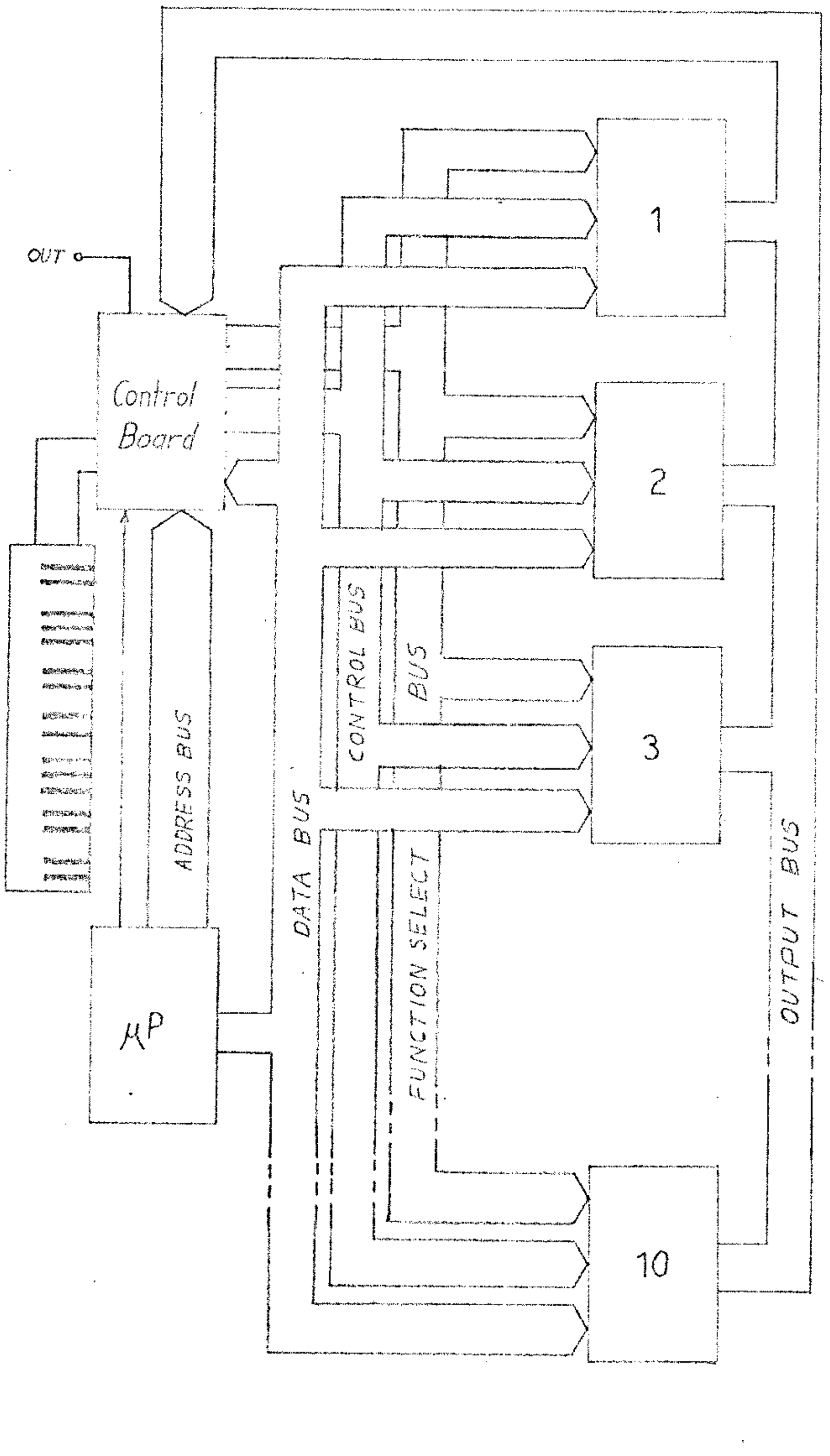


Fig 2.2 Overall Block Diagram.

Control Board

General

Fig. 2.3 shows a block diagram of the circuitry on the control board. Buffers are used on all lines having an input loading greater than one standard TTL load.

Address lines 12 - 15 are used as overall select lines, the board only being selected when a 9 appears on these lines, so as to distinguish the synthesiser from the microprocessor RAM and ROM. The decoder gives 20 select lines for the frequency division factor latches (two on each phonic board), 10 function magnitude RAN select lines, a keyboard select line and a timer select line. As far as the microprocessor is concerned, it is just addressing memory locations for any key, latch or RAM access.

Address lines 0 - 6 are used for key access, lines 0 - 2 being octave select and 3 - 6 being semitone select lines.

The octave select lines are put through a 1 of 10 decoder, which selects one of the eight possible octaves. The semitone select lines are fed into a 1 of 16 encoder. If in the selected octave the selected semitone is being played a logical ‘1’ is displayed on microprocessor data line 7. This means that when a key is down the microprocessor sees a negative number on the data line and a positive number when it is up. This makes the test software very simple. The output of the timer is also readable at the same time on data line 0.

Address lines 0 - 4 are also used for accessing each location in the 32 x 8 bit function magnitude RAMs. Each RAM being selected by the more significant address lines. As mentioned, on the Motorola 6800 microprocessor used, data is valid only when the phase 2 clock

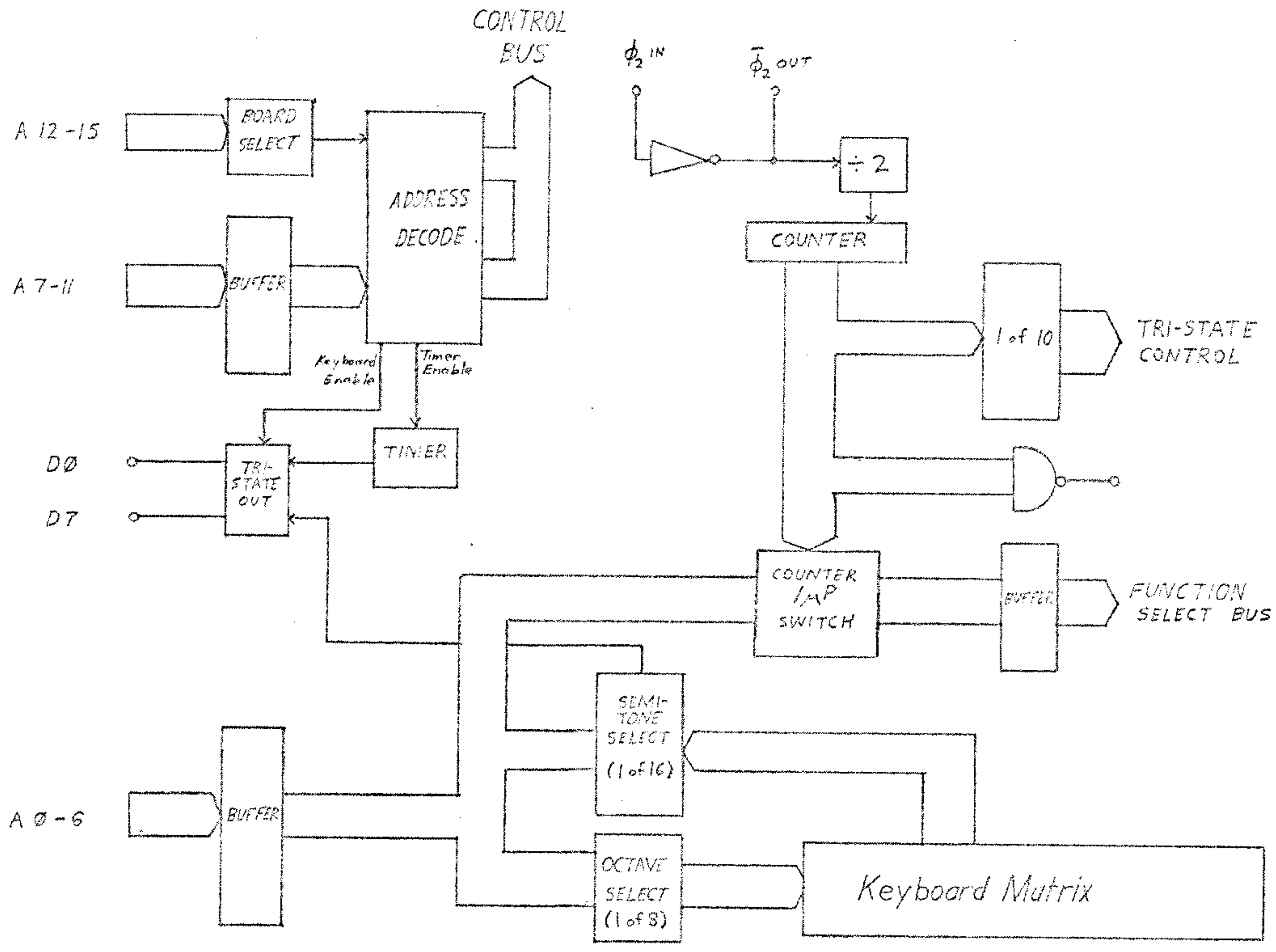


Fig 2.3 Control Board.

is high. This fact was utilised to make the microprocessor's loading of data into the function magnitude RAMs transparent to the hardware. Thus, when phase 2 is high, the RAM address lines are connected to microprocessor address lines 0 - 4 and the selected RAMs R/W is put in the read position. Then on phase 2 falling low the hardware selects the function and all the RAMs are in the write mode.

A counter cycles through each function on the boards synchronously allowing their respective magnitudes to be accumulated. When the count value reaches 32, a 'count complete' signal is generated which transfers each result to a storage latch and then resets the accumulator to zero. During the next count to 32, the 4 most significant bits of the counter output, through a 1 of 10 decoder, are used to make the value in each storage latch available on the output bus for processing on the control board.

Address Decoder (Fig. 2.4)

The input lines are buffered so as to only provide a unit loading (U.L.) on the input buses. 7408 quad AND gates were used for this purpose.

32 output lines were obtained by using two 74154 4 line to 16 line decoder/demultiplexors, with an inverter on microprocessor address line 11 (All). This means that chip 1 is selected when line A11 is high and chip 2 when it is low.

The other enable pins are controlled by a circuit which gives a low output only if VMA, inverted phase 2, A12, A15 are high and A13, A14 are low. This ensures that no part of the synthesiser is connected to the microprocessor when the address and data lines are not valid, or when the most significant bits (A12 - A15) do not show a 9 (1001 binary).

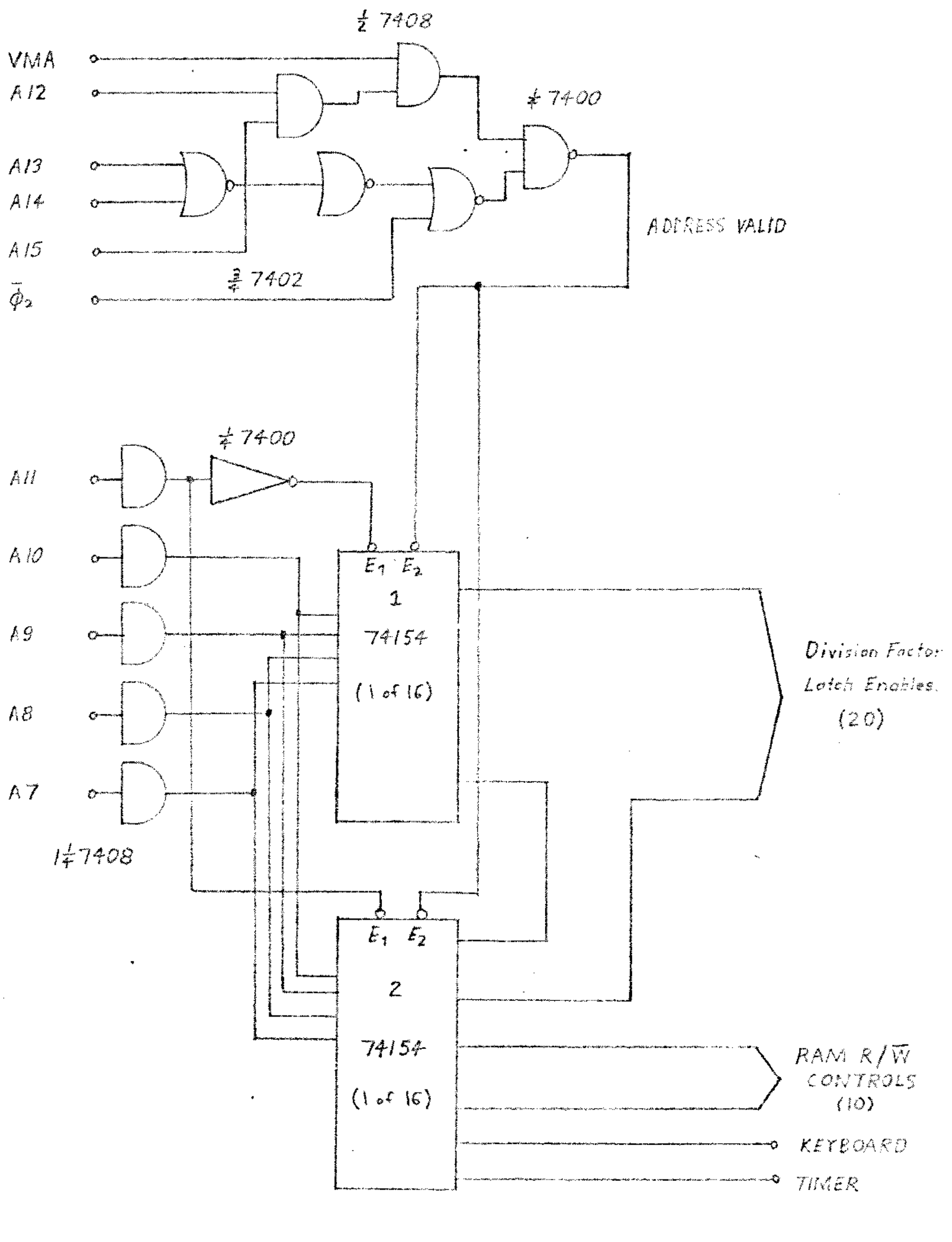


Fig 2.4 Address Decoder.

Component Control Counter (Fig. 2.5)

The counter is made from 7493 4 bit binary counters. The first divide by 2 stage means that the counter changes state every time the phase 2 clock goes high. For the count complete signal a 7430 8-in NAND gate is used with the unused inputs disconnected.

A 7442 BCD to decimal decoder acts as a 1 of 10 selector for the output bus tristate controls. In the prototype built, this line was actually disconnected and shorted to earth (permanently enabled) on the phonic board. This was so that only a simple digital to analog converter was necessary on the end of the output bus.

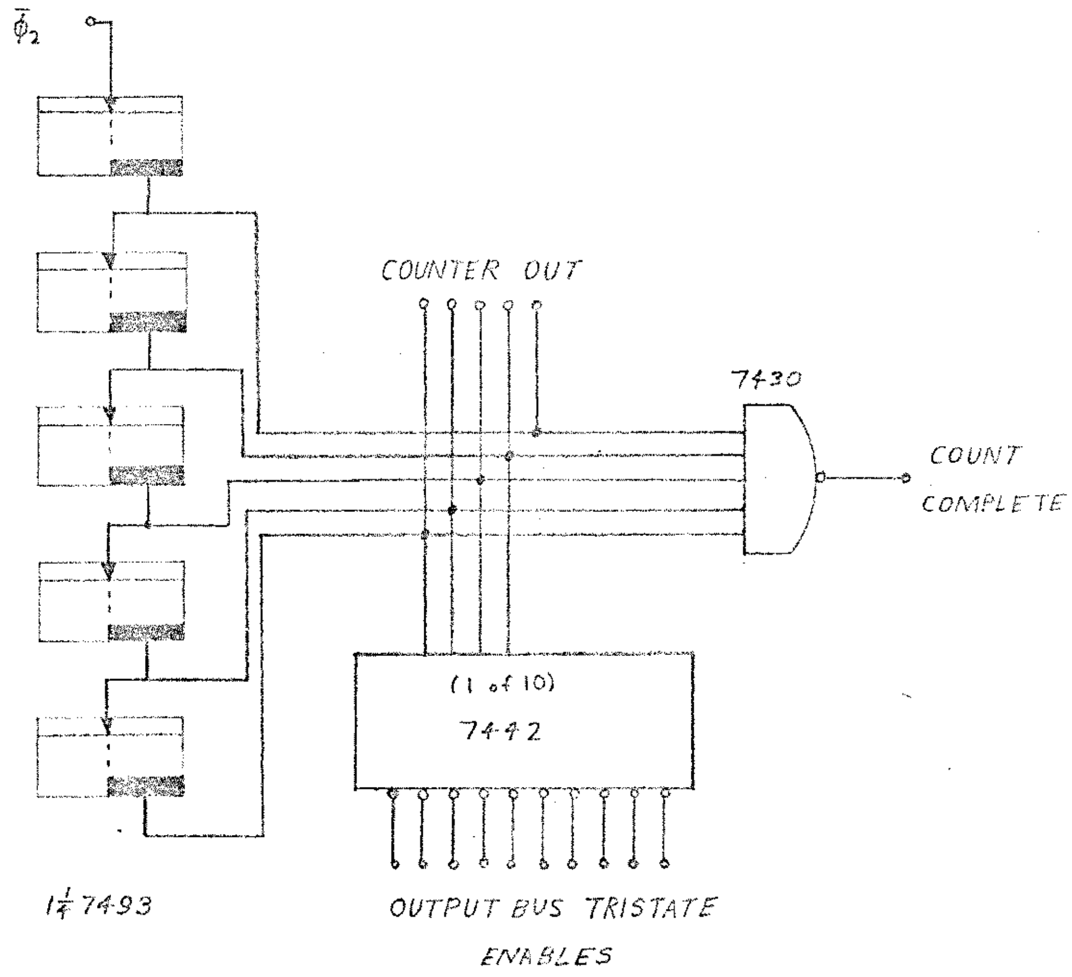


Fig. 2.5 Function Counter

Counter/Microprocessor Switch (Fig.2.6)

This selects whether the address lines of the function magnitude RAMs on the phonic boards are in the control of the hardware counter or the microprocessor. This is done on the phase 2 signal. The switch is made from a 74157 quad 2 to 1 data selector and a single 'discrete gate' selector made from already available gates.

To ensure that the address lines can handle the expected maximum load, 7437 quad 2-in NAND buffers where used. These have an output capability of 30 U.L's. Inverters are used first so as to correct the NAND's inversion. One of these NAND buffers was also used to buffer the inverted phase 2 clock signal.

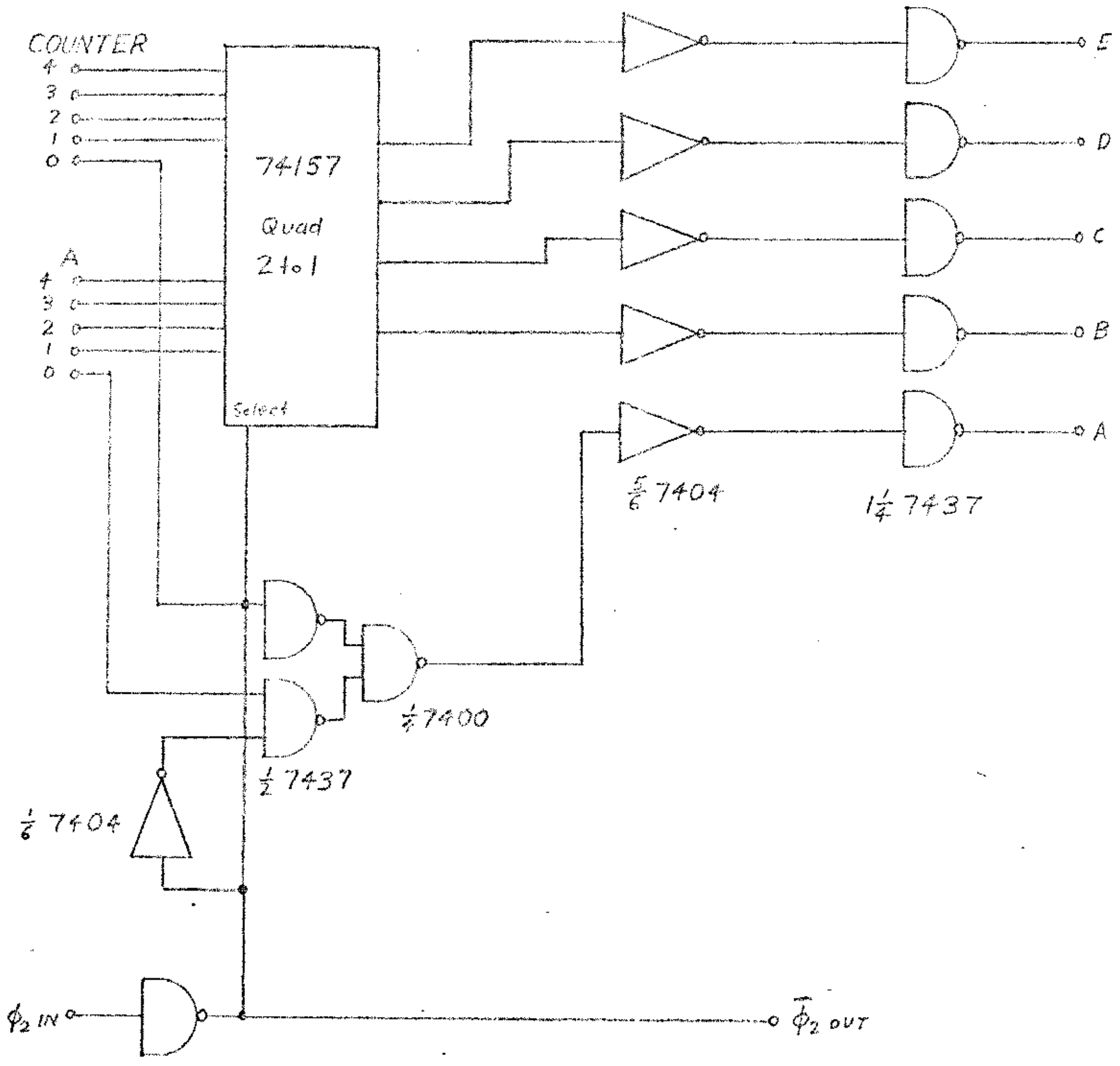


Fig 2.6 Counter/μP Switch.

Keyboard Matrix Control & Timer (Fig. 2.7)

A full keyboard would consist of a 16 x 8 matrix of SPST switches. The 8 octave lines are selected through a 7442 BCD to decimal decoder with its most significant address bit earthed. The output lines go through the matrix into a 74150 16 line to 1 line multiplexor. Thus, when a key is selected, the input multiplexor input line goes low. The output of the multiplexor inverts this and so an inverter is used on its output. The outputs of both the keyboard and the timer are controlled by the keyboard select line. This line is used to enable two of the tristate outputs of the 8097 hex tristate buffer. This particular chip is used as two of the buffers work on one enable line and the other four on a separate enable. This meant that the others could be permanently enabled and used as normal buffers. The other buffers used are a 7408 quad AND gate.

The timer is an NE 555 monostable circuit with its input connected straight to the enable line from the address decoder.

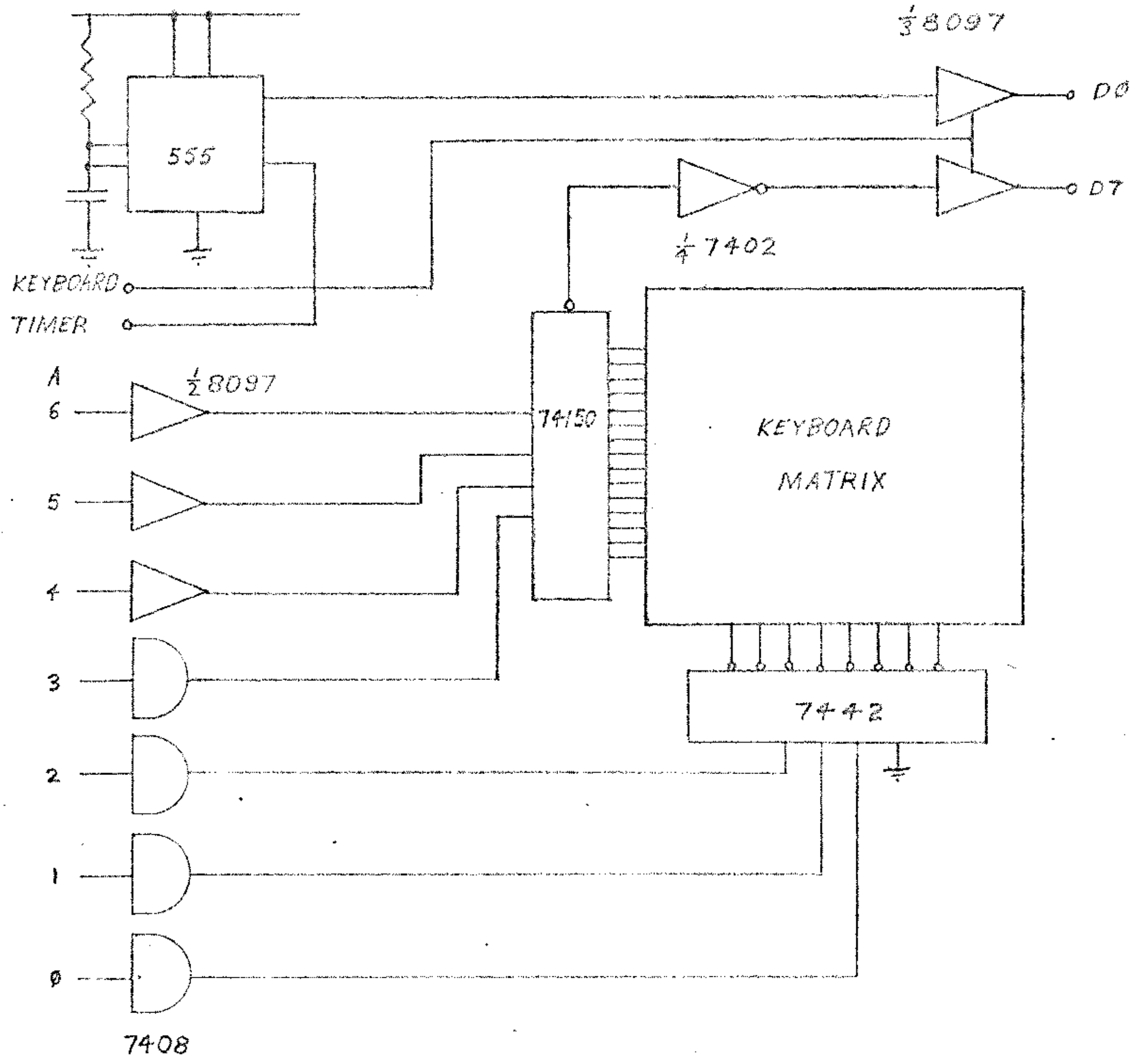


Fig 2.7 Keyboard Matrix & Timer.

Digital To Analog Converter (Fig.2.8)

An Analog Devices AD7531 12 bit digital to analog converter I.C. was used as it was readily available. The circuit is straight from a manufacturers applications note and worked very well. The circuit accepts 12 bits in 'offset binary' notation and converts this to a bipolar analog signal. Offset binary is 2's complement notation with the sign bit inverted. The output is inverted and so FFF (hex) gives an output of -Vref and 000 gives +Vref. Zero is denoted by 800 (hex). This is equivalent to adding half the absolute range to the number and using the normal binary notation for this sum.

The output of the digital to analog converter is a current, so the LM301 operational amplifiers are used in virtual earth mode. I.C.1's feedback resistor of 10k is in the converter chip. When an input is high, its current is steered through output 1 into I.C.1 giving a negative output. When an input is low its current is steered through output 2 which feeds the negative of the current into I.C.1, forcing the output positive. This gives a push-pull bipolar effect. This circuit does halve the resolution to 11 bits, but doubles the range of output levels.

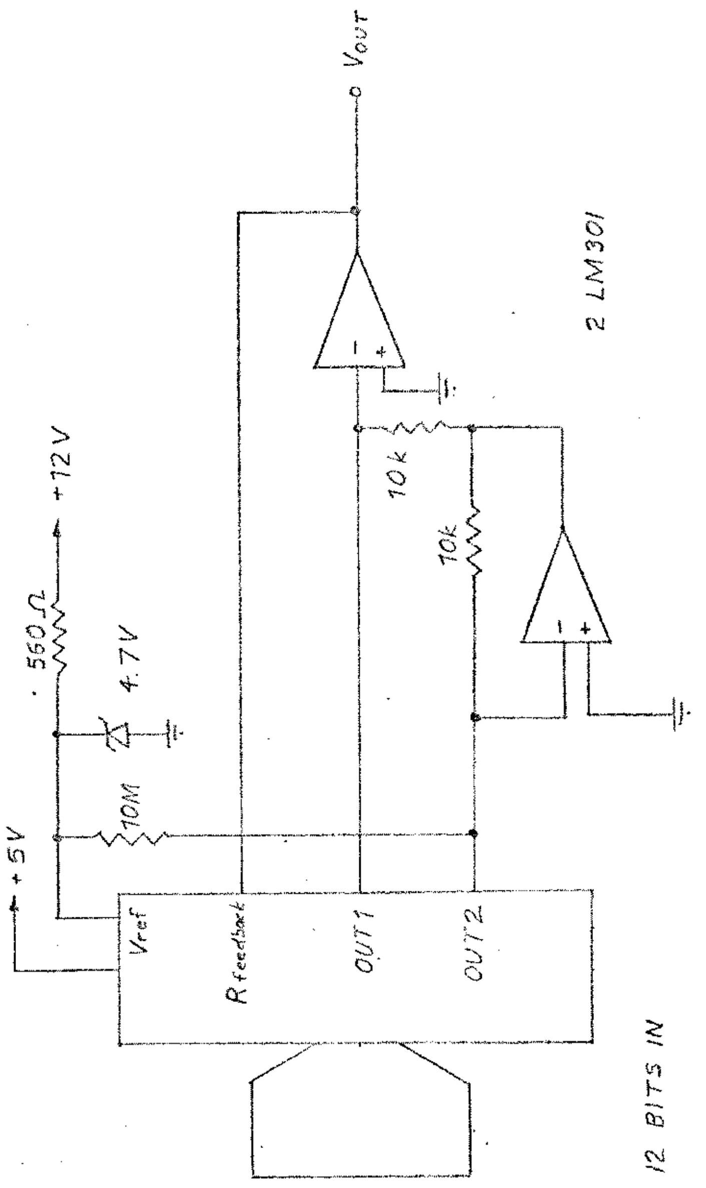


Fig 2.8 Digital-Analog Converter.

Phonic Board

The phonic board (Fig. 2.9), generates the 32 walsh functions with the correct base frequency. These then control an adder which accumulates the correct magnitudes for each sample period. When the sum of all 32 magnitudes is stored, it is made available for sending on the output bus and the accumulator is cleared.

The highest frequency square wave able to be included without error is 1 MHz / (32 x 2) = 15.6 kHz. This is because at least 2 samples must be taken for each cycle of the desired waveform and 32 functions are to be included in each sample period. Hence the sampling frequency must be 1ess than 1/64 of the system clock rate. The first full octave (C to C), which has no component greater than this frequency is 4186 to 8372 Hz. Note that if this octave where to be played, only the first function, sal (1), could be used of the 32 used, as all the others would be above the maximum permissible frequency.

This is not as bad as it seems, however, as the next component up, sal (2), has a fundamental frequency range of 8372 to 16744 Hz which is very close to the limit of hearing. Due to this fact it was felt that the 1 MHz sample rate might be sufficient.

The frequency generator originally used was a pulse rate multiplier. This required a clock rate of only 250 kHz to get sufficient accuracy and uses only a 16 bit adder and a shift register.

A division factor is loaded into a latch and the adder adds this to the (initially zero) contents of the shift register on each clock pulse. When the adder overflows an output pulse is generated, the remainder being 1eft in the shift register. A new cycle then occurs. Thus, for example, if the latch contains a value equal to half the maximum value storable in the adder, a divide by two counter would result.

By leaving the remainder in the shift register it is possible to divide by non-integral values and this is why only a low frequency input clock is required. However, the mark - space ratio continually varies and this was found to be totally unsatisfactory for the generation of stable walsh functions.

Using the alternative of normal integral division techniques, an input clock rate of around 16 MHz is necessary if the top playable octave is to be generated within the required accuracy. This figure was used and gives an accuracy of ± .1% which is within the maximum ± .3% change in frequency.

The actual semi tone divisors range in value from 451 to 239 (decimal) so 12 bits are sufficient for their description.

Problems such as a sample being taken during a change in output state of a walsh function can occur if the sampling clock is not locked to some integral multiple of the walsh function generator clock. For this reason the 16 MHz clock had to be locked to the microprocessor system clock of 1 MHz.

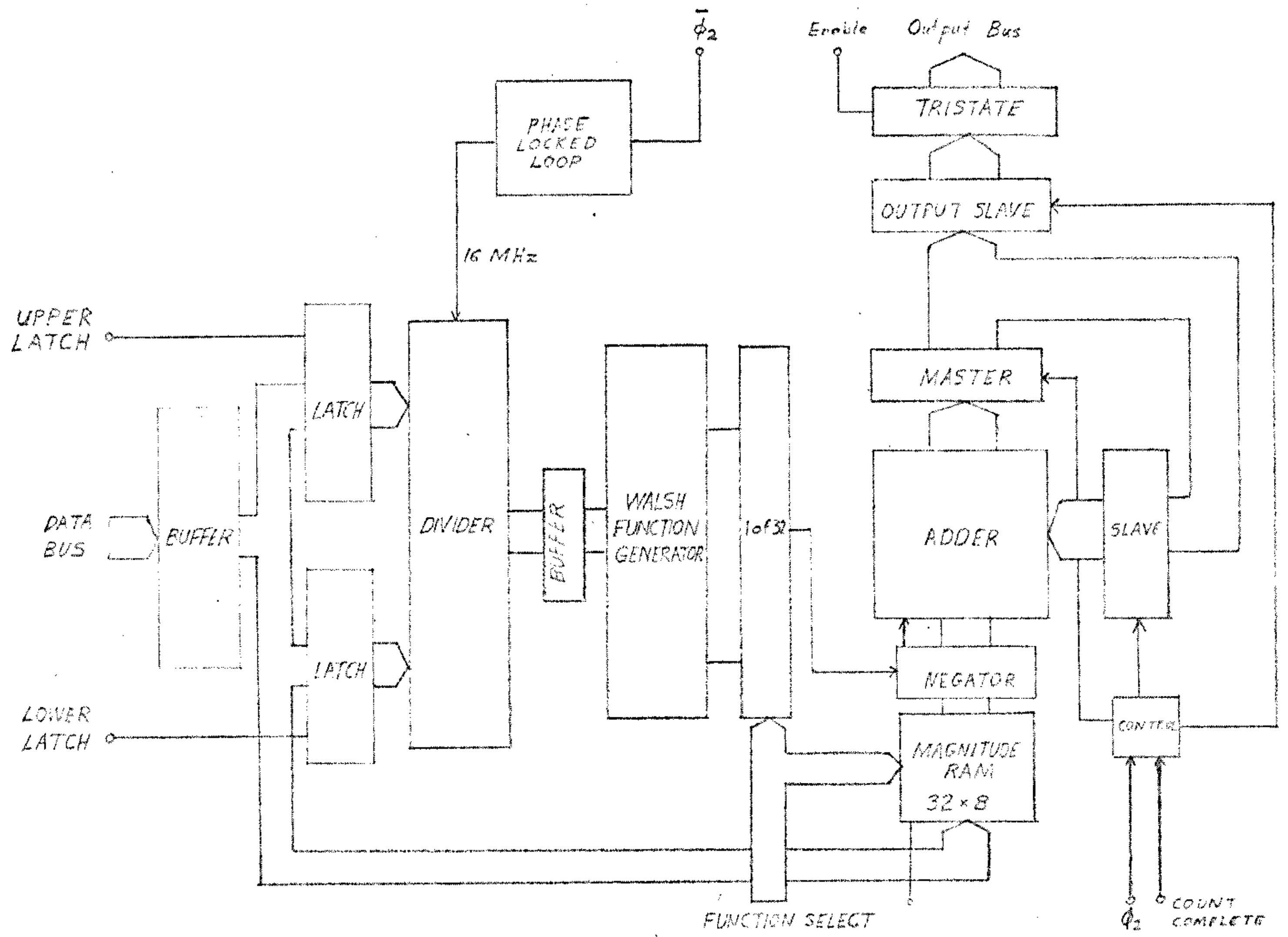


Fig 2.9 Phonic Board

An NE562 phase locked loop was used (Fig.2.10) as a frequency multiplier for this purpose. The figure of 16 MHz is just outside the guaranteed lowest maximum operating frequency of 15 MHz, so it was considered that it would probably be capable of the required service.

In the frequency multiplying mode, the 1 MHz clock is put into one of the inputs to the phase comparator.

The other comparator input is taken from a divider (a 7493 4 bit binary counter). The input of the divider is taken from one of the voltage controlled oscillator outputs. The loop thus locks in when the oscillator frequency is 16 times the input frequency, that is, 16 MHz. A separate voltage controlled oscillator output is used as the 16 MHz output line.

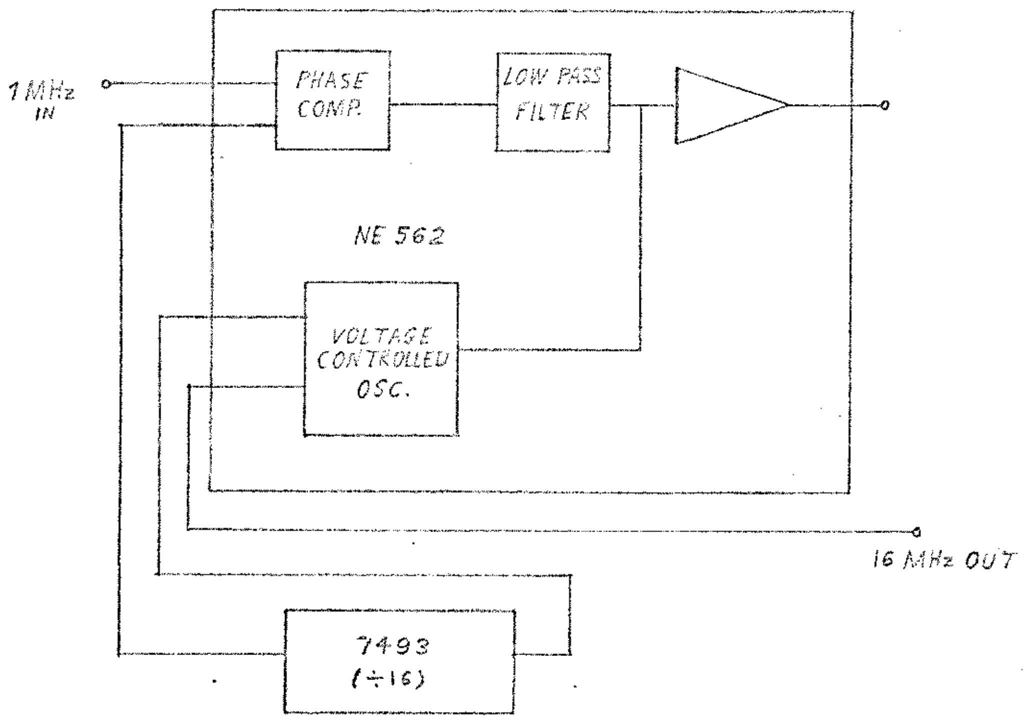


Fig 2.10 Phase Locked Multiplier.

Fig. 2.11 shows a diagram of the actual divider system. Five 74193 up/down presettable binary counters were used, three as down counters and two as up counters.

The required division factor is loaded into latches from the microprocessor data bus. The latches are two 74116 dual 4 bit latches, with the input lines buffered by two 7408 quad AND gates. Enables for the latches come from the address decoder on the control board.

The semi tone and octave factors are split up in the software, so that only the octave factor needs to be changed when using the standard chromatic scale. The semitone division factor of 12 bits is loaded into the first three 74193's from the latches. The 74193's then count down from this value until they reach zero. When this occurs a carry out is generated which also enables the loading in of the latch contents into the counter and this process is continuously repeated.

As an example, if the value '3' is in the latch, the counter, on receiving the first clock pulse, will decrement this stored value to '2', on the second to '1' and then on the third to '0'. At this stage a clock pulse is given out and the counter reloads with '3'. Thus, the circuit has actually divided by 3, which is the value stored in the latch. This is an extremely easy division circuit to use and to write software for.

It was found, when using the chromatic scale, that the most significant of the 12 bits in the semi tone factor is always '0' and so this was hard wired to this value. This means that only 11 bits actually need to be sent.

The remaining 5 bits were split into two groups, the least significant 3 bits dial a true division by a factor of 2. This can be in the range of 1 to 64.

This octave divider unit was implemented by using a 7442 BCD to decimal decoder as a 1 of 8 selector. As the output of this is active low (the selected output goes to '0'), the two 74193's were in this case connected as up counters, the mode of operation being the same as for the semi tone divider, except that the input data is inverted and the counter counts up to zero.

This system gives a possible range of seven octaves. Using the input clock rate of 16 MHz this enables the sal(l) terms to vary in frequency between 8 Hz and 1 kHz. This gives a very good low frequency range but lacks in high frequencies. To get a reasonable top frequency of, say, 2 kHz, an input clock of 32 MHz would be necessary which is impractical with commonly available phased locked loops. Another system was thus necessary for a reasonable range of output frequencies.

As mentioned earlier, no frequency above 15 kHz can be fed into the circuit because of its 1 MHz sample rate, also the magnitudes of walsh functions containing frequencies above this are set to zero in the function magnitude RAM. There is, therefore, no reason to generate these higher frequencies. So, to generate the octave above 1 kHz (1046 to 2093 Hz) one of the divide by two stages is skipped and two stages are skipped for the next octave (2093 to 4186 Hz). A very reasonable range of frequencies results.

This is actually implemented using a 74151 8 channel digital multiplexor as a 4 to 1 data selector with the two most significant bits of the upper divisor byte controlling the selection.

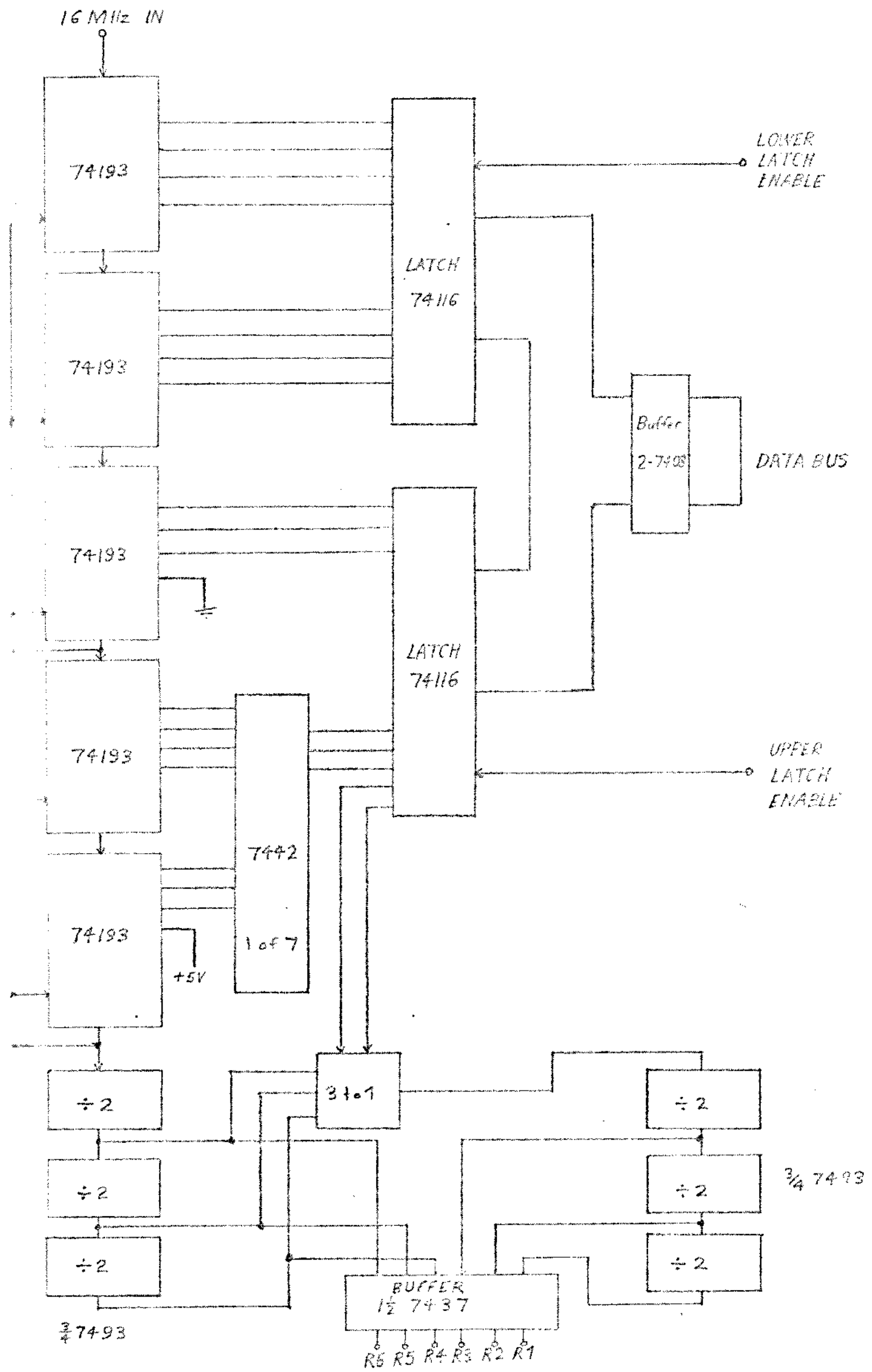
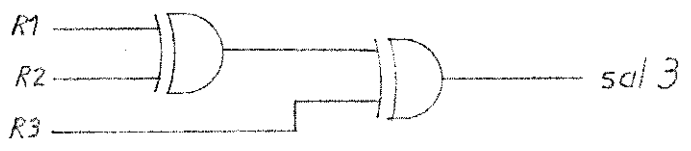


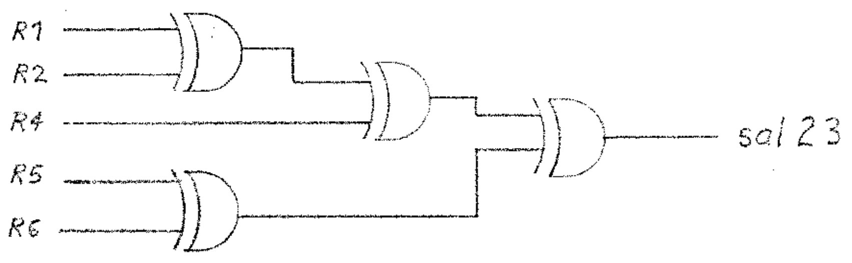
Fig 2.11 Divider System

The resultant square waves are buffered by 7437 quad NAND buffers as the loading is 16 U.L., the logical inversion not being important here. They are then fed into the 'walsh function generator'. The generator is actually a bank of 64 2-in exclusive OR gates, set up as either 3 input or 5 input gates (Fig. 2.12). This gives the 32 sal components required. Sixteen 7486 quad exclusive OR gates were necessary for the generator.



½ 7486

3-in exclusive OR



1 7486

5-in exclusive OR

Fig 2.12

Only one function is required at anyone time and so a 32 to 1 data selector is used, its 5 address lines are connected to the address lines of the function magnitude RAM (Fig. 2.13). The 74150 16 line to 1 line multiplexors were used to make this data selector. The inverted outputs mean that the NAND gate actually works like an OR gate.

An N82S09 64 x 9 RAM was used as the function magnitude RAM. However, much cheaper RAMs have been found since that decision was made. The high speed afforded by Schottky devices is not necessary in this situation as the usual access time of 450 nS is quite sufficient. The alternative tristate output devices would also be preferable to using passive pullup resistors on the N82S09's open collector outputs.

For this RAM the output is the inversion of the input data. This feature is actually very useful in this situation as the microprocessor data bus buffers apply a logical inversion to the data. Thus, the function magnitudes do not need to be inverted before output even though it is still necessary for the frequency divider data.

The RAM' s input 1ines are connected through an input buffer to the microprocessor data lines. The RAM's chip select is earthed so that the RAM is permanently selected. The read/write line is switched by the control board so that only the RAM on the correct board is in the read position when the microprocessor is in control of the RAM address lines.

The RAM output lines go into a set of gates which allow the data to go through unchanged if the particular walsh function selected is in the '+1' state (logical '0'). It gives the 2's complement of the data when the selected walsh function is in its '-1' (logical' 1') state. This is equivalent to a multiplication of the function magnitude selected by +1 and -1 respectively. This means that only an adder is required in the next stage rather than an adder/subtractor.

The implementation of this circuit requires only two 7486 quad exclusive OR gates. The circuit uses the fact that the 2's complement of a number is the inversion of the number plus 1. The plus 1 is provided by supplying a carry in to the adder. The output is sign extended to 13 bits to correspond to the 13 bit adder used.

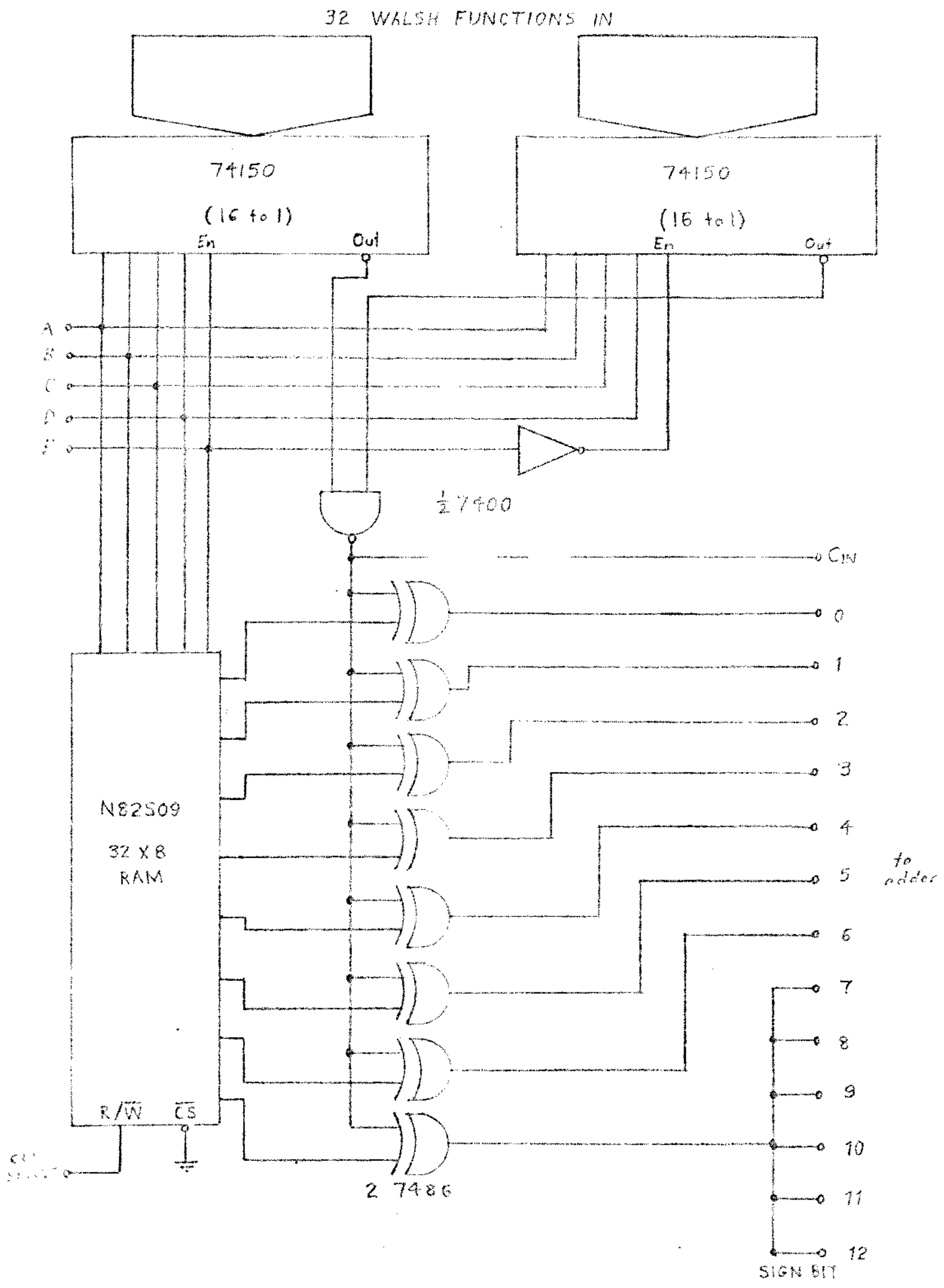


Fig 2.13 RAM and Negator.

The adder (Fig. 2.14) is made up from four 7483 4 bit full binary adders, with the top three bits left unconnected. The adder's output goes into a 1 master /2s1ave latch circuit.   Each of the three latches is implemented using two 74116 dual 4 bit latches. The master latch, which is connected to the adder's output, is enabled by the phase 2 clock going high. This is when the microprocessor assumes control of the RAM and walsh function select lines. On the falling of the phase 2 clock and the resuming of hardware control, one of the slave latches is enabled.

In the presence of a count complete signal from the control board the output latch is enabled and the other latch is cleared. When this signal is not present the output latch continues to hold the previous total and the other latch receives the information in the master. The output of the latter latch is fed into the second adder input.

Thus, for each cycle of phase 2, the control board counter is incremented; a walsh function magnitude selected which is either negated or not depending on the value of its corresponding walsh function; it is then added to an accumulated sum. On the 32nd cycle (count complete true), the value of the accumulator added to the last magnitude is put into the output latch and the accumulator is cleared.   This whole cycle is then repeated.

The output latch is connected through two 8095 tristate buffers enabled from the control board, on to the output bus.   The most significant bit is first inverted for direct input in offset binary notation.

This inverter and the phase locked loop clock would both be on the control board in a full system.

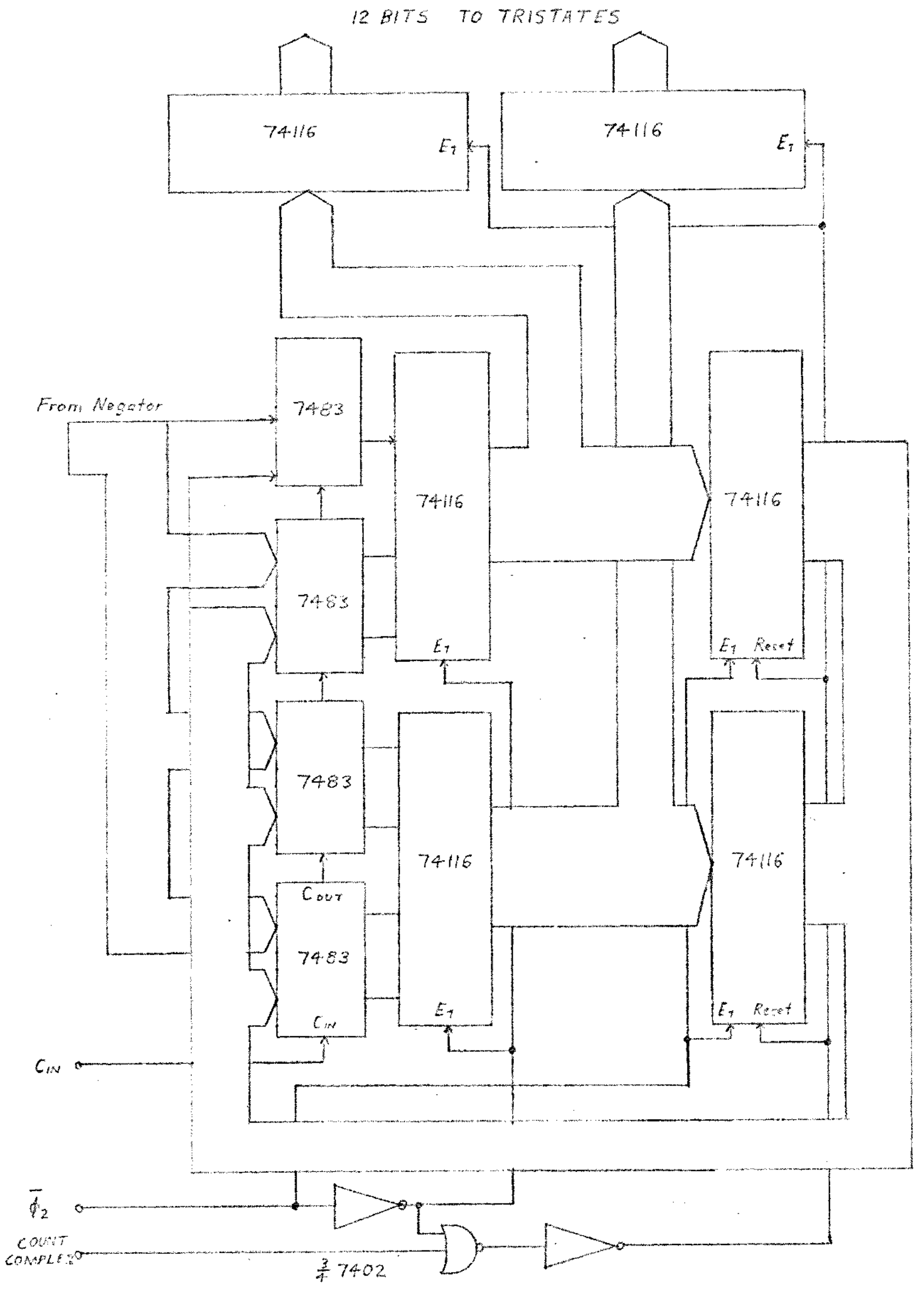


Fig 2.14 Adder & 1M/2S.

Microprocessor Software

The keyboard search and sorting techniques used were taken from a Bachelor of Engineering Thesis by Morris Swift (4) completed in 1976. Swift describes an efficient algorithm for scanning the keyboard and making the correct decisions from the data thus found. He also describes various allocation algorithms for assigning notes to the available output channels. The keyboard sorting algorithm involves the use of two tables labeled new and old.

A much simplified flow chart of the system appears in Fig. 3.1 and a program listing is in Appendix 2.

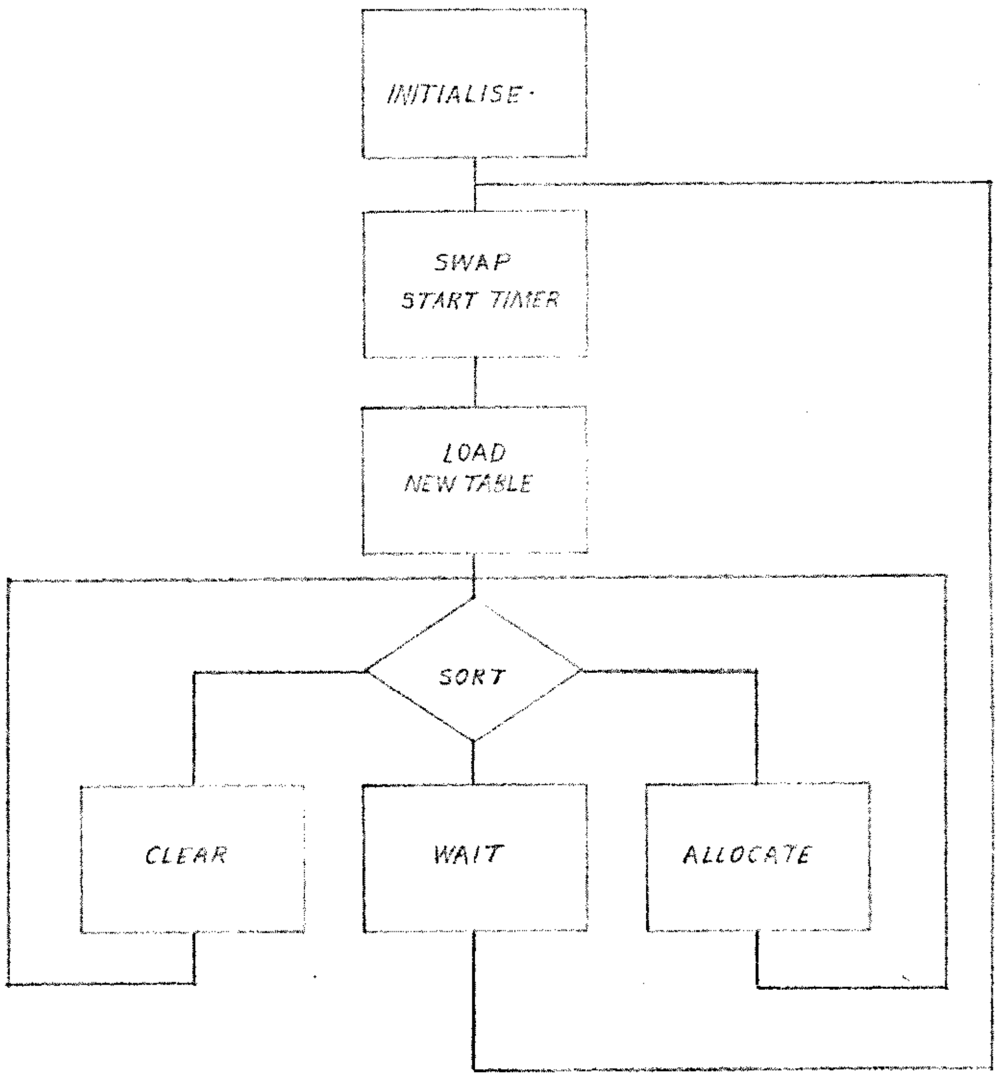


Fig 3.1 Overall Flow Chart.

The initialising clears all the program variables and the new, old and oscillator tables. It also sets up base addresses and loads the new table (which will be the old table after the base address swap occurs) with the

EOI marker of 7F (hex). This ensures that the program can be run with different function magnitudes without being reloaded into the microprocessor.

The next block changes the function of the new and old tables by swapping the base addresses. This is a far more efficient method than the obvious one of shifting the information from the new table into the old table before reloading the new table. This section also resets the contact debounce timer, this is done by addressing memory location 9F00 (hex).

The details of the keyboard scan algorithm are shown in Fig.3.2.

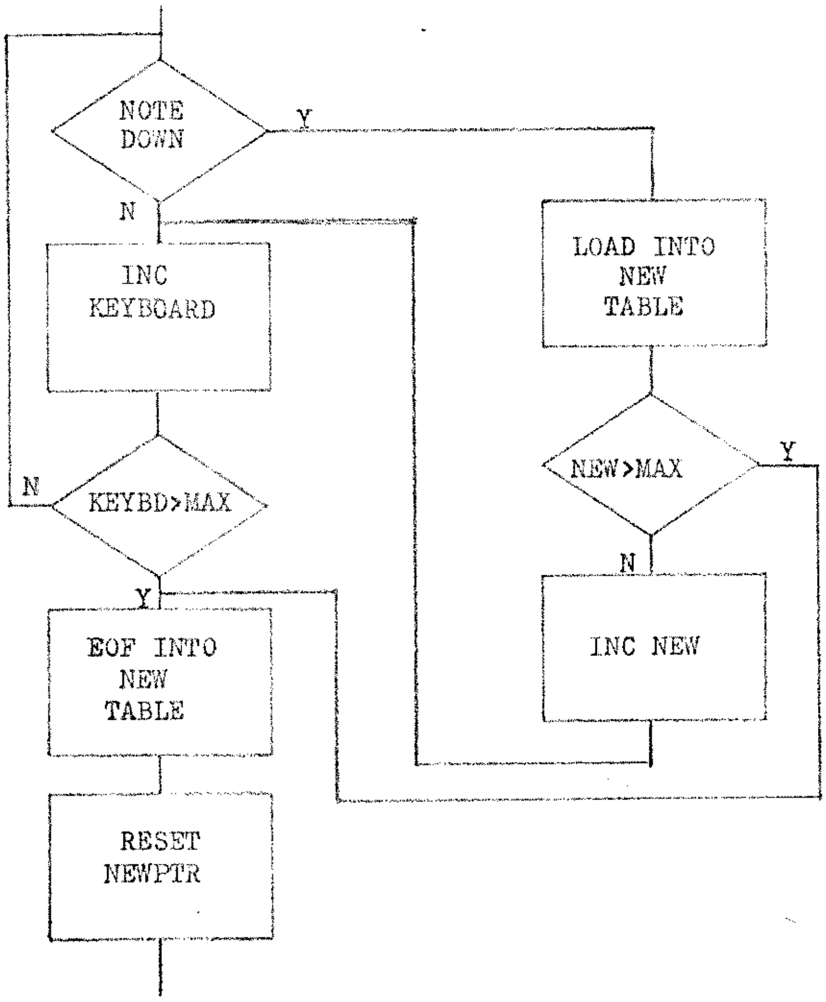


Fig 3.2 Keyboard Scan.

Each location is addressed with accumulator A acting as a counter. If the most significant bit is high (data bus tests as negative) the value in accumulator A is stored as the key code in the new table. The new table pointer is then incremented. The process is terminated when either the keyboard has been completely scanned or when the next table is full. The EOI character is then placed into the new table and the new table pointer (NEHPTR) is reset to the base value for the sort routine.

This method requires a maximum of 32 comparisons for tables with 16 locations, which is 2 orders of magnitude better than any other method mentioned by Swift. When a keyboard scan occurs the new tab1e is loaded with a code for each key down, the old table ho1ding the corresponding information for the previous scan. The tables are then searched and the appropriate action is taken. If a number occurs in both tables, then the corresponding key was down in both circumstances and no action need be taken. If the number occurs in the old table only then it has just been released and needs to be cleared. If it occurs only in the new table then the key has just been depressed and an output channel should be assigned to it.

As the code runs in ascending order these processes can be implemented by comparing the values in the tables as shown in Fig. 3.3. To provide termination an end of information (EOI) marker is put at the end of the new table after each scan. This is in fact just a number greater than the code for any available key. It is taken by the sorting algorithm to be a normal key code and provides the correct operation until the value in both the old and new tables equals EOI, at which point the sort has been completed. The new and old tables then swap meaning and the new 'new' table is loaded from the next keyboard scan.

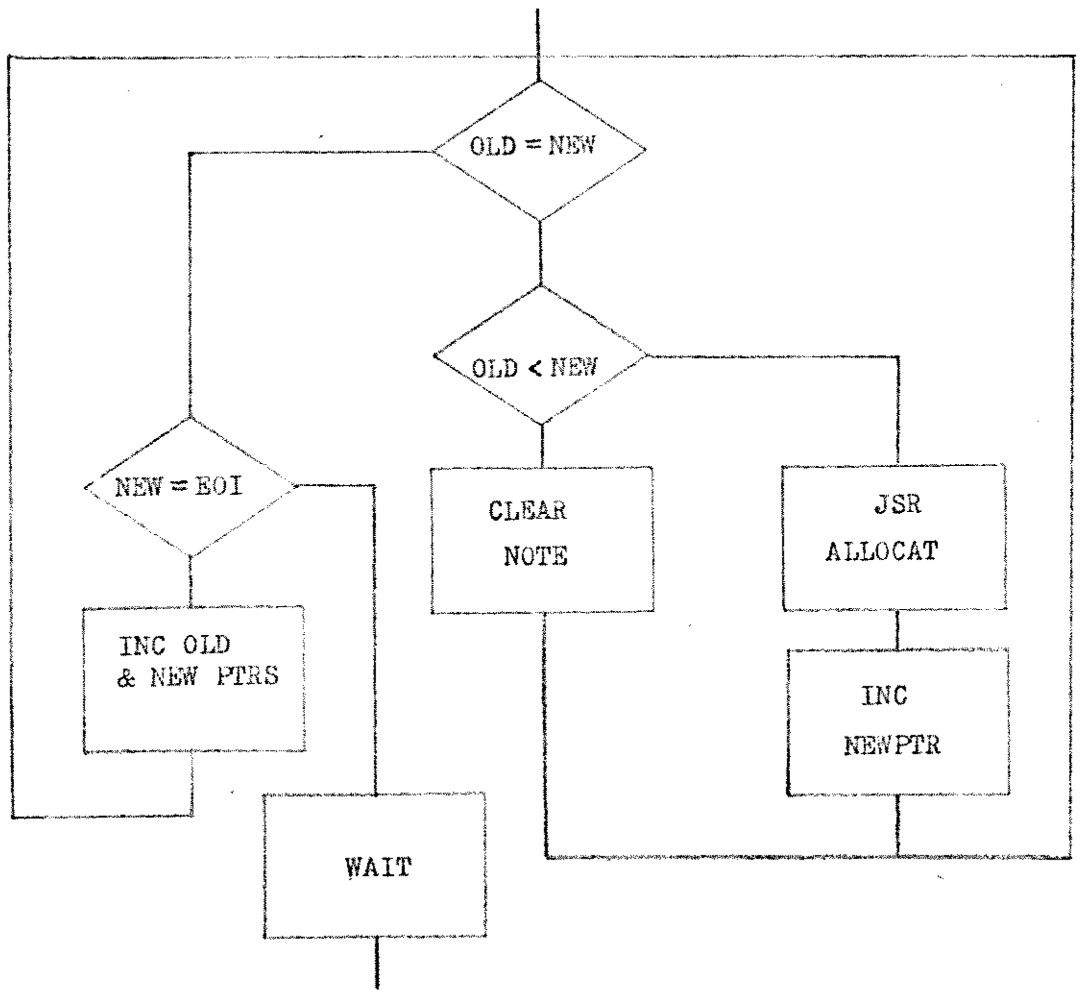


Fig 3.3 Sort Routine.

The code used for a key is actually an 8 bit word. The three least significant bits indicate in which octave the note occurs and the next four bits indicate the semitone. The most significant bit is used later as a flag.

To keep track of which key has been assigned to which output channel and whether that key is down or not an 'oscillator table' is kept. The information in this table is the note code, as above, with the most significant bit being high if the key is down.

In an expanded system this flag bit would be used to indicate whether the envelope of the note should be either in its decay or attack mode.

As the present system does not have envelope control the note is switched off by putting the magnitude of each walsh function to zero.

The allocation algorithm implemented features, using Swift's terminology, non-strict rotation and inhibited multiple loading. This involves the allocation of notes only to channels that are clear except that, if possible, the same channel is used if a note is played in quick succession.

The flow chart of the allocation subroutine is shown in Fig. 3.4. To find a channel for a note, the output table is first searched for a match with the keycode, then for a free space, then for a key which has been released. This utilises a subroutine, Fig. 3.5, which searches through the oscillator table until either a match is found or the whole table has been searched. The actual comparison made depends on the value in accumulator A, if it is zero then the value in the oscillator table is compared with the contents of accumulator B, otherwise a check is made on the sign bit (key down flag).

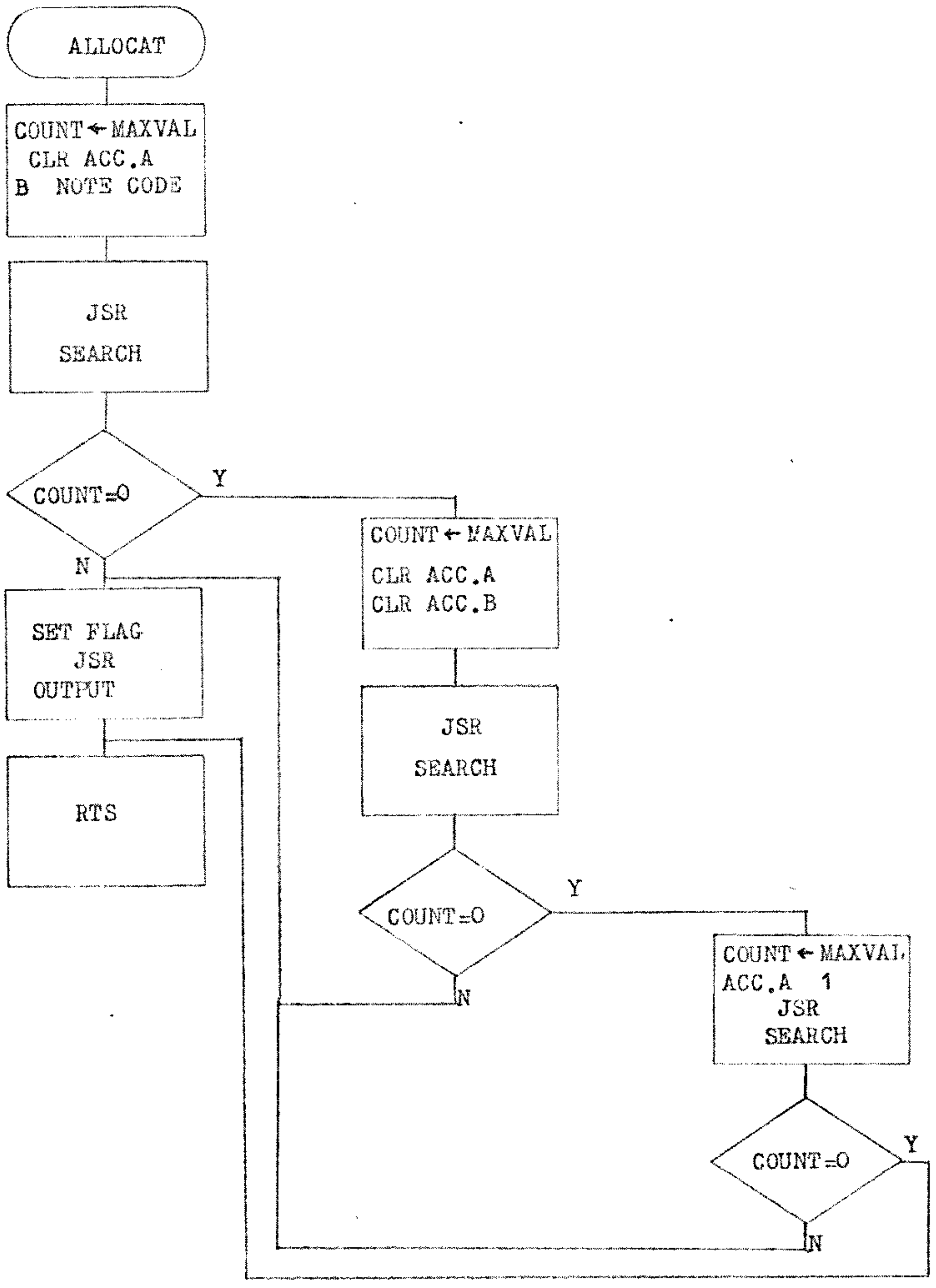


Fig 3.4 ALLOCAT Subroutine.

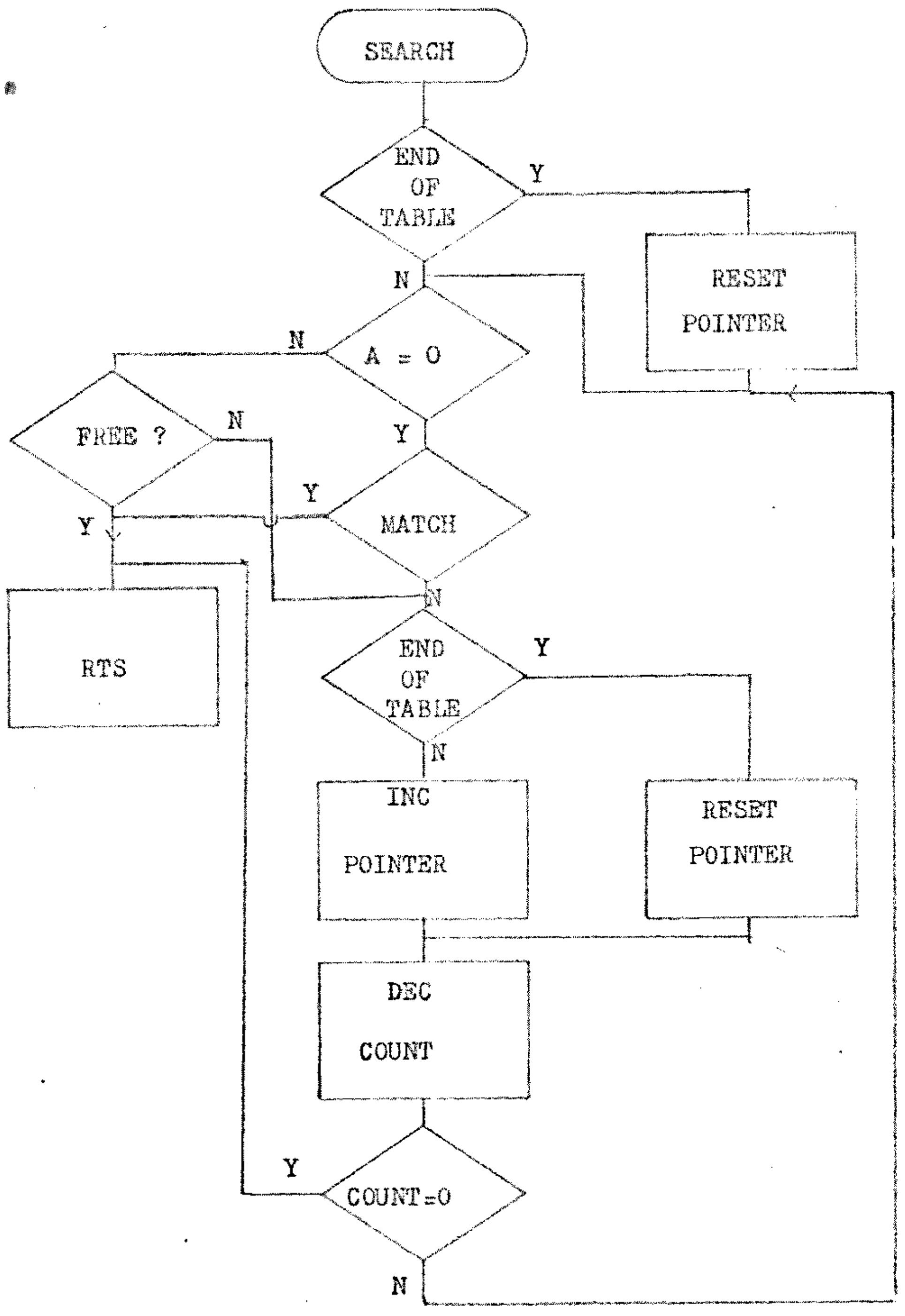
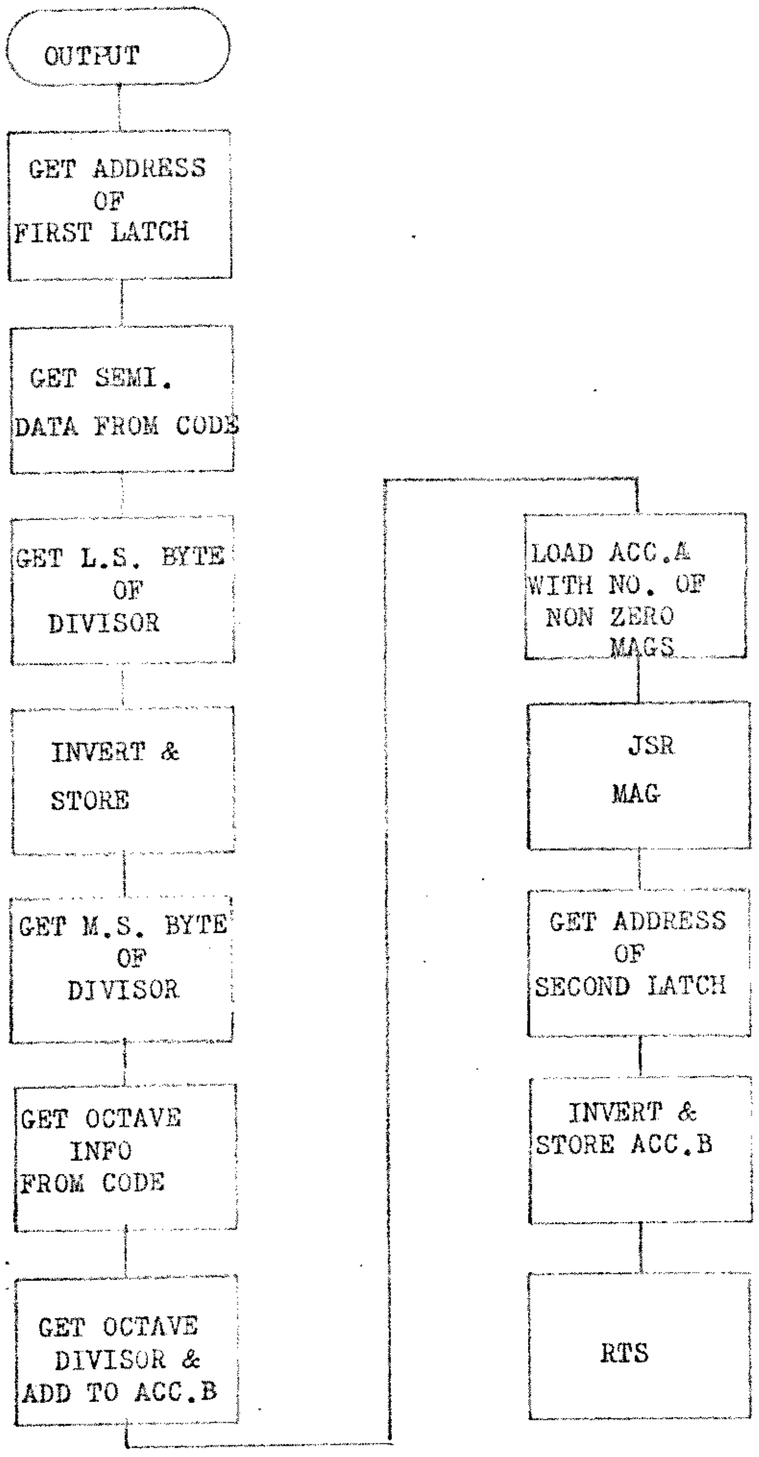


Fig 3.5 SEARCH Subroutine.

The check for a free space is only useful at start up in the present system. However, it is conceivable that in an expanded system, the contents of the oscillator table corresponding to an output channel which has completed its envelope decay, would be cleared. This would ensure that new notes would be assigned preferentially to those channels which have fully completed their output cycles. To see whether the search has been successful or not, the allocation subroutine checks the value of 'count', that is, it checks whether or not each value in the oscillator table was accessed before completion of the search subroutine.

If a usable output channel has been found another subroutine, 'OUTPUT', takes over (Fig. 3.6). This subroutine stores the correct division factor in the phonic board divider latches.

When OUTPUT is called, the least significant byte of the index register contains the code number of the phonic board which has been assigned to the key whose code lies in accumulator A. This needs to be converted to the latch address.



OUTPUT
GET ADDRESS OF FIRST LATCH
GET SEMI. DATA FROM CODE
GET L.S. BYTE OF DIVISOR
INVERT & STORE
GET M.S. BYTE OF DIVISOR
GET OCTAVE INFO FROM CODE
GET OCTAVE DIVISOR & ADD TO ACC.B
LOAD ACC.A WITH NO. OF NON ZERO MAGS
JSR MAG
GET ADDRESS OF SECOND LATCH
INVERT & STORE ACC.B
RTS

Fig 3.6 OUTPUT Subroutine.

In Fig. 3.7 the addresses of each latch are tabulated. As can be seen, the most significant byte contains the phonic board number with an offset of 90(hex). To get the information in the index register into this form, a 3 byte memory space 'OSCADD' is used. The index register is stored in the first two bytes and then the first latch address can be read straight from the second two most significant bytes after 90 (hex) has been added to the second most significant byte.

ADDRESS (hex)	LATCH
9000	0 Lower
9080	Upper
9100	1 Lower
9180	Upper
9200	2 Lower
9280	Upper
9300	3 Lower
9380	Upper
9400	4 Lower
9480	Upper
9500	5 Lower
9580	Upper
9600	6 Lower
9680	Upper
9700	7 Lower
9780	Upper
9800	8 Lower
9880	Upper
9900	9 Lower
9980	Upper

MAGNITUDE RAM

No.	Address
0	9A00 - 9A1F
1	9A80 - 9A9F
2	9B00 - 9F1F
3	9B80 - 9B9F
4	9C00 - 9C1F
5	9C80 - 9C9F
6	9D00 - 9D1F
7	9D80 - 9D9F
8	9E00 - 9E1F
9	9E80 - 9E9F

Fig 3.7 Addresses.

The semi tone division factor is stored in the 'pitch table' and the information to go into the first latch (least significant byte of divisor) is read straight from this.

To get the table address the semitone data is isolated from the key code by shifting right three times. It is then shifted left once, as the semitone division factor takes up two bytes and only the first is required, a new factor starting at every second byte. An offset is added to this va1ue and the table address is complete and is placed in 'STORE'. The octave division code must still be added. This is implemented by removing the semi tone information from the keycode and then rotating it left. This value is then loaded into the index register. The octave division factor stored at this address, with offset 80 (hex), is then added to accumulator B to form the most significant byte of the complete division factor and accumulator A is also loaded with the value at the next location in the table.

This value stored in accumulator A is the number of walsh functions that can have non zero magnitudes in each particular octave. This value is used by the subroutine 'MAG' which is called at this stage. MAG loads the function magnitude RAM on the phonic board with the correct values. After MAG has been completed, the address of the second latch is found by storing 80 (hex) (Fig. 3.7) in the third byte of OSCADD, which still contains the address of the first latch and loading the index register with the second and third byte of OSCADD. The inversion of the value in accumulator B is then stored at this address.

The note clearing routine, Fig. 3.8, searches through the oscillator table until the code in the table matches the key code, with flag bit, stored in accumulator B. If no match is found the note has not been allocated an output channel and no action need be taken. If action is necessary, both the flag and accumulator A are then cleared. The function magnitude subroutine MAG is then called under the name MAGC. This clears all the magnitudes stored in the RAM on the phonic board with needs to be switched off. The 'OLDPTR' is then incremented and the sorting algorithm continued.

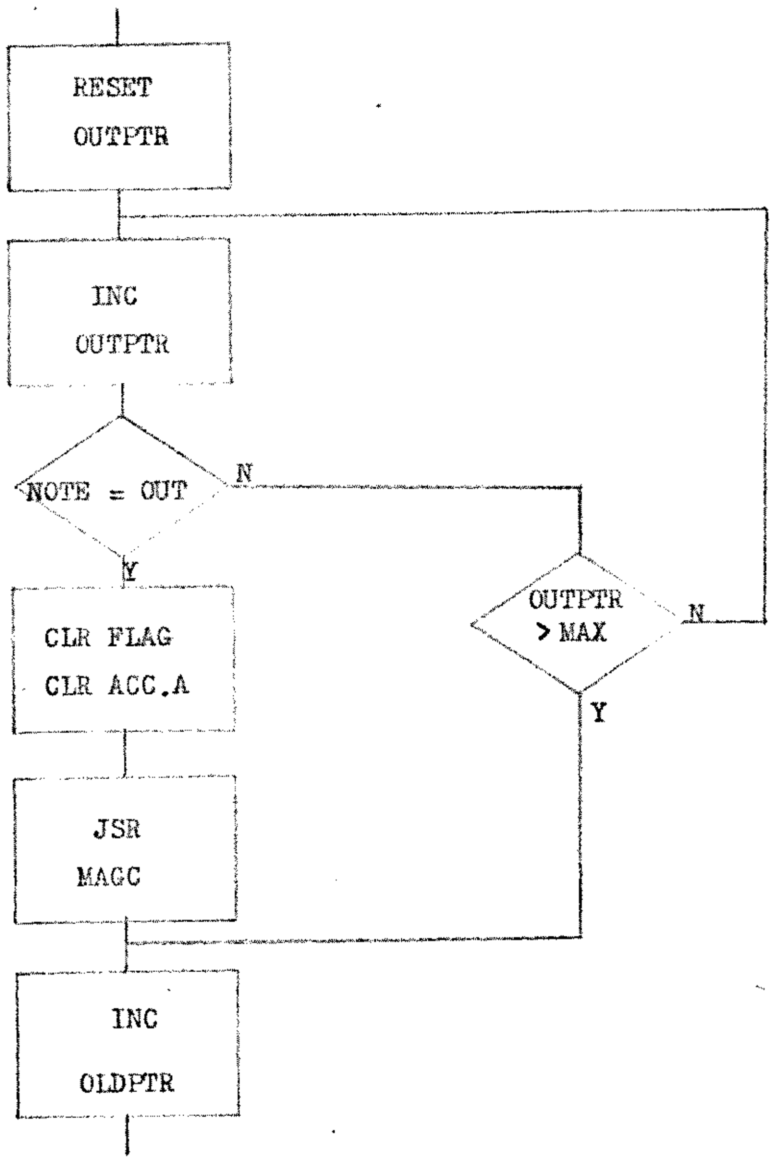


Fig 3.8 Note clearing.

MAG, Fig. 3.9, is, as mentioned, used to store the correct function magnitudes in the RAM on the correct phonic board. It does this by storing the coefficients found in the magnitude table in the RAM until a counter is equal to the value preset in accumulator A. The rest of the magnitudes are then set to zero. In this way it can be used for notes in any octave and also be used to 'switch off' a board by setting all of its walsh function magnitudes to zero.

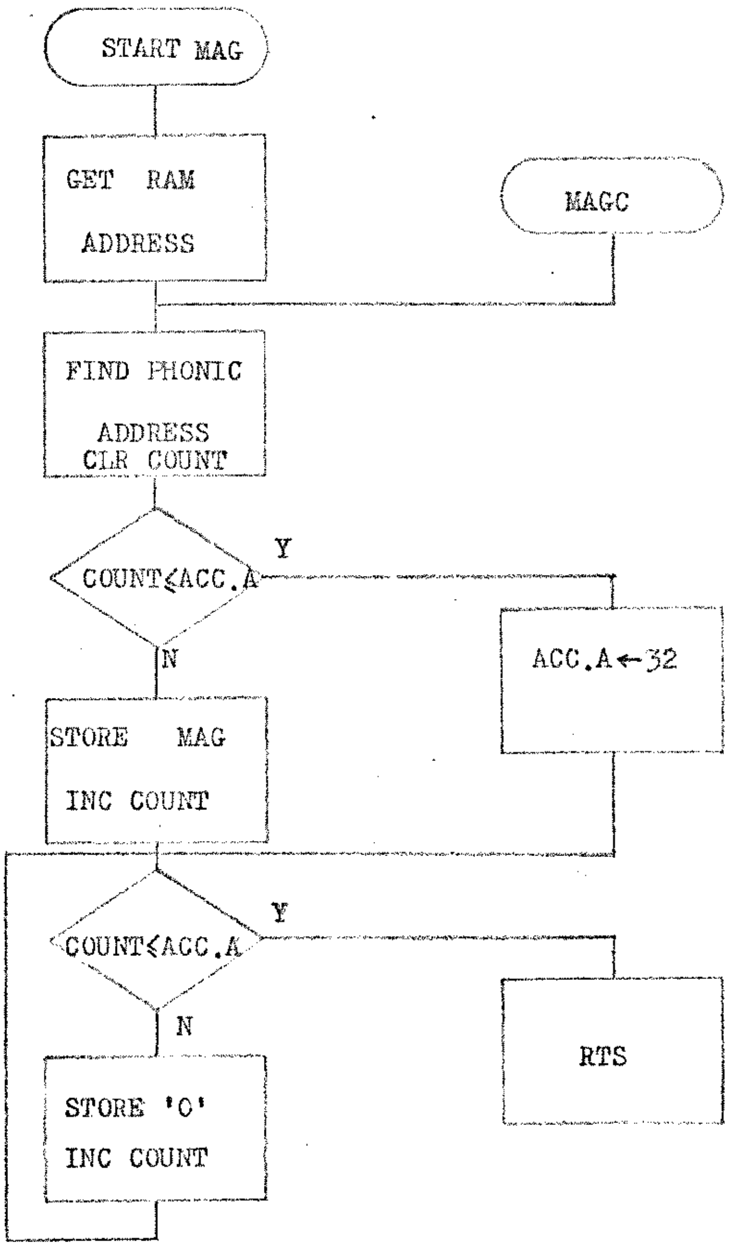


Fig. 3.9 MAG Subroutine.

The RAM address, Fig. 3.7, is derived from the phonic board number. When MAG is called by the note clearing algorithm this is already stored in accumulator B, but when called by the channel allocating subroutine 'ALLOCAT', it needs to be derived from the latch address.

For this reason the note clearing algorithm calls the subroutine under the name MAGC which misses out the board number derivation routine.

If the board number is even, 9A (hex) needs to be added to half the board number to form the most significant byte of the RAM address. If odd an additional 80 (hex) needs to be added to the least significant byte of the address. Division is accomplished by a right shift, with the resulting value of the carry giving an indication of whether the number was odd or even.

COUNT is first cleared and then incremented every time a magnitude is stored. This continues until COUNT equals accumulator A. Accumulator A is then set to 32 (decimal) and the process continues, storing a zero magnitude until COUNT is again equal to accumulator A. If, as in the note clearing situation, accumulator A is first set to zero, the first set of loads is skipped and all the magnitudes stored in the RAM are set to zero.

When the sorting algorithm has come to the end of both the new and old tables a wait period is initiated until the timer resets. This is so as to provide keyboard switch debouncing by fast scanning, with a delay between scans which is longer than the contact bounce time (4).

Fast Walsh Fourier Transform Program

One of the problems in using the synthesiser system described to generate a given waveform, is obtaining the coefficients for the walsh functions. To facilitate this a program was written, in a language called Pascal, which evaluates the coefficients when the equation for the desired waveform is placed in the program.

This program uses the method of the Fast Walsh Fourier (or Fast Hadamard) Transform and can be used to generate coefficients for any number of walsh functions. The program listings and sample outputs are in Appendix 2.

A fourier coefficient is actually the average value of the waveform to be approximated, multiplied by the approximating waveform. This is usually found by finding the integral of the product and dividing by the period of the waveform as shown:

$$C = \frac{2}{T}\int_{0}^{T} f_1(t).f_2(t)\ dt$$

In the walsh function case, the approximation waveform is digital in nature and thus has only discrete values.

It is not expressable in the form of a continuous function over the period and thus needs to be broken up and expressed as a constant value between each of its discontinuities, e.g. x = sal(l) can be expressed by

$$\begin{aligned} x &= 1 && \text{for} \quad 0 < t < 0.5 \\ &= -1 && \text{for} \quad 0.5 < t < 1 \end{aligned}$$

with the resultant formula:

$$C_1 = \frac{2}{T}\left(\int_{0}^{T/2} f_1(t)\ dt - \int_{T/2}^{T} f_1(t)\ dt\right)$$

As a digital computer cannot evaluate integrals, the expression still needs to be changed. This is done by taking the value of the waveform to be approximated at a number of points along its length. The values thus found are taken to be the magnitude of the waveform for each segment between each point at which it is evaluated and the next point on the right, Fig. 4.1. Note that the discontinuities in the approximating waveform must occur at the same place as the discontinuities in the segmented waveform.

The resulting expression is very easily solved:

$$\begin{aligned} C_1 &= \frac{2}{T}(A.t + B.t + C.t + D.t + E.t - F.t - G.t - H.t) \\ &= \tfrac{1}{4}(A+B+C+D-E-F-G-H \quad (\text{as } 8.t = T) \end{aligned}$$

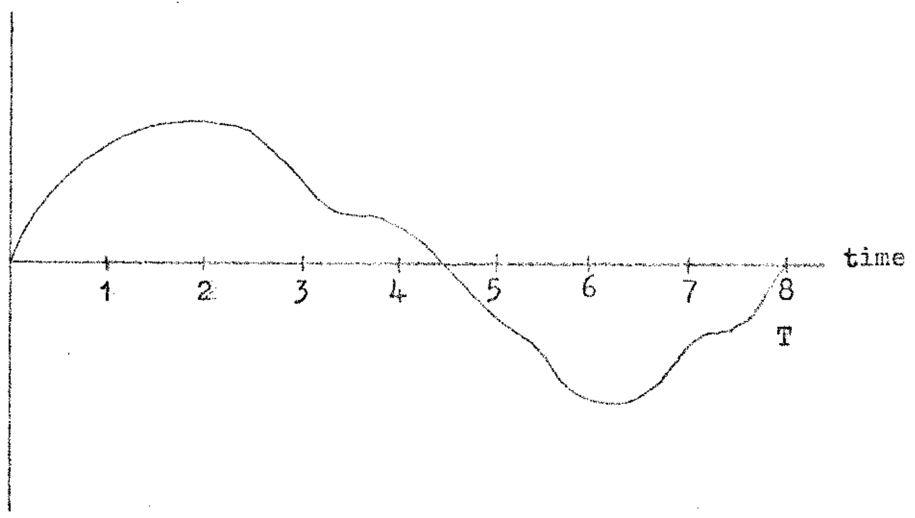
where A,B ... H are the magnitude of the waveform at 0, t, 2t ... 7t

It is in fact convenient to make the discontinuous in the segmented waveform occur at the same spot as each of the discontinuities in the highest sequency walsh function used in the approximation. For this purpose, this walsh function actually needs to be a Rademacher function (square wave).

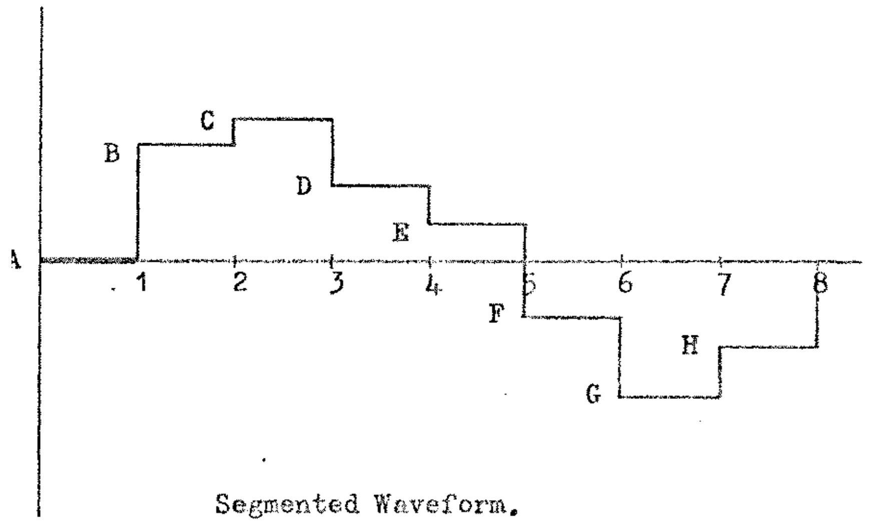
Using this system on the waveform in Fig. 4.1 the coefficients can be found as follows:

$$\begin{aligned}
\text{coefficient of } W(0) &= A+B+C+D+E+F+G+H \quad \text{(the D.C. value)}\\
S(1) &= A+B+C+D-E-F-G=H\\
C(1) &= A+B-C-D-E-F+G+H\\
S(2) &= A+B-C-D+E+F-G-H\\
C(2) &= A-B-C+D+E-F-G+H\\
S(3) &= A-B-C+D-E+F+G-H\\
C(3) &= A-B+C-D-E+F-G+H\\
S(4) &= A-B+C-D+E-F+G-H
\end{aligned}$$

This is the normal Walsh Fourier Transform and each of these coefficients can be calculated separately.



Original Waveform.



Segmented Waveform.

Fig 4.1 Segmented & Original Waveform.

It can be seen from these equations that the terms 'A+B' and 'A-B' each occur four times and this is true of all the other single pairs of terms 'C+D' etc. Thus, great savings can be made if these common pairs are evaluated first, for all the coefficients. The resultant simplified expressions also show the same behaviour, with each pair occurring twice. The Fast Walsh Fourier Transform uses this fact to greatly speed up the calculation of the coefficients.

A resultant signal flow graph from a paper by J.L.Shanks (5) is shown in Fig. 4.2. The resultant coefficients are found in a bit reversed gray code ordering in this particular form of the transform.

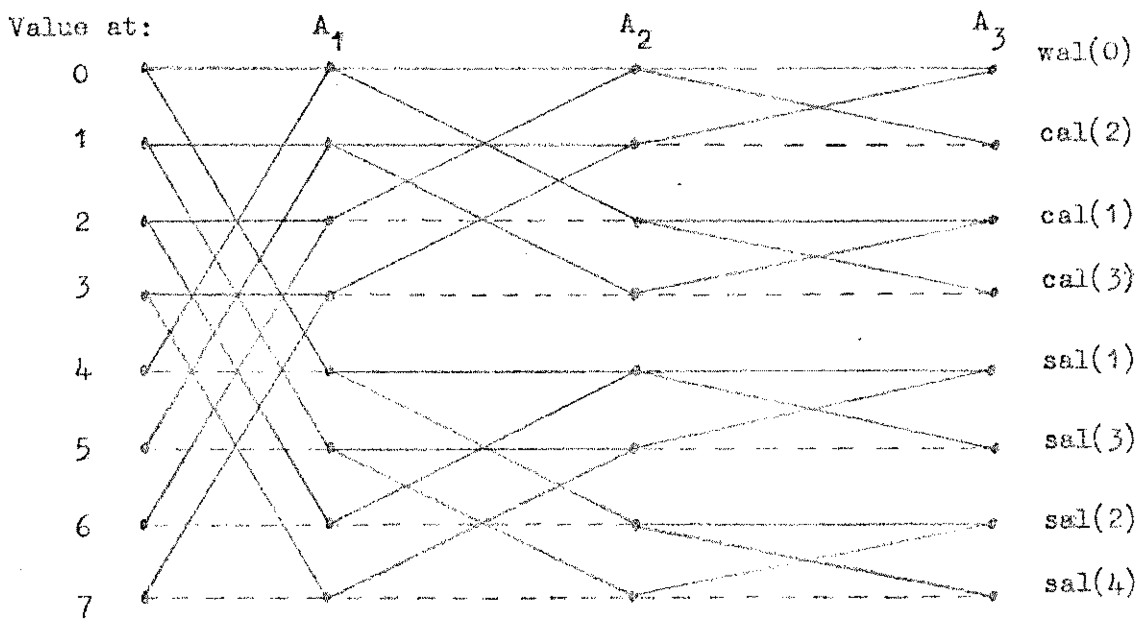
The savings using the Fast Walsh Fourier Transform (FWFT) are obvious even at such low numbers of coefficients as used in this example. For the normal method 7 x 8 = 56 operations are needed, the FWFT only requiring 3 x 8 = 24 operations.

However, the complexity of converting the output into a logical sequence offsets this saving greatly.

A Fortran program for the FWFT is given in (7) but the output sorting algorithm is far more complex than the FWFT, involving many calculations of powers.

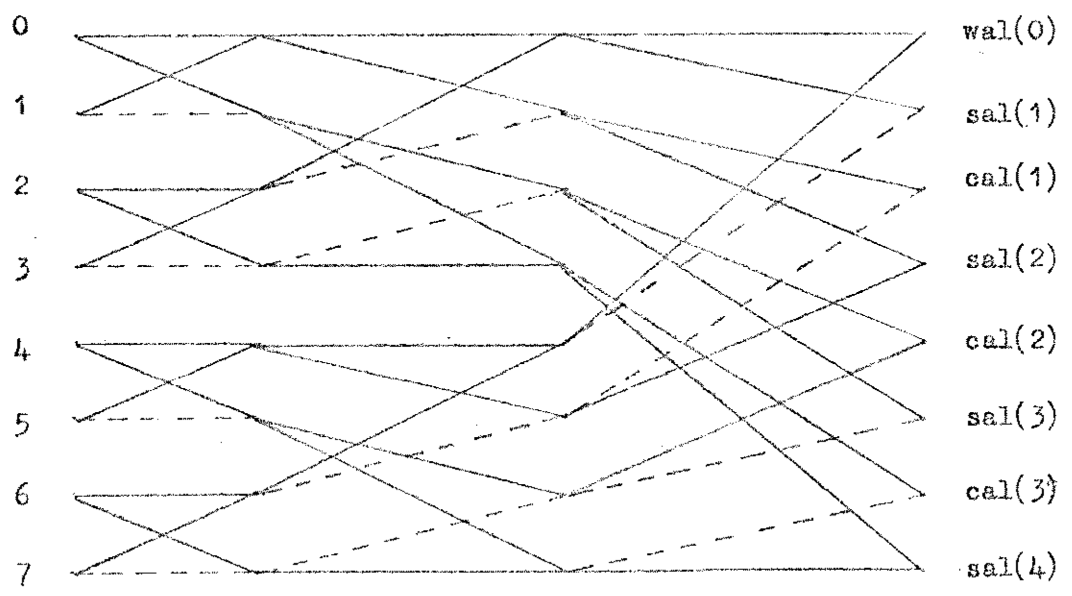
Due to inexperience in Fortran programming an attempt was made to translate this program into Pascal, but this proved to be difficult as Pascal does not have a standard power solving routine.

For this reason and because the complexity of efficiently sorting the coefficients in all the examples of the FWFT found, a new flow graph was devised which results in normal ordering of the coefficients. This is shown in Fig. 4.3.



——— denotes add.
– – – denotes subtract.

Fig 4.2 Shank's FWFT.



——— denotes add.
– – – denotes subtract.

Fig 4.3 Alternative FWFT.

The algorithm for this scheme also proved to be very easy to implement. Even though Pascal is a rather verbose language, the resultant program was only two lines longer than the fortran program mentioned earlier.

The program also does a search and uses this to scale all the coefficients so that the biggest has a magnitude of 127 decimal (7F hex). This is the biggest signed number that can be stored in the 8 bits allocated to the coefficients. This ensures that the waveform uses all the available accuracy and is not lost in noise.

The program is easily changed for differing numbers of coefficients by changing two constants L and N, given that N+1 = 2L+1, where N+1 is the desired number of coefficients.

Two versions of the program were written, one tabulates the data in the more standard notation for walsh functions of wal(0), wal(l) (= sal(1)), wal(2) (= cal(l)) etc. The output file of the other version can be directly assembled by the UNIX 6800 assembler for loading of the microprocessor. This version only gives the coefficients of the sal terms, as these are the only terms generated by the synthesiser hardware.

This latter version is extremely useful as no manual loading, with the resultant inaccuracies, is necessary.

Alternatives

As mentioned in Section 1, walsh functions can be generated by the exclusive oring of square waves which are at octave intervals (Rademacher functions).

With six Rademacher functions available, 128 (26) different combinations are available, of these, 64 are sal terms, 63 are cal terms and one is wal(0), i.e. a D.C. level. In the system described only the sal terms are required. In fact, the sal terms occur when the number of Rademacher functions exclusive ored together is odd, the cal occurring when the number is even and wal(0) when none are combined.

Thus, a 5 bit binary counter (Fig. 5.1) combined with an odd parity generator (an exclusive NOR circuit) will give 6 lines which will count through the 32 sal terms. The order in which the terms appear would not be in order, however.

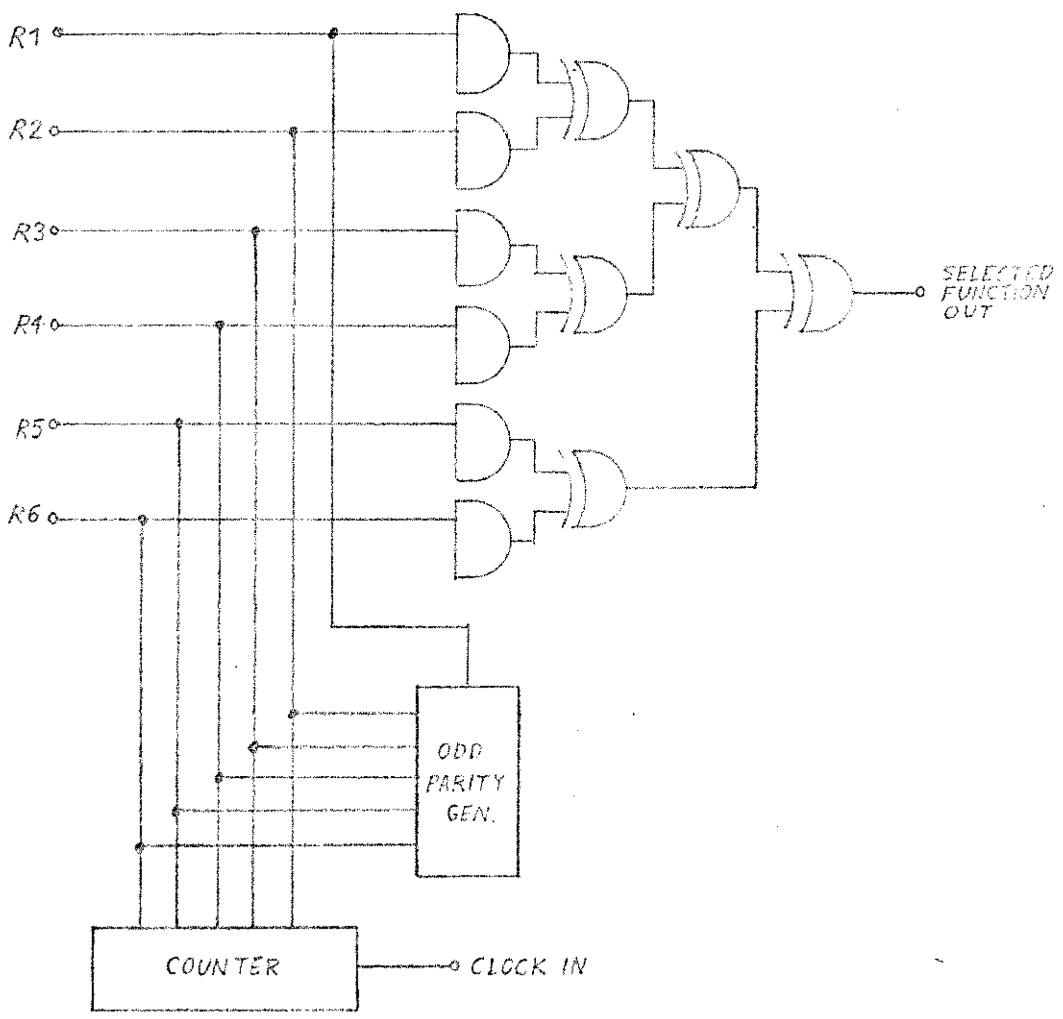


Fig 5.1 Alternative Function Generator.

A 6 bit gray code counter running at twice the desired frequency would also have the same effect and give the values in order, as long as it's output were latched at the desired frequency of 1 MHz.

The method involving the binary counter is very easily implemented, requiring only a quad exclusive OR gate and an inverter. The present system of generating each function separately requires 16 quad exclusive OR gates plus buffers and a 32 to 1 multiplexor (2 I.C. 's plus 1 inverter) so the saving in hardware is marked (17 I.C.'s).

The software required to take care of the altered output order would not be too difficult.

It would be made easier with a microprocessor equipped with indirect addressing such as the Signetics 2650, but it is still fairly easy with the Motorola 6800. This system has not been tried yet but seems very simple and reliable.

The method used to obtain square waves of the correct frequency on each phonic board reduces the generality of the synthesiser and makes it difficult to have other than the normal definition of an octave, i.e. the top frequency is twice the lower frequency, but it is necessary when using a lower clock frequency. If a clock of 32 MHz was used, a more general system could be made, consisting only of down counters. For the normal chromatic scale the division factors for each semi tone in the lowest octave would be stored and divided by two (shifted right) the correct number of times for the other octaves. The phase locked loop circuitry would be more difficult to implement at this higher frequency, so a basic clock of 32 MHz with a divide by 32 stage giving 1MHz for the microprocessor clock would probably best solution. The 1 MHz clock would need to be a 2 phase, non overlapping clock system. The use of a higher sampling rate such as 2 MHz, would also be possible with this system and would result in the greater accuracy in the upper octaves.

Review of Method

In order to give some numerical value to the soundness of using walsh functions, a numerical example will be discussed.

To simplify the mathematics, the case where 16 sal components are combined to give a sine wave with a frequency of 100 Hz will be used. The values obtained for the coefficients were found using the previously described FWFT program. In Appendix 2, the following weights are given for 16 sal terms:

127, -7, -57, -3, -11,1, -27, -1, -3, 0,1,0, -6, -14, -1.

When these are applied to their respective walsh functions and summed the waveform illustrated in Fig. 6.1 results. This figure also shows the original sine wave and the resultant error waveform.

This error waveform is sawtooth in shape, with a beat frequency twice that of the original waveform. The beat shows that the error is largely composed of two waves, of almost equal magnitude as shown by the beat minimum going close to zero. They differ in frequency by two times the original waveform frequency (f).

Using this fact and the Fourier Series representation of a sawtooth the peak amplitude of each of these waves can be found and is approximately the peak amplitude of the error waveform, which is 21. This is because, for a sawtooth waveform, the magnitude of the fundamental is twice that of the sawtooth waveform (6).

As already mentioned, the beat frequency is 2f (200 Hz). The frequency of the error waveform is 32 times the original frequency and so is 32f.

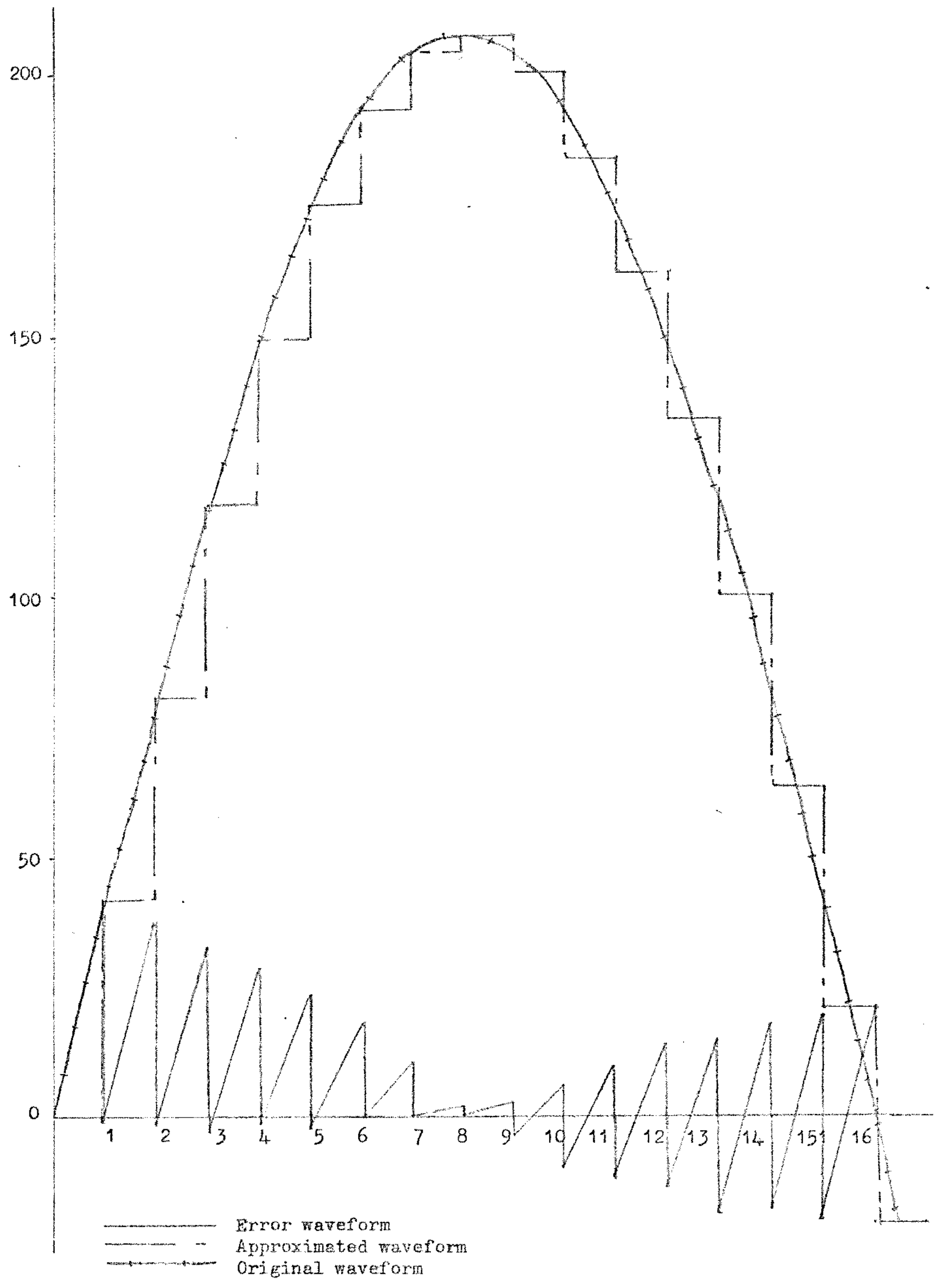


Fig 6.1 Waveforms.

Therefore:

$$32f = \frac{f_1+f_2}{2}$$

$$\text{i.e. } f_1+f_2 = 64f \qquad 1.$$

$$\& \quad f_1-f_2 = 2f \qquad 2.$$

1. + 2. gives

$$2f_1 = 2f + 64f = 66f$$

$$f_1 = 33f$$

sub in 1. gives

$$33f_2 + f_2 = 64f$$

$$f_2 = 31f$$

So the major harmonics are the 31st and 33rd. Each of these having a magnitude of 21 (peak).

The original waveform has a magnitude of 208 (peak) so the 31st and 33rd components are both 20dB below their fundamental.

In this case of f = 100 Hz, they have frequencies of 3.1 kHz and 3.2 kHz which are still within the audible range.

When the actual number of sal components used is 32, as in the synthesiser described, these error frequencies would be the 63rd and 65th harmonics. These would have frequencies of 6.3 kHz and 6.5 kHz with magnitudes of 10.5 (peak). This means that they would be 26dB below the fundamental, which is subjectively a 1/4 to an 1/8 th of the loudness of the fundamental.

These frequencies are still within the audible region. Both are large enough and high enough harmonics for no masking by the fundamental to take place.

For these reasons the present system would not be very useful for approximating waveforms at low frequencies.

Even with the doubling of the number of sal terms used to 64 this problem still exists as the error frequencies would still be within the audible frequency range and be only 32 dB down.

This error would give the synthesiser its own characteristic tone, while this would not be unmusical, it would limit its usefulness.

An obvious solution is to use a tracking filter on the output to remove these higher harmonics. This method is easily implemented, but if put on each output channel would require the use of ten digital to analog convertors instead of one.

The system was found to exhibit random changes in output waveform. The waveforms were quite stable once started, but were often lacking in the higher walsh functions. It would thus appear that, on occasions, only some of the magnitudes were successfully loaded into the magnitude RAMs. The actual source of this fault was not found.

Conclusion

The use of walsh functions in a synthesiser has many advantages. The functions are extremely easy to generate and manipulate using digital techniques. The necessary coefficients can also be easily found using a digital computer.

However, the resultant waveforms are rich in high harmonics, which would need to be removed before ‘smooth’ waveforms which have low fundamental frequencies could be reproduced.

This is not an impossible task and so walsh functions still have the possibility of being extremely useful in musical synthesisers.

Bibliography

1. Walsh, J.L., A closed set of orthogonal functions, Am. J. Math., vol. 55, pp. 5-24, January 1923.
2. Harmuth, H.F., Transmission of Information by Orthogonal Functions. New York/Berlin 1969: Springer-Verlag.
3. Henderson, K.W., Some Notes on the Walsh Functions, IEEE Trans. on Electronic Computers, vol. EC-13, pp.50-52, February 1954.
4. Swift, M., Polyphonic Keyboards for Music Synthesisers, B.E. Thesis, University of N.S.W. 1976.
5. Shanks, J. L., Computation of the Fast Walsh-Fourier Transform, IEEE Trans.on Computers, vol. C-18, pp. 457-459, May 1969.
6. Haliday, D. & Resnick. R., Physics for Students of Science and Engineering, 1966, J. Wiley & Sons.

Appendix 1 – Microprocessor Listing

Appendix 2 – FWFT PROGRAM LISTINGS AND SAMPLE OUTPUTS

(a) Version 1. Sample output for:

Sine with N =31 and 63

```
PROGRAM WAL (INPUT, OUTPUT);
CONST N = 63;
      L = 5;
      PI = 3.1415926;
VAR   A, B, C, I, J, X, Y, Z, TI : INTEGER;
      VAL : ARRAY[0..1,0..N] OF REAL;
      R : REAL;
      PLUS : BOOLEAN;
BEGIN
      A := 0; B := 1;
      FOR I := 0 TO N DO
            VAL[A,I] := SIN(2 * PI * 1 / N);
      TI := 1     {TI = 2**I}
      FOR I := 0 TO L DO
            BEGIN
            C := B; B := A; A := C; {SWAP A & B}
            X := 0; Y := 0; Z := TI; PLUS := TRUE;
            REPEAT
                  FOR J := 0 TO 1 DO
                        BEGIN
                        IF PLUS THEN VAL[A,X] := VAL[B,Y] + VAL[B,Y+TI]
                              ELSE VAL[A,X] := VAL[B,Y] - VAL[B,Y+TI];
                        PLUS := NOT PLUS;
                        END;
                  Y := Y + TI;
                  IF Y = Z THEN
                        BEGIN
                        Y := Y + TI;
                        Z := Z + 2 * TI;
                        END;
                  IF I <> 0 THEN PLUS := NOT PLUS;
            UNTIL X > N;
            TI := 2 * TI;
            END;
      R := ABS(VAL[A,0]);
      FOR I := 1 TO N DO
            IF R < ABS(VAL[A,I]) THEN R := ABS(VAL[A,I]);
      FOR I := 0 TO N DO
            WRITELN(‘WAL(‘,I:2,’) =’,ROUND(127/R*VAL[A,I]):3);
END.
```

[Output Omitted]

Version 2. Sample outputs for:

Sine with N =31 and 63
Sawtooth with N=63
Triangle with N=63.

```
PROGRAM WAL (INPUT, OUTPUT);
CONST N = 63;
      L = 5;
      PI = 3.1415926;
VAR   A, B, C, I, J, X, Y, Z, TI : INTEGER;
      VAL : ARRAY[0..1,0..N] OF REAL;
      R : REAL;
      PLUS : BOOLEAN;
BEGIN
      A := 0; B := 1;
      FOR I := 0 TO N DO
            VAL[A,I] := SIN(2 * PI * 1 / N);
      TI := 1     {TI = 2**I}
      FOR I := 0 TO L DO
            BEGIN
            C := B; B := A; A := C; {SWAP A & B}
            X := 0; Y := 0; Z := TI; PLUS := TRUE;
            REPEAT
                  FOR J := 0 TO 1 DO
                        BEGIN
                        IF PLUS THEN VAL[A,X] := VAL[B,Y] + VAL[B,Y+TI]
                              ELSE VAL[A,X] := VAL[B,Y] - VAL[B,Y+TI];
                        PLUS := NOT PLUS;
                        END;
                  Y := Y + TI;
                  IF Y = Z THEN
                        BEGIN
                        Y := Y + TI;
                        Z := Z + 2 * TI;
                        END;
                  IF I <> 0 THEN PLUS := NOT PLUS;
            UNTIL X > N;
            TI := 2 * TI;
            END;
      R := ABS(VAL[A,0]);
      FOR I := 1 TO N DO
            IF R < ABS(VAL[A,I]) THEN R := ABS(VAL[A,I]);
      WRITELN(‘ ORG X’’00D0’);
      WRITE(‘ FCB ‘,ROUND(127/R*VAL[A,1]):1);
      FOR I := 0 TO N DIV 2 DO
            WRITELN(‘,’ROUND(127/R*VAL[A,1+2*I]):1);
      WRITELN;
END.
```

[Output Omitted]

Appendix 3 – KEYBOARD, LATCH AND RAM ADDRESSES

[Text Omitted]