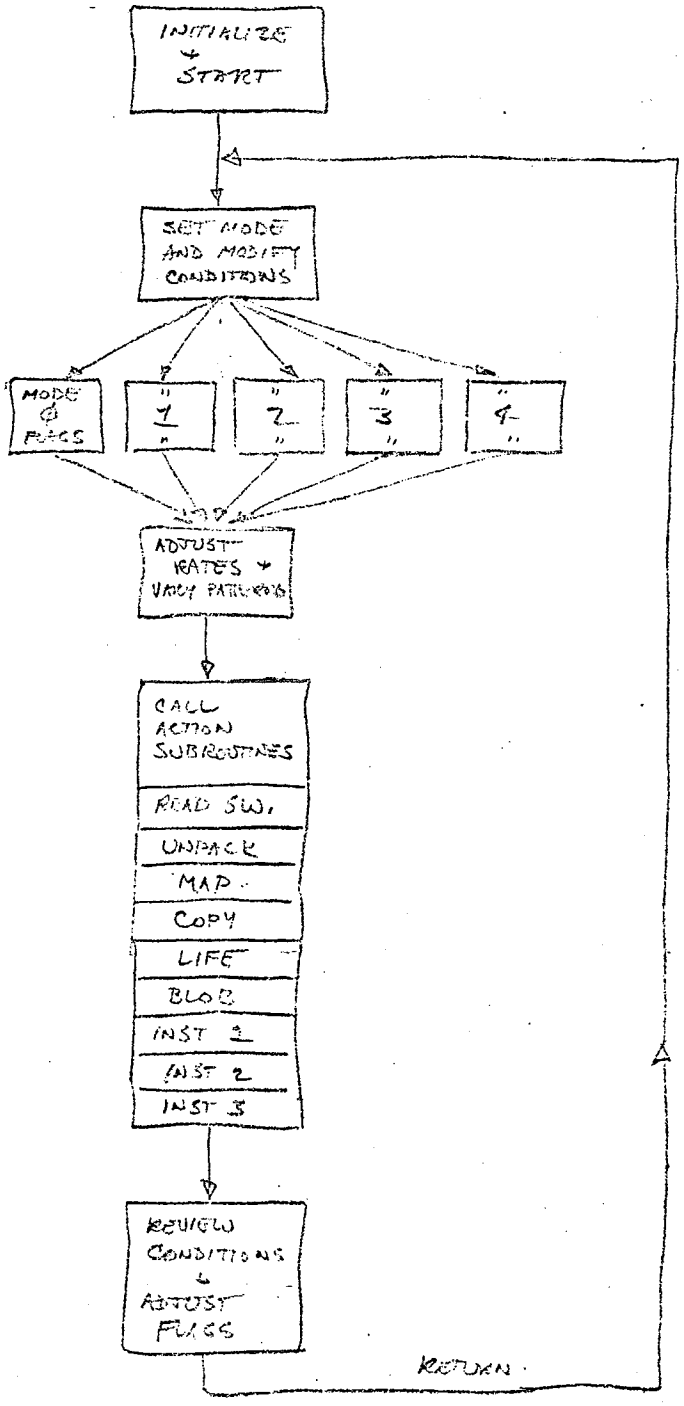
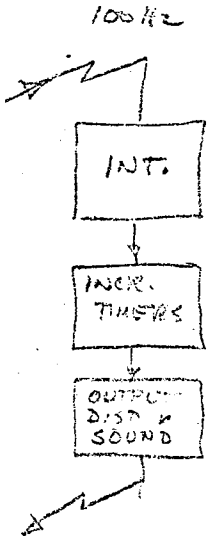
SEATTLE SOUND-LIGHT SYSTEM.

SIMPLIFIED PROGRAM FLOW.





DISP
RESON

I/O INSTRUCTION EXAMPLES

SYSTEM DEVICE CODE = 76

CLEAR DISPLAY :

```
SUBZR   0,0
MOVOR   0,0     ; SETS AC0 ← 1400000
DOCP    0,76    ; LOADS C REG + PULSES
                  CLEAR LINE ( NIOC)
```

LIGHT A LAMP :

```
SUBZR   0,0
MOVZR   0,0     ; SETS AC0 ← 0400000
DOC     0,76    ; LOADS C REG AND ENABLES
                  STROBE 6 (LIGHT)
LDA     1, LAMP ; LAMP ADDRESS : BITS 4-7 = X
                                 BITS 10-16 = Y
DOA     1,76    ; LOADS A REG AND PULSES LOAD
NIOP    76      ; BUS, LAMP STAYS LIT UNTIL
                  NEXT CLEAR
```

DO NOT COMBINE

PULSE A RELAY :

```
SUBO    0,0     ; AC0 ← 0
DOC     0,76    ; ENABLE STROBE 5

LDA     1, RELAY ADDRESS ; RELAY ADDRESS =
                           1___XXX
DOBP    1,76    ; OUTPUTS
                    BITS 1-5
```

* NOTE, WAIT AT LEAST 100 μS BEFORE REPEATING AS STROBE 5 HAS BEEN STRETCHED

OUTPUT A SOUND COMMAND :

```
SUBZR   0,0     ; AC0 ← 1000000
DOC     0,76    ; ENABLE STROBE X

LDA     1, SOUND ; BITS 1-5 = 0___
                   BITS 6-15 = DATA TO D/A
DOBP    1,76    ; OUTPUT.
```

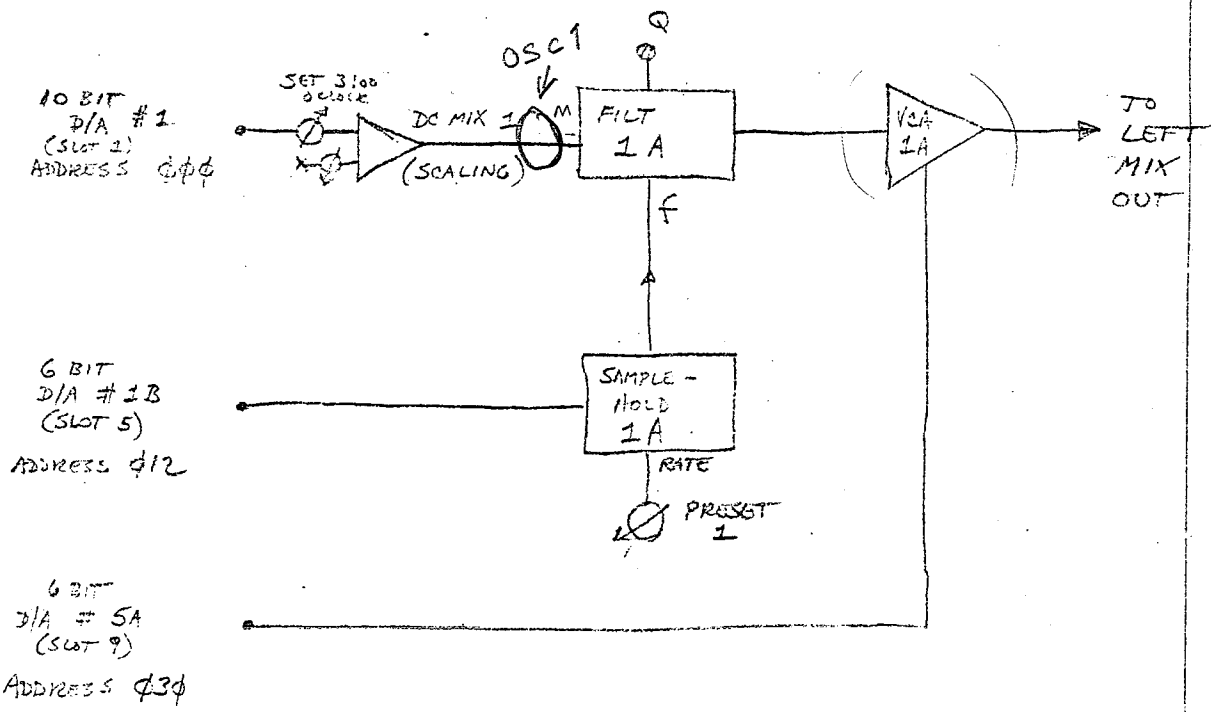
INPUT : SEE .LOC 1000 ff for EXAMPLE.

A REG BITS 13-15 MUST BE LOADED 0-3 for SWITCH WORDS 0,3

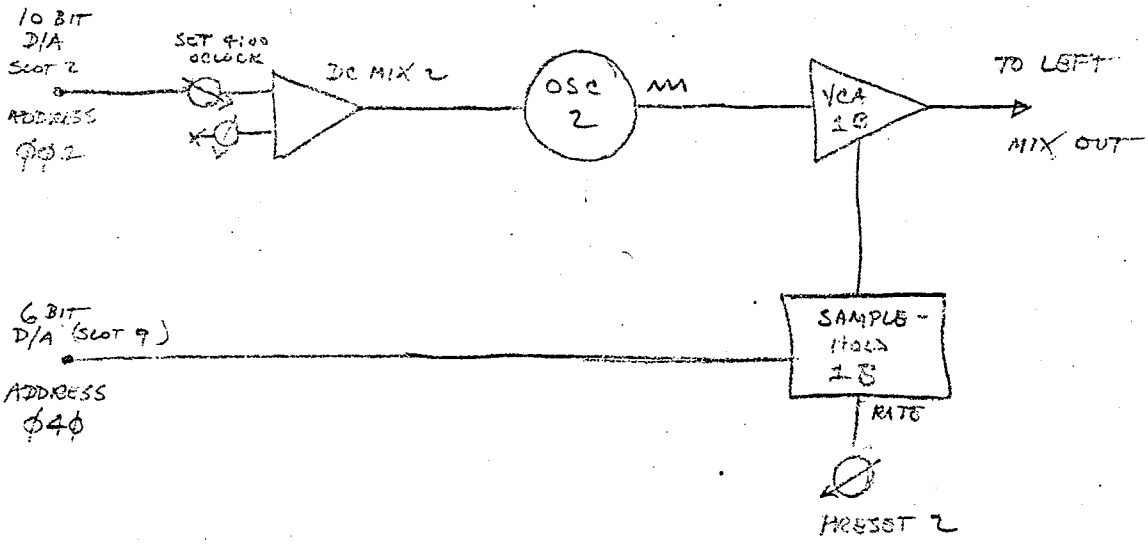
THEN DIA AC,76 INPUTS ENABLED WORD

INSTRUMENT 1

BLOCK DIAGRAM

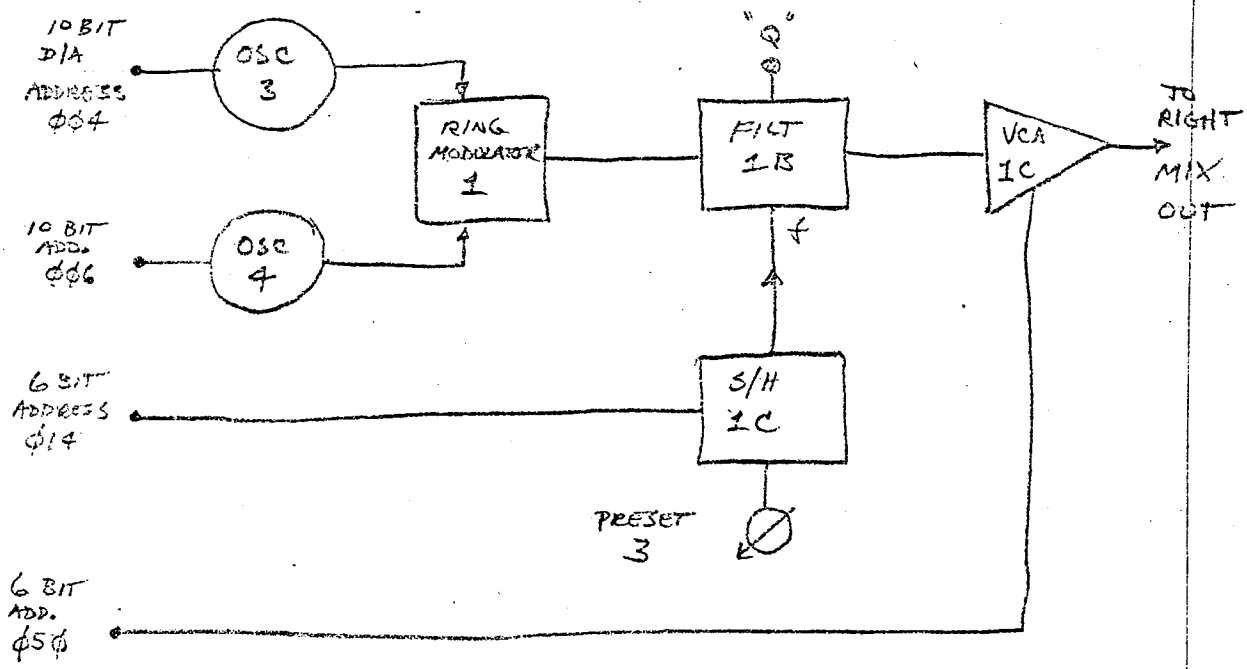


INSTRUMENT 2
BLOCK DIAGRAM

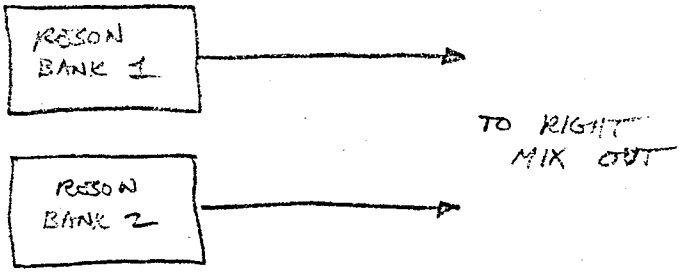


INSTRUMENT 3

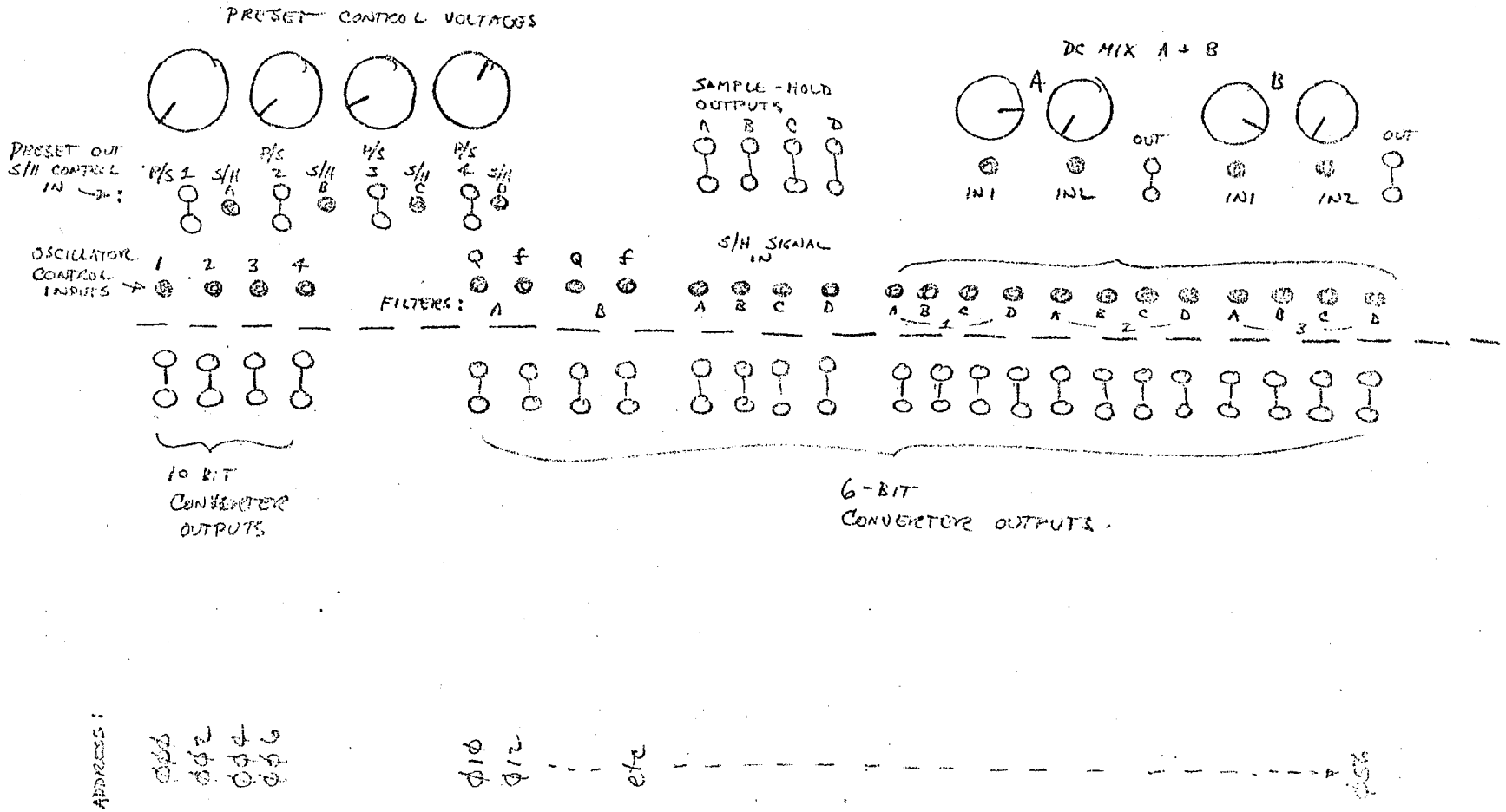
BLOCK DIAGRAM



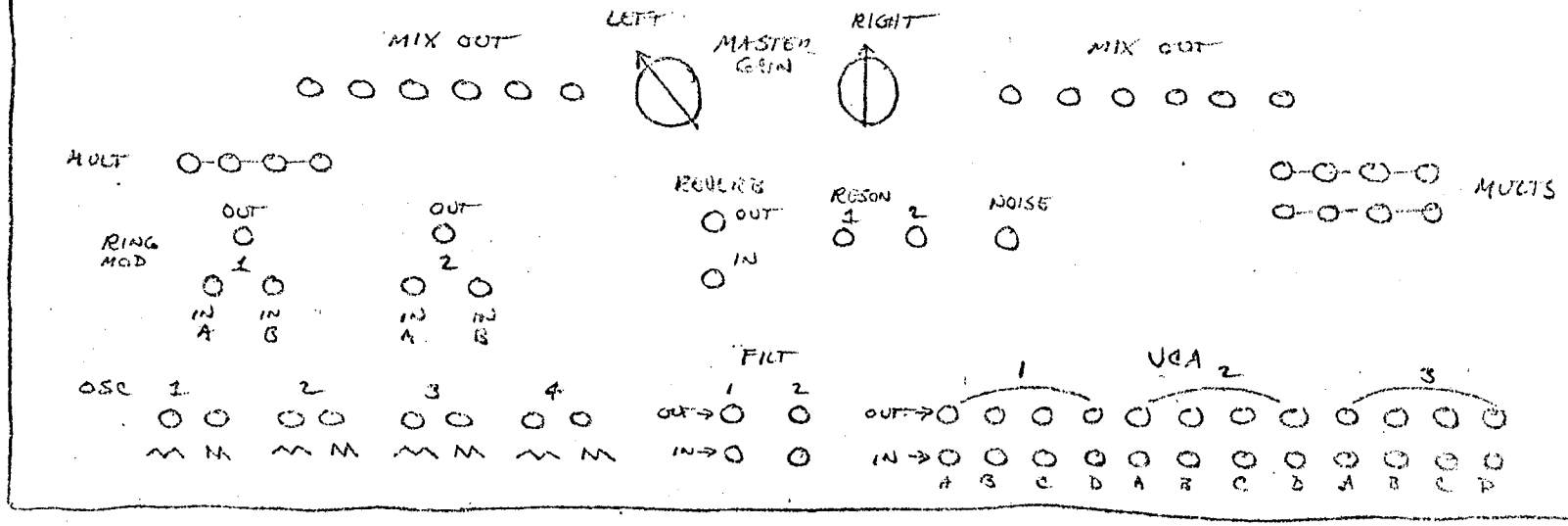
RESONATOR.



CONTROL VOLTAGE PATCHING



AUDIO PATCH PANEL

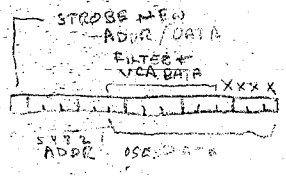


INSTRUCTION MANUAL PART 1

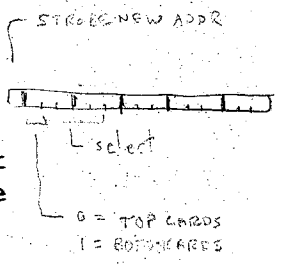
The following is a description of the Test Programs and a discussion of their operation.

The Test Program is a completely self-standing program which may be loaded with the other main program parts and will co-exist with it in core. Its starting location is 16ØØØ. The first routine is used to light the entire display. The program then halts. Pressing "Continue" will then initiate a scan by horizontal rows going upward through the entire display. The program serves as a very severe test of the display since the initial lighting only addresses each lamp once. Therefore, if one can observe that any lamp does not light, there may be a dynamic problem with the appropriate lamp-driver card and not necessarily a static problem that would be obvious in normal running. The rate of scanning can be varied by changing the number stored at location KDLY.

The second routine begins at 16Ø5Ø and is used to test the converter and audio units. The format of a sound output instruction consists of six left-most bits which are the address of the device. The ten right-most bits are the data content of the output word. In the case of the sound addresses, bit Ø, the left most of the left six bits will always be a zero. One additional note, in the case of the six bit D/A converter addresses, only the six most significant of the right hand ten bits of the output word mean anything.



To use the program, set starting address 16Ø5Ø and press start. Then set the address of the desired sound address in the bits one through five and the data in bits six through fifteen. Raising bit Ø switch will cause that word to be output so that one can go through a series of addresses loading them with a pattern of data in order to observe the results.



The third part of the test program begins at location 161ØØ. This routine pulses the resonator addresses. The resonator addresses are the upper most 32 of the 64 possible sound addresses, accordingly, bit Ø of the address is always a one. In this particular routine, bit one is set by the program itself so there is no need to do it with the switches. This was done in order to permit use of bit Ø as a load switch in the sound address test, the previous test. Operation of this routine is done by starting at 161ØØ, loading a resonator address in bits one through five, ignore bits six through fifteen. The resonator will then be pulsed at a rate dependent on the number and location RT.K. The number at this location times .2 seconds equals the rate of pulsing. The initial loading of the program sets its value to 3 so that the resonators will be pulsed at a rate of about once each 6/10th of a second.

The operations of the program are exceedingly straightforward, almost trivial, so I won't go into any detail as the actual coding is perfectly obvious from the listing.

Discussion of the Life Subroutine

The following discussion will be in the form of a guided tour through the listing of the life subroutine. The general operation of the life game should first be considered. This is a mathematical game which is based on two principle rules. It is played in a cellular space and a cell is said to be alive or dead. A cell has eight possible neighbors, the orthogonal and the diagonal neighbors. The two rules are: if a cell is now alive and has two or three live neighbors, it will remain alive; the second rule is that if a cell is dead but has exactly three live neighbors, it will come alive in the next generation. The game, therefore, goes generation by generation. It gives rise to a very wide variety of patterns which are stable, some oscillate, some propogate through the space, some grow almost without limit, some behave in very peculiar ways. It is a highly suitable routine for generating visual displays and that is the reason it is used here, although there is one fundamental change in the rules in order to prevent a too-dense growth of patterns. The sub-routine is called each time the main program loop is run. It is only carried out if there are bits in a list called the live list, in other words, if there are coordinate pairs listed and said to be live. NLive is the total of the live list. If you go back to the previous discussion, the copy routine, copy is from a temporary list which represents lamp coordinates that have been formed from switch input. This temporary list is copied into the live list and NLive set to equal the total. Therefore, entering the life subroutine, the first thing that occurs is the initialization of all the lists. Then NLive is tested for nonzero and if it is Ø, the subroutine exits.

The first thing done is that each live coordinate pair is taken and examined by generating eight offsets from it. Each offset is then compared with the original live list looking for identity. Every time that an identity is found, the conclusion is made by the program that the original bit had a neighbor at that location. A running total is kept of these neighbors. This sub-routine can be called eight times. In the sub-routine in which the offsets are performed, this by the way is called the L Test subroutine and it is called under the routine LEACH. Now I'm going to go to the L Test subroutine before coming back and finishing this first routine since it calls its own subroutine.

Two possible conditions can occur while in L Test. It can either find that there is a neighbor, the offset cell, or that there is not. If there is not a neighbor, the sub-routine has to then decide whether or not it is a potential birth location and the way this is done is that the offset square which did not find an identity in the original list is then tested by offsetting it the eight possible ways or at least enough times until you find that the first offset square has too many neighbors to make it able to be born. In other words, if the first offset square were not alive to begin with, which is found out by the first subroutine, you then look at its neighbors in order to see how many are in the live list or how many of the first offsets are alive. The subroutine will stop running once it bumps out at a total of four because that conforms to the rule that if more than three are alive, it will not become born. If a total of four is not found, the subroutine will be run eight times. This is the B Test sub-routine will be run

eight times after the program returns back into the body of the L Test subroutine. The total number of found live neighbors will be examined and tested to see if it is less than two. The result of either of these tests is that if the first offset location had less than three or more than three live neighbors, the program goes to DSGA in which that location, that coordinate pair, is listed in the abort list (the A List). If it had exactly three neighbors, it is listed in the birth list (the B List). A listing in the B list exits to list also in the abort list. The abort list is kept in order to know the number of squares that cannot become born and once a square has been marked to be born, or found to have too many or too few neighbors to be born, the result is the same in both cases - you don't want to waste time testing that square if it should be the neighbor of another in the original live list, in order to make the program as efficient as possible. After these so-called B-A tests have all been run calling the B Test subroutine as many times as necessary and then making the determination, the program returns back into the original L Test subroutine. Once the original full set of eight calls of the L Test subroutine has been run, the results are evaluated in terms of how many neighbors the original square had. This then leads to a designation that that square may be killed if it had less than two or more than three neighbors. The killing is done by making an entry on a third list - the K List - but this entry is the address of the original coordinate pair in the live list. We are going to come back later and kill coordinate pairs by loading birth coordinate pairs on top of them, so it is important to know the distinction between the live list, the abort list and the birth list, which are all lists of coordinate pairs, and the kill list which is a list of addresses. In this routine the L Test is finally concluded and goes back to the main program which then sees if there are any more in the original live list and if there are, it goes all the way back to LEXAM where it then runs again the whole series of L Tests, calls and the possible B Test or the B A routine with its B Tests in the L Test subroutine.

Once the original live list has been completely run, the first section of the life subroutine is finished, the program goes then to CINIT. This routine begins with an initialization of all the lists and then the overwrite procedure is started. In this case determination is made of how many are to be killed and how many are to be born. The program branches in such a way as to guarantee that you will first of all kill locations in the live list by means of the addresses in the kill list by overwriting entries in the birth list on top of them. If you run out of locations to be born while you still have locations to kill, the program jumps to a routine called GARB for "garbage". The cleanup of the list routine in this instance sets a pointer to the bottom of the live list and entries are taken out of the bottom and transferred to overlay on top of the remaining kill addresses until all of the list has been cleaned up and all the rest of the locations to be killed have been killed. If you run out of locations to be killed before you run out of births, the births are then added on to the bottom of the live list.

The final section of the life sub-routine is update 2 (UPDT 2) in which two important things are done. First the lists are initialized and a test is made to see if they are any left in the live list. A good

deal of the time the life pattern will be computed to have died out and NLive will be Ø so it is necessary at this point to set flags which affect the operation of the blob subroutine so that it will begin the transition into the blob state and wait until another pattern has been called on the switches. Once this flag setting occurs, if it occurs, the program exits. If NLive was nonzero, the transfer routine LLP is entered, in which the live list is copied into the display list. The assumption has been made that most of the time life is in operation the blob routine will be on the way out so that the occasional conflict caused by moving the pointers around for the display list and thus disturbing the blob list will just have to be lived with because the focus of the viewer's attention is going to be on the pattern he is creating. In effect, the live list is copied into the display list, the blob list, if any, added and the totals are set and the life subroutine exits back to the main program.

The following is a discussion of the subroutine which reads the touch plate switches.

This subroutine beginning at location 1ØØØ is run each time the main program loop is run. The manner in which the switches are read and the decision made as to whether to unpack them or not is as follows: A word is read into an accumulator, another accumulator is loaded with the previous value of that word, one is subtracted from the other and if there is any difference, a flag is set. All four words are then read with current values being loaded into the locations CSWØ, CSW1, CSW2 and CSW3 after the comparison has been made. If at the end of this routine any word was different from its previous value, another accumulator will have been set and will constitute a flag which will indicate a difference in the setting of the switches has occurred and the program will then decide in the main program loop whether to call the unpacking routine.

The next routine beginning at location 11ØØ, the next subroutine, is the unpacking and listing in the form of coordinates. This routine is operated in the following manner: Pointers are set for the two lists, two lists are kept - the current switch list and the previous switch list - at the moment we are not really using the information in the previous switch list for anything but at the time the routine was written, it seemed like a good idea to have the information available. The transfer is not accomplished by copying but simply by switching pointers so that the routine begins by setting the pointers so that the current switch pointer now points to the old switch list and not the one that is going to be loaded. A word is then gotten, CSWØ for example, and the unpack subroutine is called. This subroutine operates by using a floating 16 bit accumulator which is either incremented or it has a constant added to it in such a way as to increment the least significant bit of the left hand byte. If you remember from the hardware description, the format of a coordinate pair is uses left and the right bytes and a meaningful coordinate pair involves the four least significant bits of the left byte as "X" and the six least significant of the right byte as "Y". Any other bits are simply lost

in the output since it is not possible to address a lamp at any other location. Carry-outs cause the display to wrap in X and/or Y. The unpack subroutine shifts left each bit in the original 16-bit packed word that comes from a given set of 16 switches and everytime carry is found to have been set, the current value is loaded in the list. The floating accumulator is incremented for each shift, and each time it has been incremented through 8 shifts then the constant is added to increment the X byte. Assuming that there were 16 bits all set, 16 entries would have then been made in the list. Finally after this unpacking has been done four times, the list will then consist of coordinate pairs taken by copying the current value of the floating accumulator at each location of the auto index so that the list now consists of formatted coordinate pairs in what will be called lamp coordinates. Outputting one of these listings would cause a lamp at that particular location to light. The initial treatment of the switch panel is that its origin be coincedent with the origin of the display. That is, the lower left corner of both would correspond but at a later part in the program, a mapping operation is done so that the 64 bits of the switch array can appear anywhere in the lamp array.

The next routine does this mapping. It is called DMAP 1. This is a very simple routine which merely transfers the listings in the current switch list into a temporary list adding a constant to them at the time and then masking the sum in order to eliminate the superfluous carry outs from the X and Y bytes. The value of this constant KMAP is controlled by another part of the background program and at the moment it is set to a random value each time the mode changes.

The next routine "copy" accomplishes the actual copying of the switch list into the live list. The reason for having the copy subroutine at all is simply because a great deal of confusion would occur if bits were put into the live list that were already alive so that the copying is actually a comparison and then a copy. Each bit in the temp list is used to search the live list to see if it is already there and only if it isn't there is it added onto the end of the live list. Without this, the live list would grow uncontrollably, choking up the life subroutine.

The following is a discussion of the interrupt routine. The interrupt routine is a portion of the main program that is probably the most crucial of all in the successful operation of the piece. Only by having the interrupt run regularly at a time controlled by the clock in the computer, is it possible for events to seem to take place at a fairly consistent rate of speed. The clock, in other words, controls the viewer's perception of real time and those events which are desired to take place at a rate which does not vary because of the load the program has to compute are assigned to the interrupt. At the beginning of the interrupt routine, the first thing that occurs is that the accumulators and carry are saved. Then a number of timing locations are either incremented or decremented according to what the requirements of the background program will be. The first of these TIM1, TIM2, TIM3 and TIM4, are used to control the functions which occur later in the interrupt program. DTIME is a timer which controls the rate at which the map subroutine will choose another

place to map the switches into the display or another offset. STIME is a free running timer which is looked at in order to determine if it is time to output sound this particular interrupt cycle or not. The sound routine is only run once every eight interrupts. X time, Y time and Z time are times used by the blob routine to control the translational rate of the blob, the rate at which its direction changes and the rate at which it grows and dies.

After the TICK routine at which the time is updated has occurred, the first thing that is done (this is done every interrupt) is that the display is output. The display is output by first of all clearing the whole display and then outputting the current display list.

This is as good a place as any to discuss the technique of input/output instruction handling. In the case of display, it is necessary to set an accumulator to 140000, output that into the C register on the I/O Board in the computer. This will enable the display clearing to occur. This is accomplished by a P pulse (the instruction is a DOCP AC,76).. This generates the clear pulse to the whole display. Then the lists are initialized or if there is nothing in the display list, an exit is made down to the resonator routine. If there is something in the display list to output, the C register on the I/O Board is then set to enable L pulses, light pulses. That is done by setting an accumulator to 040000 and issuing a DOC without a P, DOC AC,76, followed by the loop in which each lamp coordinate pair is output. A very important point here is that there is a good deal of propogation delay in the logic so that the DOA AC,76 which outputs the lamp coordinates has to be followed by an N10P 76 so that the actual P pulse which lights the lamp will not get there until after the logic has settled down in the state that address the proper lamps. When this is completed, the interrupt then moves on to the reson routine. In all input-output operations the "76" is the device address of the whole light-sound system.

We've now reached the part called the Reson Routine. In this routine the resonators are called output, and the timing is controlled which regulates the duration or the time that must elapse before another resonator will be output. Now, the way this works is best understood by considering the format of a listing in the resonator list. The left hand six bits will be the resonator address and in order to trigger the resonator all that is necessary is to address it. The 10 rightmost bits in the instruction word are therefore meaningless as output, and in this case are used to contain a number which represents the duration of that particular resonator event in interrupt cycles. This number is put into RTIME at the time Reson is on and will then be counted down at the rate of the interrupt, and when it bumps out at 0, the next listing will be extracted from the resonator list and output. there is a resonator modifier routine which is controlled by a flag that is set by the main program. This provides for the following servicing of the resonator list: The word is taken out, and if the flag is set, the duration part of the word is temporarily held while the previous duration is looked up from its buffer, and if the flag is set, the decision is made to use the previous duration. This previous duration is not necessarily the previous event, but it may be the previous

one left from the last time the list was run - the last time the resonator output was achieved. I should note at this point that the resonators are coupled to the switches by the following means, (by software means). Whenever there is anything in the live list or when NLive is nonzero, NLive is actually RFLAG - it's not actually RFLAG, but NLive is copied into RFLAG every time the main program is run, so that when anyone is pushing the switches down, or as soon as bits appear in the live list RFLAG is going to get set, consequently the resonator part of the routine will be activated. If MRFLG, which is the modify resonator flag happens to be set at the same time, what happens is that the duration part of a resonator command is stripped and held and shifted by one so that the list is constantly shifting its duration arguments and its address arguments. This causes a fairly short resonator list to have its effect multiplied and many many possible combinations of rhythmic patterns will occur. The operation of the routine is perfectly straightforward with those considerations in mind. RSHFT is the little routine that does the swapping. ROUT sets up the C register on the Computer I/O Board with Ø's so that a DOC is output to set up with and a DOBP will then pulse the resonator. Now, one final note: this routine guarantees that bit Ø of the output word is always set so that if for any reason a resonator address comes up without a one in the first bit, the routine will supply it. In the sound routine which is, as mentioned before, only run every eight times the interrupt is run, you look at the current value of STIME, masking out all but the last three bits, and only proceed if the last three bits happen to be zero, which will occur once every eight times.

Now, the sound routine is a fairly long series of routines run depending on whether or not the function flags are set. The function flags are set by the background program in the instrument routines, instruments one through three. Instruments one through three control the timing of sound events for themselves and only set their appropriate flags when it is time to begin a new event, thus communicating to the interrupt program the necessity of updating the value of the function. Now there are four function routines which are identical and they are run successively if the flags are set, (actually function four is a dummy since there is really no function four routine in the program, but it is there in case in the future it might be desirable to have another function). They are called functions because they represent, in the ultimate sense, a voltage function of time, which is produced by the synthesizer, and these functions will in practice all be ramps, either up or down. The function routine is a general purpose routine. It can be handed a base value and a displacement, and a number of steps to carry out the servicing of the arithmetic addition or subtraction of the displacement to the base value, so that the function routine does not really care what the values are, it just is handed them by the background program and the flag is set. It then begins to operate on them. The values will actually be output by the interrupt routine, but the interrupt routine does not control the settings of new pitch or filter settings or any of the other data that instrument one through three routines control, but only the time dependent servicing of the values of the so-called envelope functions which are their prime reason for existing.

Now let's take a look at just one function routine in detail and see what happens. First of all, you look at the function flag and see if it is set. The function is really dependent on the displacement. If the displacement is one, the ramp will go as slow as one value out of the 1Ø24 possible values that the ten-bit converters can put out, and at the rate the interrupt runs, that takes about 10 seconds. This, of course, would be with a full or zero value of the base value. The case of function one, F1BSE is the base value. F1DSP is the displacement. F1DSP is a signed number, and if it is plus it will increase the base value, if it is minus, it will decrease it. F1TOT is a decrementing location which is loaded with a number which represents the number of times the displacement is to be added or subtracted. In the case of instruments one and three, the ramp functions are all automatic - they start at a level and decay downward. So that, just to make it simple, every time an event is initiated and the function flag is set, a value of F1TOT is set which is sufficient to guarantee that the decay will run its full course. In the case of instrument two, since instrument two can generate up and down going ramps, this will be discussed more fully when they talk about the instrument two routine, but instrument two will take the displacement value and divide it into the change or the difference between the new base values in order to come up with an approximate number of steps, and that number will be handed to the interrupt routine in the form of F1, or in this case, F2TOT. At any rate, the function routine has one very important task - it checks F1BSE for overflow or for zero. If F1BSE ever becomes negative after the subtraction of a displacement, it will just simply set itself at zero. If it ever overflows, as shown by a carry out into the eleventh significant bit, it will set itself to maximum and turn its own flag off so that the thing will not waste computing time once a ramp has gone as far as it can go. Function two, function three and function four are identical with the exception of function four, which is really a dummy. When this is concluded, the accumulators and carry are restored, the clock is reset, the interrupt is turned back on and the interrupt gets back to the main program by an indirect jump through zero.

The following discussion is a description of instrument one subroutine. Instrument one subroutine is run as called by the main program and is dependent on the mode control part of the main program. In the case of instrument one, it is actually run all the time since it is always part of the five possible modes of operation. Instrument one has two primary things to do. First of all, it has to set new pitch and filter values and it has to extract the data from the envelope list and prepare it to hand to the function routine which is in the interrupt. It then calculates an event time which is actually a subfunction of the envelope information and uses this event information to determine when to actually execute the new event, or when to change the pitch and the filter setting, then at the point initiate the setting of the function flags so that the next time around, the interrupt will output the change in envelope conditions. Just to go stepwise through instrument one, the first thing you look at is if instrument one flag is set. Then you look to see if TIM1 has gone negative.

Later in this routine, TIM1 will be set with the value of the event time, and it is decremented down by the interrupt. This controls when the next event will occur. Instrument one is unique in that it determines whether or not modification routines are run on the list data. In the main program loop, there is a flag which has four states and this flag controls which modification will be run. The first time instrument one is run after the flag has changed state. QMOD will detect that it has changed and will send you to a routine which will load MSRPT with the pointer that points to the appropriate one of three possible modify subroutines. In the case that M flag one is zero (the modifier flag), this is the case in which no modification is done. The three possible modifications, A Mod, X Mod and R Mod, are as follows: A Mod is a simple addition of an offset which is taken out of the modification list. X Mod is a process of forming the exclusive OR of the data from the modification list and the data from the various other lists. R Mod is adding a random number which is formulated each time to the data from the other lists. The first list processing that occurs is the processing of the modify list. This is typical of all the lists, so we will consider it in detail.

This particular technique is used with all the lists, so we can put it in detail. First of all, each list has two pointers that point to the beginning of it. They are intitially set. In the case of the M list, it begins at 5700. This is slightly different from the others in that it is a list of only 100 octal locations. The other lists are ~~lists are~~ lists of 400 octal locations. MPT1A initially contains 5700 and so does MPT1B. The routine loads a mask which will mask carry outs into the eighth most significant bit, loads the list pointer, increments, and masks it. Then it loads another accumulator with a number called MOVFL in this case, which is an overflow indicator. It then negates the value of the masked pointer, adds it to the overflow number and looks to see if the sign bit is set. If an overflow occured, the value of the pointer is set to zero, and now we add MPT1B to the zero'd or otherwise checked operand part of the pointer, and then put that value in MPT1A and load the data out of the list by an indirect LDA through the pointer. Now this fetches the next entry in the list, and the list will then circulate running up to the overflow and resetting at zero, actually using the first value instead of the zero value that the pointer is initially set to. This procedure is used in instrument one, first of all to fetch out the modify instruction. This modify part will only be run in instrument one. The next step would be to run the oscillator list and get the next list out of that. In this case we use a slightly different format, since we are running a 400 entry list, 400 octal, so the mask is different, but otherwise the same as the modifier. The next list servicing is the filter list, in this case, filter frequency one - FILF1. The sound system is not set to use controllable "Q" of the filters, but only the frequency. The "Q" is controlled by setting a trimmer. In the case of oscillator one and filter frequency one, once the data from the list has been fetched, looking at the listing at Location 3062, PPT1A has been addressed indirectly in order to get the next entry in the list. Now we look at the mod flag. If the mod flag, M Flag 1 is set, we go to the MSR pointer, which has been set to one of three possible modification subroutines. When you come back, the data has been modified. It

has been masked again, to get rid of any extraneous bits other than what's in a valid argument - the right hand ten bits of the word, in this case. Then the PAD1 location is fetched. This contains the address of the particular function that you are working on, in this case, oscillator one's address is zero zero zero for the first six bits of the word and that is what the value of PAD1 is. That is added to the argument and it is put in PNEW1 holding register for the new value of the pitch function. Likewise, filter frequency one. Envelope one is somewhat different. The envelope data is kept in pairs of words in the list so that the list will first fetch one word, then increment the pointer and fetch another. It is important to remember that EOVFL, which is the overflow indicator for the envelope list, has to be an even number, since it is very important not to get the pairs staggered, since the data in the list is set, knowing that one word will be split into two particular arguments and the third one will be used to control the duration, so that the second word will be a third argument which will control the duration. Otherwise, this is virtually the same procedure as before, with the exception that the first word taken from the list is separated into the right hand ten bit section, and the left hand six bit section. The left hand six bit section is interpreted by instrument one as an unsigned number which is the value of the displacement. The displacement is considered always to be negative, since instrument one (and this is true of instrument three as well) will execute ramp functions that begin at the initial value and go down. Instrument two is different in that this six bit portion of the first envelope data word is a signed number. The sign bit determines whether it is a plus or a minus displacement. We'll get to that in a minute. With instrument one, this six bit number is shifted all the way to the right hand part of the word and then the whole word is negated, so the displacement is a sixteen bit word, but its value will only be six significant bits. The original right hand ten bit part of the first word is going to be the new E base value, and it is combined with the value EAD1 which is the address of this particular convertor that is servicing envelope function one; the two portions are added together and loaded into ENEW1. The event time is determined by incrementing the pointer, getting the next data word out of the list, masking it with a mask which limits its value. This mask can be set by the background program at various times in order to provide a correlation between fast movements of blob and relatively fast movements of the sound articulation. This masked number, which came out of the list, is loaded into TIM1 which is used to determine when the next time instrument one is called it will actually be run. This number will have to be decremented down by the interrupt in order to go negative, at which time instrument one is capable of running. This is an attempt to let the interrupt control the rate of the instruments. Obviously, if the program is heavily loaded, the interrupt will probably decrement these values down to the point where they go negative before instrument one gets around again, if they are fairly low values. But, with moderate to large values of duration, there will not be any noticeable loading until the program is really almost completely loaded down with life activity, and at that point, there is so much more visual interest going on that it probably won't be noticed.

Event one, then, generally speaking, determines when instrument one will get to do another event. In other words, when it will get to service the pitch in frequency and envelope list again.

The final point of this routine is FSET. This is setting the function values, and this is all done together because it has to be done with the interrupt turned off in order to prevent accidental changing of values of things in the middle of an interrupt, or having an interrupt get in while you are trying to change values, (while the background program is trying to change values). So, it is all compressed together to take place in one series of instructions. An accumulator is set to 1ØØØØØ, three other accumulators are loaded, the interrupt is turned off, the sound output is enabled and the first three sound outputs go out, then the remaining necessary are to set the FlBSE, which is the new base value. In the case of instrument one and three, these will be values which are simply set and then begin to decay downward at a rate controlled by how the size of FlDSP, which is the displacement, in this case a negative number that was taken out of the list, and ETOTl in the case of instrument one and three, is going to be 2,000, which is a large enough number to mean that the greatest possible number of steps in the graph will not count this number completely out. Then the flag is set telling the interrupt it now has a new envelope ready to process, and the subroutine returns.

In the listing following instrument one, are found the modification subroutines. A Mod is simply adding two numbers together and returning with the sum. X Mod is the standard data general exclusive OR routine, entering the subroutine with the two numbers and returning with the exclusive OR in an accumulator. In this particular case, I think it is accumulator zero. R Mod takes a look and RANDP, which is the location on page zero, which is continuously furnished with a changing random number. There are actually a number of means of doing this, and the program can be patched to do either. It might look at the current value of location zero, which is pretty close to being random, or it might look indirectly through zero. There are a number of possible routines in the main program, which, at this point, have really not been evaluated as to which will produce the most satisfactory random number.

I would like to talk about instrument two subroutine in the respects in which it differs from instruments one and three. This subroutine, in all other respects, is identical to instrument one, except for the manner of computing envelope function arguments. In this case, a rule has been defined that if an examination of the data that comes from the envelope list shows that the displacement is a negative number and the new face value, after being compared with the existing base value (and that is the one that exists at the moment this routine is run, since that's a value that is constantly changing by the interrupt), if this difference is positive, the data will be interpreted as a setting of maximum and a ramp down at the rate of as many increments as displacements as are necessary to get it down to the new value. If the displacement is up and the difference is less, the new value will be set to zero and the ramp will go up as many steps as necessary to get to the new value. If the displacement and the difference have the same sign, the ramp will go from the current base value to the new value in the proper direction at a rate dependent on how many steps of displacement are necessary to get there plus or minus on an error of one unit, which will be indistinguishable out of the 64 possible volume levels.

To recapitulate: a data word has been taken from the envelope data list, and, incidentally each instrument has its own pointer in the case of envelope. There is only one pitch pointer and only one modify pointer and only one filter pointer, but there is an envelope pointer for each instrument. The reason for this is to make the data in the envelope list not always coincident, so that effect of rhythmic counterpoint will be achieved. At any rate, a data word has been taken, and separated into six bits on the left as the displacement, and then ten bits on the right as the data. In the case of instrument two, this displacement is a signed number, and the routine QSIGN determines what the sign is in the process of shifting that six bits all the way to the right of the word.

The next routine WVG stands for "Which Value Greater", and refers to the base value part of the word. First of all, the current value of function two, F2BSE, is fetched, and the base value that comes out of the data list will be a positive number, so that the difference is now obtained. The difference is gotten and also by means of two routines in the case in which one is bigger than the other, so that carry gets set in the process of comparing the numbers so that at the end of the routine, we know what the absolute value of the difference is, whether it is plus or minus. In other words, if the current value is greater than the value that has just come out of the list, we call that the negative difference. If the current value is less, we call it a positive difference. Then, in the process of these two routines, WVG and CVG, it is decided that if the difference is plus, but the displacement is minus, you go to NVMAX set the value at the full and ramp down to the current value. The corrolary to that is if the displacement is plus and the difference is minus, you set the value to zero and ramp up to the current value. And these ramps go at the rate of the displacement, and in the direction of its sign. If you go back you'll see that if it is a minus displacement, you are going to ramp down, if it is a plus displacement, you will ramp up. In the case where the sign of the difference and the displacement is the same, the IDIV, or Integer Divide Subroutine is located at the end of instrument two. It is a standard Data General division subroutine. You go to it with the divisor being the displacement, or the absolute value of the displacement, not the signed value. The absolute value has previously been computed and put in the location called DISPL. That's the divisor, and the dividend is the absolute value of the difference between the current function level and the new base value that came out of the data list. You return from that subroutine with a number which represents the number of steps that the interrupt routine will service the data - will subtract or add the displacement to the current value each time the interrupt is run. This determines the number of steps in the ramp, and the value of the steps is the value of the displacement. These two possiblities exit to a routine called NOENV - no envelope, and in this case the quantities are organized and copied over into the registers where they will be held to their output. Event 2 is just like it would be in instrument one. It takes the second data word and uses that to determine the duration, or the time that must elapse before the next event of the instrument. Then FSET2 routine outputs the pitch data. All this instrument has is an oscillator and envelope data, so it outputs those, and it hands the interrupt routine the numbers for the function to process. That's the end of instrument two.

Very briefly, instrument three is virtually identical to instrument one. The only difference is that it has two oscillator functions - two pitch numbers are prepared. In this case they will be sequential pitches, or they will be sequential listings in the pitch list, so it ought to be apparent that the numbers in the pitch list have to be pitches which can sound concurrently as well as sequentially, and this is borne in mind in deciding on them. Envelope three routine in this instrument three is exactly the same as in instrument one. That concludes the discussion of the instruments.

I would like to turn now to the beginning of the program, the main program loop - the main executive program which is found at locations 0400. This is the starting point of the program, and the first series of instructions is an initialization routine in which virtually everything under the sun is set to zero, or else set to one, in the case of certain values that have to count down - (that are decremented down and would cause problems because of skips if they accidentally started at zero). The next thing that is done is that real time clock is set. The real time clock is controlled by the number in the location FREQ - this has to be a 2 in order to produce a 1ØØ cycle clock rate. The interrupt is then turned on and the main program loop begun. The first thing you do is copy NLive into RFLAG - this is the way in which the life routine is coupled to the resonators, because as long as life is running, as long as NLive is nonzero, the resonator routine will be run in the interrupt.

The routine, continuing at 473-5ØØ is a timing routine which controls the modification and mode control flags. There are five possible modes of operation of the main program, and four possible modification stakes so that permutation of that gives you many effective possible modes that will be run before repeat. These routines are very straightforward - the one is simply a way of counting up to four and resetting, and the other a way of counting up to five and resetting. There is a series of skip tests which branch you out to the different modes at this point. A typical mode, mode zero, is merely a sequence of flag settings. The instruments are turned on and the modification of resonator flag is set, or in the case of mode zero, instruments two and three are turned off and instrument one is turned on. Mode one, instrument three is turned off, instruments one and two are turned on, and the resonator modification is turned on and so on. This is able to be changed easily as there are a lot of dummy locations for flag settings.

After one of these five possible routines is run, the exits all come together into EMODE. EMODE first of all computes a random setting of the speed of the blob. It services RANDP with a new random number which it sets up as a new mapping of switches into the display, and then it exits to the routine called CALL. This routine is the beginning of the series of subroutine calls, which is the heart of the main program. Most of the subroutines are self controlling. If they have something to do, they'll do it. If they don't, they know and exit. There are one or two exceptions. In the case of Read Switches (RDSW)...Read Switches is always done, but the output of Read Switches is a setting of UPKFL, the unpack flag, and only if the unpack flag is set, will Switch List (SWLST) be run. Then BMAP is run, COPY is run, LIFE is run. Then a JSR@GPT is run, the blob routine. The next three routines are Instrument I, Instrument 2, and

Instrument 3.

The following section is quite important because of its control function in the visual part of the system: In order to have a smooth transition between the blob and the life patterns, it is necessary for each to know more or less what the other is doing. There is one anomaly, in that the blob routine in order not to interfere with the life pattern or the life routine does not primarily take any information from the life list. It looks at NLIVE but it doesn't bother the actual live list or any of the other life lists. It, on the other hand, does work directly with the display list since that is the only way it can get anything displayed. So that it is necessary in the case where life reaches a static pattern to break out of that. Otherwise, the blob will not start because once life goes to Ø, the display list will still contain listings and you can't simply Ø the display list every time life goes to Ø or you will never display anything except when life is on. So the break routine looks to see if the value of NLIVE is the same as it was the previous time around for a certain number of times and that certain number is controlled by the location KSTAG. At the moment I think it is set at 3777 octal. At any rate, if a life pattern happens to become static and no one is in the room or no one is paying too much attention, rather than have the whole system frozen just endlessly recomputing the static value, after a certain number of generations it will break out and at this point it will set the blob flags to begin and clear LIFE and DISP.

There is one final routine which is called LIMIT. This is a way of stopping the life after a certain number of generations taking care of oscillating patterns. Certain oscillating patterns oscillate between two or three successive values so that the break routine will not detect a stagnant condition, but LIMIT will guarantee that after some period of time, life will be terminated and the flow of the entire system will go on.

The blob speed routine is a very simple matter of taking a random number in this case the value of location Ø, the last interrupt return address, and masking it off and using it to control the location, actually it is pointed to by MMTPT which is one of the time controls of blob.

Page Ø Description:

Location Ø is kept empty for the interrupt and the next location is the pointer to the interrupt. Then there are a group of other pointers, in this case, the list pointers, (some of the list pointers). Then locations 20 through 37 are kept vacant since they are the auto index registers. The next group from 40 to 56 are other pointers, in this case, list pointers and the overflow constants which are set initially to a fail safe level of 7. This guarantees that the program won't blow up if for any reason you forget to load the data overlay. I should say at this point that the third part of the main program is the data overlay which is simply the data in the list together with the overflow constants and that constitutes the composition part or the music part of the sound program. All of the sounds that are produced are derived from data in those lists and this is a completely separate thing from the operating program. There can be an indefinite number

of separate overlays although only one more or less optimal one will be provided but the option exists for this to be changed in the future. At location 6Ø there are more pointers and these are mostly the sub-routine pointers. At location 1ØØ are various constants CSWØ through CSW3, these are the current switch buffers. Number of current switches, number of previous switches, number in the display list, etc. Some constants for the life routine, various timers, various flags, various masks, some constants. At location 16Ø are the life list pointers, and the pointers to the instrument sub-routines. At location 2ØØ are some more timers. From here on mostly the locations used by the function generators, the instrument flags, the modify flags, the step count buffers, the E Speed buffers, the new pitch, new filter and new envelope data buffers. At location 24Ø the addresses for the sound instrument routines and a couple of miscellaneous ones, KSTAG and LMXT which are used in the break and the limit routines that break you out of stagnant life patterns. At location 26Ø some more function buffers, base value for the four functions, displacement value for the four functions, step count for the four functions. At location 3ØØ the interrupt saves accumulators and carry, the function flags, some more masks. At location 34Ø is a group of constants which appear in the blob program. Since the blob program was assembled separately space is left for them. In the case of 341/342 and 344 through 346, those have to be listed in this part of the program since they are referenced and have to be in the assembly. At location 36Ø are the constants which are used to generate the offsets in the life routine, eight of these, for the eight possible offsets of a given bit. That concludes the description of all the portions of the main program and the test program. The blob part of the program will be discussed on a separate cassette as well as some other material relevant to the sound data overlays. This concludes the series of three cassettes which will be the description of the main program.

The following is a description of the tall rack console with a detailed description of each card, slot, rack and the controls.

The first unit at the top is a double rack filled with the resonator cards. The cards are identified starting at the left top row as address zero then address two then four, six, ten, twelve, fourteen, sixteen, etc. increasing in octal numbers (always even numbers). The first top rack contains then the first sixteen resonator addresses and the second rack contains the second sixteen.

The next unit down is the audio patch panel. Starting at the top going from left to right, the first row of six jacks are the left mixer inputs. Next is a knob which controls the left channel master volume and should be set to about 10 o'clock. There is a mark on the panel to indicate the proper setting. The next knob is the right channel master volume and to right of this are its 6 input jacks. This control should be set straight up. Going back to the left there is then a row of four jacks and all the way across on the right two groups of four jacks. These are multiple connection jacks and are not normally used. Coming back to the left and starting back again are two groups of three jacks in a little pyramid shape. The left hand group is used and on the bottom are the two inputs and above is the output of the first ring modulator. The second group is not used. It is the second ring modulator inputs and output. Coming farther to the right, almost directly underneath the left main output channel control, are two jacks which are the input and output to the reverberation unit (which is not in use). The next two jacks which are in use are the outputs from the two resonator mixers. A third jack is not in use, this is the noise generator output. Once again, going to the left and starting across the bottom row of jacks, will be found four pairs of jacks. This are the outputs from oscillators one, two, three and four respectively and are in pairs with the triangle wave form on the right and the sawtooth wave form on the left for each oscillator. At least one output of each oscillator is in use. About the center of the panel are two groups of jacks, one above each other, the filter inputs and outputs for filter 1A and 1B (input on bottom). The remaining jacks in the row, twelve sets of pairs above each other, are the inputs and outputs (inputs on bottom) to the twelve channels of voltage control amplifiers, the first three of which are in use. This completes the audio patch panel.

Directly beneath the audio patch panel is a card rack with sixteen slots and in the slots are to be found the following cards: In the first four slots, with red handles, are oscillators one through four. In the next slot, card slot five, having no handle but four blue trimmer adjustments, is filter 1A and 1B. In the next slot, having a green handle, card slot six, is the quad sample and hold unit, sample and holds A, B, C and D. In the seventh eighth and ninth slot, with black handles, three cards are to be found of which only the first one is plugged all the way in. These are quad voltage control amplifiers, voltage control amplifier 1A through D is the only card in use (in slot seven). The next two slots, ten and eleven, contain the two ring modulators having blue handles with yellow tips, only the first one is actually plugged in - ring modulator 1, in slot eleven. Slot twelve contains the reverberation amplifier. It is not actually in use but is plugged in. Slot thirteen is not used, slot fourteen contains

a reson card which is adjusted to function as the resonator bank 1 mixer. Slot fifteen is not in use and in sixteen a resonator functioning as resonator bank 2 mixer.

The next unit, a patch panel with banana jacks, is the control voltage panel and consists of actually two panels, the upper one having controls on it is the input and modifying voltage control panel and the narrow panel below with pairs of jacks is the output panel from D/A converters, (digital to analog converters). Starting again at the left, on this panel will be found at the top, four knobs. These are preset control voltage knobs and their outputs are the pairs of white jacks directly below them. Immediately to the right of each pair of white jacks is a black jack which is one of four control inputs of the sample and hold unit and since it is normally used with preset control voltage, it has been located next to the preset outputs. Coming further across to about the middle of the panel are four pairs of white jacks. These are the sample and hold outputs and directly beneath them are four black jacks which are the signal inputs for the sample and hold. Continuing across the top there are two pairs of control knobs with black jacks below them and two pairs of white jacks slightly to the right and below each set. These controls are the DC control voltage mixers with the black jacks being the inputs, the controls are the input volume controls associated and the pairs of white jacks are the outputs. On the very bottom of this upper panel is a group of four black jacks at the left and a long row of black jacks continuing across the bottom. The four black jacks at the left are the control voltage inputs to the four oscillators. The next four jacks which begin the long row are the filter "Q" and frequency inputs for filter A and "Q" and frequency inputs for filter B. The next four jacks directly beneath the pairs of white jacks which are the sample and hold output were previously discussed and are the sample and hold signal inputs. The remaining twelve black jacks across the bottom are the twelve voltage controlled amplifier control inputs. On the smaller panel directly below the one previous discussed are found a set of four pairs of white jacks on the left, then after a small space, a long row of pairs of white jacks. These are the outputs from the D/A converters and the first four are from the 10 bit D/A converters whose addresses are ØØØ, ØØ2, ØØ4 and ØØ6. The remaining row of the 6-bit D/A converters which are in pairs A and B of the first one, having the addresses Ø1Ø and Ø12. The first of these actually in use is the address Ø12 which is presently used as the filter of instrument one. The next pair or the next pair of pairs addresses Ø14 and Ø16 will be found to have Ø16 used as the filter address of instrument three. Continuing across the row, the addresses increase in the same fashion as resonator addresses being even octal three digit numbers. This completes the control voltage patch panel.

The next card rack below this is the card rack for the D/A converters themselves. Slots one through four contain the cards with blue handles having red tips and are the four 10-bit D/A converters. The next slots from five to sixteen of which only five to fourteen are actually usable slots are found the dual 6-bit D/A converters which are actually in use. Five converters are actually plugged in, the next rack below that are card racks for small cards having green handles. In the first slot (with red tips) is the primary device select decoder for the sound system. Slots two, three, four and six through nine contain the secondary device select decoders for the sound system. Slot five is never used because these addresses are not used

and the remainder of the rack from ten to sixteen is always empty. Continuing down to the next rack (for large cards) is the rack in which all of the input/output cards are kept. Slots one through three are always empty. Slots four and five having cards with black handles with white tips contain the two secondary visual system decoders and slot six, black handles with red tips, contains the primary visual decoder. Slot seven, eight, and nine contain the input and output adapter cards, white handled with red tips, green tips and black tips. Slots ten and eleven contain the input selecter cards and are the cards with the ribbon wire connectors attached to them. Slots twelve and thirteen, fourteen and fifteen are not currently used and in slot sixteen is the noise generator having a black handle with blue tips.

The remainder of the cabinet is taken up with four racks containing the visual lamp driver cards, large cards with no handles. There are 64 of these cards, one for each lamp column. One important note must be remembered in connection with these cards - they should never be unplugged or plugged while the high voltage to the lamps is turned on. It is perfectly all right to take the cards in and out or change them around while the power is on with just one exception - the lamp driver cords must never be removed or plugged in with the lamp voltage on. All of the cards in the rack previously described have been keyed and slotted so that no card may be plugged into a slot where it does not belong, either right side up or upside down. Each card will fit in only one unique way so that if a reasonable amount of care is exercised, the card can be checked before inserted just to make sure an unreasonable force will not be applied to push it in if it happens to be put into the wrong slot.

The following brief discussion concerns the controls on the panel of the small rack which is the power unit. On the middle panel will be found a key operated switch. This is the main power switch for the entire system. If this switch is turned so that the key is horizontal, it is on but it must be actuated by pressing the red button to the left of it. This causes a relay to engage supplying power to the whole unit. If a power failure should occur, this relay will trip out and require the entire to be re-started. Two toggle switches above, one on the right controlling the low voltage power - power to the computer and teletype and the ventilating fan - and the switch on the left controls the high voltage power to the lamps. The switch on the left will not actually turn on unless the switch on the right is already on. The proper procedure for starting the system would be to turn on the key switch, push the reset button after which the other two toggle switches can be turned on, first the right switch then the left, then the computer may be turned on, set to the starting address of the program, which is 4ØØ, reset pressed and then start. The program should then start without incident. That completes the description of the rack units and the control functions.

The following will be a description of the patching interconnections for three instruments. Please make reference to the appropriate instrument block diagram, also the diagram of control voltage patch panel and the audio patch panel.

Instrument one consists of one oscillator, oscillator no. 1, whose control voltage input is first processed by DC mixer 1. Its output goes to filter 1A. From filter 1A output to voltage controlled amplifier 1A and then to the left mixing input bank and on to the left channel output. Filter 1A frequency control, is controlled by the output of the sample and hold 1A whose input comes from 6-bit D/A converter 1B that is at address Ø12. Its rate control is taken from preset one, voltage control amplifier 1A control input comes from 6-bit D/A converter 5A at address Ø3Ø. The original pitch control voltage comes from D/A converter No. 1 at address ØØØ.

Instrument two consists of a similar oscillator with its control input scaled by means of DC mixer two. Its original control voltage input comes from 10-bit D/A converter no. 2 at address ØØ2. The oscillators output goes directly to a voltage controlled amplifier, voltage control amplifier 1B, whose control voltage is prepared by sample and hold 1B. The signal input of sample and hold 1B comes from D to A converter no. 7A at address Ø4Ø. The sample and hold rate is controlled by preset 2. The output from this instrument again goes to the left mixing output.

Instrument three, the most complex of the three insturments, consists of two oscillators whose control voltages come directly from 10-bit converters 3 and 4 at address ØØ4 and ØØ6 respectively. The oscillator outputs are intermodulated in ring modulator no. 1 whose output then goes via filter 1B and voltage control amplifier 1C to the right mixing output. The filter's "Q" is controlled by means of the trimmer, its frequency controlled by the output of sample and hold 1C whose input comes from 6-bit D to A converter at address Ø14. Its rate is controlled by preset 3. The voltage controlled amplifier for this instrument has a control voltage coming from a 6-bit converter at address Ø5Ø. The resonator patching is directly from resonator bank 1 and bank 2 respectively to the right mixing output. All patch cord connections for the above instruments are made with patch cords of appropriate lengths. There is no electrical difference between cords having different color handles. The only difference being the primary one between banana jacks which are used for control voltage patching and miniature audio plugs which are used for audio signal patching.

The following is a description of the Blob routine.

Blob generates, maintains and kills a moving blob outline. It is called by the executive program after the termination of a Life pattern (Life subroutine).

There are 3 flags which control Blob's actions: KBF, MBF and BBF. They are checked in the order given above for nonzero contents. The first nonzero flag causes a jump to one of 3 parts of Blob.

If BBF is found to be that flag, Blob initiates the birth of the blob. This is accomplished by reading the basic pattern of lamp coordinates from the list BPAT starting from a random point within that list into the list BLIS. The basic pattern consists of 32 lamp coordinates. The number 32 is then placed in NBLIV, and 1 is placed in NBDIS. The NBDIS of the NBLIV lamp coordinates are added to a basic displacement coordinate ORG (origin) and placed at the tail of the display list (which starts at 10201 octal) so it does not destroy any life lamp coordinates. So initially only 1 lamp coordinate of the blob is displayed. But counter XTIME is counted down every 0.01 second from a number stored in MRT till it reaches zero, when NBDIS is incremented. This results in 2 Blob lamps to be lit. This goes on till NBDIS equals NBLIV, the pattern is complete and the flag MBF is set. The basic pattern is not changed in any way, until Life lamp coordinates have been read from the switches, when the executive routine sets KBF. As long as KBF is set every time XTIME is counted down before from -50 to 0, NBDIS is decremented by 1 so that one less lamp element will be displayed. This continues until

no blob elements remain, or until the life elements die, whichever comes first.

At all points Blob is careful not to destroy NDBIS, the number of total lamps turned on, by life and by the blob.

The blob also moves randomly. This is accomplished as follows: at the beginning of birth a pair of moves is chosen at random from a list of 16 paths. The paths, stored in the list MOVV, are in fact signed increments to be added to the base displacement ORG to obtain a new base displacement. They represent (in the first quadrant) the following movements schematically:

This represents roughly angles of movement 0, /8, /4 and /8 radians respectively. There are 12 more, representing the above 4 projected into the other 3 quadrants.

Having chosen a pair of moves, which are stored in MOV1 and MOV2, the first move (MOV1) is executed, the origin is incremented by adding ORG to MOV1, masking out unwanted bits with LMSK2, storing the new ORG, and calling the routine BDISP to add each of NBDIS lamp coordinates in BLIS to the base ORG and store them in the display list. After ZTIME has counted down to Ø from the number set in MMT the next move (MOV2) is made. Thus MOV1 and MOV2 are alternately used (M1OR2 is Ø if MOV1, 1 if MOV2) to move the blob around the display. When 3 pairs of moves

have been executed (the negative of the number in M4), a new move pair is chosen at random, and so on.

If the pattern moves off either vertical edge it appears on the other vertical side; that is, it wraps around horizontally. But an unusual situation arises if it travels off the top or bottom edge. In that case, BDISP detects an illegal X (vertical) coordinate (by the way, X is used for the vertical axis and y for the horizontal despite the fact that's exactly backwards in conventional mathematics notation), finds if the violation is on top or bottom and sets EDGEF to a 2 or 1 respectively. It then calls BMDIR to select a new pair of moves, the first one of which contains a component in the opposite vertical direction to the one previously used. Program control then returns to BDISP to try again. The result is that Blob appears to bounce at some random angle from the top and bottom edges.

Accessory routines include a random number generator BGR16 which replaces the contents of ACØ with a random 16-bit number obtained by adding the contents of location Ø to the byte-swapped copy of location Ø. An extension of this is BGRØ which, on input, contains a number in n in ACØ. On leaving, ACØ contains a random number modulo n. This is useful for finding a random element from within a list of size not equal to 2^m for some interger m. BGRØ simply calls MULT to multiply n by a random number obtained from BGR16. MULT multiplies two 16-bit integer numbers in ACØ and AC1, and leaves the result in ACØ and AC1 with AC1 containing the low order bits. Lastly, RSAVE and RRET save and return registers ACØ through

AC2 in a user specified location. The calling sequence is as follows:

```
        JSR RSAVE
        B
        .
        .
        .
B:      .BLK 3
```

where B is a storage location block of 3 locations where ACØ through AC2 are stored. RSAVE returns user control 2 locations following the JSR.

All subroutine calls destroy AC3, so many routines save AC3, immediately after entry, into locations with names like BRTN1 or BS5.

All symbol name terminating with a period (like BLIS.) are pointers - e.g. BLIS. points to BLIS-1.

The interrupt is never disabled in the Blob routine.

That completes the description contained in the handwritten pages. Additional information can be obtained from: Michael Delay
633-1002
(In Seattle).

SEATTLE SOUND - LIGHT SYSTEM

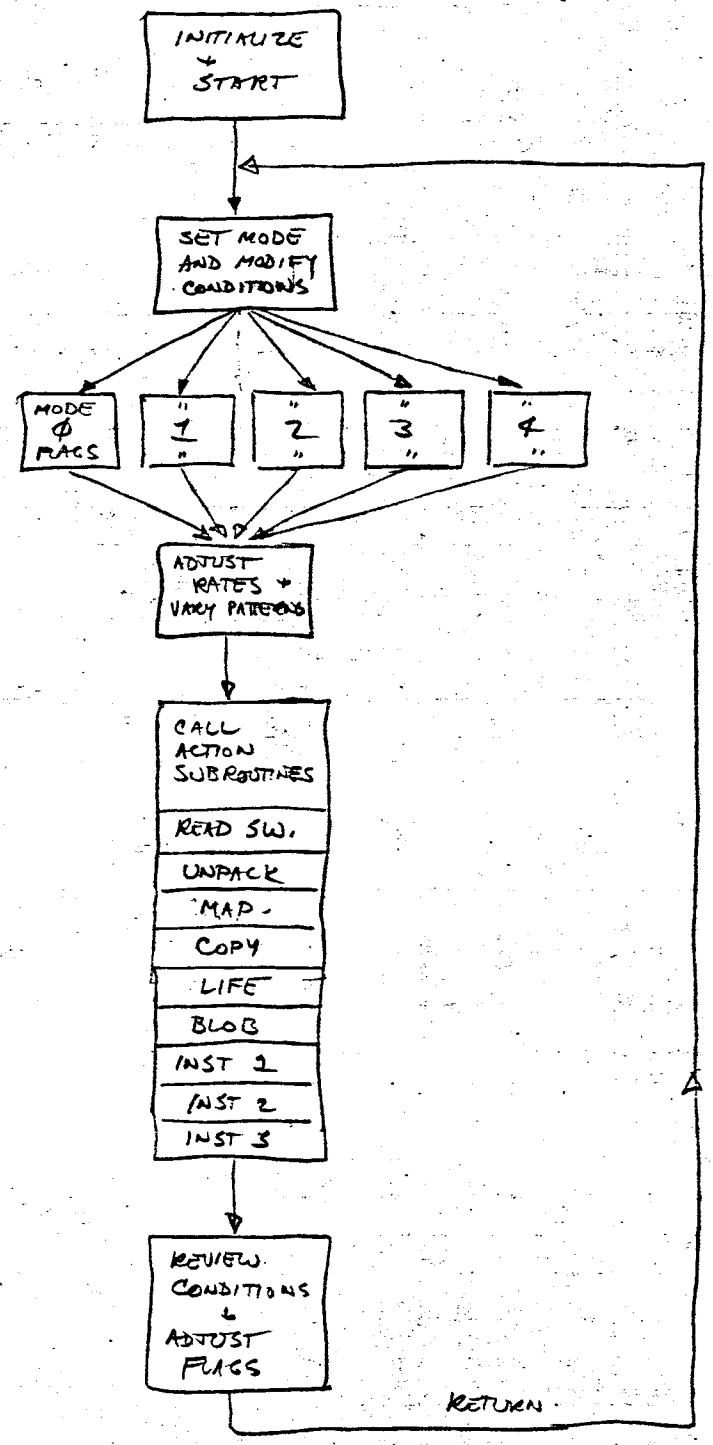
CORE MAP

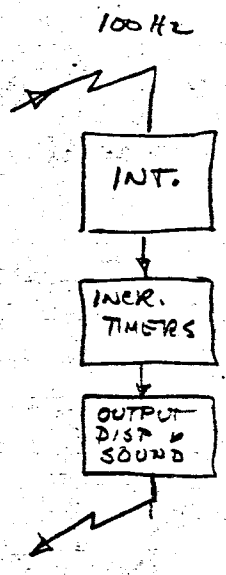
ϕ - 37ϕ7	MAIN OPERATING PROGRAM
4$\phi\phi\phi$ - 41$\phi\phi$	PITCH LIST
44$\phi\phi$ - 4442	FILTER LIST
5$\phi\phi\phi$ - 51$\phi\phi$	ENVELOPE LIST
57$\phi\phi$ - 572ϕ	MODIFY LIST.
6$\phi\phi\phi$ - 6674	BLOB SUBROUTINE
74$\phi\phi$ - 7577	PUNCH (NOT USUALLY RESIDENT)
1$\phi\phi\phi\phi$ - 14777	LIFE AND DISPLAY ASSOCIATED LISTS
15$\phi\phi\phi$ - 154$\phi\phi$	RESONATER LIST
16$\phi\phi\phi$ - 16123	TEST ROUTINES
176$\phi\phi$ - 17777	RESERVED FOR LOADERS

ALL OTHER LOCATIONS NOT USED.

SEATTLE SOUND-LIGHT SYSTEM.

SIMPLIFIED PROGRAM FLOW.





I/O INSTRUCTION EXAMPLES
SYSTEM DEVICE CODE = 76

CLEAR DISPLAY :

```
SUBZR  0,0
MOVOR  0,0     ; SETS AC0 ← 140000
DOCP   0,76    ; LOADS C REG + PULSES
                 CLEAR LINE ( HOLD)
```

LIGHT A LAMP :

```
SUBZR  0,0
MOVZR  0,0     ; SETS AC0 ← 040000
DOC    0,76    ; LOADS C REG AND ENABLES
                 STROBE L (LIGHT)
LDA    1, LAMP ; LAMP ADDRESS : BITS 4-7 = X
                                BITS 10-16 = Y
```

DO NOT COMBINE:

```
DOA    1,76    ; LOADS A REG AND PULSES LOAD
NIOP   76      ; BUS, LAMP STAYS LIT UNTIL
                 NEXT CLEAR
```

PULSE A RELAY :

```
SUBO   0,0     ; AC0 ← 0
DOC    0,76    ; ENABLE STROBE S

LDA    1, RELAYADDRESS ; RELAY ADDRESS =
                         1___XXX
DOBP   1,76    ; OUTPUTS   ( BITS 1-5 )
```

* NOTE, WAIT AT LEAST 100 μS BEFORE REPEATING AS STROBE S HAS BEEN STRETCHED

OUTPUT A SOUND COMMAND :

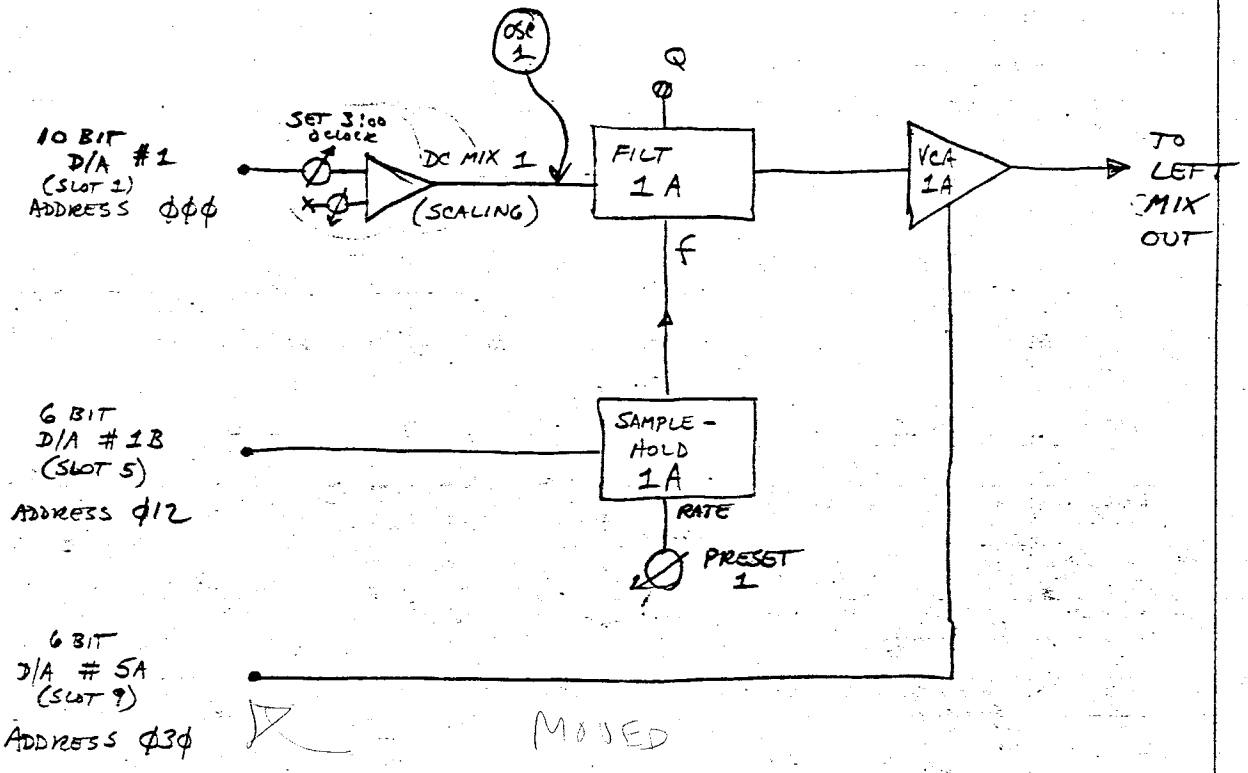
```
SUBZR  0,0     ; AC0 ← 100000
DOC    0,76    ; ENABLE STROBE X

LDA    1, SOUND ; BITS 1-5 = 0___
                  BITS 6-15 = DATA TO D/A
DOBP   1,76    ; OUTPUT.
```

INPUT : SEE .LOC 1000 ff for EXAMPLE.
A REG BITS 13-15 MUST BE LOADED
0-3 for SWITCH WORDS 0, 3

THEN DIA AC, 76 INPUTS ENABLED WORD

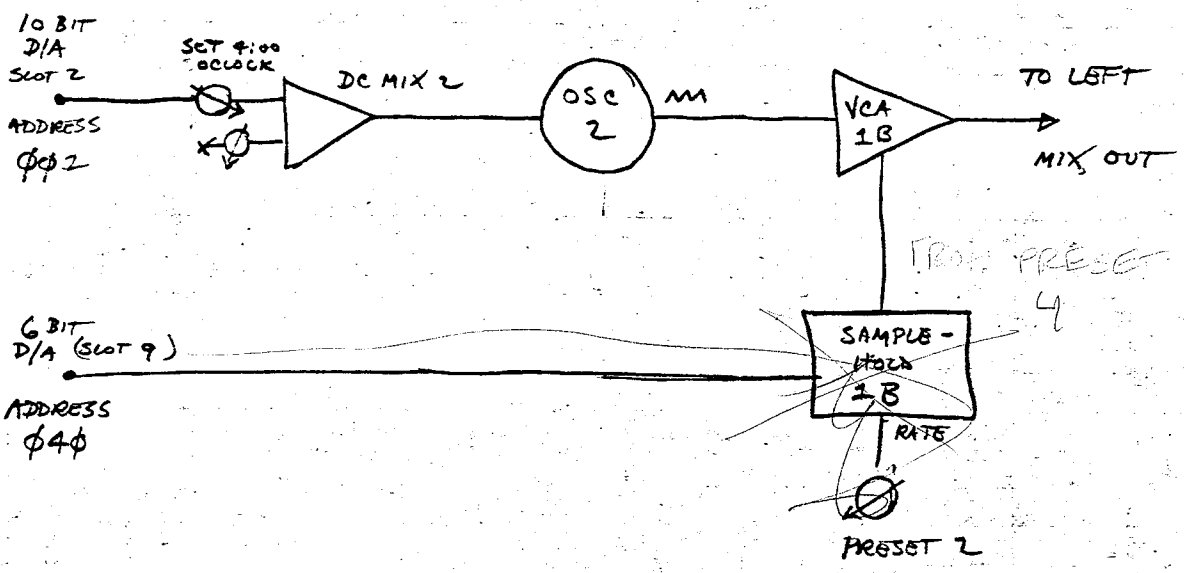
INSTRUMENT 1

BLOCK DIAGRAM

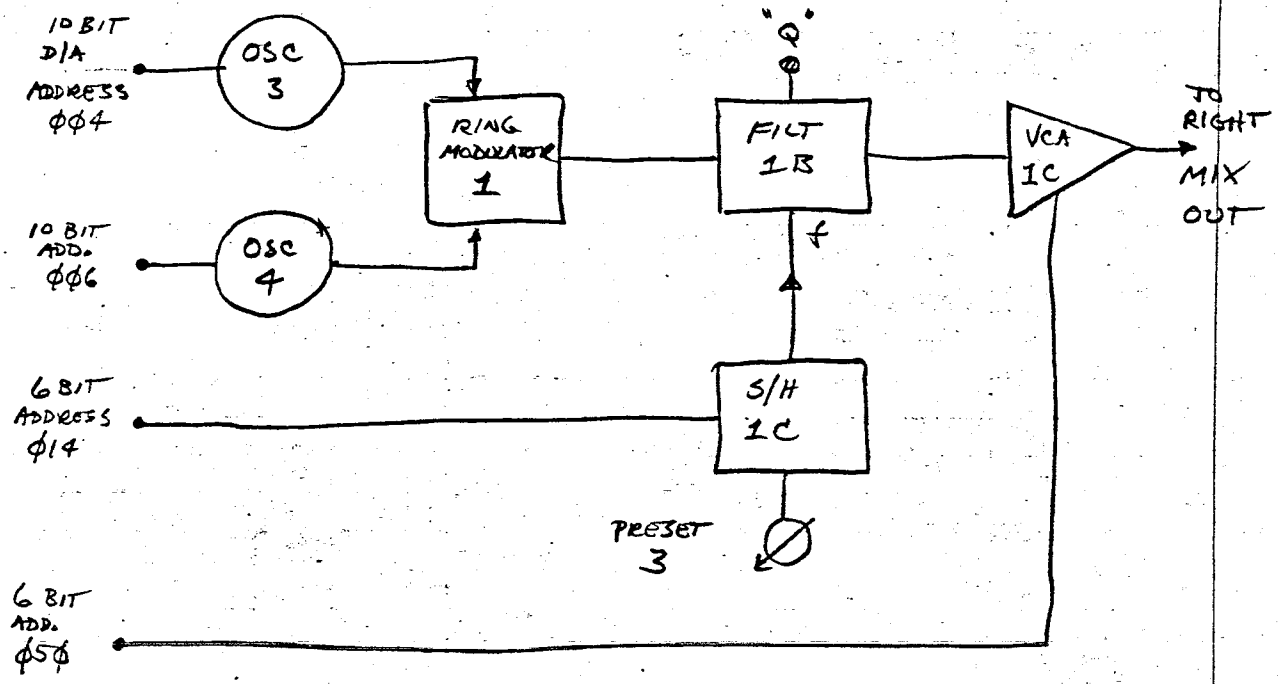


MOVED TO PRESET 3

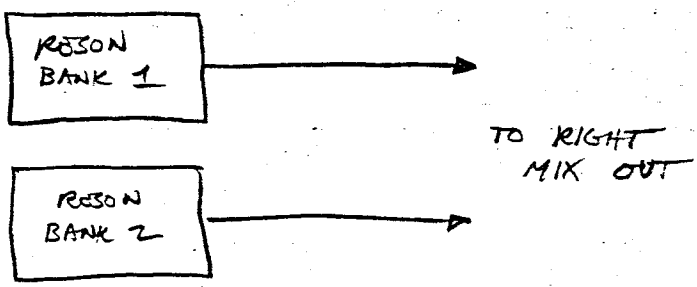
INSTRUMENT 2
BLOCK DIAGRAM

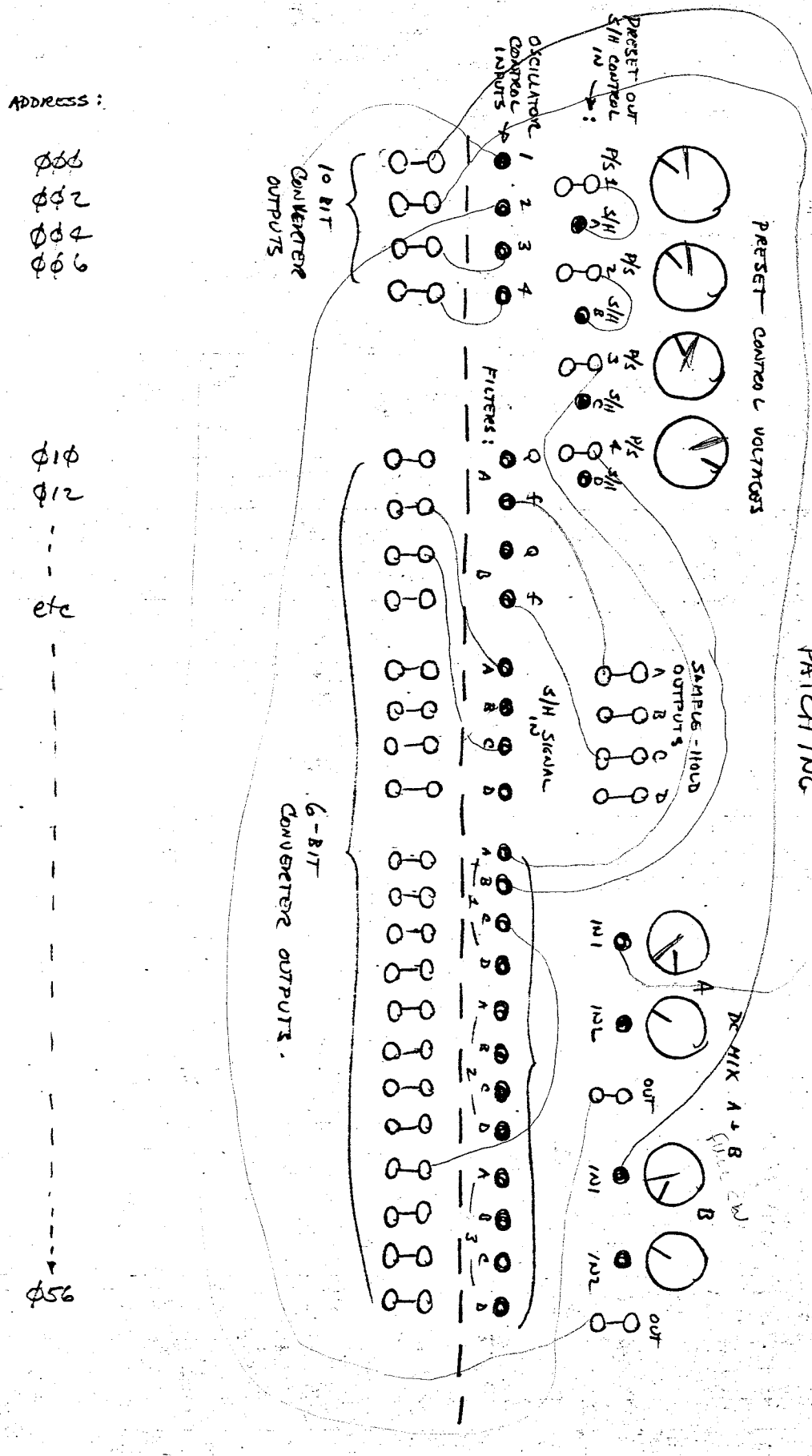


INSTRUMENT 3
BLOCK DIAGRAM



RESONATOR.



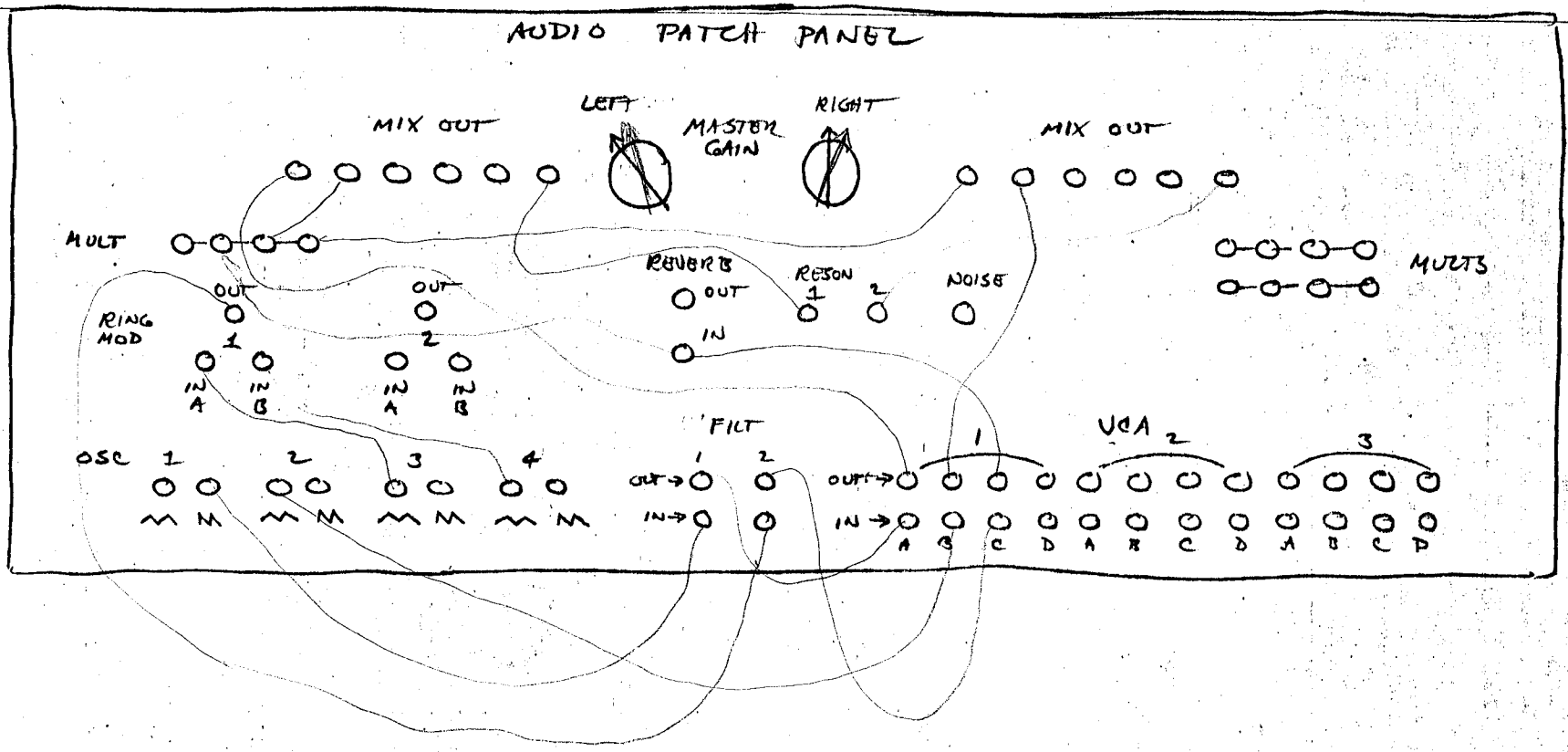


ADDRESS:

ϕϕϕ
ϕϕ2
ϕϕ4
ϕϕ6

ϕ1ϕ
ϕ12
---
etc

ϕ56


AUDIO PATCH PANEL
MIX OUT
LEFT
MASTER GAIN
RIGHT
MIX OUT
MULT
REVERB
OUT
IN
REASON
1
2
NOISE
MULTS
RING MOD
OUT
OUT
1
2
IN A
IN B
IN A
IN B
OSC
1
2
3
4
FILT
1
2
OUT
IN
VCA
1
2
3
OUT
IN
A
B
C
D
A
B
C
D
A
B
C
D

MAC AT 10:30

~~These movements can be~~ the
There are 12 more, representing the above 4 projected into other 3 quadrants.

Having chosen a pair of moves, which are stored in MOV1 and MOV2, the first move (MOV1) is executed, i.e. the ~~pattern~~ origin is incremented by adding ORG to MOV1, masking out unwanted bits with LM3K2, obtaining the new ORG, and calling the routine BDISP to add each to NODES lamp coordinates in BLIS to the base ORG and store them in the display list. After ETIME has counted down to Ø from the number set in MMT, the next move (MOV2 if CM10K2 is Ø if MOV1, 1 if MOV) is made. Thus MOV1 and MOV2 are alternately used to move the blob around the display. When 3 pairs of moves have been executed (the negative of the number in M4), a new move pair is chosen at random, and so on.

If the pattern moves off either ~~horizontal~~ vertical edge it appears on the other vertical side; that is, it wraps around horizontally. But an unusual situation arises if it travels off the ~~side~~ top or bottom edge. In that case, BDISP detects an illegal x (vertical) coordinate, finds if this violation is on top or bottom and sets EDGEF to Ø or 1 respectively (?). It then calls BMPR to select a new move pair, the first one of which contains a component in the opposite ~~direction to~~ vertical direction to the one previously used. Program control then returns to BDSP to try again. The result is that BOB appears to bounce at some random angle from the top and bottom edges.

~~Other routines~~ Accessory routines include a random number generator BGR16 which replaces the contents of ACØ with a random 16-bit number obtained by adding the contents of location Ø to the byte-swapped copy of its contents location Ø. ~~Finally~~ An extension of this is BGRØ which on input contains a number n in ACØ. On leaving, ACØ contains a random number modulo n. This is useful for picking a random element from within a list of size not equal to 2^m for some integer m. BGRØ simply calls MULT to multiply n by a random no. obtained from BGR16. MULT ~~is a 16-bit~~ multiplies two 16-bit unsigned numbers in ACØ and AC1, and leaves the result in ACØ and AC1 with AC1 containing the ~~least significant bits~~ low order bits. Lastly, SSAVE and RRET save and return registers ACØ through AC2 in a user-specified location. The calling sequence is as follows:

```
        JSR SSAVE
        B
        :
B:      .BLK 3
```

where B is the range of 3 locations where ACØ ... 3 are stored. SSAVE returns user control to 2 locations following the ~~call~~ JSR.

BLOB description

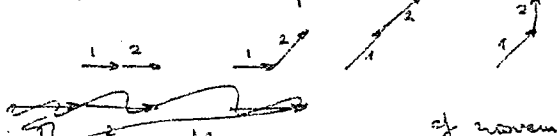
BLOB is generates, maintains, and kills a moving blob on line. It is called by the executive program after the termination of a BIFF pattern.

There are 3 flags which control BLOB's actions: KBF, MBF and BBF. They are checked in order given above for nonzero contents. The first nonzero flag causes a jump to one of 3 parts of BLOB.

If BBF is found to be that flag, BLOB initiates the birth of the blob. This is accomplished by reading the basic pattern from the list BPAT of lamp coordinates from a random point within that list into the list BLIS. The basic pattern consists of 32_{10} lamp coordinates. The number 32_{10} is then placed in NBLIV, and 1 is placed in NBDIS. Then NBDIS of the NBLIV lamp coordinate coordinates are added to a base displacement ORG and placed at the tail of the display list so it does not destroy any life lamp coordinates. So initially only 1 lamp coordinate of the blob is displayed. But a number stored in MBLT counter XTIME is counted down every .01 sec from -50_{10}, till it reaches zero, when NBDIS is incremented, and this causes another blob element to be displayed. This results in 2 lamps to be lit. This goes on till NBDIS equals NBLIV, the pattern is complete, and MBF flag is set. The basic pattern is not added to in any way, until life lamp coordinates have been read from the changed window, when the executive routine sets KBF. From then on As long as KBF is set every time XTIME is counted down as before to -50_{10} to $\emptyset$, the next blob element in the NBDIS is decremented by 1 so that one less lamp element will be displayed. This continues until no blob element remain, or until the life elements die, whichever comes first.

At all points BLOB is careful not to destroy NBDIS, the no. of total lamps displayed on, for life and for the blob.

The blob is also moves randomly. This is accomplished as follows: At the beginning of birth a pair of moves is chosen at random from a list of 16 pairs. The pairs, stored in the list MOVV, are in fact signed increments to be added to the base displacement ORG to obtain a new base displacement. They represent (in the first quadrant) the following movements schematically:



of movement

1 roughly angles of $0, \frac{\pi}{8}, \frac{\pi}{4}$, and $\frac{3\pi}{8}$ radians, respectively

SEATTLE 5 - PART 3
TEST PROGRAMS

```
016000 .LOC    16000

               ;ROUTINE TO LIGHT ENTIRE DISPLAY FOR TESTING

               ;START AT 16000, AFTER HALT PRESS CONTINUE; DISPLAY
               ;WILL SCAN.  RATE OF SCAN CONTROLLED BY KDLY.

16000 102620 LTUNE:  SUBZR   0,0
16001 101240         MOVOR   0,0
16002 063376         DOCP    0,76
16003 102620         SUBZR   0,0
16004 101220         MOVZR   0,0
16005 063076         DOC     0,76
16006 102440         SUBO    0,0
16007 024433         LDA     1,STOP
16010 123000         ADD     1,0
16011 040427         STA     0,CNT.1
16012 061076         DOA     0,76
16013 060376         NIOP    76
16014 101400         INC     0,0
16015 010423         ISZ     CNT.1
16016 000774         JMP     .-4
16017 063077         HALT

16020 102440 SCN:    SUBO    0,0
16021 126400         SUB     1,1

16022 102620 LOOP.:  SUBZR   0,0
16023 101240         MOVOR   0,0
16024 063376         DOCP    0,76
16025 102620         SUBZR   0,0
16026 101220         MOVZR   0,0
16027 063076         DOC     0,76
16030 065076         DOA     1,76
16031 060376         NIOP    76
16032 125400         INC     1,1
16033 020410         LDA     0,KDLY.
16034 040405         STA     0,CNT.2
16035 010404         ISZ     CNT.2
16036 000777         JMP     .-1
16037 000763         JMP     LOOP.

16040 000000 CNT.1:  0
16041 000000 CNT.2:  0

16042 170000 STOP:   170000
16043 170000 KDLY.:  170000




016050 .LOC    16050

               ;ROUTINE TO TEST CONVERTER AND AUDIO UNITS
               ;SEE INSTRUCTION MANUAL FOR USE

16050 024412 STUNE:  LDA     1,SM.1
16051 030412         LDA     2,SM.2
```

```
---
   16052 102620 S.DO:    SUBZR   0,0
   16053 063076          DOC     0,76
   16054 060477          READS   0
   16055 113405          AND     0,2,SNR
   16056 000772          JMP     .-6
   16057 107400          AND     0,1
   16060 066376          DOBP    1,76
   16061 000767          JMP     ST.UNE

   16062 077777 SM.1:    77777
   16063 100000 SM.2:    100000



         016100 .LOC     16100

                ;ROUTINE TO PULSE RESONS, RATE CONTROLLED BY RT.K

   16100 020420 RTUNE:   LDA     0,RT.K
   16101 040421          STA     0,RC.2
   16102 064477          READS   1
   16103 152620          SUBZR   2,2
   16104 034417          LDA     3,RT.M
   16105 137400          AND     1,3
   16106 157000          ADD     2,3
   16107 102440          SUBO    0,0
   16110 063076          DOC     0,76
   16111 060076          NIO     76
   16112 076376          DOBP    3,76
   16113 014406          DSZ     RC.1
   16114 000777          JMP     .-1
   16115 014405          DSZ     RC.2
   16116 000775          JMP     .-3
   16117 000761          JMP     RTUNE

   16120 000003 RT.K:    3

   16121 000000 RC.1:    0
   16122 000000 RC.2:    0
   16123 076000 RT.M:    76000


                .EOT
```

```
---
CNT.1    016040
CNT.2    016041
KDLY.    016043
LOOP.    016022
LTUNE    016000
RC.1     016121
RC.2     016122
RTUNE    016100
RT.K     016120
RT.M     016123
SCN      016020
SM.1     016062
SM.2     016063
STOP     016042
STUNE    016050
S.DO     016052
```

SEATTLE 5
SYMBOL TABLE

DOES NOT INCLUDE BLOB

AGN2	002651
ALL	001624
ALPT	000163
AMOD	001050
AMOD1	003020
AMSAV	001053
APT	000061
ATALY	002576
BACK	~~000526~~ 000765
BATST	002475
BBF	000344
BCNT	002604
BEACH	002517
BEGIN	000467
BEXAM	002513
BINIT	002510
BLPT	000162
BORN1	000120
BORN2	000121
BREAK	000714
BSAV	002602
BTALY	002736
BTEST	002542
BVAPT	000075
BVARY	000700
BVSAV	000710
CALL	000666
CC03	003440
CC20	003437
CC98	003456
CC99	003465
CINIT	002605
CMPR	001607
COMP1	003045
COMP2	003227
COMP3	003507
CONT0	002455
CONT1	000674
CONT2	000676
CONT3	000700
CONT4	000702
CONT5	000704
CONT6	000706
CONT7	000710
CONT8	000712
COPPT	000067
COPSV	001625
COPY	001600
CSW0	000100
CSW1	000101
CSW2	000102
CSW3	000103
CSWPT	000040
CT8A	002572
CT8B	002601
CVG	003344
DABS	003315
DELAY	002061
DELET	002701
DISP	002032

```
---
DISPL   003434
DISPT   000042
DLOOP   002046
DMAP1   001400
DMAPT   000066
DOTST   002465
DSAV    001644
DSGA    002534
DSGB    002537
DSGK    002452
DTIME   000130
DTST2   002551
DUP     001653
DUPL    001626
E2BSE   000236
EAD1    000246
EAD2    000247
EAD3    000250
EAD4    002341
EBUF1   003212
EBUF2   003433
EBUF3   003707
EMODE   000642
EMSK1   000311
ENEW1   000233
ENEW2   000234
ENEW3   000235
ENV1    003122
ENV2    003260
ENV3    003615
EOVFL   000056
EPT1A   000047
EPT1B   000006
EPT2A   000050
EPT2B   000007
EPT3A   000051
EPT3B   000010
ESET1   003142
ESET2   003300
ESET3   003635
ESPD1   000221
ESPD2   000222
ESPD3   000223
ESPD4   000224
ESTP1   000211
ESTP2   000212
ESTP3   000213
ESTP4   000214
ETOT1   000215
ETOT2   000216
ETOT3   000217
ETOT4   000220
EVAL    002740
EVNT1   003162
EVNT2   003404
EVNT3   003655
EXITM   001427
EXTAV   000144
EXTEN   002645
F1ADJ   002167
```

```
---
F1BSE   000260
F1CLR   002165
F1DSP   000264
F1FLG   000305
F1MAX   002211
F1OUT   002201
F1TOT   000270
F1ZER   002206
F2ADJ   002227
F2BSE   000261
F2CLR   002225
F2DSP   000265
F2FLG   000306
F2MAX   002251
F2OUT   002241
F2TOT   000271
F2ZER   002246
F3ADJ   002267
F3BSE   000262
F3CLR   002265
F3DSP   000266
F3FLG   000307
F3MAX   002311
F3OUT   002301
F3TOT   000272
F3ZER   002306
F4ADJ   002327
F4BSE   000263
F4CLR   002325
F4DSP   000267
F4FLG   000310
F4MAX   002351
F4OUT   002341
F4TOT   000273
F4ZER   002346
FFAD1   000244
FFAD2   000245
FILF1   003075
FILF2   003570
FLINC   000506
FM2     002374
FMAX    002373
FMSK1   000154
FMSK2   000312
FNEW1   000231
FNEW2   000232
FOVFL   000055
FPT1A   000046
FPT1B   000005
FREQ    000123
FSET1   003170
FSET2   003412
FSET3   003663
FUNC1   002160
FUNC2   002220
FUNC3   002260
FUNC4   002320
FUNCN   002360
GARB    002657
GET     001112
```

< FMAX. 002215

```
---
GINIT   002401
GLPT    000164
GPT     000073
I1EV1   003004
I1FLG   000203
I1PT    000165
I1SAV   003074
I2EV2   003224
I2FLG   000204
I2PT    000166
I2SAV   003357
I3EV3   003504
I3FLG   000205
I3PT    000167
I3SAV   003566
I4FLG   000206
I4PT    000170
IDIV    003450
IGNOR   000736
INST1   003000
INST2   003220
INST3   003500
INT     002000
INTPT   000001
IS4     002603
K17     000156
K2TLY   002741
K8      002571
KBF     000346
KDONE   002642
KLIV1   000116
KLIV2   000117
KLPT    000161
KMAP    000142
KMTIM   000202
KNEG    003436
KPT     002737
KPT2    002742
KRAMP   000151
KSTAG   000251
KTALY   002735
KVAR    000712
KX      001157
LAST    001150
LCNT    002565
LCSET   000740
LEACH   002431
LEXAM   002421
LHLD1   002566
LIFE    002400
LIFPT   000070
LIMIT   000745
LKILL   000725
LLP     002722
LLPT    000160
LMXT    000252
LMXTC   000764
LOOK    002670
LSAV    002564
LTEST   002460
```

```
---
LTSAV   002574
MAPM    000145
MARG1   003030
MAXLV   002744
MBF     000345
MFLG1   000207
MFLG2   000210
MINC    000513
MLOOP   001420
MMSK1   000152
MMTPT   000076
MODE0   000527
MODE1   000546
MODE2   000565
MODE3   000604
MODE4   000623
MODPT   000065
MOVFL   000053
MPT1A   000044
MPT1B   000003
MRFLG   000137
MSK10   000155
MSK3    000146
MSK6    000150
MSK7    000147
MSKL    002573
MSRPT   000171
MTIM1   000200
MTIM2   000201
NBDIS   000410
NBLIV   000407
NCNT    002443
NCSW    000104
NDISP   000106
NDSPM   001433
NFLAG   003435
NLIVE   000115
NOENV   003400
NOKIL   002635
NOLIV   001645
NOT3    002530
NOTIN   001640
NPSW    000105
NREF    000744
NRES    000110
NSND    000107
NTBA    000113
NTBB    000112
NTBK    000114
NTEMP   000111
NTOT1   002567
NTOT2   002577
NV0     003370
NVMAX   003371
OSC1    003046
OSC2    003230
OSC3    003510
OSC4    003540
OVER    002750
OVWR2   002675
```

```
'---
OVWRT   002623
PAD1    000240
PAD2    000241
PAD3    000242
PAD4    000243
PMSK1   000153
PMSK2   000313
PNEW1   000225
PNEW2   000226
PNEW3   000227
PNEW4   000230
POVFL   000054
PPT1A   000045
PPT1B   000004
PSWPT   000041
QMOD1   003007
QSIGN   003306
RAND    000143
RANDP   000074
RD0     001005
RD1     001013
RD2     001022
RD3     001031
RDMP1   001431
RDMSK   001432
RDSPT   000060
RDSW    001000
RESD    002060
RESON   002070
RESTR   002361
REST.   002151
RET     002370 < RET. 002060
RFLG    000141
RMOD    001060
RMOD1   003026
RMSAV   001072
RMSK1   002055
RMSK2   002056
RMSK3   002057
RNDSV   000752
ROT     001134
ROUT    002135
ROVFL   000052
RPT     000062
RPT1A   000043
RPT1B   000002
RRAND   000740
RSHFT   002131
RTBUF   000140
RTIME   000131
RTNRD   001044
RTNSW   001153
SAME    001617
SAV0    000300
SAV1    000301
SAV2    000302
SAV3    000303
SAVC    000304
SAVE    002002
SCNT    002575
```

```
SERV    001140
SET1    000503
SETC    000464
SETUP   001111
SLOOP   001633
SMASK   002054
SOUND   002142
STAGC   000743
START   000400
STIME   000132
SWLPT   000063
SWLST   001100
T1      000360
T2      000361
T3      000362
T4      000363
T5      000364
T6      000365
T7      000366
T8      000367
TEMPT   000072
TICK    002010
TIM1    000124
TIM2    000125
TIM3    000126
TIM4    000127
TMSK    000136
TPT1    002570
TPT2    002600
TPTPT   000071
TRANS   001413
TRY     002503
TSTFL   001040
TT1     000474
TT2     000476
UPDT2   002707
UPK     001125
UPKFL   000122
UPKSV   001154
UTALY   002743
VMSK    000711
WAY     002563
WVG     003322
XCNT    001155
XMOD    001164
XMOD1   003023
XMSAV   001172
XPT     000064
XTIME   000133
YCNT    001156
YES     002473
YES2    002557
YMSK    001160
YTIME   000134
ZTIME   000135
```

SEATTLE 5 - PART 3
TEST PROGRAMS

```
016000          .LOC    16000

                ;ROUTINE TO LIGHT ENTIRE DISPLAY FOR TESTING

                ;START AT 16000, AFTER HALT PRESS CONTINUE; DISPLAY
                ;WILL SCAN.  RATE OF SCAN CONTROLLED BY KDLY.

16000 102620 LTUNE:  SUBZR   0,0
16001 101240         MOVOR   0,0
16002 063376         DOCP    0,76
16003 102620         SUBZR   0,0
16004 101220         MOVZR   0,0
16005 063076         DOC     0,76
16006 102440         SUBO    0,0
16007 024433         LDA     1,STOP
16010 123000         ADD     1,0
16011 040427         STA     0,CNT.1
16012 061076         DOA     0,76
16013 060376         NIOP    76
16014 101400         INC     0,0
16015 010423         ISZ     CNT.1
16016 000774         JMP     .-4
16017 063077         HALT

16020 102440 SCN:    SUBO    0,0
16021 126400         SUB     1,1

16022 102620 LOOP.:  SUBZR   0,0
16023 101240         MOVOR   0,0
16024 063376         DOCP    0,76
16025 102620         SUBZR   0,0
16026 101220         MOVZR   0,0
16027 063076         DOC     0,76
16030 065076         DOA     1,76
16031 060376         NIOP    76
16032 125400         INC     1,1
16033 020410         LDA     0,KDLY.
16034 040405         STA     0,CNT.2
16035 010404         ISZ     CNT.2
16036 000777         JMP     .-1
16037 000763         JMP     LOOP.

16040 000000 CNT.1:  0
16041 000000 CNT.2:  0

16042 170000 STOP:   170000
16043 170000 KDLY.:  170000




016050          .LOC    16050

                ;ROUTINE TO TEST CONVERTER AND AUDIO UNITS
                ;SEE INSTRUCTION MANUAL FOR USE

16050 024412 STUNE:  LDA     1,SM.1
16051 030412         LDA     2,SM.2
```

```
                                  STUNE   16050   024412  LDA   1,SM.1
                                          16051   030412  LDA   2,SM.2
---
16052 102620 S.DO:    SUBZR   0,0
16053 063076          DOC     0,76
16054 060477          READS   0
16055 113405          AND     0,2,SNR
16056 000772          JMP     .-6
16057 107400          AND     0,1
16060 066376          DOBP    1,76
16061 000767          JMP     STUNE

16062 077777 SM.1:    77777
16063 100000 SM.2:    100000



        016100 .LOC     16100

               ;ROUTINE TO PULSE RESONS, RATE CONTROLLED BY RT.K

16100 020420 RTUNE:   LDA     0,RT.K
16101 040421          STA     0,RC.2
16102 064477          READS   1
16103 152620          SUBZR   2,2
16104 034417          LDA     3,RT.M
16105 137400          AND     1,3
16106 157000          ADD     2,3
16107 102440          SUBO    0,0
16110 063076          DOC     0,76
16111 060076          NIO     76
16112 076376          DOBP    3,76
16113 014406          DSZ     RC.1
16114 000777          JMP     .-1
16115 014405          DSZ     RC.2
16116 000775          JMP     .-3
16117 000761          JMP     RTUNE

16120 000003 RT.K:    3

16121 000000 RC.1:    0
16122 000000 RC.2:    0
16123 076000 RT.M:    76000


               .EOT
```

CNT.1	016040
CNT.2	016041
KDLY.	016043
LOOP.	016022
LTUNE	016000
RC.1	016121
RC.2	016122
RTUNE	016100
RT.K	016120
RT.M	016123
SCN	016020
SM.1	016062
SM.2	016063
STOP	016042
STUNE	016050
S.DO	016052

SEATTLE SOUND - LIGHT SYSTEM

CORE MAP

ϕ - 37ϕ7	MAIN OPERATING PROGRAM
4$\phi\phi\phi$ - 41$\phi\phi$	PITCH LIST
44$\phi\phi$ - 4447	FILTER LIST
5$\phi\phi\phi$ - 51$\phi\phi$	ENVELOPE LIST
57$\phi\phi$ - 572ϕ	MODIFY LIST
6$\phi\phi\phi$ - 6674	BLOB SUBROUTINE
74$\phi\phi$ - 7577	PUNCH (NOT USUALLY RESIDENT)
1$\phi\phi\phi\phi$ - 14777	LIFE AND DISPLAY ASSOCIATED LISTS
15$\phi\phi\phi$ - 154$\phi\phi$	RESONATER LIST
16$\phi\phi\phi$ - 16123	TEST ROUTINES
176$\phi\phi$ - 17777	RESERVED FOR LOADERS

ALL OTHER LOCATIONS NOT USED.