File under: R Date: 17 Jul 1979 Name: Robert Poor

Name: Robert Poor

Project: 1 Programmer: ROB

File Name: LOWER.TXT[SAM,MUS]

File Last Written: 21:29 4 Apr 1979

Time: 7:45 Date: 17 Jul 1979

Stanford University
Artificial Intelligence Laboratory
Computer Science Department
Stanford, California

```
COMMENT ⊛   VALID 00024 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00003 00002    An uninhibited guide to LOWER.FAI...
C00008 00003    Internal data to lower
C00011 00004    Opcode definitons
C00012 00005    Generator definitions
C00014 00006    Modifier definitions
C00015 00007    Pointers to Conos, etc.
C00016 00008    Arguments to the bind procedure
C00017 00009    Opcode definitions for Cono, etc.
C00018 00010    Mode and field definitions
C00019 00011    Macros for set_field and set_mode
C00020 00012    Finally the code... begins with set_mode:
C00022 00013    The get procedure!
C00024 00014    Give, Decode and Relative
C00025 00015    The fabulous bind procedure...
C00026 00016    Initialize, Getptr, Newptr, Chksm and Box error
C00028 00017    Flush and friends
C00029 00018    Load delay
C00030 00019    Set (error, procedure, channel and output)
C00032 00020    Macros for unpacked commands
C00035 00021    Macros for loading large fields
C00037 00022    The packed macros are used for commands which have 2 commands in them.
C00039 00023    The End!
C00040 00024    Ø22-Sep-78  2207        DGL     RAIDing LOWER
C00043 ENDMK
C⊛;
```

An uninhibited guide to LOWER.FAI...
the low level assembler for the Systems Concepts Digital Synthesizer

[KS 4-Apr-79] This document is somewhat out of date. In particular, Bind has been extensively changed, and many pages shuffled around. Thus many of the references in this document will be meaningless.

The first fact to know is that LOWER.FAI (hereafter referred to as lower) is read by a program known as MAKDEF. MAKDEF turns out a file generally called LOWER.DEF. In order to perform this marvel of the modern age, it is necessary for the definitions in lower to have a certain format. To state it briefly, the format depends on the number of left arrows in the symbol definition. For example, f ← 0 will NOT be put in lower. BUT, op_sweep ←← 13 will. So, two arrows will pass the symbol to LOWER.DEF. Also, all comments of the format "comment • ... •" are passed to LOWER.DEF. And how does MAKDEF know the end of the definitions? The answer a statement of the form "; End of definitions". Perhaps someday this funny syntax will be replaced by macros such as SDEF(symbol,value) and REMARK(text).

Another important note is the form of the identifiers. g_ indicates this declaration refers to a generator. m_ a modifier, d_ a delay unit. p_ is a pointer.

This guide will take each page one by one, so a listing of lower.fai should be on the right hand, a copy of lower.txt on the left hand. Here we go!

The first comment is strictly for MAKDEF to pass directly to lower. It includes all the external definitions a typical SAIL program needs to take. Next follow the internal definitions for the loader to find. Also USERERR is external'ed since it might be called by BOX_ERROR.

Now for the first macro definitions... ptr(name,size,lsb) creates a byte pointer shifted to the right 18 bits, i.e., without the address bits. This is so we can save a memory reference (and WHY not?) by doing a HRLI rather than a MOVE or HRL. bit(name,lsb) is a byte pointer definition for a one bit field. Otherwise it is the same as ptr.

Now for the first definitions... id_x, where x is generator, modifier, sum_memory and delay are the identifiers passed to GET in order to grab an appropriate resource. x_data is the pointer (generally 20 bits) for the data field. x_op a pointer to the opcode field and x_number is the pointer to the resource number.

Bias definitions come next. They were added in order to write MERGE, but they might be useful otherwise.

Internal data to lower

Next, we focus our attention on the internal arrays used by the lower level routines. In order to make SAIL happy and the machine language programmers unhappy, all arrays are defined with the ARRAY macro. It's arguments are the address of the array, the size of the array and the lower and upper bounds. Note that the sum memory array is sliced up into quadrants.

data_begin marks the point in the data storage area where the BIG BLT can start taking place. Next are all the resource arrays. Currently, it just acts as a boolean array... But should the need ever arise, it could very easily become an integer array.

dp_begin marks the beginning of the dual pointer arrays. Dual pointers are used to save the box from making too many memory references. The halves hold the relative address (index) into the buffer where the command for a particular command can be found. For example, take Mmode and Msum... Under packing mode, they can share a command word. SO, if a bind(m_id,mode,...) is done, then a pointer to the word where the set mode command is will be found at mmodesum[m_id]. Furthermore, the pointer will be found the left half!

mode_x holds the modes for the resources. This is so the various mode commands can set a new run mode without affecting the other modes, such as the envelope mode. sum_x holds the sum memory locations, but not for any present reason.

Most of the following definitions do NOT need a comment, so I won't! max_id is used by GET to determine whether the program has run out of resources.

Opcode definitons

This page contains the opcode definitions for various (surprise) opcodes. op_x means that this value is suitable for application to either y_op, where x is an opcode for a resource y. However... o_ definitions exist so that there is a uniform way to decode opcodes for BOTH generators and MODIFIERS. Therefore, all o_x definitions are created with g_op in mind. Since op_ definitions for modifiers are 5 bits, then the o_ definition for modifiers are the op_ definition shifted to the right by 1 bit.

Generator definitions

xxxlsb is the least significant bit; xxxmask is the mask for the modes.
All commands are shifted to the left by xxxlsb. Not much else to say ...

Generator pointers

These are pointers to various fields and bits in generator commands.
The pointers are defined through the use of the ptr and bit macros.
For all the dope on these macros, see the first page.

Modifier definitions

fnlsb stands for the function least significant bit. The • is used to shift the value into the right (i.e., proper) field.

Modifer Pointers

What can I say here that I didn't say for generators?

Pointers to Conos, etc.

These are pointers defined for creating the command stream. As of this writing, 22 June 1977, it had not been decided where the command stream would go. The pointers for the delay field are also on this page.

Arguments to the bind procedure

These are the arguments that are defined and given as the second argument to the bind procedure. A few things should be noticed. First, note that the commands overlap. This may be a bad idea, I'm not sure, but it did allow me to be able to use the identifier "sum_memory" for both modifiers and generators without preference. These are offsets into a table, which you will see soon enough.

Opcode definitions for Cono, etc.

These are the opcode definitions used to define certain modes in the command stream and also in the delay memory. Right?

Mode and field definitions

These definitions are used by set_field and set_mode. Their use is strictly limited to these two commands. These two commands are commands to control the internals of lower. The set_field command is also used to create command stream stuff AND for internal stuff. Specifically, size_buffer, size_commands and packing_mode are used to control the size of the code buffer, the size of the command stream buffer and the packing mode for numbers.

Macros for set_field and set_mode

These macros are defined for use by the table which these two procedures call by indirection. Note how there are jrsts to literals. This is because we want to have just one word in the jump table. Note also that co_field sets a field and co_bit sets a bit. A misc_bit (or field) is used to set a field in a so called "miscellaneous" command. See the infamous cram sheet for more details on this as well as timer and ticks, both of which are also command names.

Finally the code... begins with set_mode:

set_mode and set_field are finally explained herein. The first thing that any procedure in lower will do is to check to see that lower has been initialized. This can be done explicitly by the initialize procedure or automatically by the init subroutine in lower. Back to set_mode. If the mode (an index) is greater than the maximum index, then an error is given, otherwise, it (the index) is used as an index into the mode table. The mode table makes use of of the relocate macro. The relocate macro makes the program counter jump to the right place, place the word (called entry) and then sets the maximum (if the offset > current offset then set current offset to offset). Then there is a reloc back to where the pc was. This macro was designed because one upon a time, Pete Samson threatened that the machine was in a great state of flux and could be changed at any time. That is why lower has all these hacks which permit things to be moved around as much as possible. Finally, after all the relocates have been given, a final relate is given to move the pc up to the end of the table. Set_field is also very much like set_mode.

The get procedure!

Here is the well known and beloved get procedure. Besides initing, the
first thing to do is to check the id against id_maximum. Note that
since DGL complained, it became necessary to add a field for the sum_memory.
SO the first thing we do is to mask off this field. Assuming the id is
ok, then we branch through the ldbtbl. If it isn't a sum_memory, then
it must be either a generator, a modifier or a delay unit. These are called
"normal". If it is normal, then we check to make sure the field isn't contaminated.
Then we get the next device by searching linearly thru the table (SKIPing
the first one!). If no free one is found, then get will return a -1.
On the other hand, if the argument was sum_memory, then we use that field
to get the proper quadrant. Note that since the field is already left shifted
by 3, we only need to shift by 3 (to make 6, or 64). The table function_table
is kept around in case anyone needs it in the future.

Give, Decode and Relative

Give is used to give back an identifier. It should be obvious.

Decode is used to tell what kind of identifier an identifier is. Relative returns the relative number of the identifier. I hope the code is obvious. But in case it isn't, the identifier is compared against the ranges of identifiers. When it is found, then it is subtracted from the bottom of this range to give the relative number of the identifier.

The fabulous bind procedure...

Here in all of its magnificence, is the bind procedure. It is pretty straight forward until the tables. The tables consist of relocate macros which have as arguments the offset (the argument given to bind) and the location of a macro which handles that type of bind. The maximum bind field is determined by the end of a table-the beginning of a table. The bind_field procedure is used to bind an argument in a specific packing mode. It does this by temporarily changing the packing mode and calling bind and then changing the packing mode back again.

Initialize, Getptr, Newptr, Chksm and Box error

Initialize (also known as init) initializes the lower level routines by clearing out the data area (from data_begin to data_end). It also resets all the pointers to various records and so forth. Finally, it sets inited, a boolean variable so that the skipn inited will be skipped and the pushj p,init also skipped.

Getptr gets a pointer to the code stream. Newptr gets a pointer to the command stream.

Length, which called by box_error, computes the length of a asciz string. This is used by box_error to make a SAIL string for usererr to gobble. The pushing of the 0s in box_error is used to create a Usererr(0,0,message) call. Note that if the user has specified a different error handler, then box_error will call that procedure with that string INSTEAD of calling USERERR.

Chksm is used by procedures in bind to check that the argument is indeed a sum_memory address. If it isn't, it will complain, believe me.

Flush and friends

Flush should be called at the end of any program which makes use of the lower level routines. Flush will flush out any of the remaining buffers as well as clobbering the dual pointer tables so that no attempt is made to share a code word that is already used!

Load delay

Load delay is used by the user to load up a delay table with data. At the time of this writing, it is still a bit incomplete.

Set (error, procedure, channel and output)

These routines are used to set things internal to lower. They set:

Set_error sets this block of accumulators to the current contents of all the acs. It also saves the pc of the return. It is possible to return to exactly where you were by use of this procedure.

Set_procedure is used by users who want to handle their own box errors. Unfortunately, the present way of handling errors is strictly through error messages, so it would be necessary to write code that decoded the strings. Perhaps this can be changed sometime. I envision a procedure which knows enough about box errors to change whatever was at fault and continue.

Set_channel is used to set the output channel for the disk. It has to be opened in mode '17. This too could probably be changed to mode '13.

Set_output is used to set the output channel for the command stream. Any comments that apply to set_channel apply to set_output. These two should probably have better names.

Macros for unpacked commands

These macros are used for the creation of code words for unpacked commands. By unpacked, we mean commands which are not able to share a word with another command. I should note at this point the strategy used in the creation of these macros: When lower was first written, I thought "geez, this should run as fast as possible, let's trade space for time". In retrospect, this was probably an error. If I had patience, I rewrite this portion of lower to use procedure calls and probably push all this stuff on the stack and call a procedure. Or something. Anyway, the way ALL of the macros I will discuss work is the following: The first thing to do is to get a pointer to the command list. Note that this process may automagically flush the buffer if the buffer pointer exceeds the length of the buffer. In any case, then we take the pointer (which was carefully packed so that it could be used in immediate mode) and make a pointer without an address. We then take the address of the word in the buffer and place it in the right half. We then use the pointer definitions defined in earlier pages to set various fields/bits.

g_loose sets a word for a generator. m_loose does the same for a modifier and d_loose does guess what for a delay unit.

Macros for loading large fields

g_load and m_load are used for loading numbers like frequency, which are often longer than the 20 bit field. The problem with the 20 bit field is solved by a hack called the extension register. The extension register is loaded with the HIGH ORDER x-20 bits (where x is the length of the field) and then a MISC command is put out to load the DX register. Then the appropriate load command is emitted. Note that some of this depends on whether the left_justified or full_word mode is turned on.

The procedure bit_check is used to see if there are any bits on the left hand side of a word. (The left hand side is defined as the 36-32 = 4 bits on the very left). If so a warning is emitted. This can be left out by removing the call to it in the _load macros.

The packed macros are used for commands which have 2 commands in them. g_packed and m_packed are used to create such commands. The call to the macro takes 4 arguments: the name of the opcode, the name of the left and right field and a flag telling us which field to set. If the x hand side of a dual pointer (where x is left if the command wanted is right and vice versa) is non zero and our side of the dual pointer is zero, then there is a word in the code buffer which we can share this command with. If not, then we get a new buffer pointer and do the thing as tho there wasn't a command there.

The generator bind routines call these macros. See the macros expand. See the macros generate gobs of code. My My. See the run modes. Aren't they different? You bet. They work in the folowing manner: They use pointers to the proper field to set the various modes. Then they generate a mode command. Note that we save the mode away. This is so when we change the mode in any way, we still have the other modes that the mode is packed in the same. Right? Right.

The Delay bind procedures are just like modifiers/generators. We even save the mode for the delay mode just like we do for generators and modifiers.

The End!

Yes, that's it. There ain't no more except for the patch space. This is for the unlikely (I hope!) event that there are bugs in lower that need to be patched. Or better yet, you can use it to write patches for your buggy programs! Good luck! (and bon appetit!)

∂22-Sep-78 2207 DGL RAIDing LOWER
 ∂21-Sep-78 2204 KS RAIDing LOWER
The next time you feel the need to RAID LOWER, you might like to try the following technique on Bind:

Put a breakpoint at the JRA R,(R) instruction in Bind. You can find it easily by stepping thru til you get there.
Display ac's A, B, C, D, and R using <cntl>; to keep them around.
While pointing at R, type <cntl><meta>] to get a dynamic display of the word pointed to by the left half of R. Both R and that word should be displayed as half-words (use <cntl>h for that).
While pointing at the dynamically displayed word, type <cntl><meta>] again, to get the word pointed to by THAT word's left half. This should be displayed as text (with <cntl>t).

Now everytime you hit your breakpoint, ac's A thru D will tell you about the arguments for that call; R will tell you what routine you are going to use (in the right half); the word pointed to by the left half of R will tell you what parameter discriptor block you are going to use (with names like %GO, %GMODE, %ML1, etc.); and the word pointed to by the left half of THAT will tell you the (English-like) name of the parameter being twiddled.

With all that information at your disposal, debugging ought to be easy; with all that organization and style, debugging ought to be unnecessary!

-- Ken

File under: R Date: 17 Jul 1979 Name: Robert Poor

Name: Robert Poor

Project: 1 Programmer: ROB

File Name: LOWER.FAI[SAM,MUS]

File Last Written: 11:38 14 Jul 1979

Time: 7:35 Date: 17 Jul 1979

Stanford University
Artificial Intelligence Laboratory
Computer Science Department
Stanford, California

```
COMMENT @   VALID 00043 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00005 00002    title   BOXASS  Low Level code assembler for the SC box
C00007 00003    Internal/external symbols
C00011 00004    Random definitions of common interest
C00013 00005    Basic structures and constants
C00022 00006    Opcode definitions
C00026 00007    Generator definitions
C00028 00008    Generator pointers
C00031 00009    Modifier definitions
C00033 00010    Modifier pointers
C00035 00011    Pointers for CONOs, Misc, Timer and Ticks (and Delay!)
C00037 00012    Opcode definitions and modes for CONO, Misc, Timer and Ticks (and Delay!)
C00039 00013    Arguments for the bind procedure
C00040 00014    Mode and field definitions and stream fields too
C00042 00015    Macros for the set_mode and set_field procedures
C00045 00016    Set_mode and Set_field procedures
C00050 00017    set_interrupt and set_code
C00052 00018    Get procedure
C00055 00019    Give and Decode and Relative
C00057 00020    PutCmd - put an arbitrary word in code stream
C00058 00021    Bind procedure (bind_field too)
C00061 00022    Vectors to routines and parameters (Bind)
C00065 00023     Parameter descriptor blocks
C00067 00024     GENERATOR parameters
C00069 00025     MODIFIER parameters
C00071 00026     DELAY parameters
C00072 00027    EASY routine - for simple cases
C00074 00028    LONG routine - for paremeters using DX
C00077 00029    SHARED routine - for parameters with friends
C00079 00030    ZEROED routine - for parameters having clear bit
C00082 00031    Generator bind routines
C00084 00032    Modifier bind routines
C00086 00033    Delay bind routines
C00089 00034    GetPtr, NewPtr
C00091 00035    Flush and its subroutines
C00095 00036    Chksm, Smquad
C00097 00037    Init
C00099 00038    Error handling - Box_error, Length, Set_procedure
C00102 00039    - TooBig, BadMem, NotMem, BadDly
C00106 00040    - errcrlf, errstr, erroct, errdec, errend
C00109 00041    - errlst
C00112 00042    Stream procedures - Set_output and Set_stream and Unset_output
C00117 00043    The End!
C00118 ENDMK
C@;
```

```
title   BOXASS  Low Level code assembler for the SC box
ENTRY SET.FI,INITIA,UNSET.,GET,PUTCMD,BIND,GIVE,BIND.F,FLUSH,RELATI
ENTRY SET.OU,SET.MO,DECODE,PASS,SET.PR
subttl  Basic definitions

.insert samdef[six,mus]         ; message definitions for the box

;History:
; Major revision of Bind by KS. JUN-78
; Various bug fixes and gratuitous modifications by KS. NOV-77 and after
; ∂ This is the corrected version, by DGL, 10/9/77, from the new spec sheet.
; comments throughout of the changes are preceeded by "∂".

; This program is dedicated to J Brown and the all night movies!

; ACs

f ← 0
a ← 1
b ← 2
c ← 3
d ← 4
t ← 5
a1 ← 6
a2 ← 7
a3 ← 10
a4 ← 11
fp ← 12
r ← 13
x ← 14
z ← 15
sp ← 16
p ← 17

; PDP-6 switch: removes all UUO calls

ifndef nouuo,<nouuo ← 0>

; used for error calls

%erstr←←0
%erchr←←1
%eroct←←2
%erdec←←3
%ercrlf←←4
%erpar←←5
%ertyp←←6
```

```
SUBTTL Internal/external symbols

; Exciting symbols to pass to the loader for SAIL programs to call

comment ⊗
external simple procedure initialize;
external simple integer procedure get(integer id,number(-1));
external simple procedure give(integer id);
external simple procedure putcmd(integer command);
external simple procedure bind(integer src,op,dest);
external simple procedure bind_field(integer src,op,dest,type);
external simple procedure set_output(integer stream,sail_channel);
external simple procedure unset_output(integer stream,sail_channel);
external simple procedure set_procedure(reference procedure p);
external simple procedure set_mode(integer name);
external simple procedure set_field(integer name,field);
external simple procedure set_stream(integer stream,field,value);
external simple procedure set_interrupt(boolean enable; integer cause);
external simple procedure set_code(integer code);
external simple procedure flush;
external simple integer procedure decode(integer unit);
external simple integer procedure relative(integer unit);
external integer array unit_generators[0:671];
external integer array generators[0:255];
external integer array modifiers[0:127];
external integer array gen_this_pass_sum_memory[0:63];
external integer array gen_last_pass_sum_memory[0:63];
external integer array mod_this_pass_sum_memory[0:63];
external integer array mod_last_pass_sum_memory[0:63];
external integer array all_sum_memory[0:255];
external integer array delays[0:31];
external integer update_tick;
external integer pass;
⊗

        external usererr

        internal initialize,get,give,bind,set_procedure,unset_output ;,set_error
        internal bind_field,unit_generators,generators,modifiers,all_sum_memory
        internal delays,set_output,gen_this_pass_sum_memory,mod_this_pass_sum_memory
        internal flush,set_mode,set_field,update_tick,relative,set_stream
        internal pass,decode,gen_last_pass_sum_memory,mod_last_pass_sum_memory
        internal set_code,set_interrupt
```

```
SUBTTL Random definitions of common interest

; Field macro definitions, shifted right to save a memory reference

define ptr(name,size,lsb) { name ←← <point size,0,lsb>●-=18 }
define bit(name,lsb) { name ←← <point 1,0,lsb>●-=18 }

; Identifier opcode definitions
; Passed to GET to specify element type

id_generator ←← 0
id_modifier ←← 1
id_sum_memory ←← 2
id_delay ←← 3
id_maximum ←← 4

; sub fields (Only allowed for sum_memory)

last_gen_pass ←← 0
last_mod_pass ←← 10
this_mod_pass ←← 20
this_gen_pass ←← 30

; Basic field definitions
; Pointers for data fields, opcode fields and element numbers for all element
; types that take commands: generators, modifiers and delay units.

ptr(g_data,20,23)
ptr(g_op,4,27)
ptr(g_number,8,35)

ptr(m_data,20,23)
ptr(m_op,5,28)
ptr(m_number,7,35)

ptr(d_data,16,19)
ptr(d_whole_data,20,23)
ptr(d_op,2,30)
ptr(d_number,5,35)

; Bias definitions, used for MERGE prog.

bias_generator ←← 0
bias_modifier ←← bias_generator+=256
bias_sum_memory ←← bias_modifier+=128
bias_delay ←← bias_sum_memory+=256
bias_end ←← bias_delay+=32
```

```
subttl  Basic structures and constants

; These are the famed unit generators. They are considered used
; when a non zero number is stored within them

define array & (address,size,lower,upper) {
        .+6
        address
        lower
        upper
        1                       ; one dimensional arrays
        1,,size
}

unit_generators:
        array(u_gen,=256+=128+=32+=256,0,=256+=128+=32+=256-1)
generators:
        array(b_gen,=256,0,=256-1)      ; generator allocation
modifiers:
        array(b_mod,=128,0,=128-1)      ; modifier allocation
delays:
        array(b_del,=32,0,=32-1)        ; delay allocation

all_sum_memory:
        array(b_sum,=256,0,=256-1)
gen_last_pass_sum_memory:
        array(b_sum,=64,0,=64-1)
mod_last_pass_sum_memory:
        array(b_sum+=64,=64,0,=64-1)
mod_this_pass_sum_memory:
        array(b_sum+=64+=64,=64,0,=64-1)
gen_this_pass_sum_memory:
        array(b_sum+=64+=64+=64,=64,0,=64-1)

; These are SAIL array pointers which can be passed by the empty
; buffer routine in flush.

        array(codbuf,codsiz,0,codsiz-1)
code_array:
        array(combuf,comsiz,0,comsiz-1)
cmd_array:

data_begin ← .                  ; begin of data area

u_gen:                          ; unit generators (all four of them)
b_gen:  block =256              ; generators
b_mod:  block =128              ; modifiers
b_sum:  block =256              ; sum memory locations
b_del:  block =32               ; delays
max_unit ← .-b_gen

; These are the shared command arrays and the zeroed command arrays.  The
;  arrays are half-word arrays, with entries being pointers into the command
;  (everywhere else in this shit called code!) buffer.  The idea here is to
;  use multiple function commands as much as possible (if optimizing).  The
;  zeroed command arrays have a share array in the left half for the command
;  which has a clear bit for the associated parameter; the right half points
;  to any known use of a non-zero load of the parameter.  A zero entry in any
;  of these arrays means no known use of the command exists.  These arrays are
;  indexed by unit number (i.e. generator, modifier, or delay unit number).
;  To insure that the re-ordering of parameter loads effected by optimization
```

```
;  is safe, all arrays are cleared when the update ticks for a pass are exhausted.
;  Consequently we can further optimize by never loading a parameter more than
;  once during a single pass.

dp_begin ← .
gnm:    glsum:  block =256      ; left is GN/GM  right is GL/GSUM
gmodfm: gkclr:  block =256      ; left is GMODE/GFM  right is GK(clearable)
mmdscsm:ml0clr: block =128      ; left is MMODE/MSCALE/MSUM  right is L0(clearable)
mrmin:  ml1clr: block =128      ; left is MRM/MIN  right is L1(clearable)
dxy:    dyclr:  block =32       ; left is DLY X  right is DLY Y(clearable)
dzp:            block =32       ; left is DLY Z/P  right is unused
dp_end ← .-1

; Miscellaneous storage

mode_g: block =256              ; generator modes
scl_m:  block =128              ; modifier scales
mode_d: block =32               ; delay unit modes
sum_g:  block =256              ; generator sum memory locations
sum_m:  block =128              ; modifier sum memory locations

; the next blocks are organized by pairs so they can be referenced by using
; the offsets code_stream and command_stream

codsiz ← =1024                  ; maximum instruction buffer size
comsiz ← =64

codbuf: block   codsiz          ; code buffer
combuf: block   comsiz          ; command buffer

bufcnt:
codcnt: block   1               ; count of words in code buffer
comcnt: block   1               ; count of words in command command buffer

bufmax:
codmax: block   1               ; real buffer size (can be set with set_stream)
commax: block   1               ; real command buffer size

bufptr:
codptr: block   1               ; code buffer pointer
comptr: block   1               ; command buffer pointer

bufprc:
codprc: block   1               ; procedure to empty code buffer
comprc: block   1               ; procedure to empty command buffer

outc:   block   1               ; out channel,olist for instruction flush
olist:  block   1               ; iowd words,buffer
        0                       ; terminator for instruction list

packmode: block 1               ; Packing: left or right justified or fullword
optim:  block   1               ; optimize instruction packing
acblk:  block   =16             ; where all the acs go when set_error is called
savblk: block   =16             ; where to temporarily save acs

data_end ← .-1                  ; end of data area

inited: block   1               ; basic init done
errorl: block   1               ; where to go to when errors happen

buffer: codbuf                  ; buffer pointers
```

```
        combuf

outchn: codchn                  ; pointers to the buffers...
        comchn
codchn: block   =16             ; channels for set_output (code channels)
comchn: block   =16             ; channels for set_output (command channels)
outcnt:
codcct: block   1               ; code stream count
comcct: block   1               ; command stream count

bufary: code_array              ; code array
        cmd_array               ; command array

; indirect table for set_stream

strmtb: buffer                  ; buffer pointers
        bufmax                  ; maximum buffer count
        bufptr                  ; current pointer
        bufprc                  ; buffer flusher

pass:   block   1               ; pass count
update_tick:
        block   1               ; update tick count
max_update_ticks:
        block   1               ; update ticks per pass (total-processing)
proc_ticks:
        block   1               ; process ticks per pass

; Maximum identifier for each unit

        -1                      ; this one's for decode
        -1                      ; trick to make it all come out in the wash (Get)
max_id: =256-1                  ; generator
        =256+=128-1             ; modifier
        =256+=128+=256-1        ; sum memory
        =256+=128+=256+=32-1    ; delay memory

; definitions for set_output procedure

code_stream ← 0                         ; code offset
command_stream ← 1                      ; command offset
max_stream ← 1

; flag for devchr

f_disk ← 200000                 ; it's a disk!
```

```
subttl  Opcode definitions

; Timer and Ticks opcode definitions

op_timer ←← 2                   ; TIMER
op_ticks ←← 3                   ; TICKS

REPEAT 0,<      ; most of these are unused now - only TIMER and TICKS used
; generator opcode definitions (placed in g_op)

op_sweep ←← 13                  ; set sweep
op_frequency ←← 04              ;  "  frequency
op_angle ←← 11                  ;  "  angle
op_ncosines ←← 07               ;  "  number of cosines
op_scale ←← 07                  ;  "  scale
op_rate ←← 06                   ;  "  rate of decay
op_exponent ←← 02               ;  "  decay exponent
op_asymptote ←← 10              ;  "  asymptote
op_gsum ←← 10                   ;  "  sum memory address
op_fm ←← 12                     ;  "  fm address
op_gmode ←← 12                  ;  "  mode

; generator opcodes as seen by g_op field (all the same for syntax)

o_sweep ←← op_sweep
o_frequency ←← op_frequency
o_angle ←← op_angle             ;  "  angle
o_ncosines ←← op_ncosines       ;  "  number of cosines
o_scale ←← op_scale             ;  "  scale
o_rate ←← op_rate               ;  "  rate of decay
o_exponent ←← op_exponent       ;  "  decay exponent
o_asymptote ←← op_asymptote     ;  "  asymptote
o_gsum ←← op_gsum               ;  "  sum memory address
o_fm ←← op_fm                   ;  "  fm address
o_gmode ←← op_gmode             ;  "  mode

; modifier opcode definitions (placed in m_op)

op_m ←← 30                      ; M0/M1
op_m_0 ←← 30                    ; M0 (Right justified)
op_m_1 ←← 31                    ; M1 (Right justified)
op_m01 ←← 32                    ;∂M0 left-ajusted, low bits from left of DX clear DX
op_m11 ←← 33                    ;∂M1 left-ajusted, low bits from left of DX clear DX
op_l ←← 34                      ; L0/L1
op_l_0 ←← 34                    ; L0
op_l_1 ←← 35                    ; L1
op_in ←← 37                     ; a input
op_rm ←← 37                     ; b input
op_msum ←← 36                   ; set sum memory address
op_mmode ←← 36                  ;  "  mode

; modifier opcodes as seen by g_op

o_m ←← op_m⊛-1                  ; M0/M1
o_m_0 ←← op_m_0⊛-1              ; M0 Right justified)
o_m_1 ←← op_m_1⊛-1              ; M1 Right justified)
o_m01 ←← op_m01⊛-1              ;∂M0 left-ajusted, low bits from left of DX clear DX
o_m11 ←← op_m11⊛-1              ;∂M1 left-ajusted, low bits from left of DX clear DX
o_l ←← op_l⊛-1                  ; L0/L1
o_l_0 ←← op_l_0⊛-1              ; L0
o_l_1 ←← op_l_1⊛-1              ; L1
```

```
o_in ←← op_in⊛-1                ; a input
o_rm ←← op_rm⊛-1                ; b input
o_msum ←← op_msum⊛-1            ; set sum memory address
o_mmode ←← op_mmode⊛-1          ;  "  mode

; the only delay opcode

op_delay ←← 1                   ; set a delay unit

; delay opcode as seen by g_op

o_delay ←← op_delay⊛-1          ; set a delay unit

>;END REPEAT 0
```

```
subttl  Generator definitions

; Generator modes

runlsb ←← 6
runmask ←← 17@runlsb

g_inactive ←← 0@runlsb                  ;       No      No      No
g_pause ←← 1@runlsb                     ;       No      No      No
a_running ←← 17@runlsb                  ;       Yes     Yes     Yes
b_running ←← 16@runlsb                  ;       Yes     No+     Yes
g_wait ←← 11@runlsb                     ;       Yes     No      No
c_running ←← 15@runlsb                  ;       Yes     Yes+    Yes
data_read ←← 7@runlsb                   ;                       Yes
data_write ←← 3@runlsb                  ;                       No
dac_write ←← 2@runlsb                   ;                       No

; Generator envelope modes

envlsb ←← 4
envmask ←← 3@envlsb
min_envelope ←← 0@envlsb
max_envelope ←← 3@envlsb

1plusq ←← 1@envlsb                      ; 1+q           ∂ was 0
1minusq ←← 0@envlsb                     ; 1-q           ∂ was 1
1expplus ←← 3@envlsb                    ; 1+2↑(-q)      ∂ was 2
1expminus ←← 2@envlsb                   ; 1+2↑(-q)      ∂ was 3

; Oscillator modes

osclsb ←← 0
oscmask ←← 17@osclsb                    ; ∂ was 7
min_oscillator ←← 0@osclsb
max_oscillator ←← 10@osclsb             ; ∂ was 5

sine ←← 0@osclsb                        ; sin(K)                ∂ was 4
sawtooth ←← 1@osclsb                    ; sawtooth
square ←← 2@osclsb                      ; square
pulse_train ←← 3@osclsb                 ; pulse train
sum_of_cosines ←← 4@osclsb              ; sum of cosines        ∂ was 0
sin_fm ←← 10@osclsb                     ; sin(J + fm)           ∂ was 5
```

```
subttl Generator pointers

REPEAT 0,<              ; mostly unused
; pointers to generator fields

ptr(p_exponent,20,23)           ; decay exponent (Q)
ptr(p_rate,20,23)               ; decay rate (P)
ptr(p_frequency,20,23)          ; oscillator frequency (J)
ptr(p_ncosines,11,19)           ; number of cosines (N)
ptr(p_scale,4,23)               ; scale of sines or sum of cosines (M)
ptr(p_asymptote,12,17)          ; asymptote (L)
ptr(p_gsum,6,23)                ; sum memory write address (SUM)
ptr(p_angle,20,23)              ; oscillator angle (K)
ptr(p_gmode,10,16)              ; oscillator mode (MODE)
ptr(p_fm,7,23)                  ; FM (FM)
ptr(p_sweep,20,23)              ; sweep frequency or memory address (O)
>;END REPEAT 0

ptr(p_g_run_mode,4,29)          ; run_mode field in mode - unbiased |
ptr(p_g_envelope,2,31)          ; envelope field too - unbiased     |(in mode_g)
ptr(p_g_osc_mode,4,35)          ; oscillator mode - unbiased        |

REPEAT 0,<
; pointers to important bits

bit(exp_just,27)                ; decay exponent adjustified left
bit(freq_just,27)               ; frequency adjustified left
bit(no_ncosines,4)              ; Don't load ncosines (N)
bit(no_n,4)                     ; Don't load ncosines (N)
bit(no_scale,5)                 ; Don't load scale (M)
bit(no_m,5)                     ; Don't load scale (M)
bit(no_asymptote,4)             ; Don't load asymptote (L)
bit(no_l,4)                     ; Don't load asymptote (L)
bit(no_gsum,5)                  ; Don't load sum (SUM)
bit(no_gmode,4)                 ; Don't load mode (MODE)
bit(no_fm,5)                    ; Don't load fm (FM)
bit(k_clear,6)                  ; clear angle (K)
>;END REPEAT 0
```

```
subttl  Modifier definitions

; Function definitions

fnlsb ←← 0                              ;DGL - was 4
fnmask ←← 37@fnlsb
fn_minimum ←← 0
fn_maximum ←← 35

m_inactive ←← 0@fnlsb                   ; inactive
u_noise ←← 2@fnlsb                      ; uniform noise ∂ was 1
tr_u_noise ←← 3@fnlsb                   ; triggered uniform noise
latch ←← 4@fnlsb                        ; latch
threshold ←← 6@fnlsb                    ; threshold ∂ was missing
delay ←← 7@fnlsb                        ; delay

notwopoles ←← 10@fnlsb                  ; two poles; no variables
two_0poles ←← 11@fnlsb                  ; two poles; M0 variable
two_1poles ←← 13@fnlsb                  ; two poles; M1 variable

notwozeroes ←← 14@fnlsb                 ; two zeroes; no variables
two_0zeroes ←← 15@fnlsb                 ; two zeroes; M0 variable
two_1zeroes ←← 17@fnlsb                 ; two zeroes; M1 variable

int_mixing ←← 20@fnlsb                  ; integer mixer
one_pole ←← 21@fnlsb                    ; one pole
mixing ←← 24@fnlsb                      ; mixer
one_zero ←← 26@fnlsb                    ; one zero
four_quad_multiply ←← 30@fnlsb          ; four quadrant multiply ∂ was 31
am ←← 31@fnlsb                          ; amplitude modulator ∂ was 30
maximum ←← 32@fnlsb                     ; maximum ∂ was 33
minimum ←← 33@fnlsb                     ; minimum ∂ was 32
signum ←← 34@fnlsb                      ; signum function
zero_crossing_pulser ←← 35@fnlsb        ; zero crossing pulser
```

```
subttl  Modifier pointers

REPEAT 0,<
ptr(p_m,20,23)                  ; M0/M1 field
ptr(p_m_0,20,23)                ; M0 field
ptr(p_m_1,20,23)                ; M1 field
ptr(p_l,20,23)                  ; L0/L1 field
ptr(p_l_0,20,23)                ; L0 field
ptr(p_l_1,20,23)                ; L1 field
ptr(p_mmode,5,12)               ; Modifier mode (MODE)
ptr(p_msum,7,23)                ; Modifier sum (SUM)
ptr(p_rm,8,15)                  ; B data address (RM)
ptr(p_in,8,23)                  ; A data address (IN)
ptr(p_mscale,4,16)              ; A and B scale (SCALE)
>;END REPEAT 0

ptr(p_a_scale,2,33)             ; A scale - unbiased   |(in scl_m)
ptr(p_b_scale,2,35)             ; B scale - unbiased   |

REPEAT 0,<
; Modifier bits

bit(no_mmode,4)                 ; Don't load mode
bit(no_msum,5)                  ; Don't load sum
bit(l0_clear,6)                 ; Clear L0
bit(no_mscale,7)                ; Don't load AABB (scale) bits of MMODE
bit(no_rm,4)                    ; Don't load rm
bit(no_in,5)                    ; Don't load in
bit(l1_clear,6)                 ; Clear L1
bit(m_just,27)                  ; M0/M1 left justified
bit(m_select,28)                ; M0 if off; M1 if on
>;END REPEAT 0
```

```
subttl  Pointers for CONOs, Misc, Timer and Ticks (and Delay!)

; CONO-A pointers

ptr(p_control,2,19)             ; Processor control ð 2,21
bit(p_rtc,20)                   ; Reset tick counter ð 22
ptr(p_spt,2,24)                 ; (Set) Inhibit/Permit Processing ticks ð 2,23
ptr(p_drb,7,31)                 ; Diagnostic ReadBack ð 25
bit(p_mr,32)                    ; Master Reset
ptr(p_pia,3,35)                 ; Priority Interrrupt Assignment

; CONO-B pointers

ptr(p_re,2,30)                  ; Reset Error ð 2,30
ptr(p_int,1,31)                 ; Interrupt enable/disable
ptr(p_cause,4,35)               ; Cause number

; MISC pointers

ptr(misc_data,20,23)            ; misc. data
ptr(misc_op,2,30)               ; misc. opcode ð 2,32
bit(misc_wait_clear,33)         ; clear all waits
bit(misc_pause_clear,34)        ; clear all pauses
bit(misc_stop,35)               ; stop!

; TIME pointers

ptr(time_data,20,23)            ; timer data
ptr(time_op,2,32)               ; timer opcode

; TICK pointers

ptr(tick_data,10,23)            ; tick data
bit(tick_op,32)                 ; tick operation

; Delay pointers

ptr(p_d_mode,4,35)              ; Delay mode - unbiased              |(data) |
ptr(p_d_scale,4,31)             ; Scaling factor for table lookup - unbiased |
ptr(p_d_size,16,31)             ; Size of delay line-1 - unbiased            |
```

```
subttl  Opcode definitions and modes for CONO, Misc, Timer and Ticks (and Delay!)

; CONO-A

c_stop ←← 1
c_start ←← 2
c_step ←← 3

ena_ticks ←← 1
dis_ticks ←← 2

; CONO-B

bad_linger ←← 1
mix_overflow ←← 2
mult_overflow ←← 3
msum_overflow ←← 4
gsum_overflow ←← 5
w_exhausted ←← 6
r_32_data ←← 7
c_exhausted ←← 10
r_exhausted ←← 16

; Misc opcodes

dx ←← 1
ttla ←← 2
ttlb ←← 3
f0 ←← 0
f1 ←← 1
f2 ←← 2
f3 ←← 3
unfiltered ←← 4
same_filter ←← 10

; Ticks opcodes

tix_processing ←← 0
tix_total ←← 1

; Timer opcodes

pass_set ←← 1
lounge ←← 2
pass_clear ←← 3

; Delay modes (for p_d_mode when in size/mode)

d_inactive ←← 0
delayline ←← 10
table_lookup ←← 12
round_table_lookup ←← 13

; delay mode definitions (placed in d_mode)

d_base ←← 0                     ; base address
d_index ←← 1                    ; index
d_size ←← 2                     ; size of delay unit
d_mode ←← 2                     ; or the mode/scale
d_scale ←← 2                    ; delay scaling factor
```

```
subttl  Arguments for the bind procedure

; Generator definitions

sum_memory ←← 0
osc_mode ←← 1
mode ←← 2
sweep ←← 3
frequency ←← 4
angle ←← 5
ncosines ←← 6
scale ←← 7
rate ←← 10
exponent ←← 11
asymptote ←← 12
fm ←← 13
run_mode ←← 14
envelope ←← 15

; Modifier definitions

; sum_memory ←← 0
add_sum_memory ←← 0
function ←← 1
; mode ←← 2
coeff0 ←← 3
coeff1 ←← 4
term_0 ←← 5
term_1 ←← 6
a_in ←← 7
b_in ←← 10
a_scale ←← 11
b_scale ←← 12
replace_sum_memory ←← 13
invoke_delay_unit ←← 14

; Delay defintions

base_address ←← 0
; mode ←← 2
delay_length ←← 3
index ←← 4
; scale ←← 6
```

```
subttl  Mode and field definitions and stream fields too

; Mode definitions

reset_tick_counter ←← 0                 ; CONO-A
inhibit_processing_ticks ←← 1           ; CONO-A
permit_processing_ticks ←← 2            ; CONO-A
reset ←← 3                              ; CONO-A
wait_clear ←← 4                         ; MISC
pause_clear ←← 5                        ; MISC
stop ←← 6                               ; MISC
optimize ←← 7                           ; set by set_mode
non_optimize ←← 10                      ; likewise

; Field definitions

control_mode ←← 0                       ; CONO-A
PIA ←← 1                                ; CONO-A
diagnostic_address ←← 2                 ; CONO-A
dx_load ←← 3                            ; MISC
ttl_load ←← 4                           ; MISC
processing_ticks ←← 5                   ; TICKS
total_ticks ←← 6                        ; TICKS
dwell ←← 7                              ; TIMER
set_passes ←← 10                        ; TIMER
clear_passes_dwell ←← 11                ; TIMER
packing_mode ←← 14                      ; internal

; packing modes (for packing_mode)

full_word ←← 0
left_justified ←← 1
right_justified ←← 2

; stream fields

address ←← 0                            ; buffer address
size_buffer ←← 1                        ; buffer size
; ptr_buffer ←← 2                               ; buffer pointer
flusher ←← 3                            ; user flusher
max_sfield ←← flusher

; End of Definitions
```

```
subttl  Macros for the set_mode and set_field procedures

define  co_field & (name,field,exeunt) {
jrst    [
        movei   a,field
        a_cono(name,exeunt)
        ]
}

define  co_bit & (name,exeunt) {
jrst    [
        seto    a,
        a_cono(name,exeunt)
        ]
}

define  a_cono & (name,exeunt) {
        push    p,a             ; save in case
        pushj   p,newptr        ; get a word
        movei   c,MCONOA        ; conoa message type
        movem   c,@b
        pushj   p,newptr        ; get pointer to the command buffer
        setzm   @b              ; to be sure
        hrli    b,p_&name       ; in the proper place
        pop     p,a             ; recover
        dpb     a,b
        jrst    exeunt
}

define  misc_field & (op,exeunt) {
jrst    [
        pushj   p,getptr
        hrli    b,misc_data
        dpb     a,b
        movei   a,op            ; MISC opcode (RR field)
        hrli    b,misc_op
        dpb     a,b
        jrst    exeunt
        ]
}

define  misc_bit & (field,exeunt) {
jrst    [
        pushj   p,getptr
        seto    a,
        hrli    b,misc_&field
        dpb     a,b
        setz    a,
        hrli    b,misc_op
        dpb     a,b
        jrst    exeunt
        ]
}

define  timer & (op,exeunt) {
jrst    [
        setzm   update_tick
        pushj   p,getptr
        hrli    b,time_data
        dpb     a,b
        movei   a,op
```

```
        hrli    b,time_op
        dpb     a,b
        movei   a,op_timer
        hrli    b,m_op
        dpb     a,b
        jrst    exeunt
        ]
}
define  ticks & (op,exeunt) {
jrst    [
        pushj   p,getptr
        hrli    b,tick_data
        dpb     a,b
        movei   a,op
        hrli    b,tick_op
        dpb     a,b
        movei   a,op_ticks
        hrli    b,m_op
        dpb     a,b
        jrst    exeunt
        ]
}
```

```
subttl  Set_mode and Set_field procedures

set_mode:
        skipn   inited          ; have we been thru once
        pushj   p,init          ; the first time!
        move    a,-1(p)         ; pick up mode index
        caig    a,max_mode-1
        skipge  a
        jrst    [
                movei   a1,[asciz /Set_mode: Mode index out of range/]
                pushj   p,box_error
                jrst    .+1
                ]
        xct     modes(a)
mode_exit:
exit2:  sub     p,[2,,2]
        jrst    @2(p)           ; return

define max(w,y) {
ifle <w-y>,{w ← y}
}

define relocate (offset,entry) {
        reloc   .+offset
        entry
        max(count,offset)
        reloc
}

        count ← 0
modes:  relocate(reset_tick_counter,<co_bit(rtc,mode_exit)>)
        relocate(inhibit_processing_ticks,<co_field(spt,dis_ticks,mode_exit)>)
        relocate(permit_processing_ticks,<co_field(spt,ena_ticks,mode_exit)>)
        relocate(reset,<co_bit(mr,mode_exit)>)
        relocate(wait_clear,<misc_bit(wait_clear,mode_exit)>)
        relocate(pause_clear,<misc_bit(pause_clear,mode_exit)>)
        relocate(stop,<misc_bit(stop,mode_exit)>)
        relocate(optimize,setom optim)
        relocate(non_optimize,setzm optim)
        reloc   .+count+1

max_mode ← .-modes

set_field:
        skipn   inited          ; same old rigamarole
        pushj   p,init
        move    b,-2(p)
        caig    b,max_field-1
        skipge  b
        jrst    [
                movei   a1,[asciz /Set_field: Field index out of range/]
                pushj   p,box_error
                jrst    .+1
                ]
        move    a,-1(p)         ; new field parameter
        xct     fields(b)
f_exit: sub     p,[3,,3]
        jrst    @3(p)

fields: count ← 0
        relocate(control_mode,<jrst [a_cono(control,f_exit)]>)
```

```
        relocate(PIA,<jrst [a_cono(pia,f_exit)]>)
        relocate(diagnostic_address,<jrst [a_cono(drb,f_exit)]>)
        relocate(dx_load,<misc_field(dx,f_exit)>)
        relocate(ttl_load,<
        jrst    [
                push    p,a
                hlrzs   a               ; left half is ttla
                misc_field(ttla,ttlrtn)
        ttlrtn: pop     p,a
                hrrzs   a               ; right half is ttlb
                misc_field(ttlb,f_exit)
                ]>)
        relocate(processing_ticks,<
        jrst    [
                movem   a,proc_ticks
                aos     b,proc_ticks    ; arg. is tick #, 1st tick is #0.
                move    c,total_ticks
                sub     c,b
                subi    c,=8            ; overhead ticks
                movem   c,max_update_ticks
                ticks(tix_processing,f_exit)
                ]>)
        relocate(total_ticks,<
        jrst    [
                move    b,a
                addi    b,2             ; first tick is #0, arg is HiTick-1, so...
                movem   b,total_ticks
                move    c,proc_ticks
                sub     b,c
                subi    b,=8            ; overhead ticks
                movem   b,max_update_ticks
                ticks(tix_total,f_exit)
                ]>)
        relocate(dwell,<
        jrst    [
                movem   a,pass
                movei   t,2
                movem   t,update_tick
                pushj   p,clrdp
                timer(lounge,f_exit)
                ]>)
        relocate(set_passes,<
        jrst    [
                movem   a,pass
                timer(pass_set,f_exit)
                ]>)
        relocate(clear_passes_dwell,<
        jrst    [
                movei   t,2
                movem   t,update_tick
                movem   a,pass
                pushj   p,clrdp
                timer(pass_clear,f_exit)
                ]>)
        relocate(packing_mode,<movem a,packmode>)
        reloc   .+count+1
max_field ←← .-fields
```