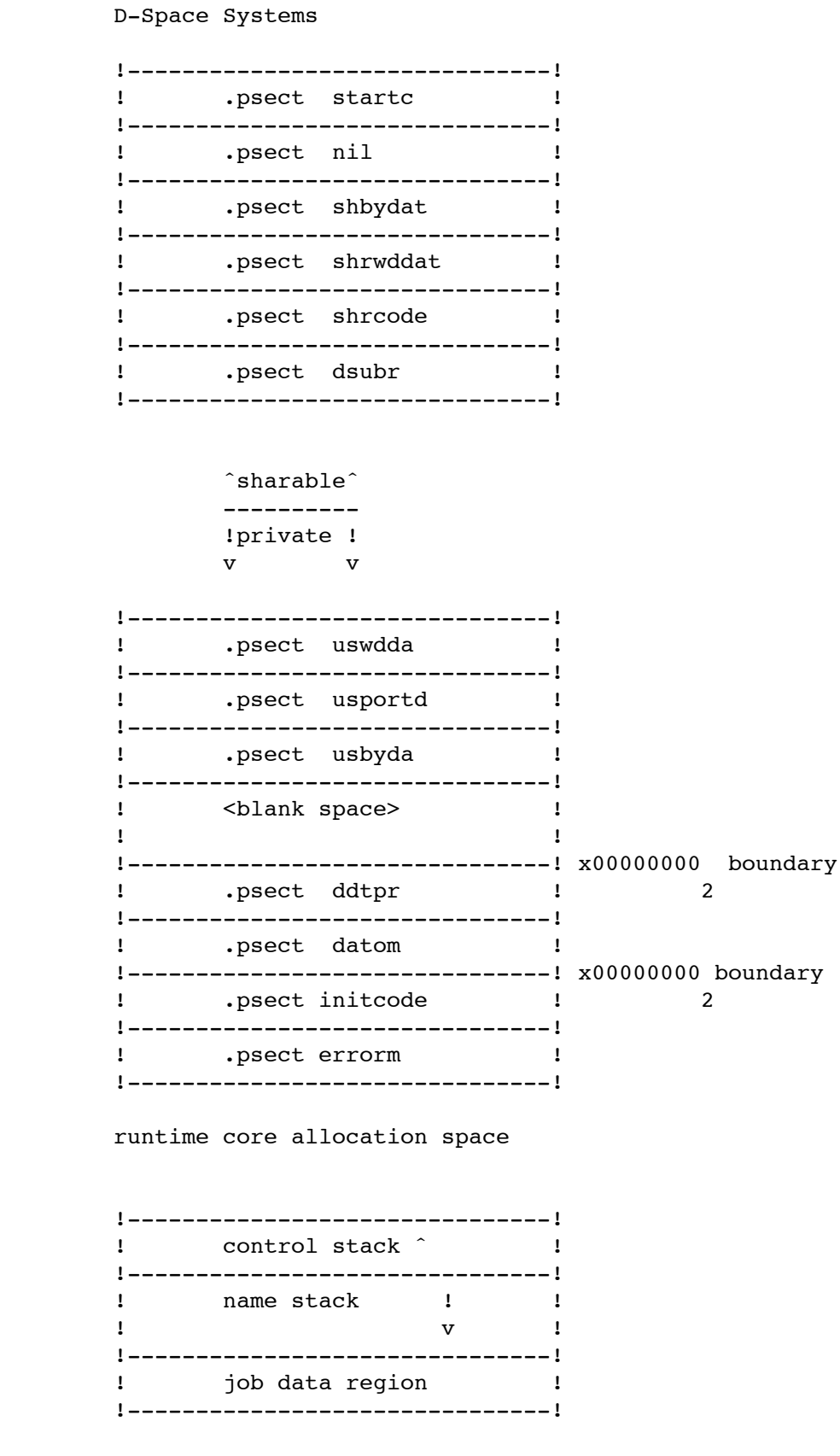
PDP-11 L110
System Description

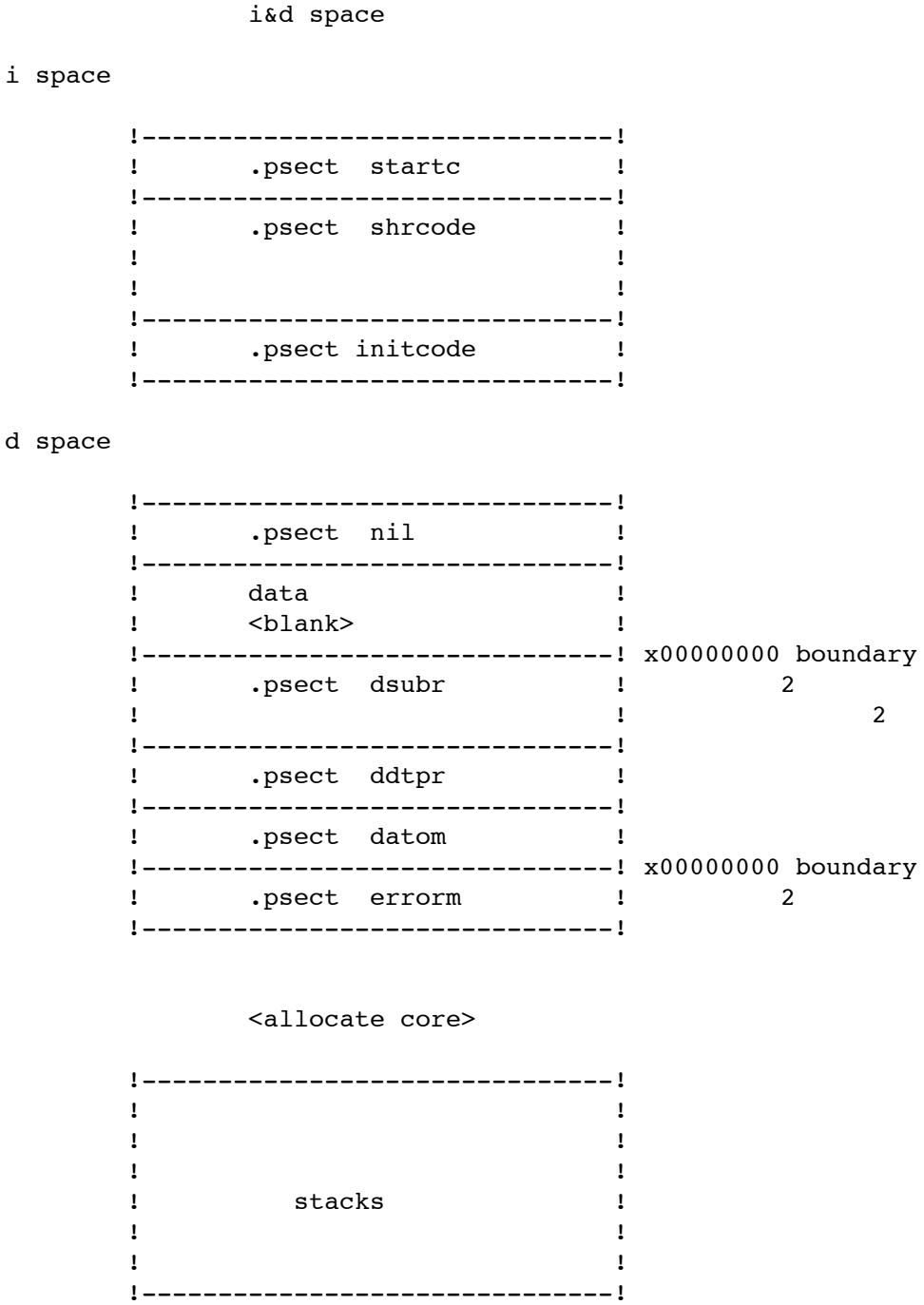
Forrest Howard

5/28/75

Core Allocation



smallints also occupy the last 1280. bytes with the stacks.



note all the data is non-sharable
smallints occupy the highest 1280. locations

One Page is 400 Octal Bytes.

As more core is needed by L110 for port buffers, or pages for atoms , ints, or dtprs, the routine GLOBALLOC searches out core for blank pages (such as those returned from ports when closed) and requests more core from UNIX as needed.

The monitor automatically keeps track of the control stack and allocates more core to it as needed.

The data region expands upwards (i.e. from 0 to 177776) in 400 Octal chunks.

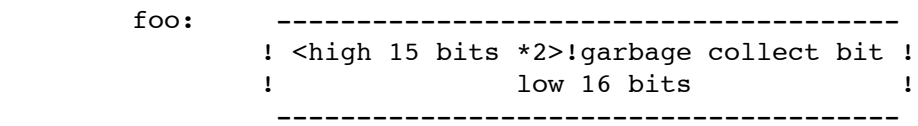
The control stack expands down (towards 0).

One should note that the psects "initcode" and "errorm" go away. Errorm has the text for the error message file; this file is written out when a "pure" lisp (one that has not had a SAVEME done on it) is called with more than one arguement (that is with more than just its name as arg). The algorithim for this writing is to do a creat on the errorfil path name; if this succeds, then write the entire psect into it, and then close it. Obviously, this requires write permission on the errorfile.

The initcode section is thrown away in data-space only lisp by being overwritten by data that is added. In i&d lisp, it is eliminated from the file by the SAVEME function.

Data Objects

INTEGERS--2 16 bit words as follow:

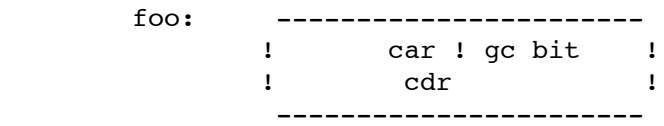


The instruction ASL x is used to clear the GC bit when forming an INTEGER; V bit will be set if the number cannot be represented.

The instruction ASR x will get the original number back; the sign bit is retained.

As an assembly switch Small Integers can also be selected. These have an address above -1280. Note that the address determines the value, not the contents. For this reason, all integer manipulation should be done thruogh the "useful numeric macros" (see below).

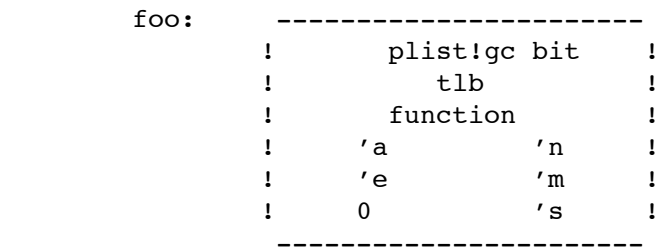
DOTTED PAIRS--2 16 bit words.



Note that all addresses in L110 are even, so that the 0 bit of the car is never set, save by the garbage collector. The 0 bit of the cdr is always 0.

Dotted Pairs always start on 0 or 4 byte boundaries.

ATOMS--length 3+<<length of print string>+1>/2 16 bit words



Plist is similar to car of dtpr.

Tlb is similar to cdr of dtpr.

Binary#Code

(For D Space)

The Length of Binary Code is Dependent on t;he number of instructions in it.

If Type is 0 if Lambda and 1 if Nlambda then:

```
foo:     ------------------------------------------------
        !       Type*100000+Args*1000+<fooend-foo> !gc  !
        !                        anil                   !
        !       jsr     pc,chas(chanl)                  !
        !                                               !
        !                    <code>                     !
        !                                               !
fooend: !               last word of code               !
         ------------------------------------------------
```

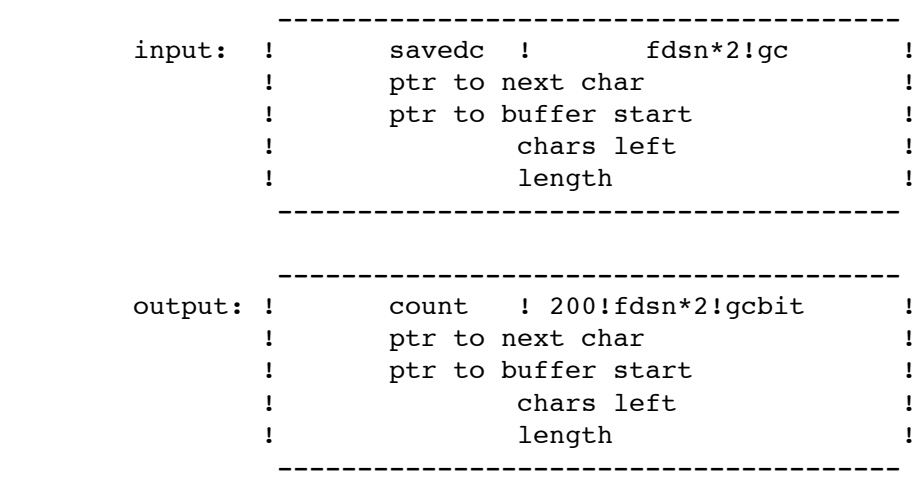
(For I&D Space)

Length is always 3 16 bit words:

```
foo:     ------------------------------------------------
        !       type*100000+args*1000+!gc               !
        !                        anil                   !
        !       pointer to instruction space           !
         ------------------------------------------------
```

Note that the Binary Code object is always in data space, although it may point to instruction space.

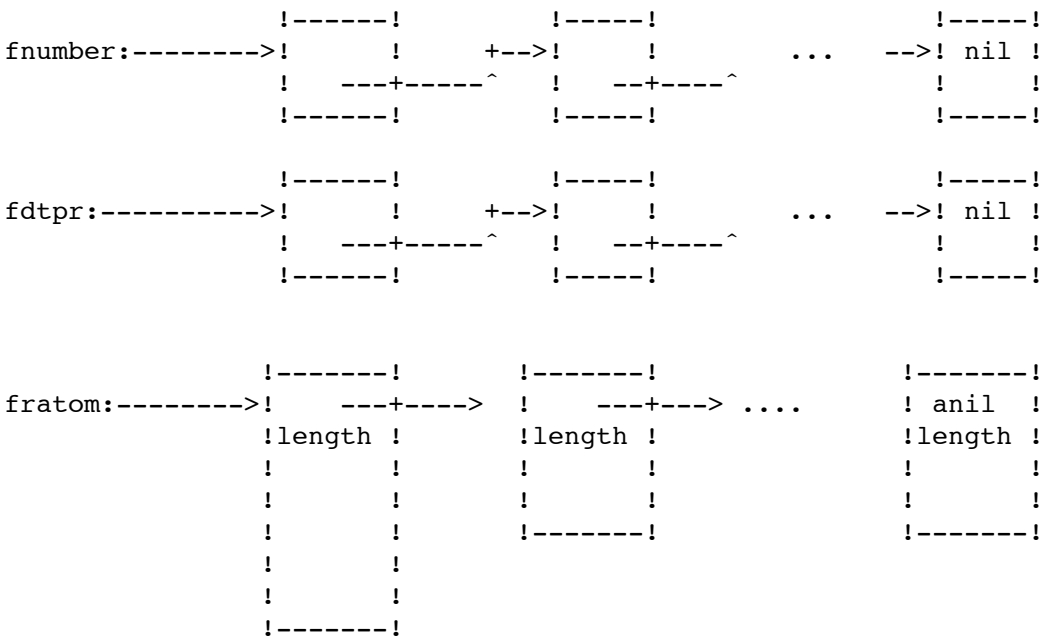
Ports--5 16 bit words



Where count in output buffer is number of characters since last Terpr.

Savedc in input buffer is the character saved by savec.

Free list formats



For i&d space, bcd free size is ignored as it is all 3 words.

All lengths are in words.

Telling what things are

Each data object has a type code:

0	int
1	dtpr
2	atom
3	bcd
4	port

In addition, there are several other types:

-1	system code (not used)
-2	i-o buffer
-3	free page owned by us
-4	stack space
-5	owned by monitor

There is a code associated with each 400 Octal byte page of which there are 400 Octal).

This type information is kept in the QMAP, which is 400 bytes long. To get the type of the address 1000 in register j1, for example, then

```
Ldtype  #1000,j1
```

expands as:

```
mov     #1000,j1
clrb    j1
swab    j1
movb    qmap(j1),j1
```

Note that due to the index, the second arguement to ldtype must be a register.

If the object whose type we wish to know is already in a register, and we want to clobber the very same register with the type code, then

```
Ldtype  Rx
```

will do the trick.

Stacks:

Stack pointers

SP=%6 is the control stack pointer.

NP=%5 is the name stack pointer.

LTOP (a core location) points to top of name stack before current function call.

CONVENTIONS

Namestack entrys are 4 byte entrys that are <atom, form> pairs. NP points to the last form pushed on. NP-2 points to the last atom pushed on. NPLIM is highest possible location to be used by namestack. Name stack grows upwards.

Control stack entrys are of several types. The first is a standard even address, i.e. return addresses, nstack addresses, etc. These are ignored during garbage collection. The second type of entrys are odd addresses. Odd address imply that the address&177776 is to be protected during garbage collection. The user is cautioned not to push spurious odd addresses on the stack. The third entry type is a Function Block:

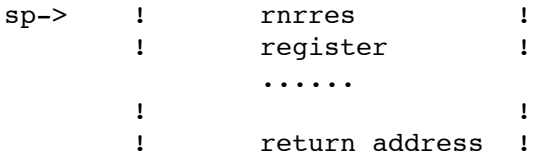
```
        sp->    !  eexit        !
                !  function     !
                !  old ltop     !
                !  form         !
                !  caller       !
```

The form and Function are protected. The fourth type of object is a "saved register" block generated by break:

```
        sp->    !       bksnag  !
                !       np      !
                !       ltop    !
                !       j3      !
                !       j2      !
                !       j1      !
                !       b       !
                !       return  !
```

None of these objects are garbage collected.

other snags recognized have to do with register saving, specifically, the r4rres, r3rres, r2rres, and r1rres snags. their format is



where there are n registers in between the snag and address. The installation of register saves blocks on the stack must be done carefully, least the user find himself with an illegal object on top of the stack. Simarily, in order to take the registers off, care must be exercised.

The prototype for using register snags is:

```
foo:    save3           ;save three registers
         ...
         ...
end:    saveret         ;which is a mov (sp),pc
                        ;which will flush regs + return to my caller
```

Debugging note

As implied earlier, each page for data objects must begin on a 400 byte boundary. When L110 is linked with DDT for debugging, it is likely that the begining of the first page will not be on a 400 byte boundary. Therefor the file filler.m11 is used to pad out the data so that the first atom (atmnil) is in the correct place.

Use of Saveme

The saveme command writes out the current lisp data enviroment to a file in an executable format. This is usefull to produce a copy of Lisp that is already initilized, has the non-primatives loaded, etc.

The function of saveme for data-space only systems is simple;make a file header, write out the text, and write out the data. For I&D space systems, life is a little harder. There are two flavors of output; one for Harvard I&D, the other for Bell I&D. Saveme has to know the difference.

Secondly, saveme cannot directly write out instruction space. Rather, it opens itself for reading. The pathname that it uses is defined in the dstuf.m11 file. It then uses the data-space information and the file to build an executable file, minus the instruction space code for initilization and for saveme itself.

A reset is then executed.

Previous to the saveme, and implicit resetio is preformed.

1) Conversion

a String to Number
Name: strnum
Args: String in strbuf, null terminated.
Return: a/ptr to Int
b/NIL

b Number to String
Name: numstr
Arg: a/ptr to int
Return: a/NIL
b/Pointer to string, null terminated.

c String to Atom
Name: Strat
Args: String in strbuf
b/length of strin in words
j2/hash index
Return: a/pointer to atom
b/garbage

2) iO

all iO routines take a port on top of the name stack
NIL is the port for the tty.

a Get a Character
Name: xgetc
call: getc (macro expanding to call xgetc)
Return: next char of port in char.

b Save a Character
Name: xsavec
Call: savec (macro expanding to call xcavec)
Arg: char in CHAR
Return: nothing

c Output a String
Name: putstr
Arg: ptr to asciz string in B
Return: Nil in b

d Read an Atom
Name: Ratomr
Args: none
Return: a/pointer to form
clobbers all registers!!!!

e Print Number, Atom, Port, Bcd
Names: numout,atmout,portout,bcdout
Arg: a/ptr to object
Return: a,b NIL

3) Form I/O

```
        Read a Form
        Name:   Reader
        Arg:    Port on top of nstk
        Return: a/ptr to form
                b,j1-j3 clobbered


b       Print a Form
        Name:   printr
        Arg:    a/ptr to form
                port on top of np
        Return: a,b NIL

4)      Integer Handlers

a       Numga---Macro
        Arg:    a/ptr to INT
        leaves: High 16 bits in A
                Low 16 bits in B

b       Numgj1

        same as above except leaves result in j1,j2

c       Numga0--Macro
        Arg:    a/ptr to int
        leaves floating representation of int in AC0

d       Numga1
        same as above except leaves result in ac1

e       Nmstore--Macro
        Arg:    a/high 16 bits of int
                b/low  16 bits of int

        Result: a/ptr to int
                b/NIL

f       Nmstac0--Macro
        Arg:    AC0 has floating representation of int
        Result: a/ptr to int

5)      Atom Handlers

a       Get Atom
        Name    Gatom
        Arg     a/number of words needed for string
        Result: a/pointer to atom with NILs in bindings

6)      Dtpr Handlers

a       CONSA--Macro calling
        Name:   xconsa
        Args:   a/car
                b/cdr

        Return: a/pointer to dtpr
```

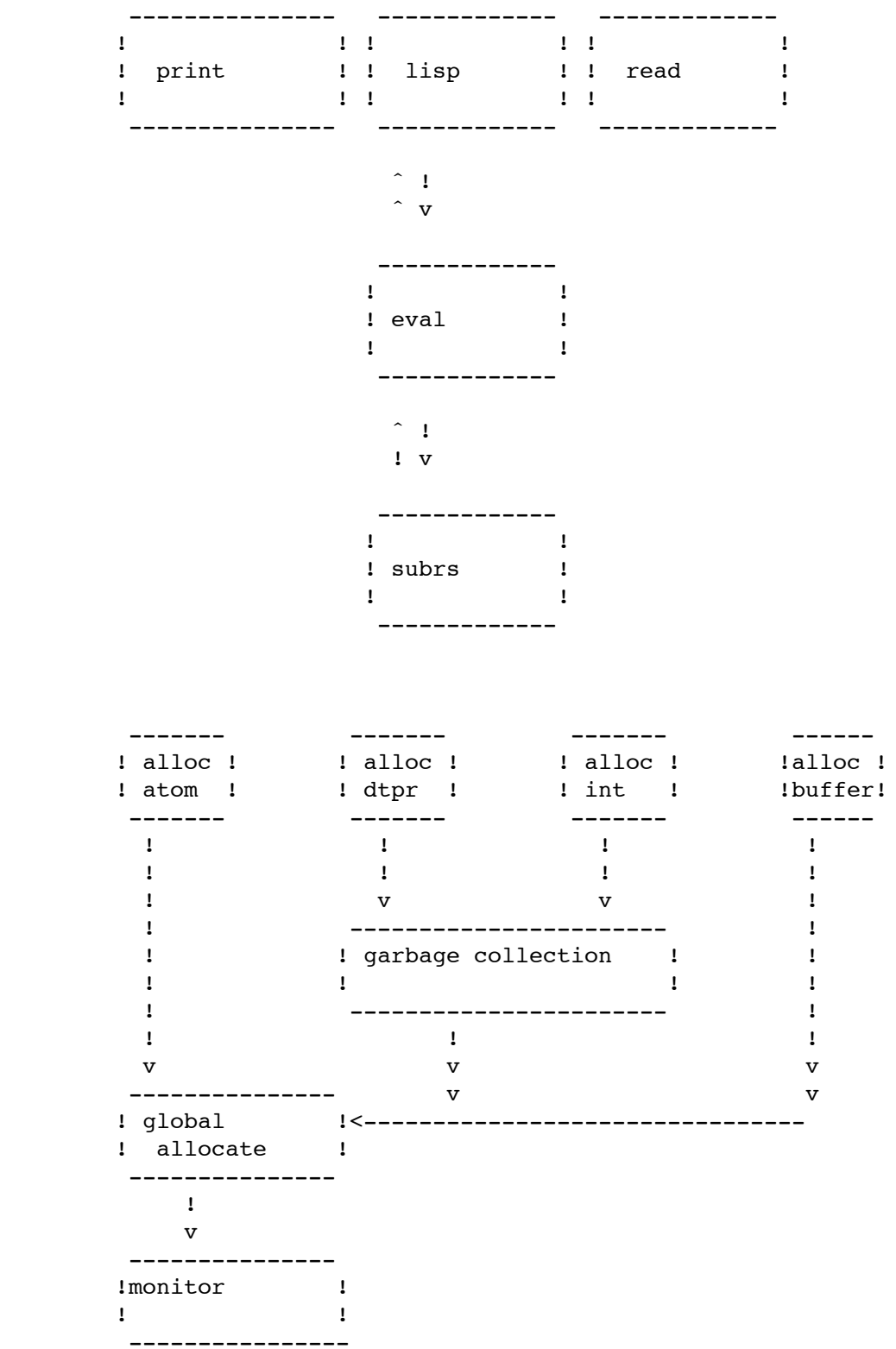
```
        CONSB--Macro calling
        Name:   xconsb
        Args:   a/car
                b/cdr
        Return: b/pointer to dtpr

        CONSBNIL--Macro calling
        Name:   xconsbn
        Args:   a/car
        Return: b/pointer to dtpr with NIL as cdr.

Consa, consb, and consbnil all protect a and b from garbage
collection

d       Allocate Dtpr
        Name:   gdtpr
        Arg:    none
        Return: a/ptr to dtpr
```

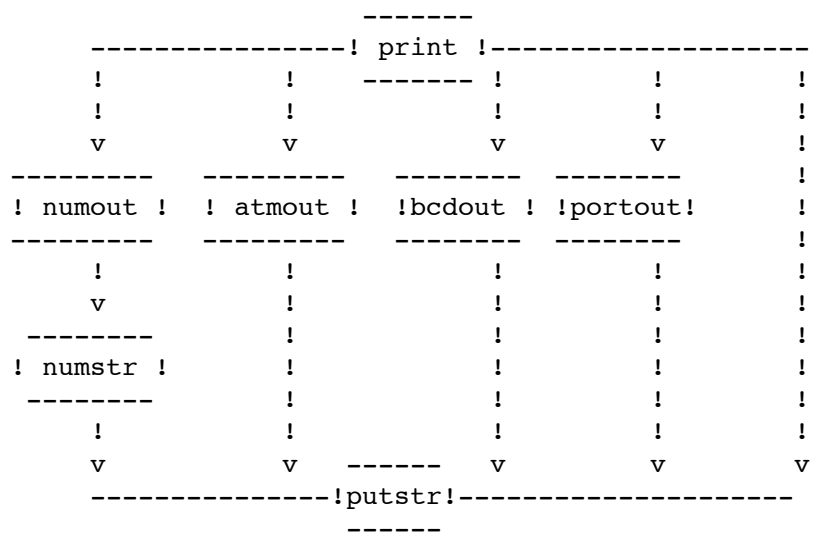
```
Useful Macros

register save

        save1   -- puts j3 in register snag on stack
        save2   -- puts j2+j2 in stack block
        save3   -- j1,j2,j3
        save4   -- b,j1,j2,j3

        saveret --Returns from register save

push    x
        puts x onto control stack

pop     x
        puts top entry of stack into x

npush   x
        pushs <NIL,x> onto stack

npop    x
        pops top entry(form) of stack into x

propush x
        pushs x onto control stack and marks it for garbage collection

unpropop x
        undoes propush and puts result in x

call    x
        jsr     pc,x

ret
        rts     pc

car     x,y
        puts car of what x points to into y

cdr     x,y
        puts cdr of what x points to into y

jmpifinil       x,y,flag
        generates code to test x=nil and branch to y if true.
        flag is for nilas0#0, when previous operation mov
        manipulated x so that condition codes were set

jmpiftrue       x,y
        generates code to test x=true and branch to y if so.

getca
        calls getc, moves char to a

loadnil x
        loads nil into x

retnil
        <loadnil        a
         ret             >
```

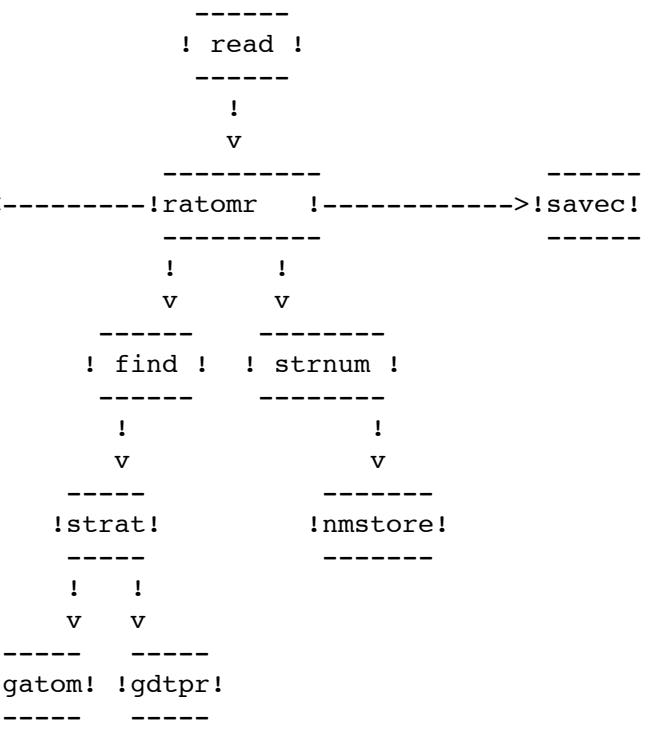
```
rettrue
        <mov    #atrue,a
         ret            >

ldtype y or ldtype x,y
        loads the type of the first arg into y.
        y must be a register

cmptype x,y,z
        using y as scratch generate code to compare the type of
        the first arguement to z.

generm  </delimited string/>

        All sorts of perversions are preformed to get the string
        into errorm psect, from which it will be written to the
        errorfile.  Most importantly, psect changes are made;
        beware therefore for local symbols. The psect eventually
        returned to is the psect of the last ".rsect" (see below).
         A template for using this is:

 noway: generm  </This is my error message/<12>//>
        mov     #tmp-<^pl errorm>,a
        call    geterr                  ;reads the error message file
        call    putstr                  ;return with b pointing
                                        ;to string or asciz
                                        ;string of digits (if
                                        ;file not available)

error   </nasty delimited message/>
error   </nastier delimited message/>,where
        code is generated to call the error package, and return to
        to "where" when continue is evaled (or to cantcont if blank).

rsect sectname
        This is esentally a psect directive, except that it assigns
        a number corresponding to the "sectname" to a variable that
        it keeps. This allows things like generm to leave the cur-
        rent psect, screw around, and then return the user to the
        place that he was in.

dispatch
        generates call to routine which returns:

        dispatch
        jmp     1$      ;if a is int
        jmp     2$      ;if a is dtpr
        jmp     3$      ;if a is atom
        jmp     4$      ;if a is bcd
        jmp     5$      ;if a is port

the 5 instructions following dispatch must be of the two word variety

outstr  x
        generates code to print the string at label x
        to port on top of np

Note!  The following macros will work only(!) if the ctable
        constants are not redefined!!!
```

```
isalph  x,y
        generates code to branch to y if the contents of x(a register)
        is alphabetic (more or less)

isnum   x,y
        generates code ... is -,0,1...9

issep   x,y
        generates code ... is space, tab, cr, lf, ...

isbrk   x,y
        generates code if seperator or (, ), ., [, or ].

isalnum x,y
        same as         isalph  x,y
                        isnum   x,y
```

Control Flow




print
numout
atmout
bcdout
portout
numstr
putstr


read
getc
ratomr
savec
find
strnum
strat
nmstore
gatom
gdtpr