14. Subprimitives

Subprimitives are functions which are not intended to be used by the average program, only by "system programs". They allow one to manipulate the environment at a level lower than normal Lisp. They are described in this chapter. Subprimitives usually have names starting with a % character. The "primitives" described in other sections of the manual typically use subprimitives to accomplish their work. To some extent the subprimitives take the place of what in other systems would be individual machine instructions. Subprimitives are normally hand-coded in microcode.

There is plenty of stuff in this chapter that is not fully explained; there are terms that are undefined, there are forward references, and so on. Furthermore, most of what is in here is considered subject to change without notice. In fact, this chapter does not exactly belong in this manual, but in some other more low-level manual. Since the latter manual does not exist, it is here for the interim.

Subprimitives by their very nature cannot do full checking. Improper use of subprimitives can destroy the environment. Subprimitives come in varying degrees of dangerousness. Those without a % sign in their name cannot destroy the environment, but are dependent on "internal" details of the Lisp implementation. The ones whose names start with a % sign can violate system conventions if used improperly. The subprimitives are documented here since they need to be documented somewhere, but this manual does not document all the things you need to know in order to use them. Still other subprimitives are not documented here because they are very specialized. Most of these are never used explicitly by a programmer; the compiler inserts them into the program to perform operations which are expressed differently in the source code.

The most common problem you can cause using subprimitives, though by no means the only one, is to create illegal pointers: pointers that are, for one reason or another, according to storage conventions, not allowed to exist. The storage conventions are not documented; as we said, you have to be an expert to use a lot of the functions in this chapter correctly. If you create such an illegal pointer, it probably will not be detected immediately, but later on parts of the system may see it, notice that it is illegal, and (probably) halt the Lisp Machine.

In a certain sense car, cdr, rplaca, and rplacd are subprimitives. If these are given a locative instead of a list, they will access or modify the cell addressed by the locative without regard to what object the cell is inside. Subprimitives can be used to create locatives to strange places.
14.1 Data Types

**data-type arg**

data-type returns a symbol that is the name for the internal data-type of the "pointer" that represents arg. Note that some types as seen by the user are not distinguished from each other at this level, and some user types may be represented by more than one internal type. For example, dtp-extended-number is the symbol that data-type would return for either a flonum or a bignum, even though those two types are quite different. The typep function (page 11) is a higher-level primitive that is more useful in most cases; normal programs should always use typep rather than data-type. Some of these type codes are internal tag fields that are never used in pointers that represent Lisp objects at all, but they are documented here anyway.

- **dtp-symbol** The object is a symbol.
- **dtp-fix** The object is a fixnum; the numeric value is contained in the address field of the pointer.
- **dtp-small-flonum** The object is a small flonum; the numeric value is contained in the address field of the pointer.
- **dtp-extended-number** The object is a flonum or a bignum. This value will also be used for future numeric types.
- **dtp-list** The object is a cons.
- **dtp-locative** The object is a locative pointer.
- **dtp-array-pointer** The object is an array.
- **dtp-tef-pointer** The object is a compiled function.
- **dtp-u-entry** The object is a microcode entry.
- **dtp-closure** The object is a closure; see chapter 11, page 180.
- **dtp-stack-group** The object is a stack-group; see chapter 12, page 186.
- **dtp-instance** The object is an instance of a flavor; see chapter 20, page 321.
- **dtp-entity** The object is an entity; see section 11.4, page 185.
- **dtp-select-method** The object is a "select-method"; see page 163.
- **dtp-header** An internal type used to mark the first word of a multi-word structure.
- **dtp-array-header** An internal type used to mark the first word of an array.
- **dtp-symbol-header** An internal type used to mark the first word of a symbol.
- **dtp-instance-header** An internal type used to mark the first word of an instance.
- **dtp-null** Nothing to do with nil. This is used in unbound value and function cells. An attempt to refer to the contents of a cell that contains a dtp-null gets an error. This is how "unbound variable" and "undefined function" errors are detected.
- **dtp-trap** The zero data-type, which is not used. This hopes to detect microcode bugs.
dtp-free
This type is used to fill free storage, to catch wild references.

dtp-external-value-cell-pointer
An "invisible pointer" used for external value cells, which are part of the closure mechanism (see chapter 11, page 180), and used by compiled code to address value and function cells.

dtp-header-forward
An "invisible pointer" used to indicate that the structure containing it has been moved elsewhere. The "header word" of the structure is replaced by one of these invisible pointers. See the function structure-forward (page 203).

dtp-body-forward
An "invisible pointer" used to indicate that the structure containing it has been moved elsewhere. This points to the word containing the header-forward, which points to the new copy of the structure.

dtp-one-q-forward
An "invisible pointer" used to indicate that the single cell containing it has been moved elsewhere.

dtp-gc-forward
This is used by the copying garbage collector to flag the obsolete copy of an object; it points to the new copy.

q-data-types
The value of q-data-types is a list of all of the symbolic names for data types described above under data-type. These are the symbols whose print names begin with "dtp-".
The values of these symbols are the internal numeric data-type codes for the various types.

q-data-types type-code
Given the internal numeric data-type code, returns the corresponding symbolic name.
This "function" is actually an array.

14.2 Forwarding

An invisible pointer is a kind of pointer that does not represent a Lisp object, but just resides in memory. There are several kinds of invisible pointer, and there are various rules about where they may or may not appear. The basic property of an invisible pointer is that if the Lisp Machine reads a word of memory and finds an invisible pointer there, instead of seeing the invisible pointer as the result of the read, it does a second read, at the location addressed by the invisible pointer, and returns that as the result instead. Writing behaves in a similar fashion. When the Lisp Machine writes a word of memory it first checks to see if that word contains an invisible pointer; if so it goes to the location pointed to by the invisible pointer and tries to write there instead. Many subprimitives that read and write memory do not do this checking.

The simplest kind of invisible pointer has the data type code dtp-one-q-forward. It is used to forward a single word of memory to someplace else. The invisible pointers with data types dtp-header-forward and dtp-body-forward are used for moving whole Lisp objects (such as cons cells or arrays) somewhere else. The dtp-external-value-cell-pointer is very similar to the dtp-one-q-forward; the difference is that it is not "invisible" to the operation of binding. If the (internal) value cell of a symbol contains a dtp-external-value-cell-pointer that points to some other word (the external value cell), then symeval or set operations on the symbol will consider
the pointer to be invisible and use the external value cell, but binding the symbol will save away
the dtp-external-value-cell-pointer itself, and store the new value into the internal value cell of
the symbol. This is how closures are implemented.

dtp-gc-forward is not an invisible pointer at all; it only appears in "old space" and will
never be seen by any program other than the garbage collector. When an object is found not to
be garbage, and the garbage collector moves it from "old space" to "new space", a dtp-gc-forward is left behind to point to the new copy of the object. This ensures that other references
to the same object get the same new copy.

structure-forward old-object new-object
This causes references to old-object actually to reference new-object, by storing invisible
pointers in old-object. It returns old-object.

An example of the use of structure-forward is adjust-array-size. If the array is being
made bigger and cannot be expanded in place, a new array is allocated, the contents are
copied, and the old array is structure-forwarded to the new one. This forwarding ensures
that pointers to the old array, or to cells within it, continue to work. When the garbage
collector goes to copy the old array, it notices the forwarding and uses the new array as
the copy; thus the overhead of forwarding disappears eventually if garbage collection is in
use.

follow-structure-forwarding object
Normally returns object, but if object has been structure-forwarded, returns the object
at the end of the chain of forwardings. If object is not exactly an object, but a locative
to a cell in the middle of an object, a locative to the corresponding cell in the latest copy
of the object will be returned.

forward-value-cell from-symbol to-symbol
This alters from-symbol so that it always has the same value as to-symbol, by sharing its
value cell. A dtp-one-q-forward invisible pointer is stored into from-symbol's value cell.
Do not do this while from-symbol is lambda-bound, as the microcode does not bother to
check for that case and something bad will happen when from-symbol gets unbound. The
microcode check is omitted to speed up binding and unbinding.

To forward one arbitrary cell to another (rather than specifically one value cell to
another), given two locatives, do
(%p-store-tag-and-pointer locative1 dtp-one-q-forward locative2)

follow-cell-forwarding loc evec-p
loc is a locative to a cell. Normally loc is returned, but if the cell has been forwarded,
this follows the chain of forwardings and returns a locative to the final cell. If the cell is
part of a structure which has been forwarded, the chain of structure forwardings is
followed, too. If evec-p is t, external value cell pointers are followed; if it is nil they are
not.
14.3 Pointer Manipulation

It should again be emphasized that improper use of these functions can damage or destroy the Lisp environment. It is possible to create pointers with illegal data-type, pointers to non-existent objects, and pointers to untyped storage, which will completely confuse the garbage collector.

%data-type x

Returns the data-type field of x, as a fixnum.

%pointer x

Returns the pointer field of x, as a fixnum. For most types, this is dangerous since the garbage collector can copy the object and change its address.

%make-pointer data-type pointer

This makes up a pointer, with data-type in the data-type field and pointer in the pointer field, and returns it. data-type should be an internal numeric data-type code; these are the values of the symbols that start with dtp-. pointer may be any object; its pointer field is used. This is most commonly used for changing the type of a pointer. Do not use this to make pointers which are not allowed to be in the machine, such as dtp-null, invisible pointers, etc.

%make-pointer-offset data-type pointer offset

This returns a pointer with data-type in the data-type field, and pointer plus offset in the pointer field. The data-type and pointer arguments are like those of %make-pointer; offset may be any object but is usually a fixnum. The types of the arguments are not checked; their pointer fields are simply added together. This is useful for constructing locative pointers into the middle of an object. However, note that it is illegal to have a pointer to untyped data, such as the inside of a FEF or a numeric array.

%pointer-difference pointer-1 pointer-2

Returns a fixnum which is pointer-1 minus pointer-2. No type checks are made. For the result to be meaningful, the two pointers must point into the same object, so that their difference cannot change as a result of garbage collection.

14.4 Analyzing Structures

%find-structure-header pointer

This subprimitive finds the structure into which pointer points, by searching backward for a header. It is a basic low-level function used by such things as the garbage collector. pointer is normally a locative, but its data-type is ignored. Note that it is illegal to point into an "unboxed" portion of a structure, for instance the middle of a numeric array.

In structure space, the "containing structure" of a pointer is well-defined by system storage conventions. In list space, it is considered to be the contiguous, cdr-coded segment of list surrounding the location pointed to. If a cons of the list has been copied out by rplacd, the contiguous list includes that pair and ends at that point.
%find-structure-leader pointer
This is identical to %find-structure-header, except that if the structure is an array with
a leader, this returns a locative pointer to the leader-header, rather than returning the
array-pointer itself. Thus the result of %find-structure-leader is always the lowest
address in the structure. This is the one used internally by the garbage collector.

%structure-boxed-size object
Returns the number of "boxed Q's" in object. This is the number of words at the front
of the structure which contain normal Lisp objects. Some structures, for example FEs and
numeric arrays, contain additional "unboxed Q's" following their "boxed Q's". Note
that the boxed size of a PDL (either regular or special) does not include Q's above the
current top of the PDL. Those locations are boxed, but their contents are considered
garbage and are not protected by the garbage collector.

%structure-total-size object
Returns the total number of words occupied by the representation of object, including
boxed Q's, unboxed Q's, and garbage Q's off the ends of PDLs.

14.5 Creating Objects

%allocate-and-initialize data-type header-type header second-word area size
This is the subprimitive for creating most structured-type objects. area is the area in
which it is to be created, as a fixnum or a symbol. size is the number of words to be
allocated. The value returned points to the first word allocated and has data-type data-
type. Uninterruptibly, the words allocated are initialized so that storage conventions are
preserved at all times. The first word, the header, is initialized to have header-type in its
data-type field and header in its pointer field. The second word is initialized to second-
word. The remaining words are initialized to nil. The flag bits of all words are set to 0.
The cdr codes of all words except the last are set to cdr-next; the cdr code of the last
word is set to cdr-nil. It is probably a bad idea to rely on this.

The basic functions for creating list-type objects are cons and make-list; no special
subprimitive is needed. Closures, entities, and select-methods are based on lists, but there is no
primitive for creating them. To create one, create a list and then use %make-pointer to change
the data type from dtp-list to the desired type.

%allocate-and-initialize-array header data-length leader-length area size
This is the subprimitive for creating arrays, called only by make-array. It is different
from %allocate-and-initialize because arrays have a more complicated header structure.
14.6 Copying Data

%b1t from to count increment
Copies count words, separated by increment. The word at address from is moved to
address to, the word at address from + increment is moved to address to + increment, and
so on until count words have been moved.

%b1t is useful for copying parts of structures, making or deleting space inside structures,
and initializing structures.

Only the pointer fields of from and to are significant; they may be locatives or even
fixnums. If one of them must point to the unboxed data in the middle of a structure,
you must make it a fixnum, and you must do so with interrupts disabled, or else garbage
collection could move the structure after you have already created the fixnum.

14.7 Returning Storage

return-storage object
This peculiar function attempts to return object to free storage. If it is a displaced array,
this returns the displaced array itself, not the data that the array points to. Currently
return-storage does nothing if the object is not at the end of its region, i.e. if it was not
either the most recently allocated non-list object in its area, or the most recently allocated
list in its area.

If you still have any references to object anywhere in the Lisp world after this function
returns, the garbage collector can get a fatal error if it sees them. Since the form that
calls this function must get the object from somewhere, it may not be clear how to legally
call return-storage. One of the only ways to do it is as follows:

(defun func ()
  (let ((object (make-array 100)))
    ...
    (return-storage (progn object (setq object nil))))

so that the variable object does not refer to the object when return-storage is called.
Alternatively, you can free the object and get rid of all pointers to it while interrupts are
turned off with without-interrupts.

You should only call this function if you know what you are doing; otherwise the garbage
collector can get fatal errors. Be careful.
14.8 Locking Subprimitive

%store-conditional pointer old new
This is the basic locking primitive. pointer is a locative to a cell which is uninterruptibly read and written. If the contents of the cell is eq to old, then it is replaced by new and t is returned. Otherwise, nil is returned and the contents of the cell are not changed.

14.9 I/O Device Subprimitives

%unibus-read address
Returns as a fixnum the contents of the register at the specified Unibus address. You must specify a full 18-bit address. This is guaranteed to read the location only once. Since the Lisp Machine Unibus does not support byte operations, this always references a 16-bit word, and so address will normally be an even number.

%unibus-write address data
Writes the 16-bit number data at the specified Unibus address, exactly once.

%xbus-read io-offset
Returns the contents of the register at the specified Xbus address. io-offset is an offset into the I/O portion of Xbus physical address space. This is guaranteed to read the location exactly once. The returned value can be either a fixnum or a bignum.

%xbus-write io-offset data
Writes data, which can be a fixnum or a bignum, into the register at the specified Xbus address. io-offset is an offset into the I/O portion of Xbus physical address space. This is guaranteed to write the location exactly once.

sys:%xbus-write-sync w-loc w-data delay sync-loc sync-mask sync-value
Does (%xbus-write w-loc w-data), but first synchronizes to within about one microsecond of a certain condition. The synchronization is achieved by looping until

 (= (logand (%xbus-read sync-loc) sync-mask) sync-value)

is false, then looping until it is true, then looping delay times. Thus the write happens a specified delay after the leading edge of the synchronization condition. The number of microseconds of delay is roughly one third of delay.

sys:%halt
Stops the machine.
14.10 Special Memory Referencing

%p-contents-offset base-pointer offset
This checks the cell pointed to by base-pointer for a forwarding pointer. Having followed
forwarding pointers to the real structure pointed to, it adds offset to the resulting
forwarded base-pointer and returns the contents of that location.

There is no %p-contents, since car performs that operation.

%p-contents-as-locative pointer
Given a pointer to a memory location containing a pointer that isn’t allowed to be "in the
machine" (typically an invisible pointer) this function returns the contents of the location
as a dtp-locative. It changes the disallowed data type to dtp-locative so that you can
safely look at it and see what it points to.

%p-contents-as-locative-offset base-pointer offset
This checks the cell pointed to by base-pointer for a forwarding pointer. Having followed
forwarding pointers to the real structure pointed to, it adds offset to the resulting
forwarded base-pointer, fetches the contents of that location, and returns it with the data
type changed to dtp-locative in case it was a type that isn’t allowed to be "in the
machine" (typically an invisible pointer). This can be used, for example, to analyze the
dtp-external-value-cell-pointer pointers in a FEF, which are used by the compiled
code to reference value cells and function cells of symbols.

%p-store-contents pointer value
value is stored into the data-type and pointer fields of the location addressed by pointer.
The cdr-code and flag-bit fields remain unchanged. value is returned.

%p-store-contents-offset value base-pointer offset
This checks the cell pointed to by base-pointer for a forwarding pointer. Having followed
forwarding pointers to the real structure pointed to, it adds offset to the resulting
forwarded base-pointer and stores value into the data-type and pointer fields of that
location. The cdr-code and flag-bit fields remain unchanged. value is returned.

%p-store-tag-and-pointer pointer miscfields pntfield
Creates a Q by taking 8 bits from miscfields and 24 bits from pntfield, and stores that
into the location addressed by pointer. The low 5 bits of miscfields become the data-type,
the next bit becomes the flag-bit, and the top two bits become the cdr-code. This is a
good way to store a forwarding pointer from one structure to another (for example).

%p-1db ppss pointer
This is like 1db but gets a byte from the location addressed by pointer. Note that you
can load bytes out of the data type etc. bits, not just the pointer field, and that the word
loaded out of need not be a fixnum. The result returned is always a fixnum.
%p-ldb-offset ppss base-pointer offset
This checks the cell pointed to by base-pointer for a forwarding pointer. Having followed forwarding pointers to the real structure pointed to, the byte specified by ppss is loaded from the contents of the location addressed by the forwarded base-pointer plus offset, and returned as a fixnum. This is the way to reference byte fields within a structure without violating system storage conventions.

%p-dpb value ppss pointer
The value, a fixnum, is stored into the byte selected by ppss in the word addressed by pointer. nil is returned. You can use this to alter data types, cdr codes, etc.

%p-dpb-offset value ppss base-pointer offset
This checks the cell pointed to by base-pointer for a forwarding pointer. Having followed forwarding pointers to the real structure pointed to, the value is stored into the byte specified by ppss in the location addressed by the forwarded base-pointer plus offset. nil is returned. This is the way to alter unboxed data within a structure without violating system storage conventions.

%p-mask-field ppss pointer
This is similar to %p-ldb, except that the selected byte is returned in its original position within the word instead of right-aligned.

%p-mask-field-offset ppss base-pointer offset
This is similar to %p-ldb-offset, except that the selected byte is returned in its original position within the word instead of right-aligned.

%p-deposit-field value ppss pointer
This is similar to %p-dpb, except that the selected byte is stored from the corresponding bits of value rather than the right-aligned bits.

%p-deposit-field-offset value ppss base-pointer offset
This is similar to %p-dpb-offset, except that the selected byte is stored from the corresponding bits of value rather than the right-aligned bits.

%p-pointer pointer
Extracts the pointer field of the contents of the location addressed by pointer and returns it as a fixnum.

%p-data-type pointer
Extracts the data-type field of the contents of the location addressed by pointer and returns it as a fixnum.

%p-cdr-code pointer
Extracts the cdr-code field of the contents of the location addressed by pointer and returns it as a fixnum.
%p-store-pointer  pointer  value
Clobbers the pointer field of the location addressed by pointer to value, and returns value.

%p-store-data-type  pointer  value
Clobbers the data-type field of the location addressed by pointer to value, and returns value.

%p-store-cdr-code  pointer  value
Clobbers the cdr-code field of the location addressed by pointer to value, and returns value.

%stack-frame-pointer
Returns a locative pointer to its caller’s stack frame. This function is not defined in the interpreted Lisp environment; it only works in compiled code. Since it turns into a "mise" instruction, the "caller’s stack frame" really means "the frame for the FHF that executed the %stack-frame-pointer instruction".

14.11 Storage Layout Definitions

The following special variables have values which define the most important attributes of the way Lisp data structures are laid out in storage. In addition to the variables documented here, there are many others that are more specialized. They are not documented in this manual since they are in the system package rather than the global package. The variables whose names start with %q are byte specifiers, intended to be used with subprimitives such as %p-ldb. If you change the value of any of these variables, you will probably bring the machine to a crashing halt.

%q-cdr-code  Variable
The field of a memory word that contains the cdr-code. See section 5.4, page 72.

%q-flag-bit  Variable
The field of a memory word that contains the flag-bit. In most data structures this bit is not used by the system and is available for the user. However, it may soon be reallocated to other purposes.

%q-data-type  Variable
The field of a memory word that contains the data-type code. See page 201.

%q-pointer  Variable
The field of a memory word that contains the pointer address, or immediate data.

%q-pointer-within-page  Variable
The field of a memory word that contains the part of the address that lies within a single page.
**%q-typed-pointer**
The concatenation of the %q-data-type and %q-pointer fields.

**%q-all-but-typed-pointer**
The field of a memory word that contains the tag fields, %q-cdr-code and %q-flag-bit.

**%q-all-but-pointer**
The concatenation of all fields of a memory word except for %q-pointer.

**%q-all-but-cdr-code**
The concatenation of all fields of a memory word except for %q-cdr-code.

**%q-high-half**
**%q-low-half**
The two halves of a memory word. These fields are only used in storing compiled code.

**cdr-normal**
**cdr-next**
**cdr-nil**
**cdr-error**
The values of these four variables are the numeric values that go in the cdr-code field of a memory word. See section 5.4, page 72 for the details of cdr-coding.

### 14.12 Function-Calling Subprimitives

These subprimitives can be used (carefully!) to call a function with the number of arguments variable at run time. They only work in compiled code and are not defined in the interpreted Lisp environment. The preferred higher-level primitive is lexr-funcall (page 27).

**%open-call-block** function n-adi-pairs destination
Starts a call to function. n-adi-pairs is the number of pairs of additional information words already %push'ed; normally this should be 0. destination is where to put the result; the useful values are 0 for the value to be ignored, 1 for the value to go onto the stack, 3 for the value to be the last argument to the previous open call block, and 2 for the value to be returned from this frame.

**%push** value
Pushes value onto the stack. Use this to push the arguments.

**%activate-open-call-block**
Causes the call to happen.

**%pop**
Pops the top value off of the stack and returns it as its value. Use this to recover the result from a call made by %open-call-block with a destination of 1.
%assure-pdl-room n-words
Call this before doing a sequence of %push's or %open-call-blocks that will add n-words to the current frame. This subprimitive checks that the frame will not exceed the maximum legal frame size, which is 255 words including all overhead. This limit is dictated by the way stack frames are linked together. If the frame is going to exceed the legal limit, %assure-pdl-room will signal an error.

14.13 Special-Binding Subprimitive

bind locative value
Binds the cell pointed to by locative to x, in the caller's environment. This function is not defined in the interpreted Lisp environment; it only works from compiled code. Since it turns into an instruction, the "caller's environment" really means "the binding block for the stack frame that executed the bind instruction". The preferred higher-level primitives that turn into this are let (page 17), let-if (page 18), and progv (page 19). [This will be renamed to %bind in the future.]

The binding is in effect for the scope of the innermost binding construct, such as prog or let—even one that binds no variables itself.

14.14 The Paging System

[Someday this may discuss how it works.]

sys:%disk-switches Variable
This variable contains bits that control various disk usage features.

Bit 0 (the least significant bit) enables read-compares after disk read operations. This causes a considerable slowdown, so it is rarely used.

Bit 1 enables read-compares after disk write operations.

Bit 2 enables the multiple page swap-out feature. When this is enabled, as it is by default, each time a page is swapped out, up to 20 contiguous pages will also be written out to the disk if they have been modified. This greatly improves swapping performance.

Bit 3 controls the multiple page swap-in feature, which is also on by default. This feature causes pages to be swapped in groups; each time a page is needed, several contiguous pages are swapped in in the same disk operation. The number of pages swapped in can be specified for each area using s1:set-swap-recommendations-of-area.

s1:set-swap-recommendations-of-area area-number recommendation
Specifies that pages of area area-number should be swapped in in groups of recommendation at a time. This recommendation is used only if the multiple page swap-in feature is enabled.
Generally, the more memory a machine has, the higher the swap recommendations should be to get optimum performance. The recommendations are set automatically according to the memory size when the machine is booted.

si:set-all-swap-recommendations recommendation
   Specifies the swap-in recommendation of all areas at once.

si:wire-page address &optional (wire-p t)
   If wire-p is t, the page containing address is wired-down; that is, it cannot be paged-out.
   If wire-p is nil, the page ceases to be wired-down.

si:unwire-page address
   (si:unwire-page address) is the same as (si:wire-page address nil).

sys:page-in-structure object
   Makes sure that the storage that represents object is in main memory. Any pages that
   have been swapped out to disk are read in, using as few disk operations as possible.
   Consecutive disk pages are transferred together, taking advantage of the full speed of the
   disk. If object is large, this will be much faster than bringing the pages in one at a time
   on demand. The storage occupied by object is defined by the %find-structure-leader
   and %structure-total-size subprimitives.

sys:page-in-array array &optional from to
   This is a version of sys:page-in-structure that can bring in a portion of an array. from
   and to are lists of subscripts; if they are shorter than the dimensionality of array, the
   remaining subscripts are assumed to be zero.

sys:page-in-pixel-array array &optional from to
   Like sys:page-in-array except that the lists from and to, if present, are assumed to have
   their subscripts in the order horizontal, vertical, regardless of which of those two is
   actually the first axis of the array. See make-pixel-array, page 137.

sys:page-in-words address n-words
   Any pages that have been swapped out to disk in the range of address space starting at
   address and continuing for n-words are read in with as few disk operations as possible.

sys:page-in-area area-number
sys:page-in-region region-number
   All swapped-out pages of the specified region or area are brought into main memory.

sys:page-out-structure object
sys:page-out-array array &optional from to
sys:page-out-pixel-array array &optional from to
sys:page-out-words address n-words
sys:page-out-area area-number
sys:page-out-region region-number
   These are similar to the above, except that they take pages out of main memory rather
   than bringing them in. Actually, they only mark the pages as having priority for
   replacement by others. Use these operations when you are done with a large object, to
make the virtual memory system prefer reclaiming that object's memory over swapping something else out.

sys:%change-page-status virtual-address swap-status access-status-and-meta-bits
The page hash table entry for the page containing virtual-address is found and altered as specified. It is returned if it was found, nil if it was not (presumably the page is swapped out). swap-status and access-status-and-meta-bits can be nil if those fields are not to be changed. This doesn't make any error checks; you can really screw things up if you call it with the wrong arguments.

sys:%compute-page-hash virtual-address
This makes the hashing function for the page hash table available to the user.

sys:%create-physical-page physical-address
This is used when adjusting the size of real memory available to the machine. It adds an entry for the page frame at physical-address to the page hash table, with virtual address -1, swap status flushable, and map status 120 (read only). This doesn't make error checks; you can really screw things up if you call it with the wrong arguments.

sys:%delete-physical-page physical-address
If there is a page in the page frame at physical-address, it is swapped out and its entry is deleted from the page hash table, making that page frame unavailable for swapping in of pages in the future. This doesn't make error checks; you can really screw things up if you call it with the wrong arguments.

sys:%disk-restore high-16-bits low-16-bits
Loads virtual memory from the partition named by the concatenation of the two 16-bit arguments, and starts executing it. The name 0 refers to the default load (the one the machine loads when it is started up). This is the primitive used by disk-restore (see page 652).

sys:%disk-save physical-mem-size high-16-bits low-16-bits
Copies virtual memory into the partition named by the concatenation of the two 16-bit arguments (0 means the default), then restarts the world, as if it had just been restored. The physical-mem-size argument should come from %sys-com-memory-size in system-communication-area. This is the primitive used by disk-save (see page 654).

si:set-memory-size nwords
Specifies the size of physical memory in words. The Lisp machine determines the actual amount of physical memory when it is booted, but with this function you can tell it to use less memory than is actually present. This may be useful for comparing performance based on the amount of memory.
14.15 Closure Subprimitives

These functions deal with things like what closures deal with: the distinction between internal and external value cells and control over how they work.

**sys:%binding-instances list-of-symbols**

This is the primitive that could be used by closure. First, if any of the symbols in list-of-symbols has no external value cell, a new external value cell is created for it, with the contents of the internal value cell. Then a list of locatives, twice as long as list-of-symbols, is created and returned. The elements are grouped in pairs: pointers to the internal and external value cells, respectively, of each of the symbols. closure could have been defined by:

```
(defun closure (variables function)
  (%make-pointer dtp-closure
    (cons function (sys:%binding-instances variables)))
```

**sys:%using-binding-instances instance-list**

This function is the primitive operation that invocation of closures could use. It takes a list such as sys:%binding-instances returns, and for each pair of elements in the list, it "adds" a binding to the current stack frame, in the same manner that the bind function (which should be called %bind) does. These bindings remain in effect until the frame returns or is unwound.

sys:%using-binding-instances checks for redundant bindings and ignores them. (A binding is redundant if the symbol is already bound to the desired external value cell.) This check avoids excessive growth of the special pdl in some cases and is also made by the microcode which invokes closures, entities, and instances.

Given a closure, closure-bindings extracts its list of binding instances, which you can then pass to sys:%using-binding-instances.

**sys:%internal-value-cell symbol**

Returns the contents of the internal value cell of symbol. dtp-one-q-forward pointers are considered invisible, as usual, but dtp-external-value-cell-pointers are not; this function can return a dtp-external-value-cell-pointer. Such pointers will be considered invisible as soon as they leave the "inside of the machine", meaning internal registers and the stack.

14.16 Microcode Variables

The following variables' values actually reside in the scratchpad memory of the processor. They are put there by dtp-one-q-forward invisible pointers. The values of these variables are used by the microcode. Many of these variables are highly internal and you shouldn't expect to understand them.
%microcode-version-number

   Variable
   This is the version number of the currently-loaded microcode, obtained from the version
   number of the microcode source file.

sys:%number-of-micro-entries

   Variable
   Size of micro-code-entry-area and related areas.

default-cons-area is documented on page 224.

sys:number-cons-area

   Variable
   The area number of the area where bignums and flonums are consed. Normally this
   variable contains the value of sys:extra-pdl-area, which enables the "temporary storage"
   feature for numbers, saving garbage collection overhead.

current-stack-group and current-stack-group-resumer are documented on page 188.

sys:%current-stack-group-state

   Variable
   The sg-state of the currently-running stack group.

sys:%current-stack-group-calling-args-pointer

   Variable
   The argument list of the currently-running stack group.

sys:%current-stack-group-calling-args-number

   Variable
   The number of arguments to the currently-running stack group.

sys:%trap-micro-pc

   Variable
   The microcode address of the most recent error trap.

sys:%initial-fef

   Variable
   The function that is called when the machine starts up. Normally this is the definition of
   si:lisp-top-level.

sys:%initial-stack-group

   Variable
   The stack group in which the machine starts up.

sys:%error-handler-stack-group

   Variable
   The stack group that receives control when a microcode-detected error occurs. This stack
   group cleans up, signals the appropriate condition, or assigns a stack group to run the
   debugger on the erring stack group.

sys:%scheduler-stack-group

   Variable
   The stack group that receives control when a sequence break occurs.

sys:%chaos-csr-address

   Variable
   A fixnum, the virtual address that maps to the Unibus location of the Chaosnet interface.
%mar-low
A fixnum, the inclusive lower bound of the region of virtual memory subject to the MAR feature (see section 27.13, page 599).

%mar-high
A fixnum, the inclusive upper bound of the region of virtual memory subject to the MAR feature (see section 27.13, page 599).

sys:%inhibit-read-only
If non-nil, you can write into read-only areas. This is used by fasload.

self is documented on page 338.

inhibit-scheduling-flag is documented on page 540.

inhibit-scavenging-flag
If non-nil, the scavenger is turned off. The scavenger is the quasi-asynchronous portion of the garbage collector, which normally runs during consing operations.

sys:scavenger-ws-enable
If this is nil, scavenging can compete for all of the physical memory of the machine. Otherwise, it should be a fixnum, which specifies how much physical memory the scavenger can use: page numbers as high as this number or higher are not available to it.

sys:%region-cons-alarm
Incremented whenever a new region is allocated.

sys:%page-cons-alarm
Increments whenever a new page is allocated.

sys:%gc-flip-ready
t while the scavenger is running, nil when there are no pointers to oldspace.

sys:%gc-generation-number
A fixnum which is incremented whenever the garbage collector flips, converting one or more regions from newspace to oldspace. If this number has changed, the %pointer of an object may have changed.

sys:%disk-run-light
A fixnum, the virtual address of the TV buffer location of the run-light which lights up when the disk is active. This plus 2 is the address of the run-light for the processor. This minus 2 is the address of the run-light for the garbage collector.

sys:%loaded-band
A fixnum, the high 24 bits of the name of the disk partition from which virtual memory was booted. Used to create the greeting message.
**sys:%disk-blocks-per-track**  
*Variable*  
Configuration of the disk being used for paging. Don’t change these!

**sys:%disk-blocks-per-cylinder**  
*Variable*

**sys:%disk-switches** is documented on page 212.

**sys:%qlaryh**  
*Variable*  
This is the last array to be called as a function, remembered for the sake of the function store.

**sys:%qlaryl**  
*Variable*  
This is the index used the last time an array was called as a function, remembered for the sake of the function store.

**%mc-code-exit-vector**  
*Variable*  
This is a vector of pointers that microcompiled code uses to refer to quoted constants.

**sys:currently-prepared-sheet**  
*Variable*  
Used for communication between the window system and the microcoded graphics primitives.

**sys:alphabetic-case-affects-string-comparison** is documented on page 144.

**sys:tail-recursion-flag** is documented on page 33.

**zunderflow** is documented on page 104.

The next four have to do with implementing the metering system described in section 32.2, page 637.

**sys:%meter-global-enable**  
*Variable*  
t if the metering system is turned on for all stack-groups.

**sys:%meter-buffer-pointer**  
*Variable*  
A temporary buffer used by the metering system.

**sys:%meter-disk-address**  
*Variable*  
Where the metering system writes its next block of results on the disk.

**sys:%meter-disk-count**  
*Variable*  
The number of disk blocks remaining for recording of metering information.

**sys:lexical-environment**  
*Variable*  
This is the list of previous stack frames used by lexical-closure.
sys:amem-evcp-vector

This is a vector of shadow locations for all these microcode variables, used in implementing closure-binding of them. The microcode does not check for the presence of external value cell pointers in the microcode locations that these variables correspond to; therefore, when a closure would otherwise try to store an external value cell pointer into one of them, it goes in this vector instead.

background-cons-area is documented on page 224.

sys:self-mapping-table is documented on page 356.

sys:gc-switches

What is this used for?

sys:a-memory-location-names

A list of all of the above symbols (and any others added after this documentation was written).

14.17 Meters

read-meter name

Returns the contents of the microcode meter named name, which can be a fixnum or a bignum. name must be one of the symbols listed below.

write-meter name value

Writes value, a fixnum or a bignum, into the microcode meter named name. name must be one of the symbols listed below.

The microcode meters are as follows:

sys:count-chaos-transmit-aborts Meter

The number of times transmission on the Chaosnet was aborted, either by a collision or because the receiver was busy.

sys:count-cons-work Meter

sys:count-scavenger-work Meter

Internal state of the garbage collection algorithm.

sys:tv-clock-rate Meter

The number of TV frames per clock sequence break. The default value is 67, which causes clock sequence breaks to happen about once per second.

sys:count-first-level-map-reloads Meter

The number of times the first-level virtual-memory map was invalid and had to be reloaded from the page hash table.
sys:%count-second-level-map-reloads  Meter
The number of times the second-level virtual-memory map was invalid and had to be
reloaded from the page hash table.

sys:%count-meta-bits-map-reloads  Meter
The number of times the virtual address map was reloaded to contain only "meta bits", not an actual physical address.

sys:%count-pdl-buffer-read-faults  Meter
The number of read references to the pdl buffer that were virtual memory references that
trapped.

sys:%count-pdl-buffer-write-faults  Meter
The number of write references to the pdl buffer that were virtual memory references that
trapped.

sys:%count-pdl-buffer-memory-faults  Meter
The number of virtual memory references that trapped in case they should have gone to
the pdl buffer, but turned out to be real memory references after all (and therefore were
needlessly slowed down).

sys:%count-disk-page-reads  Meter
The number of pages read from the disk.

sys:%count-disk-page-writes  Meter
The number of pages written to the disk.

sys:%count-fresh-pages  Meter
The number of fresh (newly-consed) pages created in core, which would have otherwise
been read from the disk.

sys:%count-disk-page-read-operations  Meter
The number of paging read operations; this can be smaller than the number of disk pages
read when more than one page at a time is read.

sys:%count-disk-page-write-operations  Meter
The number of paging write operations; this can be smaller than the number of disk
pages written when more than one page at a time is written.

sys:%count-disk-prepages-used  Meter
The number of times a page was used after being read in before it was needed.

sys:%count-disk-prepages-not-used  Meter
The number of times a page was read in before it was needed, but got evicted before it
was ever used.
sys:%count-disk-page-write-waits Meter
The number of times the machine waited for a page to finish being written out in order to evict the page.

sys:%count-disk-page-write-busys Meter
The number of times the machine waited for a page to finish being written out in order to do something else with the disk.

sys:%disk-wait-time Meter
The time spent waiting for the disk, in microseconds. This can be used to distinguish paging time from running time when measuring and optimizing the performance of programs.

sys:%count-disk-errors Meter
The number of recoverable disk errors.

sys:%count-disk-recalibrates Meter
The number of times the disk seek mechanism was recalibrated, usually as part of error recovery.

sys:%count-disk-ecc-corrected-errors Meter
The number of disk errors that were corrected through the error correcting code.

sys:%count-disk-read-compare-differences Meter
The number of times a read compare was done, no disk error occurred, but the data on disk did not match the data in memory.

sys:%count-disk-read-compare-rereads Meter
The number of times a disk read was done over because after the read a read compare was done and did not succeed (either it got an error or the data on disk did not match the data in memory).

sys:%count-disk-read-compare-rewrites Meter
The number of times a disk write was done over because after the write a read compare was done and did not succeed (either it got an error or the data on disk did not match the data in memory).

sys:%disk-error-log-pointer Meter
Address of the next entry to be written in the disk error log. The function si:print-disk-error-log (see page 643) prints this log.

sys:%count-aged-pages Meter
The number of times the page aget set an age trap on a page, to determine whether it was being referenced.
sys:%count-age-flushed-pages  Meter
The number of times the page ager saw that a page still had an age trap and hence made it "flushable", a candidate for eviction from main memory.

sys:%aging-depth  Meter
A number from 0 to 3 that controls how long a page must remain unreferenced before it becomes a candidate for eviction from main memory.

sys:%count-findcore-steps  Meter
The number of pages inspected by the page replacement algorithm.

sys:%count-findcore-emergencies  Meter
The number of times no evictable page was found and extra aging had to be done.

sys:a-memory-counter-block-names  Variable
A list of all of the above symbols (and any others added after this documentation was written).