Interlisp-D Opcodes

```
Written by: Masinter, van Melle, Sybalsky
Stored on [Eris]<LispCore>Internal>Doc>Opcodes.TEdit
RamVersion:        26,24 (recently incremented)
LispVersion:            113400


/************************************************************************/
/*
*/
/*    Copyright 1989, 1990 Venue                                        *
/
/*
*/
/*    This file was work-product resulting from the Xerox/Venue         */
/*    Agreement dated 18-August-1989 for support of Medley.             */
/*
*/
/************************************************************************/
```

UFNs—Handling undefined op-codes

When the microcode (or C emulator) doesn't handle an opcode, it "punts" to the UFN for that opcode: a Lisp function that does what the opcode should do.

To find out what function to call, the microcode looks at a 256-cell block of storage called the "UFN table" (pointed to in Lisp by \UFNTable). The UFN table contains, for each opcode

`(FNINDEX WORD)`	Atom number (really "definition index") of the function to be called.
`(NEXTRA BYTE)`	# of extra bytes to be pushed as argument to the UFN (either 0, 1, or 2).
`(NARGS BYTE)`	# of arguments to call the UFN function with.

The Op-code descriptions

In multibyte opcodes (len-1>0), alpha is byte 1, beta is byte 2, and gamma is byte 3.

`TOS` refers to the argument on the top of the stack; `TOS-1` is arg one back, etc.

`@[x]` is the contents of the word pointed to by `x`.

#	name	len-1	stk level effect	UFN table entry
0	-X-			

used only to denote end of function, never executed.

#	name	len-1	stk level effect	UFN table entry
1	CAR	0	0	\CAR.UFN

If `arg` not `LISTP`
    If `NIL`, return `NIL`
    else call UFN

If cdr code=0, follow indirect pointer. Take value of car field & return it. (Cons cells are 32 bits: first

8 are cdrcode, rest are "carfield")

[required by diagnostics (except car[NIL]); implemented in all ucodes]

#	name	len-1	stk level effect	UFN table entry
2	CDR	0	0	\CDR.UFN

```
If arg not LISTP
      if NIL, return NIL
      else call UFN
```

if cdrcode=0, follow indirect pointer
if cdrcode=200q, return NIL
elseif cdr code gt 200Q, CDR is on same page as cell,
in cell page+2*(cdrcode-200Q)
else CDR is contained in cell at PAGE+2*(cdrcode).
(Cons cells are 32 bits: first 8 are cdrcode, rest are "carfield")

[required by diagnostics (except cdr[NIL]); implemented in all ucodes]

#	name	len-1	stk level effect	UFN table entry
3	LISTP	0	0	LISTP

Return arg if LISTP (NTYPX=LISTPType), else NIL

[required by diagnostics; implemented in all ucodes]

#	name	len-1	stk level effect	UFN table entry
4	NTYPX	0	0	NTYPX

Return type number of arg (right half of word at `MDSTypeTable + [tos rsh 9]`)

[required by diagnostics; implemented in all ucodes]

#	name	len-1	stk level effect	UFN table entry
5	TYPEP	1	0	\TYPEP.UFN

return arg if type=alpha byte, else NIL

[required by diagnostics; implemented in all ucodes; similar to LISTP]

#	name	len-1	stk level effect	UFN table entry
6	DTEST	2	0	\DTEST.UFN

return arg if typename=(alpha,beta), else call UFN or atom number 372 (`\DTESTFAIL`) with tos and (alpha,beta). (typename is word 0 of type's `DTD`; `DTD` is `DTDBase+(type# lsh 4)`)

[required by diagnostics; implemented in all ucodes]

#	name	len-1	stk level effect	UFN table entry
{ 7	CDDR	0	0	CDDR

TAKE CDR Twice [not currently used or implemented or emitted.]}

REPLACED BY :

7	UNWIND	?	?	\UNWIND.UFN

(N is the alpha byte, KEEP is the beta byte) Unwinds the dynamic stack of the current frame to absolute stack depth N, performing any unbinding indicated by bind marks found along the way. If KEEP is 0, the original top of stack is discarded, otherwise it is pushed after unwinding everything else. This opcode is essentially the same as UNBIND or DUNBIND, except that you stop when the stack depth is N, rather than stopping as soon as you have processed the first bind mark.

The stack depth N is measured in cells (doublewords) starting at the base of the pvar region. N=0

means the stack is utterly empty (including the pvar region; i.e., the end of stack pointer (pointer to next stack block) would be the same as PV). Of course, N=0 cannot be used at all in the present architecture, since there is always at least a quadword pad between the frame header and the start of the dynamic stack. If we get rid of that quadword, then N=0 could have meaning in a frame that had an empty pvar region, though that is not true of any closure target, the current sole user of this opcode.

Note that taking the stack depth as alpha byte means this opcode cannot unwind to any deeper than depth 255. For sake of reference, the largest pvar region in Full.sysout is for the function \CURVE, whose pvar region is 92 cells long (59 locals and 32 fvars), which means it could still achieve a dynamic depth of an additional 173 cells before UNWIND would care (it actually never exceeds a depth of 30).

```
{let SP be the stack pointer; i.e., TOS = @SP}
TOP ← loc[pvar0]-2 + 2*N
if KEEP neq 0
   then TEMP ← TOS
until (SP ← SP - 2) = TOP
  do if @SP is bind mark
        then perform its unbinding
if KEEP neq 0
   then push TEMP
```

#	name	len-1	stk level effect	UFN table entry
10	FN0	2	1	
11	FN1	2	0	
12	FN2	2	-1	
13	FN3	2	-2	
14	FN4	2	-3	

call fn (alpha,beta) with N args [required]

#	name	len-1	stk level effect	UFN table entry
15	FNX	3	FNX	

call fn (beta,gamma) with alpha args [required]

#	name	len-1	stk level effect	UFN table entry
16	APPLYFN	0	-1	

call fn (tos) with (tos-1) args after popping tos & tos-1 [required. Right now, it goes to \INTERPRETER if TOS isn't a litatom. May add requirement that will work with code blocks.]

#	name	len-1	stk level effect	UFN table entry
17	CHECKAPPLY*	0	0	\CHECKAPPLY*

If TOS is a literal atom whose definition cell has CCODEP on and ARGTYPE=0 or 2, return it, otherwise call UFN. Note that CHECKAPPLY* is always immediately followed by an APPLYFn. If it would save some time, you might be able to immediately jump to the APPLYFN code. (note: definition cell: bit 0 is CCODEP, bit 1 is "fast" {this fn has empty nametable}, bits 2-3 are ARGTYPE)

[not required; implemented on Dorado]

#	name	len-1	stk level effect	UFN table entry
20	RETURN	0	0	\HARDRETURN

```
do return except when:
      slow bit in returner is on
       and
         returnee usecount not 0
          or
         returners BF usecount is not 0
```

```
 or
returnee not immediately followed by
   a free block
     or
   the basic frame of the returner.
```

In any of those conditions, call UFN or context switch to hardreturn context (which?). [required]

#	name	len-1	stk level effect	UFN table entry
21	BIND	2		

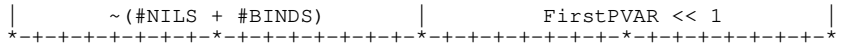
push binding mark, bind variables, popping values of stack. [required]

alpha byte is [#NILS <<4 + #BINDS].

beta byte is [FirstPVAR], which is 1-origin (i.e., 0 is PVAR1?? it looks like --JDS)

BIND takes #BINDS values off the top of stack and binds FirstPVAR and successive PVARs to those values. It then sets the #NILS PVARs beyond that to NIL.

Finally, a “binding mark” is pushed on the top of the stack:



Binding marks are identified on the stack because they’re negative: The high bit is guaranteed to be 1.

#	name	len-1	stk level effect	UFN table entry
22	UNBIND	0		

remember tos, pop until binding mark, unbind variables, push old tos [required]

#	name	len-1	stk level effect	UFN table entry
23	DUNBIND	0	(DUNBIND)	

pop until binding mark, unbind variables [required]

#	name	len-1	stk level effect	UFN table entry
24	RPLPTR.N	1	−1	\RPLPTR.UFN

deleteref value at @(tos-1)+alpha.
addref (tos)
store (TOS) at @(tos-1)+alpha [leave high byte of destination intact]
pop (return (TOS-1)).
If reference count failure, call GCTABLESCAN (atom ????) on punt [not required; in Dorado, 12K]

#	name	len-1	stk level effect	UFN table entry
25	GCREF	1	0	\HTFIND

perform ref count operation on TOS according to alpha byte:
0 - addref (add 1 to reference count)
1 - delref (subtract 1 from reference count)
2 - stkref (turn on "stack reference" bit)

If DELREF causes new refcnt to go to 0 & stk bit off, return arg, else always return NIL. On reference count failure, call UFN (no GCTABLESCAN). [not required; in D0, Dorado]

#	name	len-1	stk level effect	UFN table entry
26	ASSOC	0	−1	ASSOC

if TOS=NIL, return NIL.
if TOS not LISTP, call UFN
if (CAR TOS) not LISTP, call UFN

if TOS-1 = (CAAR TOS), return (CAR TOS)
set TOS_(CDR TOS), reiterate, checking for interrupts

[not required, in 12K]

#	name	len-1	stk level effect	UFN table entry
27	GVAR←	2	0	\SETGLOBALVAL.UFN

Do RPLPTR on VALSPACE+2*(alpha,beta) of TOS [not required; in Dorado, D0. May want to change to UFN if high bit of val cell is on]

#	name	len-1	stk level effect	UFN table entry
30	RPLACA	0	−1	RPLACA

if TOS-1 not LISTP, call UFN
Fetch @[TOS-1].
if cdrcode=0, follow indirect
Do RPLPTR with TOS
pop (return (TOS-1)).

[not required; in Dorado, 12K]

#	name	len-1	stk level effect	UFN table entry
31	RPLACD	0	−1	RPLACD

if tos-1 not listp, call ufn
fetch @ tos-1
if cdrcode=0, follow indirect
if cdrcode<200Q
    rplptr cell+2*cdrcode with tos
elseif TOS is NIL
    if CDRCODE#200, deleteref cell+2*(cdrcode-200)
    change cdrcode to 200
elseif TOS is on same page as cell
    addref TOS
    if cdrcode#200, deleteref cell+2*(cdrcode-200)
    change cdrcode to 200+(cell# of TOS)
else (can call UFN on this case)

(this punts on cases where RPLACD must allocate space) [not required; in Dorado, 12K]

#	name	len-1	stk level effect	UFN table entry
32	CONS	0	−1	CONS

Cons pages start with two word header:

```
    word 0: [cnt, nxtcell]   (two 8-bit fields: count of available cells
                                  on this page, and word# of next free cell
                                  on this page)
    word 1: nextpage         (page# of next cons page)
```

DTDs (data type descriptors) have (ucode relevant fields in caps)

```
    word 0:     NAME
    word 1:     SIZE
    words 2,3:  FREE
    words 4,5:  descrs
    words 6,7:  tyspecs
    words 10,11:      POINTERS
    words 12,13:      oldcnt
    word 14:    COUNTER
    word 15:    NEXTPAGE
\CDR.NIL= 200q
```

LISTPDTD is the DTD for type LISTP, i.e., at DTDbase + (LLSH 5 4)

Subroutine MAKECONSCELL[page] (given page, return new cell from it):

```
      new cell is at page + page:nxtcell
      new CNT is old CNT - 1; punt if CNT was zero
      new NXTCELL is new cell's cdr code

Subroutine NEXTCONSPAGE:
      if LISTPDTD:NEXTPAGE # 0 then return it, else punt
      (lisp code scans for page with cnt>1)

CONS(X Y) // note: this may not be right. Check sources for truth
If Y is NIL:
      get NEXTCONSPAGE
      MAKECONSCELL on it
      store new cell with \CDR.NIL in cdrcode (hi byte)
      X in rest of cell

Elseif Y is a listp and the CNT in Y's page > 0, then
      MAKECONSCELL[Y's page]
      store X as CAR, CDR code = ([(LOLOC Y) and 377q] rsh 1) + 200q
Else:
      get NEXTCONSPAGE
      MAKECONSCELL on it
      store Y in new cell (hi byte 0)
      (remember this as Z)

      MAKECONSCELL on same page
      store X in new cell, with hi byte= [(LOLOC Z) and 377q] rsh 1

ADDREF X
ADDREF Y
increment LISTPDTD:COUNTER
DELREF result

[not required, in Dorado, 12K]
```

#	name	len-1	stk level effect	UFN table entry
33	CMLASSOC	0	-1	CL::%%SIMPLE-ASSOC

Takes to two arguments off the stack and returns the of the simplest case of cl:assoc. Equivalent to ASSOC opcode, except punts if the key argument is not an immediate datum. [not required, not implemented on 4K, Dorado]

#	name	len-1	stk level effect	UFN table entry
34	FMEMB	0	-1	FMEMB

if TOS=NIL, return NIL
if TOS is not LISTP, call UFN
if (CAR TOS)=TOS-1, return TOS
else TOS_(CDR TOS), do jump to . [i.e., iterate]
Be sure to allow interrupts.

[not required; in 12K]

#	name	len-1	stk level effect	UFN table entry
35	CMLMEMBER	0	-1	CL::%%SIMPLE-MEMBER

Takes to two arguments off the stack and returns the of the simplest case of cl:member. Equivalent to FMEMB opcode, except punts if the key argument is not an immediate datum. [not required, not implemented on 4K, Dorado]

#	name	len-1	stk level effect	UFN table entry
{ 36	PUTHASH	0	-2	PUTHASH

[not required, not implemented]}

REPLACED BY :

36	FINDKEY			

```
alpha = arg#
tos = key
for z from arg# to numargs - 1 by 2
   if arg(z) = key then return(z + 1)
return(NIL)
```

#	name	len-1	stk level effect	UFN table entry
37	CREATECELL	0	0	CREATECELL

Create a new cell of type TOS (a smallposp):
DTD _ DTDSpace + (type lshift 4)
NewCell _ DTD:FREE (2 words)
DTD:FREE _ @(NewCell) (2 words)
if DTD:FREE is now NIL, signal a gc punt at end of opcode
increment DTD:COUNTER, signal a gc punt if counter goes negative
Zero out DTD:SIZE words starting at NewCell (always an even number)
Deleteref NewCell
TOS _ NewCell

[not required; in Dorado, D0]

#	name	len-1	stk level effect	UFN table entry
40	BIN	0	0	\BIN

If TOS is not of type STREAM (13q) then PUNT

Format of stream is (only some fields are used by microcode):

```
word 0: COFFSET              ; a byte offset from BUFFER
word 1: CBUFSIZE           ; size of input buffer in bytes
word 2&3: flags [byte] = READABLE (bit 0), WRITABLE (bit 1),
                         EXTENDABLE (bit 2), DIRTY (bit 3),
                         PEEKEDCHARP (bit 4), ACCESSBITS (bit 5-7)
      BUFFER [24 bits]   ; pointer to data
word 4: BYTESIZE
          CHARSET         ; 8 bits each
word 5: PEEKEDCHAR       ; valid when PEEKEDCHARP true
word 6: CHARPOSITION
word 7: CBUFMAXSIZE      ; maximum size of output buffer
```

If COFFSET >= CBUFSIZE then PUNT [buffer overflow]
If READABLE is off then PUNT
Fetch and remember the byte at BUFFER + COFFSET[byte offset]
    Note that this address is guaranteed to be valid at this point,
    but it could pagefault.
Update the stream:
    store COFFSET _ COFFSET + 1
Return the remembered byte as a small positive number.

[not required; in Dorado, 12K]

#	name	len-1	stk level effect	UFN table entry
41	BOUT	0	-1	\BOUT

If TOS-1 is not of type STREAM (13q) then PUNT. (see format under BIN)
If TOS is not a small positive number (< 400Q) then PUNT.
if WRITABLE is off then PUNT
if BUFFER is NIL then PUNT
if COFFSET >= CBUFMAXSIZE then PUNT
deposit byte from TOS at BUFFER + CCOFF[byte offset]
Update the stream:
    store COFFSET _ COFFSET + 1
    set DIRTY flag to 1 [if it isn't already]
return the smallposp one (1)
[not required; not implemented; not even generated by compilers (3/13/89)]

#	name	len-1	stk level effect	UFN table entry
42	PROLOGOPDISP	0	0	none

```
Implements the Prolog Opcode Dispatch.  Uses the Prolog registers PC, N,
USQbase, and uSQtablebase.
It takes one arg (DEST).  In pseudo-RTL:
      if smallp(DEST) then PC ← PC + DEST
            else PC ← DEST
      N ← logand(PC↑ 00FF'x)
      opcode ← lrsh(PC↑ 8)
      if uLMBase(opcode) = 1 then
            { LispPC ← USQbase + uSQtablebase(opcode)
              return to Lisp }
            else
            { PC ← PC + 1
              (run microcode version) }
```

#	name	len-1	stk level effect	UFN table entry
{43	LIST1	0	0	CONS

(perform (CONS TOS NIL)] not required, not implemented}

REPLACED BY :

43	RESTLIST			

alpha = skip -- number of args to skip
tos = last -- last arg#
tos-1 = tail
IF tail = NIL THEN
    page _ NEXTCONSPAGE
    GOTO make
ELSE
    AddRef tail
    page _ CONSPAGE[tail]
    GOTO make
make:
    get [cnt,,next] from page
make1:
    tail _ CONSCELL (CAR = IVar(last), CDR = tail)
    AddRef IVar(last)
    IF skip = last THEN GOTO fin
    last _ last - 1
    GOTO make1
noroomonconspage:
fin:
    store updated [cnt,,next]
    update ListpDTD:COUNTER
    DelRef tail
    IF noroomonconspage THEN UFN
    ELSEIF ListpDTD:COUNTER overflow then GCPUNT
    ELSEIF overflow entries then GCHANDLEOVERFLOW
    ELSE NEXTOPCODE

#	name	len-1	stk level effect	UFN table entry
44	MISCN	2	1 + (-n)	\MISCN.UFN

Miscellaneous opcode for opcodes needing n args from the stack. The alpha byte contains the sub-opcode number and the beta byte contains the number of arguments on the stack. This opcode was added specifically for bytecode emulated implementations, where the opcodes could be written in C. This opcode provides the same functionality of the SUBRCALL opcode, except it has the added flexability of having the opcodes UFN (on both Suns & D-Machines). The UFN vectoring routine is written to adjust the stack according to the number of arguments stated in the beta byte, and there is a UFN handler for each sub-opcode. The opcode is generated using the (MISCN NAME &REST ARGS) macro & optimizer defined in LLSUBRS. The NAME parameter must be registered in \MISCN-TABLE-LIST list, which is of the form (name index ufn-name). The \INIT-MISCN-TABLE function initializes the MISCN's sub-opcode UFN vector.

The predefined MISCN sub-opcodes are as follows:

index	name	function
0	USER-SUBR	This is for the user-supplied subr C coded subrs. It contains its own sub-opcode division based on the 1st argument on the stack. Like MISCN, USER-SUBR requires that the user-subrs be registered with the variable \USER-SUBR-LIST (name index ufn) by calling the \INIT-USER-SUBR-TABLE function. Thus user-defined subrs can each have thier own ufn handler which will be indexed through the MISCN & USER-SUBR UFN mechanism. This opcode can be generated using the (USER-SUBR NAME &REST ARGS) macro found in LLSUBRS.
1	CL:VALUES	Return multiple values
2	CL:SXHASH	Common Lisp hash-bits function for EQUAL hash-tables
3	CL:EQLHASHBITSFN	[Not currently implemented]
4	STRINGHASHBITS	IL hash-bits function for STREQUAL harrayp's
5	STRING-EQUAL-HASHBITS	IL hash-bits function for String-EQUAL harrayp's
6	CL:VALUES-LIST	Return a list of multiple values.

To reserve new MISCN & USER-SUBR entries, you should set the global values for \MISCN-TABLE-LIST and \USER-SUBR-LIST in the LLSUBRS file & re-write the file to insure that you will have unique numbers. The funcion WRITECALLSUBRS whould also be called to generate a new subrs.h file, which contains the C constant definitions for the proper indexes in the C code.

The args to the MISCN UFN routines consist of (INDEX ARG-COUNT ARG-PTR), where INDEX is your sub-opcode number, ARG-COUNT is the number of args to be found on the stack, and ARG-PTR is a pointer to the 1st arg found on the stack. The rest of the args can be found by using (\ADDBASE ARG-PTR (LLSH n 1)) for the n-1th arg.

USER-SUBR UFNs have similar args of (USER-SUBR-INDEX ARG-COUNT ARG-PTR), where USER-SUBR-INDEX is the user-subr sub-opcode index, and ARG-COUNT & ARG-PTR are the same as in MISCN UFNs.

CAUTION: Since the stack affect is variable, thus not known to the compiler, the optimizer may do something funny to the stack args around your call. You should check the emitted code to be sure that things compiled correctly. Putting your calls in small functions will help.

#	name	len-1	stk level effect	UFN table entry
45	<unused>	0	-1	(was ENDCOLLECT)

[not required; not implemented, will be eliminated]

#	name	len-1	stk level effect	UFN table entry
46	RPLCONS	0	-1	\RPLCONS

takes two args (LST ITEM):
check (LISTP LST)
LST's pages CNT field # 0 (see CONS above),
LST's cdrcode = 200q.
call UFN if any of these are not true
MAKECONSCELL on LST's page
store ITEM as in cell, with cdr code = 200q (\CDR.NIL)
store as LST's new cdrcode (((LOLOC newcell) and 377) rsh 1) + 200q.
ADDREF item
increment LISTPDTD:COUNTER
return new cell
[not required; in 12K]

#	name	len-1	stk level effect	UFN table entry
50	ELT	0	-1	ELT

(ELT array index)

Check if TOS-1 is type ARRAYP, call UFN if not
Check if TOS is smallpos, call UFN if not

Array descriptor:

```
word 0,1: Flags(8),,base(24)
```

Flags = Orig(1), unused(1), Readonly(1), unused(1), type(4)

```
word 2: Length
word 3: Offset
```

Compute index = (TOS) - Orig
if index < 0 or index >= length, call UFN.
index _ index + Offset
dispatch on type (note that index*2 may overflow):

[0] (byte) return (GETBASEBYTE base index)
[1] (smallpos) return (GETBASE base index)
[2] (fixp) return 32 bits at base+index*2 as a fixp (possibly smallp)
[3] (hash) return (GETBASEPTR base index*2)
[4] (code) same as byte
[5] (bitmap) same as smallpos
[6] (pointer) return (GETBASEPTR base index)
[7] (float) return 32 bits at base+index*2 as a floatp
[11.] (double-pointer) same as hash
[12.] (mixed) same as hash

[not required; not implemented yet]

#	name	len-1	stk level effect	UFN table entry
51	NTHCHC	0	-1	NTHCHARCODE

Same as ELT, except type of TOS-1 is STRINGP, the type of the array is always 0, and (optionally) return NIL instead of calling UFN when index is out of range. [not required; not implemented]

#	name	len-1	stk level effect	UFN table entry
52	SETA	0	-2	SETA

(SETA array index value)

Check array and compute index as with ELT.
If ReadOnly is true, call UFN.
In all cases, leave value on stack on exit.
Dispatch on type:

[0] (byte) perform (PUTBASEBYTE base index value)
[1] (smallpos) perform (PUTBASE base index value)
[2] (fixp) unbox integer value, deposit 32 bits at base+index*2
[3] (hash) perform (RPLPTR base+index*4 value)
[4] (code) same as byte
[5] (bitmap) same as smallpos
[6] (pointer) perform (RPLPTR base+index*2 value)
[7] (float) unbox float value, deposit 32 bits at base+index*2
[11.] (double-pointer) same as hash
[12.] (mixed) same as hash

[not required; not implemented]

#	name	len-1	stk level effect	UFN table entry
53	RPLCHARCODE	0	-2	RPLCHARCODE

[SPECIFICATION INCOMPLETE]

[not required; not implemented]

#	name	len-1	stk level effect	UFN table entry
54	EVAL	0	0	\EVAL

```
takes single argument ARG
If ARG=NIL, T, or smallp, return ARG
If ARG is an atom, attempt free variable lookup:
 If bound, return value
        If top value is not NOBIND (atom #1), return top value
 else ufn-punt
```

```
[optional: if ARG is FIXP, FLOATP, return ARG]
[optional: if ARG is LISTP, punt to \EVALFORM (atom 370q)]
else ufn-punt
[not required; in Dorado, 4K]
```

#	name	len-1	stk level effect	UFN table entry
55	(was EVALV)			

#	name	len-1	stk level effect	UFN table entry
56	TYPECHECK.N	1	0	\TYPECHECK.UFN

identical to DTEST; only UFNs different

#	name	len-1	stk level effect	UFN table entry
57	STKSCAN	0	0	\STKSCAN

TOS is VAR.
If TOS is not litatom, punt.
Returns 24 bit pointer to cell where VAR is bound.

Note: must check VAR=NIL, and return pointer to NIL's value cell. (Free variable lookup algorithm fails if given NIL, at least on Dorado.)

If variable was bound on stack, the value returned will be a pointer into stack space. If variable is not bound, value will be pointer to top level value cell.

[not required; in Dorado (I think), not in DLion? In Maiko emulator]

#	name	len-1	stk level effect	UFN table entry
60	BUSBLT	1	-3	\BUSBLT.UFN

Talks to the BusMaster peripheral adapter.
Alpha bytes:

0	WORDSOUT
1	BYTESOUT
2	BYTESOUTSWAPPED
3	NYBBLESOUT
4	WORDSIN
5	BYTESIN
6	BYTESINSWAPPED
7	NYBBLESINSWAPPED

[not required; in 12K only]

#	name	len-1	stk level effect	UFN table entry
61	MISC8	1	-7	\MISC8.UFN

Miscellaneous opcode for operations needing 8 args.
Alpha bytes:

Alpha	name	function
0		IBLT1 - special-purpose halftone-drawing routine for spectrogram creation
1		IBLT2 - ditto

[not required; in 12K only]

#	name	len-1	stk level effect	UFN table entry
62	UBFLOAT3	1	-2	\UNBOXFLOAT3

in 12K only
Alpha bytes:

0	POLY
1	3X3
2	4X4
3	133

```
      4     331
      5     144
      6     441
for matrix multiply, polynomial evaluation
 alpha byte 7: Unboxed ASET
```

#	name	len-1	stk level effect	UFN table entry
63	TYPEMASK.N	1	0	\TYPEMASK.UFN

similar to TYPEP, except checks if high byte of type table AND with alpha is non-zero, returns TOS if so, NIL otherwise.

#	name	len-1	stk level effect	UFN table entry
64	PROLOGREADPTR			
65	PROLOGREADTAG			
66	PROLOGWRITETAGPTR			
67	PROLOGWRITE0PTR			
70	PSEUDOCOLOR			
72	EQL			

#	name	len-1	stk level effect	UFN table entry
73	DRAWLINE	0	−8	\DRAWLINE.UFN

takes 8 (!) args from top of stack, does line draw inner loop

#	name	len-1	stk level effect	UFN table entry
74	STORE.N	1	0	\STORE.N.UFN

takes quantity at TOS and stores it at TOS-alpha.

#	name	len-1	stk level effect	UFN table entry
75	COPY.N	1	1	\COPY.N.UFN

pushes quantity at (TOS-alpha/2). COPY.N 0 = COPY

#	name	len-1	stk level effect	UFN table entry
76	RAID	0	0	RAID

[used only for UFN]

#	name	len-1	stk level effect	UFN table entry
77	\RETURN			

used only for UFN for LLBREAK

#	name	len-1	stk level effect	UFN table entry
100–106	IVAR	0	1	

push IVAR#(opcode-100) [required]

#	name	len-1	stk level effect	UFN table entry
107	IVARX	1	1	

push IVAR#alpha
[required]

#	name	len-1	stk level effect	UFN table entry
110–116	PVAR	0	1	

push PVAR#(opcode-110)
[required]

#	name	len-1	stk level effect	UFN table entry
117	PVARX	1	1	

push PVAR#(alpha)
[required]

#	name	len-1	stk level effect	UFN table entry
120-126	FVAR	0	1	
127	FVARX	1	1	

Push the indicated FVAR.
[required]

#	name	len-1	stk level effect	UFN table entry
130-136	PVAR←	0	0	
137	PVARX←	1	0	

Set the indicated PVAR from tos, do not pop.
[required]

#	name	len-1	stk level effect	UFN table entry
140	GVAR	2	1	

Push @(VALSPACE+2*(alpha,beta))

[required; may want to change to check if high order bit on, and UFN]

#	name	len-1	stk level effect	UFN table entry
141	ARG0	0	0	\ARG0

check TOS smallp, call UFN if not
check TOS between 1 and #args in current function
replace TOS with value of Ith variable, counting from 1
[to do range check, must fetch flags; if not fast, fetch BLINK.
#args is computable from difference of BLINK and IVAR]
[not required; not implemented yet]

#	name	len-1	stk level effect	UFN table entry
142	IVARX←	1	0	

store TOS as new value of IVAR alpha
[required]

#	name	len-1	stk level effect	UFN table entry
143	FVARX←	1	0	

free variable assignment. When value cell is global, perform GVAR_ operation
[can call \SETFREEVAR.UFN (atom# ???) instead]

#	name	len-1	stk level effect	UFN table entry
144	COPY	0	1	

push TOS again
[required]

#	name	len-1	stk level effect	UFN table entry

#	name	len-1	stk level effect	UFN table entry
145	MYARGCOUNT	0	1	\MYARGCOUNT

Push as a smallpos the number of arguments in current frame.
See ARG0. (probably should use common subroutine)
[not required; not implemented]

#	name	len-1	stk level effect	UFN table entry
146	MYALINK	0	1	

Returns stack-index of beginning of ALINK of current frame.
This pushes the "ALINK" field of the current frame, with the low bit turned off less ALINK.OFFSET (= 12Q).
[required]

#	name	len-1	stk level effect	UFN table entry
147	ACONST	2	1	

Push {0, (alpha,beta)}
[required]

#	name	len-1	stk level effect	UFN table entry
150	'NIL	0	1	
151	'T	0	1	
152	'0	0	1	
153	'1	0	1	

Push the indicated constant.
[required]

#	name	len-1	stk level effect	UFN table entry
154	SIC	1	1	
155	SNIC	1	1	
156	SICX	2	1	

Push:
alpha as a smallposp,
alpha as a smallneg (extend leftward with 1's),
(alpha,beta) as smallposp, respectively.
[required]

#	name	len-1	stk level effect	UFN table entry
157	GCONST	3	1	

Push {alpha, (beta,gamma)}
[required]

#	name	len-1	stk level effect	UFN table entry
160	ATOMNUMBER	2	1	

same as SICX. Different opcode for benefit of code walkers.
[required]

#	name	len-1	stk level effect	UFN table entry
161	READFLAGS	0	0	\READFLAGS

TOS is a virtual page# as a smallposp
TOS _ virtual memory flags of that page, as a smallposp
Flags are:
bit 0: referenced
bit 2: write-protect
bit 3: dirty
Vacant is denoted write-protect + dirty
[This is the same as XNovaOp ReadFlags, with AC0 -> loloc[TOS]]

[required]

#	name	len-1	stk level effect	UFN table entry
162	READRP	0	0	\READRP

TOS is a virtual page# as a smallposp
TOS _ the corresponding real page, as a smallposp
[This is the same as XNovaOp ReadRP, with AC0 -> loloc[TOS]]
[required]

#	name	len-1	stk level effect	UFN table entry
163	WRITEMAP	0	−2	\WRITEMAP

TOS-2 is a virtual page# as a smallposp
TOS-1 is a real page as a smallposp
TOS is a word of flags as a smallposp
Make the indicated virtual page# be associated with the given
real page, with status flags. Real page is immaterial if flags = VACANT
Return the virtual page #
[This is the same as XNovaOp SetFlags, with AC0 -> loloc[TOS-2],
AC1 -> loloc[TOS-1], AC2 -> loloc[TOS]]
[*not yet in Dorado]

#	name	len-1	stk level effect	UFN table entry
164	READPRINTERPORT	0	+1	\READPRINTERPORT

TOS _ current value from printer port, as a smallposp
Ufn if machine cannot do this.
[not in 4k]

#	name	len-1	stk level effect	UFN table entry
165	WRITEPRINTERPORT		0	0 \WRITEPRINTERPORT

Printer _ TOS, interpreted as a smallposp
Ufn if machine cannot do this.
[not in 4k]

#	name	len-1	stk level effect	UFN table entry
166	PILOTBITBLT	0	−1	\PILOTBITBLT

Performs Pilot-style bitblt.
TOS is constant zero, which can be used for maintaining state.
TOS-1 is a pointer to a bitblt table, which is 16-aligned.
[not required, implemented]

#	name	len-1	stk level effect	UFN table entry
167	RCLK	0	0	\RCLKSUBR

Store into words pointed to by TOS the processor clock [up to 32 bits, left justified].
[required]

#	name	len-1	stk level effect	UFN table entry
170	MISC1	1	0	\MISC1.UFN
171	MISC2	1	−1	\MISC2.UFN

These are miscellaneous opcodes that dispatch on alpha to provide
infrequent and/or machine-specific operations. To save microcode space (currently), the two opcodes share the same dispatch table, i.e., the alpha's do not overlap. There are two opcodes principally so that there can be a reasonable ufn handler: MISC1 takes 1 arg, MISC2 takes
2. Current values for alpha:

0 STARTIO[bits] Currently only for Dolphin ethernet. Perform the "StartIO" function with bits given as smallp TOS. (Resets Ethernet to known quiet state).

1 INPUT[devreg] Perform input from some device. TOS is smallp

device register specification (on Dolphin: 4 bits of
task, 4 bits of device reg; on DLion: 4 bits absolute).
Returns TOS = smallp value input from device.

DLion codes:
for INPUT {alpha = 1 mod 4}
TOS = 00 mod 16, _ EIData
TOS = 01 mod 16, _ EStatus
TOS = 02 mod 16, _ KIData
TOS = 03 mod 16, _ KStatus
TOS = 04 mod 16, _ uSTATE
TOS = 05 mod 16, _ MStatus
TOS = 06 mod 16, _ KTest
TOS = 07 mod 16, MP code 9122
TOS = 08 mod 16, _ Version
TOS = 09 mod 16, <12K> _ BusExt L <4K> MP code 9122
TOS = 10 mod 16, <12K> _ BusExt M <4K> MP code 9122
TOS = 11 mod 16, <12K> _ uFLmode <4K> MP code 9122
TOS = 12 mod 16, MP code 9122
TOS = 13 mod 16, MP code 9122
TOS = 14 mod 16, MP code 9122
TOS = 15 mod 16, MP code 9122

2 OUTPUT[value, devreg] Perform output to some device. TOS is smallp
device register spec as with INPUT; TOS-1 is the smallp
value to output.
for DLion:
for OUTPUT {alpha = 2 mod 4}
TOS = 00 mod 16, <12K> BusExt L _ <4K> IOPOData _
TOS = 01 mod 16, IOPCtl _
TOS = 02 mod 16, <12K>uFLmode _ <4K> KOData _
TOS = 03 mod 16, KCtl _
TOS = 04 mod 16, EOData _
TOS = 05 mod 16, EICtl _
TOS = 06 mod 16, DCtl _
TOS = 07 mod 16, uBBTime _ {display rate}
TOS = 08 mod 16, uLispOptions _
TOS = 09 mod 16, PCtl _
TOS = 10 mod 16, MCtl _
TOS = 11 mod 16, <12K> BusExt M _ <4K> MP code 9120
TOS = 12 mod 16, EOCtl _
TOS = 13 mod 16, KCmd _
TOS = 14 mod 16, <12K> PPort _ <4K> MP code 9120
TOS = 15 mod 16, POData _

9 Dorado only, RWMUFMAN

#	name	len-1	stk level effect	UFN table entry
172	RECLAIMCELL	0	0	\GCRECLAIMCELL

Check type of TOS; let DTD be pointer to DTD of this type
If not LISTP then punt
Reclaim list:

```
        code_PTR:cdrcode
        if (code and 200q) = 0 then punt        [or optional: if code = 0 then punt]
        FreeListCell(PTR)
        val_ deleteref(PTR:carfield)            * deleteref CAR
        if code # \CDR.NIL
          then PTR_PTR:pagebase + (code lsh 1) * point to cdr or lvcdr
                [if (code and 200q) = 0         * optional
                  then FreeListCell(PTR)                * cdr indirect--free cell
                       PTR_ GetBasePtr(PTR)]
                if deleteref(PTR)                       * deleteref CDR
                  then val_PTR
        return val
FreeListCell(PTR):
        PAGE _ address of PTR's page
        if PAGE:Nextpage < 0 then punt          * only when page was full
```

PTR:cdrcode _ PAGE:nextcell
PAGE:nextcell _ word# of PTR
PAGE:count _ PAGE:count + 1

How to reclaim other types, roughly (needs type table change):
if Type bit "ok to reclaim" is off, call UFN
store DTD:FREELST in first two words of DATUM
store DATUM in DTD:FREELST
[not required; implemented for Listp on D0, non-listp on Dorado?, ? for 12K]

#	name	len-1	stk level effect	UFN table entry
173	GCSCAN1	0	0	\GCSCAN1

scan HTMAIN from (TOS)-1 to 0 for a cell with
collision bit on or else stack bit & reference cnt both are 0
if none found, return NIL
else return new index.
note: design allows NWWInterrupts to be processed
note: can actually perform GCRECLAIMCELL on the
cell indicated if stack bit off and ref cnt=0)
[not required; in all]

#	name	len-1	stk level effect	UFN table entry
174	GCSCAN2	0	0	\GCSCAN2

similar to GCSCAN1, but scan for word
with collision bit on or stack bit on.
Note: can optionally turn stack bit off, check if
count is 1 and zero entry, continue scanning
Note: design allows NWWInterrupts to be processed
[not required; in all]

#	name	len-1	stk level effect	UFN table entry
175	SUBRCALL	2		

Call Bcpl subr number alpha with beta arguments.
The following have some microcode on the DLion:
17'b Raid
15'b Logout
06'b BackGround
11'b DspBout
20'b Pup
22'b SETSCREENCOLOR
23'b ShowDisplay

#	name	len-1	stk level effect	UFN table entry
176	CONTEXT	0	0	\CONTEXTSWITCH

switch to context (TOS).

#	name	len-1	stk level effect	UFN table entry
177	(was audio)			

[not required; not currently implemented]

#	name	len-1	stk level effect	UFN table entry
200-217	JUMP	0	JUMP	
220-237	FJUMP	0	CJUMP	
240-257	TJUMP	0	CJUMP	
260	JUMPX	1	JUMP	
261	JUMPXX	2	JUMP	
262	FJUMPX	1	CJUMP	
263	TJUMPX	1	CJUMP	
264	NFJUMPX	1	NCJUMP	

265	NTJUMPX	1	NCJUMP

Assorted jumps. The offset of the jump is given in the succeeding bytes, sign-extended to the left in the case of the single-byte offsets. The offset is relative to the start of the instruction. The opcodes with implicit offset run from +2 thru +21q.
JUMP* are unconditional.
FJUMP* and TJUMP* perform the jump only if TOS is NIL or non-NIL, respectively.
NFJUMPX and NTJUMPX perform the jump only if TOS is NIL or non-NIL, respectively. Additionally, they pop the stack only if the jump is not taken.
[required]

#	name	len-1	stk level effect	UFN table entry
266	AREF1	0	−1	%AREF1

Perform a one-dimensional array access:

(AREF1 array index)

1.) Check that array is a oned-array -- if not punt

2.) Check that 0 <= index < total size for array

3.) Compute (index + offset for array)

4.) Extract base, and type number -- and pass base, type number, index + offset to array-read subroutine and return result on top of stack.

#	name	len-1	stk level effect	UFN table entry
267	ASET1	0	−2	%ASET1

Perform a one-dimensional array set:

(ASET1 new-value array index)

1.) Check that array is a oned-array -- if not punt

2.) Check that 0 <= index < total size for array

3.) Compute (index + offset for array)

4.) Check array not read-only

5.) Extract base, and type number -- and pass newvalue, base, type number, index + offset to array-write subroutine and return newvalue on top of stack.

#	name	len-1	stk level effect	UFN table entry
270-276	PVAR←↑	0	−1	

Store TOS into indicated PVAR, pop stack.
[required]

#	name	len-1	stk level effect	UFN table entry
277	POP	0	−1	

Pop stack.
[required]

#	name	len-1	stk level effect	UFN table entry
300	POP.N	1	(variable)	

POP (alpha+1) elements off top of stack, POP.N 0 = POP, POP.N 1 = POP POP, etc.

#	name	len-1	stk level effect	UFN table entry
301	ATOMCELL.N	1	0	

if TOS is atom (0,,low), then replace with (alpha,,low+low), with carry into alpha. This is used for getting the PList, Def, Val cell of litatoms. If TOS HI is not 0, call UFN. This will allow assigning definitions, plists and values to non-litatoms.

#	name	len-1	stk level effect	UFN table entry
302	GETBASEBYTE	0	-1	\GETBASEBYTE

Retrieve byte at offset TOS from (TOS-1).

#	name	len-1	stk level effect	UFN table entry
303	INSTANCEP	2	0	\INSTANCEP.UFN

return T if typename is subtype of (alpha,beta), else return NIL.
(typename is word 0 of type's DTD; DTD is DTDBase+(type# lsh 4) not locked down
supertype is word 15 of DTD, 0 means no supertype)
[currently only in 12k Dandelion]

#	name	len-1	stk level effect	UFN table entry
304	BLT	0	-2	\BLT

(BLT destinationaddr sourceaddr nwords)
Move nwords from source to destination. If nwords < prespecified constant (currently 10q), then operation is uninterruptable, else must be prepared to service interrupts. On page fault or interrupt, update stack according to how much is moved, and back up pc. Words are moved right to left (high addresses to low), if it makes a difference. Result is unspecified.

#	name	len-1	stk level effect	UFN table entry
305	MISC10	2	-9	\MISC10.UFN

Perform miscellaneous operation on 10 arguments.

alpha	operation
0	PIXELBLT

[not required; in 12k only]

#	name	len-1	stk level effect	UFN table entry
306	(unused)			

#	name	len-1	stk level effect	UFN table entry
307	PUTBASEBYTE	0	-2	\PUTBASEBYTE

Store TOS at offset TOS-1 from (TOS-2), punting if TOS is not smallposp.
Currently ucode punts and ufn errors if offset isn't a smallp.

#	name	len-1	stk level effect	UFN table entry
310	GETBASE.N	1	0	\GETBASE

TOS _ @(TOS+alpha) as a smallposp.

#	name	len-1	stk level effect	UFN table entry
311	GETBASEPTR.N			

TOS _ 24-bit pointer @(TOS+alpha).

#	name	len-1	stk level effect	UFN table entry
312	GETBITS.N.FD			

take 1 arg on stack (PTR) and 2 bytes (n, fd). fetches the "field" fd from the word PTR + n. fd is a mesa field descriptor: the left 4 bits is the number of the "first" bit of the field, while the right 4 bits is the width of the field-1. E.g., 0:17 is the full word, 0:0 is the leftmost bit.

#	name	len-1	stk level effect	UFN table entry
313	Unused			

#	name	len-1	stk level effect	UFN table entry
314	CMLEQUAL	0	−1	CL:EQUAL

Takes two arguments off the stack and performs some cases of the cl:equal predicate. Punts if either argument is a not an immediate datum or a number.
[not required, not implemented on 4K and Dorado]

#	name	len-1	stk level effect	UFN table entry
315	PUTBASE.N	1	−1	\PUTBASE.UFN

Store TOS as word at location (TOS-1)+alpha
Pop (Return TOS-1).
Punt if TOS not smallposp. Note that UFN will specify extra byte for punt.

#	name	len-1	stk level effect	UFN table entry
316	PUTBASEPTR.N	1	−1	\PUTBASEPTR.UFN

Takes (PTR, NEWVAL) on stack, leaves PTR on stack, stores NEWVAL at PTR+N . (note: no punt case)

#	name	len-1	stk level effect	UFN table entry
317	PUTBITS.N.FD	2	−1	\PUTBITS.UFN

Takes (PTR, NEWVAL) on stack, stores bits of NEWVAL at FD field of PTR+N. Returns PTR.
Punt (UFN) if NEWVAL is not smallposp.

#	name	len-1	stk level effect	UFN table entry
320	ADDBASE	0	−1	\ADDBASE
321	VAG2	0	−1	\VAG2
322	HILOC	0	0	
323	LOLOC	0	0	

as before

#	name	len-1	stk level effect	UFN table entry
324	PLUS2	0	−1	PLUS
325	DIFFERENCE	0	−1	DIFFERENCE
326	TIMES2	0	−1	TIMES
327	QUOTIENT	0	−1	QUOTIENT

(same as I- versions, except UFN different. Optionally perform as F- opcode if one of arguments is floating.)

#	name	len-1	stk level effect	UFN table entry
330	IPLUS2	0	−1	\SLOWIPLUS2
331	IDIFFERENCE	0	−1	\SLOWIDIFFERENCE
332	ITIMES2	0	−1	\SLOWITIMES2
333	IQUOTIENT	0	−1	IQUOTIENT
334	IREMAINDER	0	−1	IREMAINDER

unbox TOS & TOS-1
(if SmallPos then 0,,loloc, if SmallNeg then -1,,loloc,
typetest if FIXP then fetch 32 bit quantity)
perform 32x32 operation, and then
if overflow occurs, punt
[used to say: call OFLOWMAKENUMBER (atom ???)
with result mod 2^32 as two 16 bit smallposps.
This can't work; what did we mean?]
If no overflow:
if hi part 0, return SmallPosHi,,lo

if hi part -1, return SmallNegHi,,lo
else need to return large integer. Two choices:
1) set up as if in call to MAKENUMBER (atom ???) with 2 args being
Hi and Lo part of result, as smallposps; or
2) Perform CREATECELL of type FIXP, and then store results
in generated box; return new box
[only smallpos x smallpos required on IPLUS, IDIFFERENCE;
Current implementation status:
Only smallpos x smallpos on ITIMES in both microcodes
only smallpos/smallpos for REMAINDER, QUOTIENT in Dorado]

#	name	len-1	stk level effect	UFN table entry
335	IPLUS.N	1	0	\SLOWIPLUS2

add TOS+alpha

#	name	len-1	stk level effect	UFN table entry
336	IDIFFERENCE.N	1	0	\SLOWIDIFFERENCE

subtract TOS-alpha

#	name	len-1	stk level effect	UFN table entry
337	unused			

#	name	len-1	stk level effect	UFN table entry
340	LLSH1	0	0	\SLOWLLSH1
341	LLSH8	0	0	\SLOWLLSH8
342	LRSH1	0	0	\SLOWLRSH1
343	LRSH8	0	0	\SLOWLRSH8

unbox TOS, perform 32 bit operation and box results
as with 2 arg fns
[smallposp -> smallposp required, can UFN in other cases]

#	name	len-1	stk level effect	UFN table entry
344	LOGOR2	0	−1	\SLOWLOGOR2
345	LOGAND2	0	−1	\SLOWLOGAND2
346	LOGXOR2	0	−1	\SLOWLOGXOR2

see IPLUS etc above
[smallposp -> smallposp required, can UFN in other cases]
[32x32 bit implemented in Dorado, D0]

#	name	len-1	stk level effect	UFN table entry
347	LSH	0	−1	LSH

shift TOS-1 arithmetically by TOS.

#	name	len-1	stk level effect	UFN table entry
350	FPLUS2	0	−1	FPLUS2
351	FDIFFERENCE	0	−1	FDIFFERENCE
352	FTIMES2	0	−1	FTIMES2
353	FQUOTIENT	0	−1	FQUOTIENT

[not required; in Dorado, 12K]

#	name	len-1	stk level effect	UFN table entry
354	UBFLOAT2	1	−1	\UNBOXFLOAT2

alpha bytes:
0 ADD x+y

1 SUB x-y
2 ISUB y-x (currently unused)
3 MULT x*y
4 DIV x/y
5 GREAT x>y (returns T/NIL rather than unboxed floating)
6 MAX (max x y) currently unused
7 MIN (min x y) currently unused
8 REM (x remainder y), i.e. x-(floor x/y)*y
9 (UBAREF A I)
Same as AREF1, except that this one returns an unboxed number
implementations: Dorado has GREAT only 12K has all but REM

#	name	len-1	stk level effect	UFN table entry
355	UBFLOAT1	1	0	\UNBOXFLOAT1

alpha byte:
0 BOX (tos -> floating box (tos))
1 UNBOX (tos -> floating unbox (tos), float if FIXP)
2 ABS (currently unused)
3 NEGATE (currently unused)
implemented all on 12K

#	name	len-1	stk level effect	UFN table entry
356	AREF2	0	−2	%AREF2

Perform a two-dimensional array access:

(AREF2 array i j)

1.) Check that array is a twod-array -- if not punt

2.) Check that 0 <= i < bound0

3.) Check that 0 <= j < bound1

4.) Compute (j + i * bound1)
5.) Extract base, and type number -- and pass base, type number, (j + i * bound1) to array-read subroutine and return result on top of stack

#	name	len-1	stk level effect	UFN table entry
357	ASET2	0	−3	%ASET2

Perform a two-dimensional array set:
(ASET2 newvalue array i j)

1.) Check that array is a twod-array -- if not punt

2.) Check that 0 <= i < bound0

3.) Check that 0 <= j < bound1

4.) Compute (j + i * bound1)

5.) Check array not read-only

6.) Extract base, and type number -- and pass base, type number, (j + i * bound1) to array-write subroutine and return newvalue on top of stack.

#	name	len-1	stk level effect	UFN table entry
360	EQ	0	−1	

return T or NIL if (tos)=(tos-1)

#	name	len-1	stk level effect	UFN table entry
361	IGREATERP	0	−1	\SLOWIGREATERP
362	FGREATERP	0	−1	FGREATERP

[IGREATERP required; FGREATERP not implemented]

#	name	len-1	stk level effect	UFN table entry
363	GREATERP	0	-1	GREATERP

Same as IGREATERP (see PLUS, etc)
[not required]

#	name	len-1	stk level effect	UFN table entry
364	EQUAL	0	-1	EQUAL

If args are EQ, return T
If either arg is litatom, return NIL
else call UFN
[not required; not implemented]

#	name	len-1	stk level effect	UFN table entry
365	MAKENUMBER	2	-1	MAKENUMBER

TOS-1 and TOS are smallposp's denoting the hi and lo halves of a 32-bit number.
Return a fixp that represents it:
 If loloc[TOS-1] = 0
  then return SmallPl,,loloc[TOS]
  elseif loloc[TOS-1] = 177777q
  then return SmallNeg,,loloc[TOS]
  else CREATECELL[\FIXP]
 Store loloc[TOS-1] and loloc[TOS] as its hi and lo halves
 return the new cell
[implemented on 4K, Dorado]

#	name	len-1	stk level effect	UFN table entry
366	BOXIPLUS	0	-1	\BOXIPLUS
367	BOXIDIFFERENCE	0	-1	\BOXIDIFFERENCE

Same as IPLUS2, IDIFFERENCE, except store result @TOS -- first arg is number box (for which optionally check) -- and no overflow check.

#	name	len-1	stk level effect	UFN table entry
370	FLOATBLT	0	-3	\FLOATBLT

Miscellaneous floating point array ops; will eventually be renamed MISC5. Provides access to just about everything the Weitek FP chip does. Operates on two arrays; puts results in a third.
args: (BASE1, BASE2, DEST, N).
Alpha bytes:

0	FLOATWRAP
1	FLOATUNWRAP
2	FLOAT
3	FIX
4	FPLUS
5	FDIFFERENCE
6	FDIFFERENCE
7	(FPLUS (ABS source1) (ABS source2))
10	(ABS (FDIFFERENCE source1 source2))
11	(ABS (FPLUS source1 source2))
20	FTIMES

[not required; implemented on 1108X only]

#	name	len-1	stk level effect	UFN table entry
371	FFTSTEP	0	-1	\FFTSTEP

Takes FFTTABLE as TOS; performs one FFT step thereupon.
[not required; implemented on 1108X only]

#	name	len-1	stk level effect	UFN table entry

```
372       MISC3         0         -1                    \MISC3.UFN
```

Miscellaneous 3-arg opcode.
Alpha bytes:

0 EXPONENT(source dest n)
source is vector of floatps, dest is vector of words
store exponent of source for n in dest

1 MAGNITUDE
source is a vector of complex, dest is a vector of float
store magnitude of source in dest

2 FLOAT
source is a vector of word, dest is a vector of float
float source & store in dest

3 COMP
source is a vector of float, dest is a vector of complex
spread source into dest, storing 0's.

4 BLKFMAX

5 BLKFMIN

6 BLKFABSMAX

7 BLKFABSMIN

8 FLOATTOBYTE
source is vector of float (must have even number of elements), dest is vector of words

9 ARRAYREAD (base typenumber index)
Dispatch on typenumber and perform a typed get.

[not required; implemented on 1108X only]

#	name	len-1	stk level effect	UFN table entry
373	MISC4	0	-1	\MISC4.UFN

Miscellaneous 4-arg opcode.
Alpha bytes:

0 TIMES

1 PERM

2 PLUS

3 DIFF

4 SEP

6 \BITMAPBIT bitmap x y newvalue (optional)

7 ARRAYWRITE (newvalue base typenumber index)
Dispatch on typenumber and perform a typed put

[some confusion on how 0,2,3 different from corresponding TIMES, PLUS, DIF [not required; implemented on 1108X only]

#	name	len-1	stk level effect	UFN table entry
374	reserved on D0, UPCTRACE on Dorado			
375	SWAP	0	0	
376	NOP	0	0	
377	=			