UTAH COMPUTATIONAL PHYSICS GROUP
OPERATING NOTE NO. 20

January 1975

Appendix -- Utah Modifications
Stanford Lisp/360 Reference Manual, Fourth Edition

by

Kevin R. Kay
Computational Physics Group
University of Utah

Table of Contents, Utah Appendix

Work supported in part by the National Science Foundation under Grant No. GJ-32181 and by the Advanced Research Projects Agency of the Office of the Department of Defense under Contract No. DAHC15-73-C-0363.

A1. Comments on the Stanford Manual

The following chapter-and-verse notes are intended as (a) clarifications of certain statements in the existing manual, or (b) brief hints of changes made at Utah, but not as a complete list of such changes.

Section	Page	Comment
2.1	4	Bits 4 and 5 denote BIGPOS and BIGNEG numbers, as the BIGNUM extensions to fixed-point representations; see section A4.
2.1.1	7	Generated numeric values are not put on the object list, but read-in numbers are currently (and thus they are searched for before INTERNing).
2.2	9	Initial stack length now is 3K words; this can be altered by CONDENSE and SETSIZE, section A5.
3.1	15	EXPT(n1,n2) -- n1 may be a BIGNUM (section A4), but n2 may not unless n1 = 0, +1, -1, +1.0, or -1.0 .
3.1	19	OVOFF/OVON routines deleted, since overflows now are handled by the BIGNUM package, and these routines can confuse it.
3.1	21	READCH returns the atom EOF if end-of-file seen.
3.1	22	REMPROP now checks every cell on the property list, instead of just the 1st, 3rd, 5th, etc.
3.1	23	RPLACA/D only replace the address portion of the cell pointed at (the lower 24 bits of the word), and keep the old high-byte with its flag bits. Thus they are not quite analogous to CONS (which works on 32 bits), and probably unsuitable for manipulating fullcells, etc.
3.1	23	SPEAK (and UNCOUNT) use 2 CONS to return a value.
3.1	24	TRACE and UNTRACE have additional capabilities, to affect function tracing and error tracebacks, section A8.
3.2	25	New atom GC* has an APVAL initially NIL, but can be reset by user to affect garbage collections; see section A5.
4.	27	Fixed-point numbers may be of any magnitude internally and any input/output length externally; section A4.
5.	28	An initial check is made for LISPOUT and LISPIN; if not provided as DD statements, error messages are given and the run is terminated (return code = 12); if provided, LISPOUT will use the JCL's BLOCKSIZ. 'B=0' now works; multiple PARM specifications now work; RESTORE affects the PDS length as well as the initial PARMs; see section A6.
6.1	29	All file-input related functions can now handle certain partitioned data sets. See section A7 for details and the extended denotation of the ddname argument.
6.1.1	29	JCL-supplied LRECL and BLKSIZE are now used if given; if not and if OPEN's second argument = NIL, say, then the SYSFILE defaults are supplied.

(continued)

Section	Page	Comment
6.1.1	30	SYSFILE is now blocked (80,1600); it was (133,665). The DCB-address property is now called OPENFILE, to distinguish it from APVALs or chance NIL references. The value of the DCB OPENFILE is now a simple fullcell, obtainable by CAR(GET(ddname,'OPENFILE)), with the complication of being negative for PDS DCB's.
6.1.2	30	Partitioned data sets are not really closed, unless the user so requests explicitly; see section A7.
6.1.3	30	If ASA is used in conjunction with an OTLL, the ASA should precede the OTLL.
6.1.4	30	OTLL(n) permits a maximum n of 120, but the user should restrict it to <= LRECL, of course. For compactness, all datasets will print up to their respective OTLL; LISPOUT is permitted, however, to start a new line if an atom prints to within 20 spaces of the selected OTLL. To turn off this feature, do OTLL((n)). Note also that LISPOUT's linelength includes the ASA control-character and the 4-space indentation, such that the user-writable length is at most n-5.
6.1.5	31	WRS(LISPOUT) has the effect of ASA(T) and OTLL(120). WRS(any-other-dataset) sets ASA(NIL) and OTLL(LRECL-8).
6.1.7	31	RDS(NIL) is equivalent to RDS(LISPIN). RDS(any-dataset) sets input linelength to LRECL-8.
6.2	32	The checkpoint facilities have been significantly extended, but there are still hazards; see section A6 for an exhaustive/exhausting discussion. All checkpoint-related I/O functions automatically OPEN the data set as a SYSFILE (if the user omits doing so), and also automatically CLOSE it (unless it is a member of a partitioned data set; see section A7).
7.1.3	35	R15 (PDL) may be used freely, except when interacting with the *MOVE and *REMOVE processes of the compiler, which set up R15 as the local routine's stack; R7 (PDS) should never be changed, except indirectly by means of one of the stack macros of 7.1.4 .
7.3.2	41	BPSMOVE and BPSZ now zero the old evacuated BPS area to avoid thwarting the garbage collector of its prey. BPSMOVE(n) can give an error "BAD ARG OR TOO BIG" if n is not an integer or would involve shifting BPS away from the end-of-BPS (mustn't clobber FCS). For the latter or to get more BPS, CONDENSE might be useful (see section A5.2). BPSMOVE will normally return the relocation done as a logical-number, signifying how far the BPS base was shifted in #-of-bytes; this will be 0 mod 8, in order to maintain double-word alignment within BPS.
7.3.2	41	EXCISE(p) will only function as EXCISE(T), because the compiler has been re-arranged with LAP360 first.

(continued)

Section	Page	Comment
7.3.2	42	OVOFF/OVON routines deleted, since overflows now are handled by the BIGNUM package, and these routines can confuse it.
7.3.3	42	In addition to interpreter-assist routines showing up as "BAL 2,nn(0,R12)", the following common routines have been open-coded for less LAP space and greater execution speed: APPEND1, ATOM, CONS, FLAGP, GET, NCONC, NUMBERP, TERPRI, and two extra: NCONS, XCONS.
8.	44	The garbage collector has a user-variable (section A5) to affect early job termination if space exhausted, and an alternate CLEANing function has been added.
10.4	50	The register dump now prints a few extra word-contents, but the average user needn't pay attention to these. The "OVER- OR UNDERFLOW" message should never occur, in principle, when using the BIGNUM package...i.e., the system's arithmetic routines mentioned in section A4. User-written LAP code is not protected.
10.5.2	51	The traceback can be selectively turned off by doing a prior UNTRACE(T); see section A8.4.
10.5.3	54	A new error has been added to OPEN, such that: D1-FILE CANNOT BE OPENED - DD STATEMENT MISSING . Errors D5 and D6 no longer exist, since SYSFILEs will be automatically OPENed. Error message D7 has been updated to reflect the new capabilities discussed in section A6, and now reads: "D7: WRONG CHKPT FILE, OR NOT ENOUGH ROOM" . Each such file (created by CHKPOINT, BPSCHKPT, WBLK) has a TYPE and a DATE in its first record (as well as some relocation information); the date is the LISP1 source edition or version date, e.g. " 120174", and is included as a precaution against users RESTOREing old files subsequent to LISP1 being patched and rebuilt. The possible reasons for "NOT ENOUGH ROOM" are: RESTORE - the file's FCS+BPS is longer than the in-core FCS:end-of-BPS; BPSRESTR- the file's BPS is longer than the in-core BPS boundaries; RBLK - the file's overlay length is longer than the in-core unused-BPS remaining.
10.5.3	55	Error R5 now should apply only to long atom names, since numbers may be any length (core permitting).

A2. LISP1 Patches for different Assemblers and Computers:

1) Many of the extended definitions of BCR instructions have been used in the source code to enhance legibility, and the MACROs for these have been included in the file. These are needed for users doing the assembly with ASMF and should be retained as is by such users. For users doing the assembly with ASMG, these MACROs are redundant and must be deleted...the following cards will do the trick:

```
./ N    10
./ R    102840  103270
*
*                  FOR ASMG ASSEMBLY, THE EXTENDED-DEFINITIONS
*                    OF BCR-INSTRUCTIONS HAVE BEEN DELETED.
*
```

2) The modification above will suffice on most 360 computers. However, those models with "imprecise" interrupts (in particular, the 360/91) will need the following additional insertions:

```
./ N    2
./ I    212052
         CLI       6(1),X'0F'           IMPRECISE NON-OVERFLOW?
         BH        TRAPSCAR               YES.
         CLI       6(1),X'00'           IMPRECISE OVERFLOW?
         BH        TRAPSOVF               YES.
./ I    212710
         BNER      0                    NO-OP; PIPELINE DRAIN FOR /91.
```

3) The modifications above for the 360/91 are reputedly sufficient for the 370 series; however, the following alteration to CONS may be used (instead of the "pipeline drain" card) if an explicit test is needed rather than relying on the "specification exception" trap:

```
./ N    2
./ D    212050
./ R    212550
CONSINST C         FREE,FOUR
./ R    212700  212710
*                          LOW COMPARISON HERE, TO SIGNAL NEED TO GC.
CONS     C         FREE,FOUR
         BNL       CONSOK
         STM       7,5,CONSAV
         BAL       14,CONS0
         LM        7,5,CONSAV           REG 6 EQU "FREE", OF COURSE.
CONSOK   ST        A,CAR(FREE)
./ I    212760
CONSAV   DS        15F
```

A3. Textual and Programming re-arrangements in LISP1

1) Nulls (X'00') have been removed from the source code, so the ECHO and ECHOKRK macros have been amended to supply nulls in atom names. In addition, ECHO will handle names written as hexadecimal and allow names of any length, following a design by Owen Saxton of SLAC.

2) The arrangement of the functions is slightly shuffled but hopefully handier and more commented for new programmers who have to read the code. The sections of the file have been renumbered.

3) In line with the modifications to the checkpoint functions, the treatment of type '40' cells (APVALs, SUBRs, BPS, etc.) is now more comprehensive; the internal BPRELOC functions have been accordingly deleted or amended.

4) Users who patch LISP1 with extra atoms or initialization code probably will need to increase the STACKSIZ. The present method of assigning core assumes no user routines are LKED above the LISP1 assembly module, so terminal-interaction code should be first.

5) Assemblies doing LIST and XREF may need more tracks allocated for SYSUT2, SYSUT3, SYSPRINT. The LISP1 source currently takes 90 tracks.

6) The codes returned from the LISP1 module upon termination now have some significance, and indicate the following conditions:

0	- normal termination (e.g. after EOF on LISPIN);
4	- termination after non-fatal error, because EXITERR(T);
8	- termination after fatal error (e.g. FCS exhausted);
12	- termination after serious error while initializing LISP.

7) N.B. Although the garbage-collector now checks the PDS for unboxed numbers (e.g. the result of CAAR 3), it still assumes (for speed) that arbitrary unboxed numbers are never stored as part of FCS structures. They are safe on the stack and can usually be passed safely as computed arguments to SUBRs or FSUBRs, but should not be bound to atoms or appear on the ALIST, etc. Otherwise, the next GC will either complain CAR TAKEN OF FULLCELL or abort with a 0C0 system error.

Usually, if LISP blows up with a 0C0 and if no interrupt message or register-contents are printed, a GC was in progress (prior interrupt) which suddenly found a spurious or unboxed number (non-fullcell) and was led astray.

A4. BIGNUMs -- Arbitrary Precision Arithmetic

A4.1 Effects to user

Lisp functions accepting a fixed-point number previously were limited to integers in the range -<2↑31> : <2↑31>-1. With the current BIGNUM code, this restriction is void; an integer may be of any magnitude ... limited only by the number of cells in FCS (@ roughly 9 decimal digits per cell).
Hence the following functions accept arbitrary integers with impunity:

ADD1, SUB1, MINUS, PLUS, TIMES, DIFFERENCE, QUOTIENT, REMAINDER, MAX, MIN, ZEROP, MINUSP, LESSP, GREATERP, EVENP, FIXP; EQUAL, RNUMB/MKATOM, READ, PRINT.

The following and their ilk (and perhaps some compiler functions) are still restricted to the old range (for speed):

LENGTH, COUNT, SPEAK.

The following are partially restricted or special:

FIX(n)	--	a large floating-point number does not become an imprecise BIGNUM, but returns 0 as before.
FLOAT(n)	--	a BIGNUM larger in magnitude than 4.3E68 or so will give the error "BFLT OVFL".
BIGP(n)	--	gives T if n is internally represented as a BIGNUM (see below); gives NIL if anything else, including an integer < 2↑31 in magnitude.
EXPT(n1,n2)	--	n1 may be a BIGNUM, but n2 may not, unless n1 = 0,+1,-1,+1.0,-1.0, else you'll get an "EXPT- BIGNUM EXPONENT" error.

A4.2 Implementation

A new numeric atom-type is defined, using bits 4 and 5 to denote positive and negative BIGNUMs: BIGPOS (X'C8') and BIGNEG (X'CC') types respectively. The arithmetic routines, and others (mentioned above) as appropriate, make software or hardware checks for atom-type or overflow and perform necessary conversions and arithmetic operations for those integers requiring more than 32 bits to express (roughly 2 billion in magnitude). The actual code is derived from that written for Lisp 1.6 (Stanford A.I. Lab's Lisp for the PDP-10), with the necessary changes for a 32-bit machine, different overflow mechanism, etc., etc.
The principal routines are almost exactly 1:1 in content, with the exception of the BIG:FLOAT conversion which uses a hexadecimal representation. The code therefore has the virtues and failings of the Lisp 1.6 rendition, which appears to be modelled after Collins' SAC system. At any rate, the intermediate scratch cells are generally returned to the FREE list to reduce GC frequency, and the 3 special-cases of the divide routine are handled a la Knuth.

A5. Garbage collector, GC☆, CLEAN, CONDENSE, SETSIZE

A5.1 Changes to the garbage collector

A5.1.1 The global variable "GC☆" may be set by the user to force a terminating error when FCS runs low, instead of going on and on collecting a few cells at a time before dying with "STORAGE EXHAUSTED". The variable's APVAL is initially NIL, which means run to exhaustion as previously; if set to some integer, say 500, then the system will abort (error GC2) if a future GC fails to reclaim that many cells.

A5.1.2 A new function has been added as a more powerful alternative to doing RECLAIM() and should only be invoked at the top-level because it clears the ALIST and all other internal holding areas to ensure that everything collectable is GC'd. The function is CLEAN() and causes pass 1 of the garbage collector (marking cells in use) to first make an extra check on user-introduced atoms: if they aren't pointed to by some FCS cell and do not have a property-list either, then they will be GC'd. If they do have some property, they are retained (presumably for future reference). This check/purge gets rid of numbers, intermediate atom names from the compiler, etc.

A5.1.3 The FREE list has its CARs cleared when collected, with a subtle indirect intent of reducing those 0C0 or car-of-fullcell errors which occurred during GCs or tracebacks (see also section A3.7).

A5.2 New functions affecting PDS and FCS sizes

A5.2.1 SETSIZE(n1,n2) will try to set the PDS length to n1 words and the FCS area to n2 cells, if the space exists. All Lisp space not in use, including high BPS, will be taken as needed; hence equivalents of "FCSMOVE" and BPSMOVE are done, but SETSIZE cannot reduce the actual size of FCS ... to do that, use CONDENSE below.
In detail, SETSIZE relocates the FCS and BPS core blocks and their inter-block pointers to atoms and SUBRs; then resets all system work cells to 0, the ALIST to NIL, the PDS to its origin, and does a RECLAIM; finally exits to the top-level EVALQT and reads a new doublet.

A5.2.2 CONDENSE(n1,n2) will compact FCS cells down in core towards the PDS, with the FREE list becoming linear in high FCS; then it uses SETSIZE(n1,n2) being able to reduce FCS (up to the number reclaimed) if so desired.
In detail, the compacting method involves: marking all cells in use, moving high cells down to the lowest unused FCS areas or niches, updating FCS and BPS pointers to the moved atoms and cells (error-prone), and completing the GC to rebuild the FREE list (above the active area).

N.B. Because the compacting affects the locations of atomheaders, some caution is needed if CONDENSE's are done in proximity with checkpointing functions; see section A6.5 for details.

A6. New Checkpoint Facilities

A6.1 Old limitations superseded

A6.1.1 Reading BPS files was essentially restricted to those files deriving immediate ancestry from the Lisp core which generated the most recent RESTORE. That is, the sequence

```
        BPSCHKPT(X1), CHKPOINT(X), RESTORE(X), BPSRESTR(X1)
```

worked in a bare Lisp, but the sequence

```
        RESTORE(CMPL), BPSCHKPT(Y), BPSRESTR(Y)
```

would blow up without warning.

A6.1.2 A BPSCHKPT from an EXCISEd Lisp (compiler and LAP360 deleted) could safely be BPSRESTR'd only into a similarly EXCISEd Lisp, because of the old relocation methods.

A6.1.3 Likewise, without some knowledgeable tinkering by the user, a BPSCHKPT from a Lisp-with-compiler could not be BPSRESTR'd into an EXCISEd Lisp.

A6.2 General comments about Utah facilities

With the present Utah system, the limitations of A6.1 have been removed and new features have been added, mainly the functions WBLK and RBLK for creating and restoring partial-BPS overlay files. My comments in paragraphs A6.3-5 are directed mostly to system builders trying to conserve core, and the following should suffice for most users:

A6.2.1 A RESTORE will reset the PDS length to whatever it was when the file was CHKPOINT'd. Thereafter the length can be changed by doing SETSIZE (section A5) without adversely affecting future BPSRESTRs. If a CONDENSE were used instead, the caution below applies.

A6.2.2 Restores may be done without regard for the particular sequence of ancestry (in the sense of A6.1.1). The only remaining need for caution is if a CONDENSE is used in proximity with a BPS file function.

A6.2.3 Re 6.1.2 and .3, BPSRESTRs will restore the BPS properly, but not the pointers in FCS; a RBLK will do the trick correctly.

A6.2.4 BPSRESTR (and RBLK) will now accept a file created by CHKPOINT as well as by their output counterparts, and will restore just the BPS-related portions from the file. A trivial example would be:

```
        RESTORE(CMPL), use CMPL, EXCISE(T), other, BPSRESTR(CMPL),
```

which would retain the FCS structures built up but reset BPS with just the compiler routines.

A6.3 CONDENSE caution

As noted in section A5, a CONDENSE involves relocating atomheaders (and other FCS cells) and updating in-core FCS and BPS pointers to them accordingly; however, BPS code in a previously-output BPSCHKPT file cannot be so updated and, if now BPRRESTR'd, might erroneously and fatally reference some FCS address whose contents were moved. Two solutions around this inconvenience are:

A6.3.1 If the user's BPS code references only system atoms and functions (and no numbers explicitly), CONDENSE can probably be used safely since anything referenced in FCS is compacted already.

A6.3.2 One general solution is to do all CONDENSE's in the ancestral system prior to involved CHKPOINT's, BPSCHKPT's, or RBLK's. That is build your system, shrink it using CONDENSE to the minimal size anyone might want (want as a user after a RESTORE), and do the appropriate checkpoint function(s) immediately.

A6.4 New functions

In the Stanford and Utah systems, BPSCHKPT and BPSRESTR are used to write and re-read entire contents of BPS. Two new functions have been implemented to handle the bookkeeping for segmenting BPS into individual overlay areas. Logical blocks of code (presumably containing logical blocks of functions) may be arbitrarily relocated in BPS, regardless of intervening BPSMOVE's and SETSIZE's, as long as the blocks and their storage territories either overlap completely or not at all. If this condition is met and the CONDENSE caution is heeded, the W/RBLK pair will handle relocations and maintenance of BPS-FCS linkages. In addition, RBLK "remembers" where a block was read in previously; if the block or another of its overlay "family" is read to the same origin subsequently, the function is quite fast; a future relocation takes a bit more time, once-only for each family.

WBLK(ddname at n) -- this function performs like BPSCHKPT, except that a range is specified by the second and third arguments for the amount of BPS to be written:

at= name of the very first F/SUBR compiled in the block. You may get a "NOT FOUND" error.

n = the address of the end of this block; e.g., the numeric equivalent of (CAR BPS) just after the last function was compiled.
Or, n may = NIL, in which case the current value of BPS is used; i.e., write from at to the end of active BPS.

The value WBLK returns is the end-of-block address used.

RBLK(ddname at n) -- this function performs like BPSRESTR, except that the second and third arguments specify what block is being read and where to store:

at= name of the F/SUBR routine which heads the block when restored. It may be "NOT FOUND" if no common ancestry exists.

n = the BPS address of the origin of the block when read into core and stored; e.g., the address portion of GET(at F/SUBR) as a number. Or, n may = NIL, in which case the current value of BPS is used as the origin.

The value RBLK returns is the first free BPS location above the block read. BPS's APVAL is unchanged even if RBLK'd above active BPS.

A6.4.1 Note that WBLK blocks can be filled in by the compiler in any sequence and at arbitrary (non-overlapping) ranges in BPS. A family of overlays, later to be RBLK'd to some common origin point, need not be built from that exact origin in the ancester's BPS; each member can have a different origin when built, if that is more convenient to the user, but if the building-grounds overlap at all, they must have the same origin.

A6.4.2 Moreover, once one member of a family is created or RBLK'd with the same origin as another member, it is not as free to relocate about in BPS as previously. Instead, the longest member of the family must move first (be RBLK'd first at the new origin). If you have trouble or want to know why, paragraph A6.5.4 may help.

A6.4.3 Examples of this last proviso:

a) The X family has two members, X1 and X2, which were built in and WBLK'd from two separate areas of the ancestral BPS. In the user's Lisp, both are RBLK'd to origin Y; if later they are to come in at origin Z, the longer of the two members should be RBLK'd first there.

b) The Y family has two members, Y1 and Y2, which were built at the same ancestral origin (at separate times). When first RBLK'd into another Lisp (or into the same Lisp at another origin), the longer of the two blocks should be read in there first.

c) The Z family also has two members which were built at the same ancestral origin (at separate times) but, after these were WBLK'd, still more functions ABC were compiled at the same origin. If later a Z member is RBLK'd over these, all is well except ABC are not recoverable. If the Z's were RBLK'd to some new origin not overlapping ABC, you might suppose ABC could still be used. They cannot, however, because their FCS addresses were relocated as if part of the Z family.

d) Similarly, if X1 were RBLK'd at one origin and X2 at another, both could be safely referenced by other user code; but if Y1 were RBLK'd at one origin and then Y2 at another, only Y2 could be called safely because Y1's addresses were relocated. Y1 would have to be re-RBLK'd somewhere before trying to call it, and so on.

e) You might ask now "In (d), if the longest of the Y's must be RBLK'd first at any new location, how could Y1 be read here one time and Y2 elsewhere the second? Shouldn't Y1 be read there first, and then Y2 on top of it?" This is true except if you use the following trick: determine ahead of time (or over-estimate) the longest member's length, add this to the effective origin of each member, and set your third argument to their WBLK's accordingly. Then, each member of the family will be the same length as far as RBLK is concerned, and any can be read at an arbitrary new origin.

f) Note: If you are trying to shoehorn overlays into lower BPS areas below other code, leave a small slop-over area above the block because the files are read in 80-byte chunks.

A6.5 Implementation details

A6.5.1 Perhaps a few details on compiled-code linkages will bear mentioning here for completeness, since these are not discussed in the main manual. The initial links generated by the compiler to called functions really might be called "slowlinks", as they search the property-list of the function named for a F/EXPR or F/SUBR attribute, and dispatch accordingly to the APPLY interpreter or to the BPS or system-routine code. If a SUBR or FSUBR is linked to, the address of the actual function code (relative to NIL) is inserted in the slowlink, making it a "fastcall".

Future use of this linkage no longer checks the atom and its property-list but just jumps directly between subroutines with minor bookkeeping (adding the current NIL, etc.). This is faster, with less overhead, but a fastcall linkage can no longer be traced nor will it respond to lambda re-definitions of the particular called function. In the Stanford version of 360/Lisp, no means was provided to uncall these linkages back to slowlinks. You can now use TRACE, however, to perform desired uncalls (section A8.3); an UNTRACE can be done whenever you're ready to let LISP make it into a fastcall again.

Be careful if you redefine system routines and compile them, since the LAP360 generator relinks any older code-definitions to the new routines in BPS. Usually this is safe, but occasionally the code for several system routines is intertwined such that the RELINK patch could wipe out part of another routine. In such a case, you're better off not trying to redefine although LAP360 can be tricked.

A6.5.2 With that by way of history, you will see that fastcalls in a WBLK would not work when RBLK'd into a different Lisp (or a BPSMOVEd Lisp). Therefore WBLK uncalls any fastcalls within the block it's working on (i.e., any calls outward from the block or between functions within the block). Likewise, if a block is read in by RBLK to a new family origin than previously, all fastcalls from the rest of BPS to this family are uncalled. On subsequent RBLK's to the same origin, there is no need to uncall, and all linkages remain speedy.
RBLK can also read files made by CHKPOINT or BPSCHKPT, if you have a need to, but remember that the latter files aren't internally uncalled.

A6.5.3 This mechanism handles the most general cases, subject to the rules of A6.4; to eliminate this uncalling altogether would, however, place too many (more) restrictions upon the user. Granted the first RBLK of a family wastes time checking the compiler if it hasn't been EXCISEd, but subsequent RBLK's pay no overhead and instead (hopefully) enjoy the greater generality.

A6.5.4 Finally, a brief discussion of how FCS relocation is handled may aid the perplexed user and explain the need for the A6.4 provisos concerning block origin and movement.

Each logical block of functions has in its WBLK file a few figures about length and position, if only so RBLK can decide if room is available. Strictly, a WBLK doesn't know which FCS atoms point to its compiled functions nor what quoted atoms it references, any more than a BPSCHKPT does; only the ancestral FCS remembers this (or retains this information in a CHKPOINT). Compiled functions in core (and in CHKPOINT and BPSCHKPT files) can have SYS or FCS or BPS addresses relative to NIL's atomheader, but a WBLK file will have just relative FCS addresses because the rest are specially uncalled. Therefore, when RBLK'd to some origin, some FCS addresses (type '40' cells) must be relocated appropriately to again reference the block. Namely, only those address-cells refering to this family and by the amount the new origin differs from the previous or ancestral origin. This delta is easily calculated from RBLK's second argument, but to shift all family references (sharing its old origin) requires the longest most-inclusive member to move first.

A7. New partitioned data set facilities

A7.1 PDS files can now be read, though not written, with blocking of (80,1600); this is also the standard blocking for the various SYSFILE functions (RESTORE, RBLK, etc). However, because only one 1600 byte buffer is used in this implementation, a few provisos:

A7.1.1 Text and checkpoint files apparently should not be members in the same PDS as LISP(LISP). The latter would require an unformatted buffer of 7280 bytes, which didn't seem necessary for this version.

A7.1.2 RDS's of text-PDS members should be closed or de-selected before doing any restore-type function involving a second PDS member. Otherwise the single buffer will be pre-empted and a subsequent read of the RDS file will get a software-forced EOF.

A7.1.3 Note: whenever a member is RDS'd, reading always restarts at its beginning, and not at the point reached when last read; this is mostly for bookkeeping ease and avoids confusing the buffer.

A7.2 To invoke just one specific member of a PDS, use the current method of a specific DD statement.

A7.3 To invoke a general PDS file for subsequent input of any member, supply a DD statement for the file itself; then within Lisp, refer to the desired member name wherever you'd currently use an atomic ddname, as the dotted-pair: (ddname . membername).

The output functions (WRS,CHKPOINT,BPSCHKPT,WBLK) will give a non-fatal ERROR message if you try to output to a general PDS file.

The input functions will likewise protest, e.g., if you try to read a member which isn't in the file.

A7.4 Note: the buffer is only allocated once and each general PDS file is really opened only once and never closed, for speed in processing subsequent member requests in the same general PDS.

The OPEN function (still only explicitly needed for a RDS or WRS) has the form: OPEN((ddname.member) SYSFILE INPUT), where the member name is optional for commenting. OPEN will protest however if you try opening the same ddname as both a simple and a PDS filetype.

CLOSE((ddname.member)) is ignored as a commenting call. To explicitly close the PDS itself, do CLOSE((ddname)).

A8. Other Changes

A8.1 ERRORSET(e,at) provides a means, as in other Lisps, to evaluate an expression within an environment protected against ERRORs. Fatal errors will terminate the job or return to the top-level EVALQT as before (according to EXITERR), but other errors, diagnostics, and user calls to ERROR are caught and contained within the current innermost ERRORSET. If none is active, the stack unwinds to the top.

As far as the user is concerned, he calls ERRORSET with an expression e and a flag at. If EVAL of e produces no errors, LIST(e) is returned: i.e., a non-atomic value.
If the user invokes ERROR(at2) during the EVAL, the PDS and ALIST are restored, and the ERRORSET returns at2. Likewise, if an internal error occurs, NIL is returned. If the user's flag at =NIL, the error message and backtrace printout will be omitted; if nonNIL, they will be printed as usual.

A8.2 POSN() returns the current cursor position of the PRINT functions; e.g. returning 1 after a TERPRI(). Heavy use of this function will cause more frequent GC's, as with any arithmetic routine, since each integer returned requires 2 CONS's to construct.

A8.3 TRACE(x) has been extended in the following ways:

TRACE(list)	- in addition to flagging each function's atomheader to signal LISP to trace it when called, in-core BPS is searched and any fastcalls to these functions are now uncalled to permit tracing compiled code (see section A6.5.1). However, for simplicity, fastcalls to RELINKed functions are only uncalled if they address the latest re-compilation of the function; older references aren't checked for presently, and are therefore still untraced.
TRACE(T)	- enables tracing of any previously-specified functions; TRACE(list) automatically sets TRACE(T) for you.
TRACE(NIL)	- temporarily disables tracing in general until another TRACE(T) is given, but retains all passivated functions.

A8.4 UNTRACE(x), if given an atomic argument rather than a list, will affect the printing of tracebacks (after errors) in the following way:

UNTRACE(T)	- turns off the traceback, though not the error message;
UNTRACE(NIL)	- turns on the traceback printing (the initial state).

A8.5 Two auxiliary CONS functions have been added, and are also open-coded during compilations:

NCONS(x)	- performs CONS(x,NIL);
XCONS(x1,x2)	- performs CONS(x2,x1).

A8.6 $$$ $(p), the ABEND function, now takes one argument to specify an immediate dump (if T) or a deferred dump (if NIL); if deferred, the dump is taken for return codes >= 4, but not for a normal termination.

A9. Summary of New Functions, Errors, and Diagnostics

A9.1 New functions in Utah Lisp/360 are:
CONDENSE, SETSIZE, WBLK, RBLK,
BIGP, POSN, NCONS, XCONS, CLEAN

A9.2 New system atoms are:
GC*

A9.3 New fatal error messages are:

```
LISPOUT DD STATEMENT MISSING - RUN ENDED
LISPIN  DD STATEMENT MISSING - RUN ENDED
```

A9.4 New non-fatal error messages are:

```
D1-FILE CANNOT BE OPENED - DD STATEMENT MISSING
D7: WRONG CHKPT FILE, OR NOT ENOUGH ROOM
BPSMOVE - BAD ARG OR TOO BIG
UNCALL FAILED TO FIND ATOM FOR THIS ADDR
R/WBLK -- FIRSTFUN F/SUBR NOT FOUND
BFLT OVFL
EXPT- BIGNUM EXPONENT
ZERO DIVISOR
OPEN: PDS NOT SYSIN OR SYSFILE (80,1600)
PDS DDNAME/ARG INCORRECT
PDS MEMBER NOT FOUND OR I/O FAULT
PDS NOT SELECTABLE FOR OUTPUT
PDS BUG - SYSFILE HIT EODAD
```

A9.5 New diagnostic messages are:

```
SOFTWARE EOF FORCED ON RDS
```