CONFIDENTIAL (1)

XANADU™

→OPERATING SYSTEM

XS-1™ for 16-bit & 12-bit machines.

Pocket Card # 1. November, 1971. Tentative.

XANADU™ is an unusual stream-oriented operating system uniquely meshing core, disk, display refreshment, file creation, editing, coupling and evolution.

SYSTEM PHILOSOPHY

The basic business of XANADU is moving back and forth in streams and pointer streams, and editing the streams.

It is in search of recursion and generality that the system has grown to its present sprawlishness. Lengths and contents must change wildly, files must couple willy-nilly. The dual-tree system structure, as elaborated here, is also intended to be an extensible plan for much bigger systems.

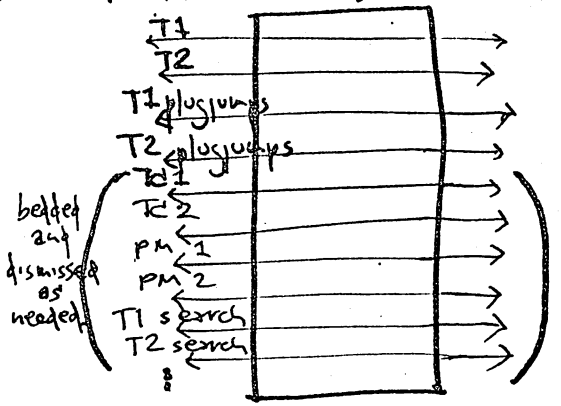
TREE PHILOSOPHY

Everything's a Tree. Programs are a tree. Data is brought to programs as a coupled tree. Things are specified by naming a certain node on the tree; contents of node in are what you want in a specific circumstance.

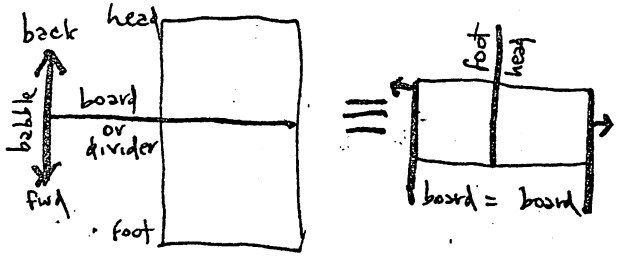
STREAM PHILOSOPHY

All data's in streams. Counts in streams determine the finding of most data, e.g. for Parallel TextFace™.

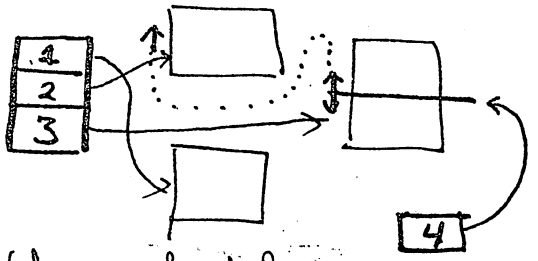
Core contains windows into n streams, each of which may be independently babbled. For the Parallel TextFace™:



THE BED, A WINDOW ON A STREAM



A scatterbed is a distributed bed whose parts may be thrown around into empty parts of core, but which may be babbled as if it were in one piece.

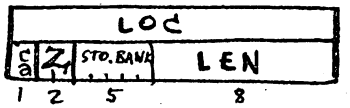


Scatterbeds are preferred for the more transient streams; but for stable streams like T1 and T2 they are less desirable.

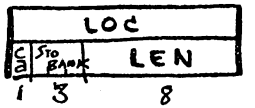
THE CRUM derived from the notion of editing streams

by meander twiddling through pointers, and making the pointer more general: Code, Recusive and Universal, for Meanders. (A meander is a loose string of data, typically in mass storage.)

16-BIT CRUM



12-BIT CRUM (pointing at 7-bit ASCII final meanders)



LOC: starting address of string

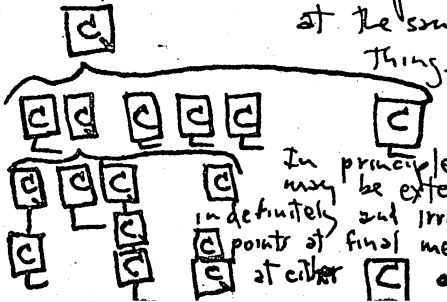
ca: Crum Again bit; string pointed to is composed of crums. To any level.

Z: Zilch bits. Indicate invalidity of first or last bytes respectively (when characters packed 2/word).

STO. BANK: which of 31 or 7 sectors of mass memory the string is in. (All zeroes = core.) Sounds cramped but can be extended through tree coupling.

The crum may first be thought of as a disk-pointer for data.

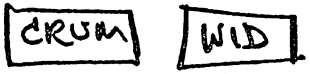
However, crums may be stacked in a hierarchy to any depth, to embrace bigger & bigger files. The crum-again bit (ca) indicates that below it are more crums pointing at the same thing.



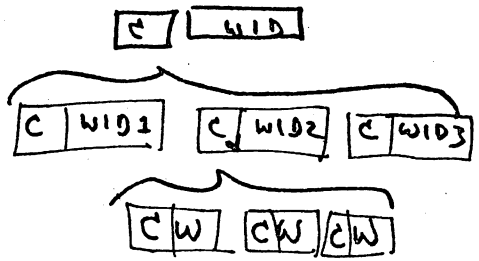
CONFIDENTIAL (2)

XS-1™ Pocket Card #2

In some cases, each crum has its own FREIGHT, a fixed-length field with some preset meaning.



In the most important case, the freight is a WID (width), telling how many data elements are in all the members it embraces.



WID = WID1 + WID2 + WID3

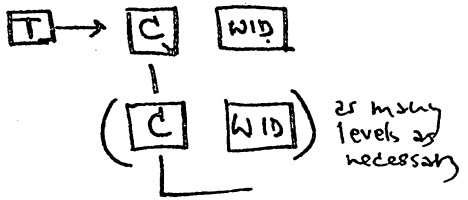
⇒ This speeds search by allowing upper-level check. and so on down.

CRUM-WORK. Editing requires repeating and rearranging crums, and placeholding for stream-counts whose crum-work has not been completed but which must still present a valid count to search routines to make their members accessible.

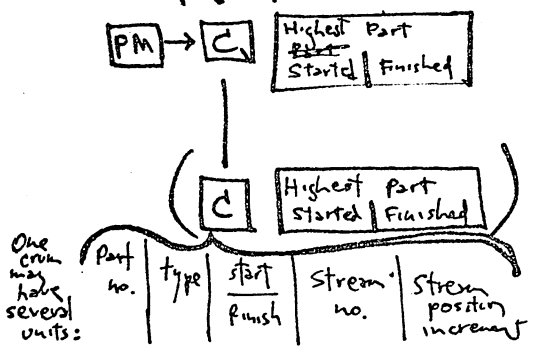
RECURSIVE RULE FOR LENGTHS: If LEN or WID is all ones, the number is too large to hold in a single pointer; therefore look down a level. Or more.

TYPES OF STREAM (tentative)

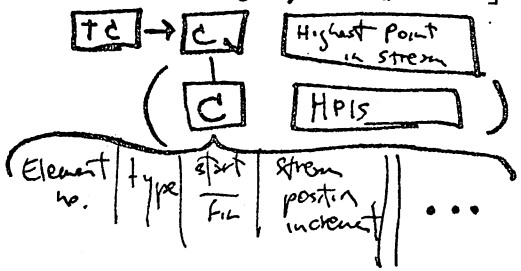
Text (T) [Data (D) is similar]



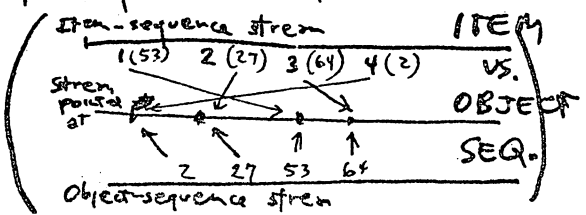
Part Map (PM)



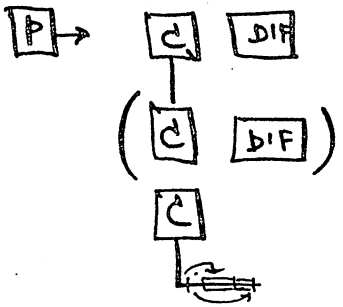
Text Control (TC) [Counterpart Map (CPM) is similar]



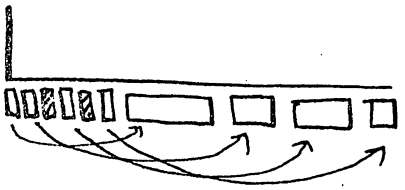
this was originally intended for paragraphs, hyperfext jumps, etc. However, by combining it with the ~~text~~ text-sequence part map, the need for a separate ~~text~~ text-sequence part map is eliminated.



Pieces (P) (esp. for coupling overlay graphics)

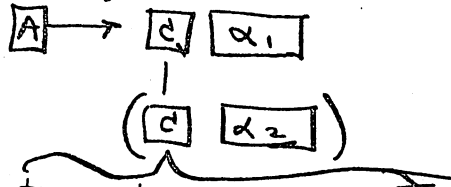


DIF here is the difference between high and low piece-numbers.

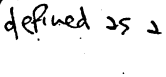


the Graphic Piece has a pointer field, corresponding to the pointer fields of other graphics pieces, and display subroutines which execute these corresponding parts.

Alphabetical Stream (A) (for pointing into directories, computer language interpreters, etc.)



Here, α represents an alphabetical parameter, α2 a further one. Alpha parameters might be defined as alphabetical increment-nos, viz.:



with the difference defined as a number mod 26.

CONFIDENTIAL (3)

XS-1™ Pocket Card #3

NON-CRUM POINTERS

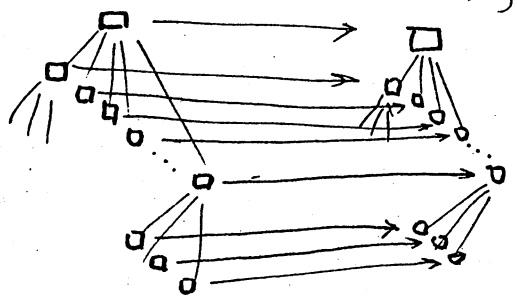
Because the crums are standard two-word pointers, it follows that other pointers — esp. into core — should also be two words. Thus they can be freely ~~[illegible]~~ intermixed.

DUAL TREE STRUCTURE OF XANADU

The crum system constitutes a parsimonious Extremely Movable Dynamic storage Hierarchy. To keep track of it we continuously map it in a dynamic storage map, or **Specification Tree**. Like everything else, the Spec. Tree is embedded in the storage hierarchy itself.

Everything in the system is reached by the tree.

SPECIFICATION TREE (identities) {TELLS WHAT IS IN THE} **LOOKUP TREE** (locations: crums; some in core, some not, always changing; some non-crum pointers, also changing.)
(usually core-resident, somewhat changeable)



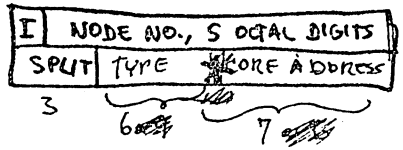
The Dual Tree Structure permits complete relocatability, context and program switching in an open-ended manner. The Spec. Tree tells what is where in the Lookup Tree, and where the LU tree elements are in core, if they are in core.

WHY TWO TREES?

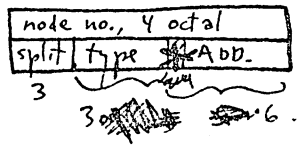
We don't want or need all the pointers in core. We don't want to have to add labels to them when they come in. But we want to have them searchable.

LEAF (element of specification tree)

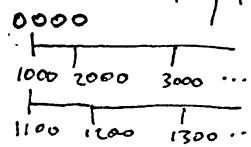
16-BIT LEAF



12-BIT LEAF



Node nos. are hereditary thus:



This facilitates testing heredity for clearing out core of unwanted crums. Note that 5 octal digits allows levels having respectively 1, 7, 49, 343, 2401 and 16807 elements, totalling 19608.

SPLIT tells how many nodes are below this one:

up to 7. may be 0.

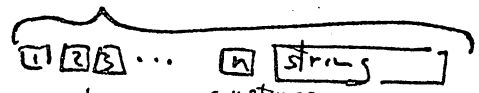
CORE ADDRESS tells where crum is in core table.

*: bit telling whether core address of corresponding crum is in a file's own crum table (0) or the general crum buffer (1). (Cf. PDP-8's page 0.)

BIZARRE FEATURE OF TREE STRUCTURE

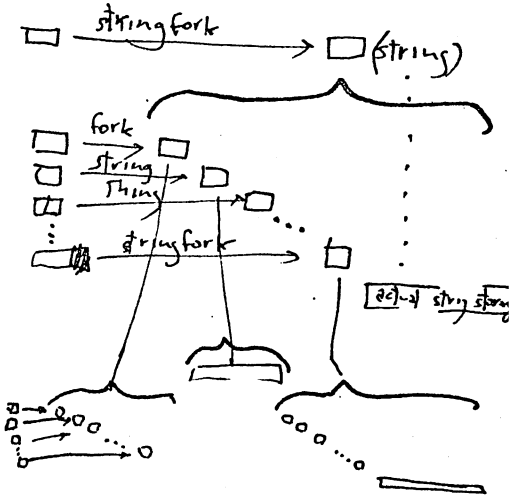
Each crum (till we get down to the data structure) typically has $n \leq 7$ crums below it, plus a string! Length of string is

LEN - length of crums below × 2
(if packed 7-bit ASCII)



As a string is sometimes named by its specifying leaf, we notate thus; as if the string were at the higher levels:

EXAMPLE OF DUAL STRUCTURE

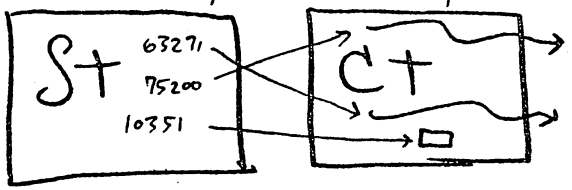


CONFIDENTIAL (4)

XS-1 TM Pocket Ctrl #4

CORE POINTERS & SYSTEM PARAMETERS

are in crum table; node codes let you send.



SWAP ROUTINE.

Clobber the crums not needed, searching node numbers to determine; push what must be saved, or leave it in core.

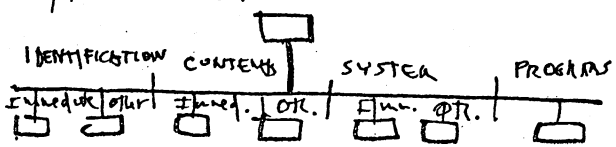
System states specify what's to save or regenerate by node numbers, or categories of node numbers.

TYPE CODE VS. NODE NUMBER

A crum can be identified in principle either by its tree position or type code; thus a trade-off. Former intents of the type code, no longer important: having spec. tree out of order; finding data types at random. PRESENT USES OF TYPE CODE: allowing changes in tree, esp. couplings; labelling parts of operation in progress.

ALL FILES HAVE THE SAME STRUCTURE

A file is sorted with this structure. The System is a file.

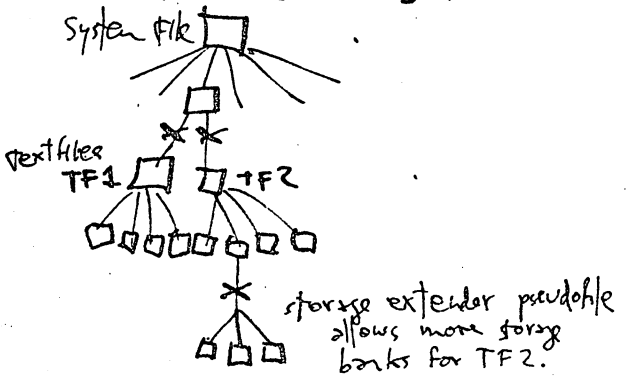


XS-1 p. 4

PRESENT FILE TYPES

System, Directory, Textfile, Interfile (file of connections & history between files), cluster (file of connections among files).

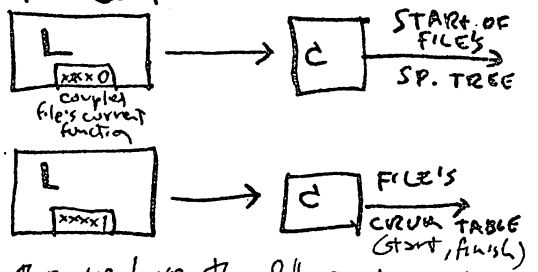
FILES MAY BE COUPLED TO FILES



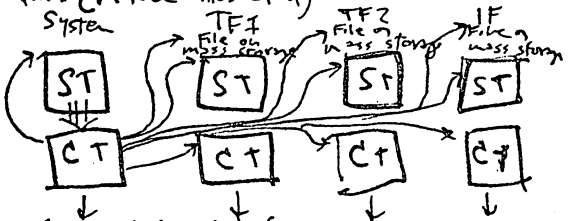
FILE COUPLING AND DISPERSED SYSTEM LISTS

Every open file has a specification tree and a crum table. Through these it handles its own storage allocations and system lists.

THE FILE COUPLER IS:



thus we have the following type of structure: (PTface illustrated)



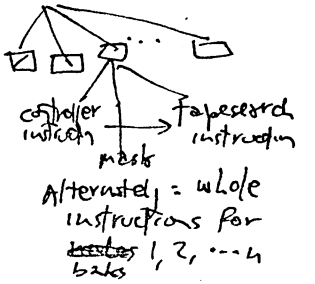
(Beds, lists, etc. for each; much in general crum buffer.)

Note that the CTs need not be as big as their address bits permit, and may overlap.

SYSTEM CRUM TABLE CONTAINS

2-WORD NON-CRUM POINTERS AS FOLLOWS

- Op in Progress (UPDOC)
- Errand stack (THROWBACK)
- Dispatch table for errands
- Dispatch table for display
- PJ Bed
- Furniture
- Character set
- Core management params
- Main core buffer
- Free space list (external storage)
- Storage Bank List



TF CRUM TABLE CONTAINS

THE FOLLOWING NON-CRUM POINTERS:

- own PJ table
- T BED, HEAD, FOOT, BOMBED, CURRENT, FIDUCIAL ELEMENT
- same for TC
- Storage Bank List
- Search crums of T, or their pos'l-location
- Fiducial element in search crums

TF CRUM TABLE CONTAINS FOLLOWING CRUMS:

- Pty
- Hist. Push
- Hist. Matter
- Own stems

NOTE THAT LEAF POSITIONS are absolute for system and files; crum-table positions and all other core allocations are accidental or arbitrary.

CONFIDENTIAL (5)

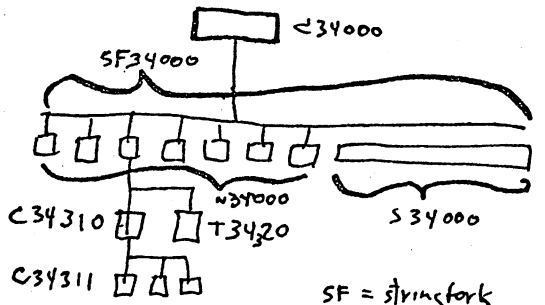
XS-1™ Pocket Card #~~4~~5

TO REVIEW: CRUMS VERSUS LEAVES; TREELOGIC

A leaf is a specifier of a string or other data and its position in a Grand Lookup Tree. Position above is given by node no., below by 'split' field. If the data is 32 bits or less it is stored at one level; if greater, it is stored as a "string," one level below.

A crum is a string address. Tree structures are snuck into the hierarchy by including up to seven crums in another string.

A USEFUL FUNNY ASSEMBLER would put files, programs, etc. into the tree structure on tape or disk with node assignment and storage laid out as requested. Possible notation:

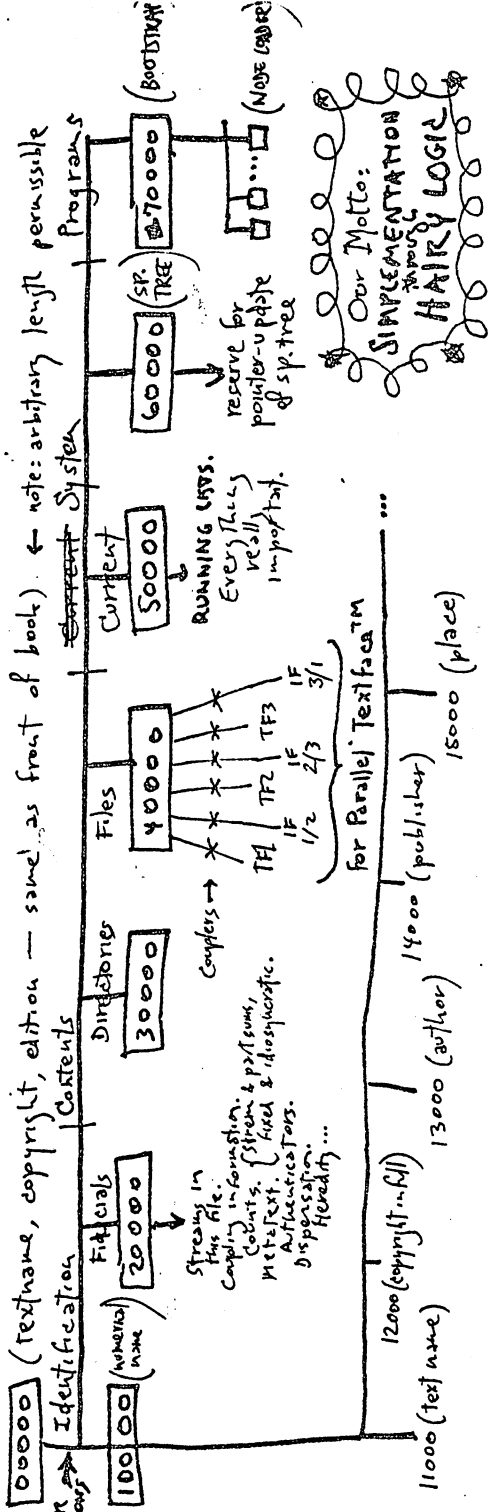


SF = stringfork
T = Thing or thingy
N = Nodes (C or T)

NODE LOADER AND BOOTSTRAP. The Node Loader calls in specified program nodes, as listed in the current stuff. The Bootstrap brings in the Node Loader, at startup or when core really has to be purged. Node loader also sorts its needs for single tape pass (if tape system). NODE LOADER INCLUDES NODE SEARCH TECHNIQUES.

XS-1 625

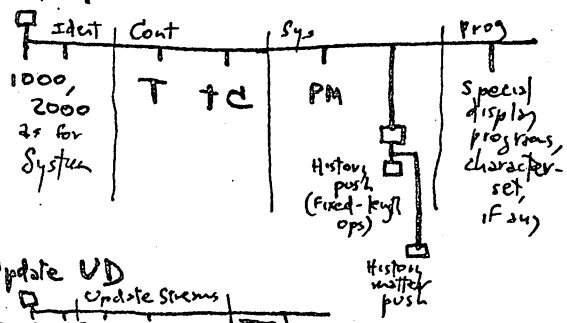
ACTUAL FILE MAP, TENTATIVE, OF WHOLE SYSTEM. Below 300000 same for all files; note that only stuff above 30000 needs immediate implementation. **Note four major divisions, 2d level.**



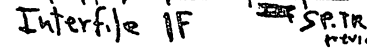
SOME FILE TYPES

Note that CRUM EXTENSIONS DON'T SHOW IN FILEMAPS: "T" in file map could have seventeen layers before you get to Redata.

Textfile TF

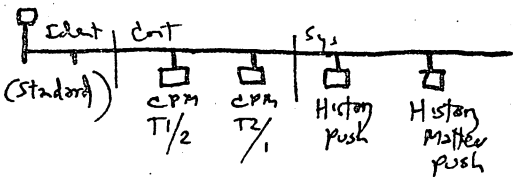


Update UD

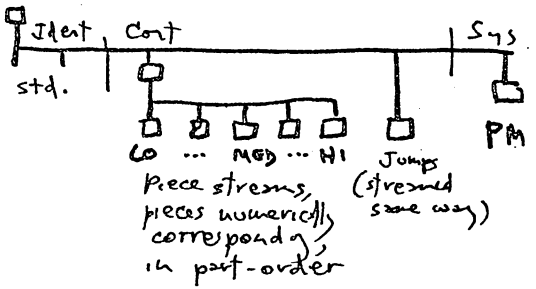


Interfile IF

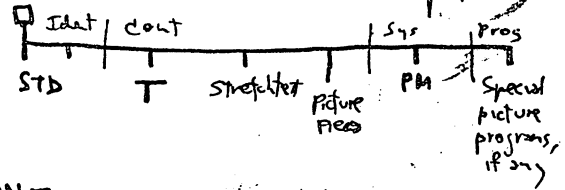
SP.TREE → crums pointing into previous crum & matter structure.



Stretchtext ST



Hypertext with a little stretchtext, some animated diagrams, jumps from both text and pictures.



UNFORTUNATELY we have forgotten to figure out just where to put the Sp. Trees, to which file crum tables are linked. Look upon this as an exciting omission. BEST GUESS: Spec. Tree shd. always be 60000.

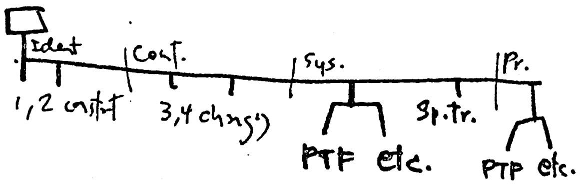
CONFIDENTIAL (6)

XS-1™ Pocket Card #6

© 1971 Theodor H. Nelson

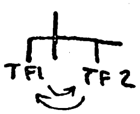
FUNCTION SWAP IN TREE XANADU

~~The dual Specification~~ The Parallel Textface is only the most urgent Xanadu™ Userface™. Others may be added as branches of the same overall system tree:



CHANGING THE SYSTEM TREE

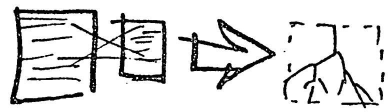
1. INTERNAL OPERATIONS
~~Some~~ since programs find their data at specified nodes, switching crums between nodes quickly makes specific selectors in the program. For instance, if in the PTF, T1 is ~~defined~~ defined as independently moving and T2 is dependently moving, then to move the second file independently we switch them:



⇨ (This is one reason each file keeps track of its own storage and lists.)

2. PROGRAM OVERLAYS AND SYSTEM FUNCTION CHANGES
Because unnecessary crums can be clobbered and necessary ones swapped out in meanders, the system can change its functions drastically. For instance, to ~~call~~ bump the PTF and call the historical backtrack program, → we clobber most screen contents, T1 and T2 and PJs and PTF program, ~~then call in the hist. backtrack program as a separate~~ Then call in the hist. backtrack program as a separate program node. Only

core saves: node loader, screen furniture, place-savers in texts.



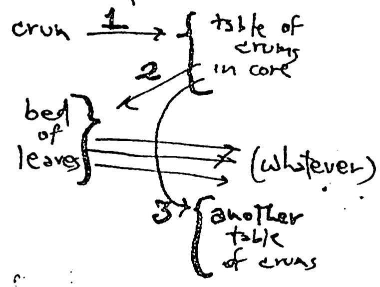
~~[illegible]~~

GENERAL SCHEME FOR FUNNY SYSPOINTERS

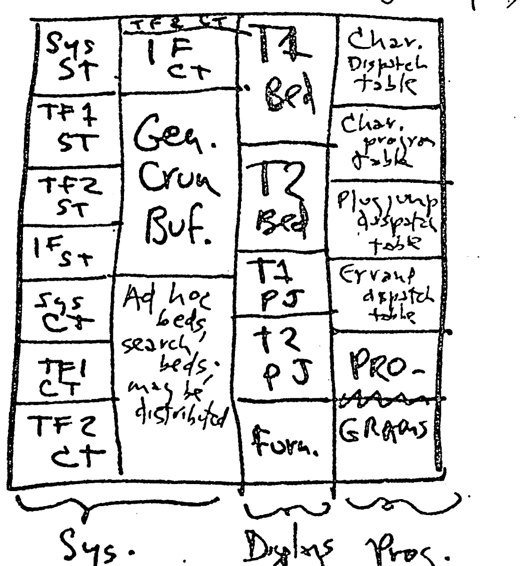
takes advantage of the Leaf/crum duality.

Data crum → Text or Data or Pointers
System crum → ≤ crums ± string

In addition, we may also define three aberrant crum types:

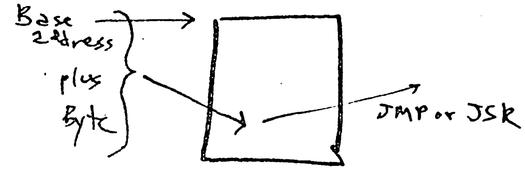


THE GENERAL LAYOUT OF CORE, THEN, IS AS FOLLOWS (for PTF) (ignore sequence)



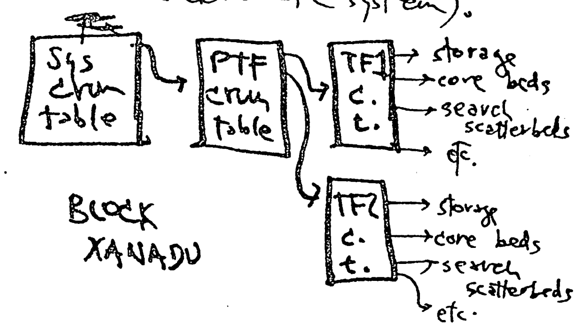
USE OF DISPATCH TABLES

For several types of operations — esp. character display, plugjumps, and the Errand Stack, I prefer the Dispatch Table method of finding out what to do.



AN OVERSIMPLIFIED VERSION

The dual Specification Tree & Crum Table are not really necessary; they foreshadow a time of much bigger systems, with many more user alternatives. A more straightforward approach would use blocks of pointers (crum tables) with the functions of each pointer given (as are the functions of each Leaf and node in the more elaborate system).



BLOCK XANADU

The same Rule of Order, however — of each functional block minding its own housekeeping pointers — seems a good system to build from.

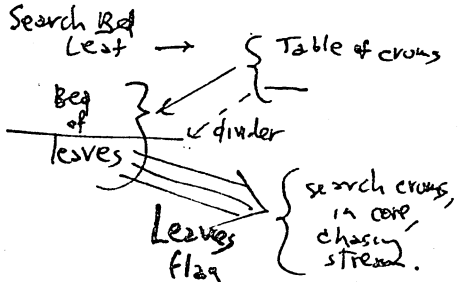
XS-1 DZ 6

CONFIDENTIAL (7)

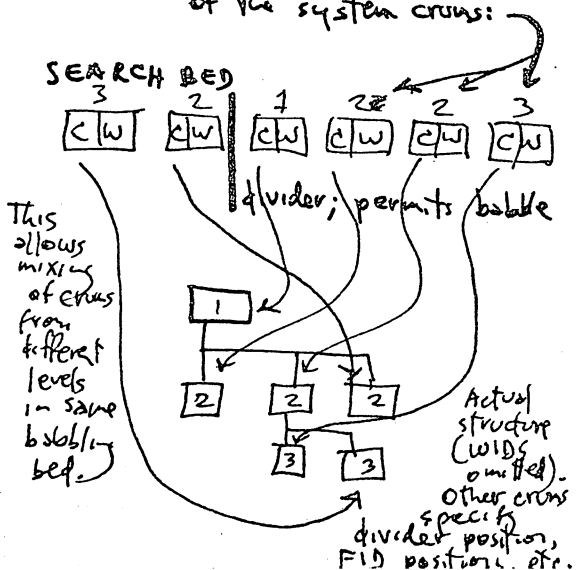
XS-1™ Pocket Card #7

MISCELLANEOUS SYSTEM FEATURES.

THE SEARCH BED is used for chasing along snaggled-up streams stored all over. This uses your Type 2 special crum:



The relative crum level of the system crums:

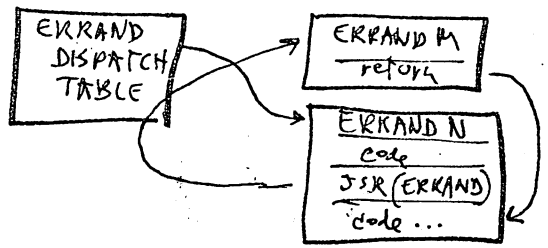


ERRAND STACK or Throwback

The basic types of deferred work to be done — edit changes, display changes, crum-work, disk or tape fetches — should be put in one or more stacks, till there's time for them.

The format will depend on the chosen pointer method. An equal-sized opcode is desirable, with 'edit matter' of course being (a separate stack of?) crums.

A favorite idea: errands to be dispatch-table subroutine jumps, themselves to be defined either by more such errands or raw ~~codes~~ code.

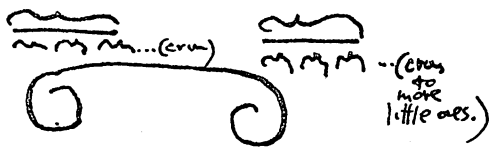


Note that errand definitions may also contain branches.

FREE SPACE

Free space on any medium begins as a contiguous block (blocks) and is whittled down.

For DECtape, presumably we want a long whittlable space at the little leftover spaces noted separately ~~for each tape~~ by tape sequence.



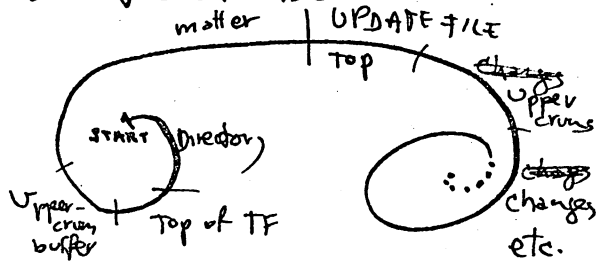
Some way of compiling the small free spaces along the tape, or at the ends would be good.

FILE STORAGE IN GENERAL

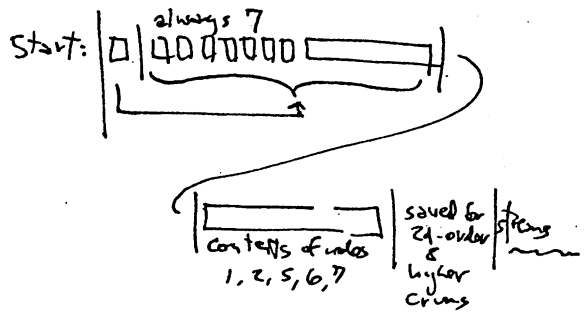
A file opens with its topmost crum, which points to everything else; its title and map (specification tree) are found at standard places.

A directory, also a file, tells where the other files begin.

DECTAPE STORAGE



SINGLE FILE ON DECTAPE



Sequence of files:

DIRECTORY, SYSTEM, CONTENT FILES, UPDATE...

any old way

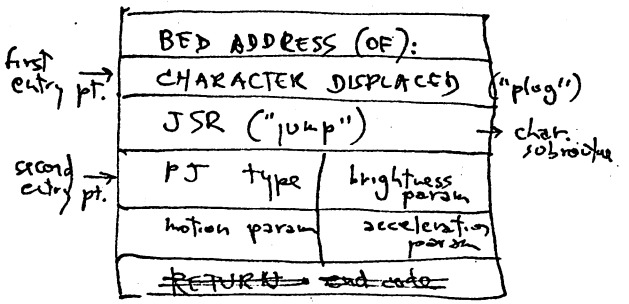
CONFIDENTIAL (8)

XS-1™ Pocket Card #8

DISPLAY SYSTEM & PLUGJUMPS

Each character to be displayed is used as a jump address into a dispatch table, which goes to the character routine. OR, if it is a plugjump, the routine first looks it up in a plugjump bed (which babbles). Plugjumps, being in sequence, may be stepped through; the bed address is an additional link to its origin.

PJ (provisional)



WHY PLUGJUMPS?

So that text can lie where it is brought into core, display jumps are inserted by the display program to make a circular list of all display matter. These PLUGJUMPS consist of what was moved, now to be displayed, followed by information on what is to be done now to the display. By this means we may have smooth text motion, growing characters, butterflies flapping around, etc.

PLUGJUMP TECHNIQUES

1. On babble in, set plugjump for end of textline or other activity, move character to PJ and replace with 'plugjump' symbol. Finish PJ as desired.

2. On display: a) show character, b) perform in-stream modification of display routine, depending on PJ type, by resetting brightness, adding acceleration param to motion param, adding motion param to current screen position.

3. On babble out, return character to bed, relink display with next plugjump.

Successive-frame animation methods (butterflies, etc.) will not be covered here.