1 An Essay on Part of the AHI Program

1A J. F. Rulifson, 8 June 1968

1B .NBL=2;

1C Foreword

1C1 This paper is a term project for Phil. 245 (Psycholinguistics), Spring Qtr. 1968. The Foreword discusses the origin and history of the thoughts presented later. The paper then loosely describes the practical, architectural goals of the NLS (On-Line System). The currently implemented NLS is subsequently analyzed from a more general, formal viewpoint. The notion of "intellectual augmentation" is discussed. A general strategy is offered for testing, evaluating and designing system features. Finally, two possible experiments are outlined.

1C2 The impetus to write a paper such as this grew out of three strong, personal feelings.

1C2A The AHI Center is supposed to "Augment Human Intelligence" but no one can give a precise explanation of what that means. There is, therefore, no measure of success and this results in a personal feeling of through frustation.

1C2B Research in the AHI Center is at a standstill, and the rut is impossible to get out of unless a fresh approach is

taken to NLS.

1C2C After all the work that has gone into building a flexible system, I want to see the flexibility used to truly improve the system. There is a wealth of potential. It can be exploited only if we understand the potential, the process being augmented, and the interconnection.

1C3 The relationship of this to psycholinguistics will be seen later. NLS is supposed to be a system that augments its user's capacity to organize and remember information. Language plays two roles, so important that they led me to Phil. 245 in search of ideas.

1C3A The user must talk to the computer. No matter what he does, he is using some form of language, even though the language may be trivial.

1C3B Part of the information which the user manipulates and organizes will be stored in natural language. Psycholinguistics bears heavily on determining what the user does or accomplishes through the transcription and manipulation of the information.

1C4 The paper grew in the following way:

1C4A During November 1967 I happened on some of Piaget's work

concerning groupements. I became very excited and tried to relate his work to NLS. A term paper for CS225 summed up these results--it was a complete disaster. The vagueness of the then NLS framework coupled with my rudimentary understanding of Piaget resulted in groping thoughts and nothing concrete.

1C4B To try and get some direction I talked with Colby, and discussed the possibility to doing some kind of testing with the current system. I then felt that only through some sort of empirical study could new, fresh thought come forth. Colby suggested that I look at an experiment designed to discover the way small children index limited, self-contained fields of information. The idea was appealing but it was not tied to the formework of the project. Slwoly I developed the attitude that there was no adaquate framework to couch an experiment in.

1C4C In early May Suppes convinced me that empirical testing must be done in parallel with clinical work. This made some problems very clear, for statistical studies require a concrete task, or problem domain, and I could not conceive of one that was related to the overall goals of NLS. What was needed was an explicit formalism that NLS could be put into. Statistics could then test statements about the formalism.

1C4D This paper now began to take form. The notions of separating the control language from the forma (defined below)

were laid out.

1C4E In mid May I had two meetings with Bill Ward. These led to the inclusion of associatve structures as a basic facet of the structural component of the forma.

2 Goals and Constraints

2A The one unifying, pragmatic mandate from the sponsers charges the AHI center with building the AHI Super Screen. It is a large collection of electrical and mechanical devices connected to a computer. The elusive ground rules for construction and guide lines for development have often made progress slow. No one knows what is being augmented, but everyone has firm ideas about features needed in the system. This characterization attempts to isolate and bound a single problem area from the many of the Center.

2A1 The tool is, like many tools, and artifact and an extension of man's basic senses. Not totally unlike paper, pencils, chalk, and blackboards, it is an information storage, retrieval, and manipulation device. Simply, it is a dynamic memory extension.

2A2 (information structure) The tool works on documents, or files. This paper was written exclusively with the current prototype of the eventual tool. The documents are called information structures to emphasize the point that they are blocks of textual and graphical information organized into hierarchical

and associative structures. The total information content is in the structure as well as the blocks, a point many automated textual systems ignore (as do many writers).

2A3 Language, symbolic reasoning, and problem solving also serve as tools [Bru66:37]. The Super Screen might someday have a similar role. It is not inconceivable that the tool could take on currently unknown features that would give its user a power of thought only realizable through the artifact, much as the power of language is reasiable only through speech, and numerical analysis through the computer and its ability to do the computations.

2A4 The unique feature of the Super Screen, over other rudimentary passive tools like paper, is that it is not only responsive but can stimulate the user while he works. In [Bru67:56] the analysis of human implement systems discusses motor, sensory, and ratiocinative capacity amplifers. The threefold amplification is found in the Super Screen.

2A4A It makes huge volumes of information availabe for viewing and manipulation through minute keystrokes.

2A4B In a way that is becoming common with computer-controled devices, the Super Screen reacts instaneously to user commands. The commands, moreover, could be given in structurally complex languages, something impossible with simpler controlling

devices.

2A4C Unlike any previous devices, the Super Screen adds a new dimension to ratiocinative implements, for it observes the operation of the user, his information structures, and his habits. These observations may then be used to provoke thought and stimulate the use of different structures.

2B Emphasis has been placed on the personal nature of the tool not from disregard of the problem of communication among fellow workers but out of caution. The development thus far could easily assimilate many arears of computer science. Further constraints, necessary to bring the problems down to practical levels, are best set by conjunctive disassociation rather than extension of the original premise, especially since the constraints are necessarily artificial.

2B1 Unfortunately, we must not attempt to radically change, through research and development, already existing cognitive tools of the Super Screen user.

2B1A Increased efficency in personal habits may readily be expected of a Super Screen user, just as increased typing speed could be expected of one using a typewriter.

2B1B [Bru66:56] states that "any implement system to be effective must produce an appropiate internal counterpart, an appropriate skill necessary for organizing sensorimotor acts,

for organizing percepts, and for organizing our thoughts in a way that matches them to the requirements of implement systems."

2B1B1 However, we consider it outside the scope of this project to research and make significant contributions toward accelerating the evolution of these skills.

2B1B2 The tool is designed for use in tasks so ill-defined that there are no measures of success or concrete working rules. No one knows of universal rules for taking notes or wrting papers; finding such rules is a project of different orientation.

2B1C Neither are we modeling the person or his thoughts. The aim is to find a mutually agreeable class of models which the user and the computer can easily work on in unison. The ultimate desire is that the information structures enhance the memory of the user beyond the capacity of more conventional artifacts.

2B2 The most difficult criterion to set is an idea of usefulness. The more a feature or aspect of the tool amplifies or stimulates the cognitive or organizational capacity of the user, the more useful it is. A measure of usefullness is untimately a function of the things being augmented. This is discussed in more

detail after the nature of the tool itself is described.

2B2A This definition is more restrictive than the one generally pursued in the AHI Center, where smoothness of the system and application to specific problems often takes priority.

2B2B This does not leave features open to testing. If usefulness is to ever have a well-defined measure, it must be related to a strong, restricted thesis.

2B3 Finally, we make the strong requirement that the services offered to a Super Screen user be limited to state-of-the-art computer science. This avoids the great hangups of natural language and artifical intelligence.

3 Forma and Model

3A The computer and the Super Screen user converse in an artifical language about specific information structures. For the user to realize the full potential of the tool he must have abstract notions of how the structures are built, connected, and manipulated.

3A1 (forma) This set of rules, from which the user can derive specific models of the structures in the computer, is called the forma of the system. The "forma" is in no way related to the programming techniques used within the software of the system. It

is a formalism used to characterize all possible allowable information structures.

3A2 The idea of a forma is similar to two more common notions.

3A2A An analogy can be drawn with logic, where axiom schema are given from which all possible formulas in the system are derivable.

3A2B The forma is loosely related to the "schema" of Piaget. [Fla63:54] considers a schema to be "a kind of concept, category, or underlying strategy which subsumes a whole collection of distinct but similar action sequences." Thus the forma is both a mathematical characterization of possible structures and a general knowledge of how the structures are built and manipulated. The forma should properly include the user heuristics for the problem at hand, but they are deliberately left out.

3A3 A forma has three components:

3A3A (lexical) The lexical component of an information structure is the content of the nodes of the structure.

3A3A1 In the prototype systems, the structures are blocks of text and pictures combined in a graph structure. Each numbered string of text in this paper is such a block. The

rules which describe the possible configurations of text and lines in such a block are the lexical component of the forma.

3A3A2 More precisely, any node in the current system may be a string of characters. The character set has 96 common characters and the maximum length of any such string is 3000 characters. Line drawing is just being implemented and its rules are much more restrictive.

3A3A3 Factors which limit the usefulness of the prototype systems are rarely concerned with the lexical component. Occasionally the need arises for a special character or font, but in general this has little bearing on forma design.

3A3B (structural) The structural component of a forma is the nature of the connections and relations that can be imposed on the nodes of the lexical compontent. [Wh65:215] suggests that the structure of adult mental processes has an associative, fast-acting level and a slower-acting, information-processing cognitive level. If we take (as in [Bru66:48]) the hypothesis that "experience is organized to correspond in some measure to the structure of language," a forma must have both hierarchical and associative aspects.

3A3B1 (hierarchical) A structure is a means of implying

information missing in the nodes of the structure. When the information is supplied purely through the structure, and is independent of the content of the nodes, it is a hierarchical structure.

3A3B1A The tree structure or outline form of this paper is a hierarchical structure. This widely used format has a curious relation to the concrete operation of Piaget [Fla63:164-196]. If information is categorized or put into class heirarchies, any of the groupments and operations can be handled within the tree even though it is not as general as a directed graph.

3A3B1B The utility of the tree suggests that it is adequate for currently conceivable formas. It may even be too general. Natural language is highly restrictive in its structure. The hypothesis mentioned above implies that is the case; one of the experiments outlined below is designed to test it.

3A3B2 (associative) When the information supplied by the structure is simply the fact that nodes are related because of common content, the structure is associative.

3A3B2A All of the attempts to permit the user of prototype systems the facility of associative structures seem to miss something. Techniques such as naming nodes

(putting a word in parentheses at the front) and embedding the names as references in other statements coupled with commands to move the viewing window to a node by pointing at its name have been the extent of user aids for associative structures.

3A3B2B This work has neglected two important aspects of the associative level of mental processes.

3A3B2B1 [Wh65:189-194] finds that associations are made by subjects and lost for further use in fractions of seconds. This means that, if the associations are to be communicated to the computer, the process must be instanteous. Typing in names takes seconds, not fractions of seconds.

3A3B2B2 Secondly, nodes are associatively related if they have a common concept. Previous work within the Center cast associations and linkages in the framework of reference and referent (pointer and name). A more realistic associative scheme would link nodes through common terms or concepts, on the order of inverted files used in information retrieval.

3A3C (dynamic) The dynamic component is the nature of the way things can be changed. It describes the vast variety of ways the text, pictures, hierarchies, and associations can be

manipulated. This aspect of the forma is the abstract way the abstract forma is manipulated. It is not the languages or commands used for the actual manipulations of information structures.

3A3C1 The trend in prototype systems has been to include everything one can think of so that no user will ever feel constrained.

3A3C2 When this is done, however, care must be take to make consistent definitions and conventions. Obscure manipulative features often lead to misunderstandings of the system.

3A4 The most important characteristic of a forma is that the user understand it. An exceedingly complicated scheme which no one can use is not a tool; it is burden. The simplicity of the formas of the prototype systems is one of their best virtues.

3B (model) It seems at times like a trivial point, but a useful distinction can be made between the information structure in the coumpter, the way a person thinks of the structure, and what he sees as he looks at the Super Screen.

3B1 A model is a specific representation of a specific document or information structure. Thus, "information structure" refers to the actual information, as it is stored in the computer. "Model,"

on the other hand, refers to the way the user thinks of his structure in terms of the forma of the system.

3B2 The actual, physical representation of the model is on the Super Screen and is different from the model and the information structure. The user sees characters laid out in a square format; he thinks of the information as being in a tree; and it is really stored using a fancy free-storage technique.

3C (special purpose) In all of the prototype systems there have been a great many special purpose features. These have included the ability to scan every node for textual content and perform arithmetic on lists of specially tagged numbers, and special formatting conventions for tables. Schemes have been proposed for allowing the user to view the structure implicit in highly structured data, in both the lexical and the structural component.

3C1 The features are difficult to discuss relative to the forma. They are aids more closely aligned to motor and sensory capacity amplifiers than to ratiocinative amplifiers.

3C2 They certainly make the systems smoother to use, and they have been invaluable in well defined tasks. They are, however, ad hoc, and have nothing to do with the problem area attacked by the notion of formas.

4 The Communication Language

4A The user communicates with the machine about models by using an artificial dialogue language. This is not the natural language of the lexical component of the forma. It is the language used to direct the machine in its editing, organizational, and retrieval tasks as well as the language used to inform the machine about structural relations and associations.

4A1 Obviously the nature of the language will dictate certain user habits. How this affects the capacity amplification of the system cannot be understood until the amplification itself is determined.

4A2 The dialogue language is nonreflexive. The user currently strikes keys, pushes buttons, and rolls a cursor-positioning device with his hand. The computer responds via the CRT.

4A3 This language plays a vital part in the system. Its flexibility and power crucially determine the success of system features. Stealing the linguistic approach to language analysis and perverting the terms "phoneme," "morpheme," and "structure" leads to rich insights about the contemporary dialogue language.

4B (phoneme) The phonetics of the dialogue language is the set of basic units of communication between the user and the machine, and vice versa.

4B1 There is a strange imbalance between the dialogue language

and natural languages.

4B1A There are about 100 keys in the user's half of the dialogue language. This is more than twice the number of phonemes in English, yet the artificial language is much more restricted.

4B1B This difference would seem reasonable if the communication speed using keys were faster then speech, but it isn't.

4B2 The dialogue language is reflexive only on the phoneme level. The characters displayed on the screen and the ones struck on the keyboard are the same.

4C (morpheme) The morphemes of the dialogue language are the basic units of discourse about the models. The morphemes need not correspond to the atomic units in the structural, lexical, or dynamic components of the forma.

4C1 This is a very important point, for it means that models can exist separately from the language used to manipulate them. In practice this is true even though it has not been previously recognized. When someone is taught about the system, terminology is used which is not in the dialogue language. Users continue to think about the models with these concepts, yet they are not

realized in the language.

4C2 Awareness of this means that the morphemes can vary without changing the formas. Historically, the morphemes of the dialogue language have been considered the forma. The results of designing without consideration of this feature can be seen in the prototype systems.

4C3 Two striking observations can be made when the user's dialogue language is compared to natural language.

4C3A The morphemes are almost identical with the phonemes. This explains the large number of phonemes. But the utility of this feature, even if it is only speed, is still untested.

4C3B The lack of symbolism or abstractness is a glaring deficiency in the language. All the words are concrete. They are always in reference to a specific action about a specific entity in the model.

4D (syntax) The structural or syntactic analysis of the dialogue corresponds precisely to similar analyses of natural language.

4D1 I know of no structurally complex man-machine interactive language. From job-control languages to highly interactive control languages, the structure is always that of a simple linear grammar. The so-called high-level languages of programming

do not apply here. They are used to describe algorithms to a computer program specifically designed to translate the algorithm to another language. These language are not interactive; writing in them takes a good deal of time.

4D1A Except for a few possible queries about the computer's status, the dialogue language is strictly imperative. It always tells the machine to do something, to change the portion under view, to modify a parameter, etc. This could explain the sufficiency of simple structure, for even in natural language imperative statements tend to be short and structurally simple.

4D1B A more adequate explanation seems to be that structural complexity is closely connected to semantic complexity. Simple concrete semantics, such as a forma, do not need structurally complex languages to describe the allowable operations.

4D2 Just as with morphemes, the syntax of the dialogue language has been confused with the dynamic component of the forma.

4E The common analysis methods applied to the study of speed and redundancy of natural language do not apply to the dialogue language.

4E1 In all of the prototype systems, the semantic complexity of the forma is so trivial that single, short commands have been used to implement the user's entire command language. With the current system, the number of commands is becoming larger than the number

of available keys. We do not, however, have anywhere to look for guidance in studing the problem. All work oriented toward the speed and redundancy of artifical language tends to be concerned with communication on the bit and hardware level.

4E2 The computer, through the display screen, can transmit information so fast that the problems again shift back to the user and his ability to read and follow the computer. Observations about response speed in time-sharing systems do not apply to the theoretical level of this discussion. The output channels from the computer handle information much faster than people can assimilate it. The slowness of the input channel is mainly due to the inadequate dialogue language. The computer can certainly handle informaton faster than people, and the limitation on input speed is neither human dexterity nor hardware capability but rather the capability of the control language..

5 Augmentation

5A Through the notions of forma and dialogue language we have attempted to bring the concept of the Super Screen into the arena of rational discussion. The next, and more difficult, problem concerns the ultimate goal--augmentation of intellectual processes. We do not solve this problem; we only offer notes for further thought.

5B The Super Screen, viewed as a capacity amplifier, augments only one portion of the user's intellect--his performance in the task of

information organization on a personal memory level.

5B1 The problem, then, reduces to the overwhelming task of discovering a theory behind the processes of the user as he does the organization, and relating these processes to specific features of the forma or the dialogue language. With such a relationship, we could make hypotheses and test them.

5B2 The notion of performance must not be restricted to the current user task. The availability of the Super Screen may well mean that the user could undertake organizational task which would otherwise be prohibitively difficult.

5B3 The user processes must be restricted if a theory is to be found. This can be done in two ways.

5B3A The first is the task the user is performing. This would be things such as writing a specific kind of paper or computer program, or taking notes while reading a book.

5B3B The other restriction defines a motivation of the user. There are three motivations that appear to lend them themselves to theory.

5B3B1 The Super Screen user could be using the system as a means of personal recall. NLS becomes an extension of his memory. Within it he can store facts, relations among the

facts, and structural information about both the facts and the relations.

5B3B2 The Super Screen can be used as a device to communicate with other Super Screen users. Entirely new approaches can be taken to paper organization if reading will always be through the system. If the intent of the user is communication, the structural devices are standard enough to hypothesize on, and maybe even to test.

5B3B3 Finally, the user may be attempting to correlate information. The process of forming associations, relations, and concepts has been highly studied. If users were set to this task, various current psychological theories could be used as the basis of test concerning both the forma and the dialogue language.

5B4 A serious problem is that individual user ability, independent of the Super Screen, overshadows the results of being augmented.

5C There is a tendency to view the whole project as the construction of a big text editor.

5C1 This is done both by outsiders when the program is explained to them and implicity by insiders when they think about expansion

and new user features.

5C2 This seems to happen for two reasons:

5C2A The text-editing problem is well defined. With simple manipulation of text, graphs, and structure the features and use are already laid out. Authors and editors feel that they know what they want.

5C2B It is very easy to get a big gain in paper production with such a system. Dozens of rough draft copies, all neatly formated, right justified, numbered, etc. accumulate in piles on the desk of the system users. This is fun, but it is also dangerous. The increase in output is often viewed as an intellectual or quality-performance gain--this is a big mistake.

5C3 It is becoming clear that text editing, even with the Super Screen, augments intellect in only a trivial way [Bru63:298-301]. The notions of the Whorfian and Neo-Whorfian [Eng62:24] hypotheses are just not enough.

6 Research Strategy

6A This has little to do with psycholinguistice; it is deliberately left out of the term paper. (It is also a cop-out because I did

finish it.)

6B introspective development

6C technical development

6D empirical research

7 Two Experiments

7A The Value of Restricted Hierarchies

7A1 This is a set of experiments instead of just one. The purpose is to study the effect of structure, as used in NLS, on the process of communication.

7A2 The general idea would be to take a paper or article, and transcribe in into the system in various versions with a variety of structures encompassing a spectrum of complexity.

7A2A The simplest format is the standard paragraph format.

7A2B A next step could be multi-sentence structure, breaking the paragraphs into related units, each unit composed of one or more sentences.

7A2C A final form might have at most a sentence, and

complicated sentences might be broken down further.

7A3 Various hypotheses could then be made about the way users prefer to read using the Super Screen. These could be tested by observing the commands used to move the viewing window over the parts of the document when subjects are asked to figure out what it says.

7A4 Another set of hypotheses could be made concerning the best way to structure the document to convey its contents to a reader [Ronco:19-21]. Subjects could then be asked to read the file, and use the system to make notes. We would test for comprehension and rentention of material.

7A4A The idea that the minimum structure necessary for good comprehension and rentention is a candidate for testing.

7A4B Another candidate is the idea that complicated structures can convey considerable information, provided the user is aware of the meaning of the structure. Here, various structures could be explained before the subjects read the file. Only selected structures would actually be used in the file. The results of the test afterward could give clues abut the utility of the forma in its present form.

7B Kinds of Concepts in Associations

7B1 The motive behind this experiment is to get some clue, however small, about the use of associations in concept formations, the speed at which they occur, how they are eventually used in hierarchicaly concept formation, and what kinds of features could be added to the Super Screen to allow users to record them and retrieve them.

7B2 The experiment is basically clinical. The idea is merely to interview children about a specific topic. I recognize all the problems connected with children, but I see no other place to start. The interviews would merely probe the child for all the information he has about a small number of highly related objects. The hope is that close analysis of the transcribed interviews will lead to ideas about indexing, associations, and the transformation of associations into structural concepts.

7B2A The area of personal relatives is well defined. Questions concerning the location of individuls, such as grandmother, and of sets, such as mother's parents, might initially be answered thourgh associations. As the questioning procedes, and the child's thoughts stabilize, the associations may progress to structures. To observe such a change and to see the origins of a structure could lend a lot to further refinement of the forma.

8 Summary

8A I feel this paper has made progress if two very simple things have been accomplished.

8A1 If the idea of the Super Screen has become firm enough to recieve meangingful criticism. In the past there have only been three kinds of comments: approval through head nodding, suggestions for generally kludgey special purpose features, and the advice that we are all wasting our time. This quickly becomes discouraging. I want to make the global concepts we work with clear enough to be understood and critized by others.

8A2 I also want the global view to be extendable to research strategy and experimental verification. The experiments must directly relate to the forma, dialogue language, and augmentation process if they are to mean anything.

8B I realize that the discussion of augmentation is lacking. It needs much more thought, and I need a lot more readi:g. However, I also feel that the experiments outlined above could be started. The results would assist in solidifying the ideas about augmentation as well as clearing up more points about the forma and dialogue language by putting them into pratise.

9 References

9A [Bru66] Jerome S. Bruner, Rose R. Olver, Patricia M Greenfield,

et al, "Studies in Cognitive Growth," John Wiley & Sons, Inc., 1966.

9B [Eng62] D. C. Engelbart, "Augmenting Human Intellect: A Conceptual Framework," Summary Report, Contract AF 49(638)-1024, SRI Project 3578, Stanford Recearch Institute, Menlo Park, California (October 1962).

9C [Fla63] John H. Flavell, "The Developmental Psychology of Jean Piaget," D. Van Nostrand Co., Inc., 1963.

9D [Ronco] Paul G. Ronco, et al, "Characteristics of Technical Reports that affect Reader Behavior: A Review of the Literature," Institute for Psychological Research, Tufts University, Medford Mass.

9E [Wh65] Sheldon H. White, "Evidence for a Hierarchical Arrangement of Learning Processes," in "Advances in Child Development and Behavior" Vol. 2, pp. 187-220, ed. by Lipsitt & Spiker, Academic Press, 1965.

/NETPL, 09/11/68 1719:29 DGC ; .NBL=2; .PLO=1; .DPR=0;

Abstract

This paper represents JEF's and JDH's current views on how the NET problems should be attacked. The problems covered are those local to the 940 Monitor and NLS.

These two areas, NLS and the Monitor, bring out most of the low-level NET communication problems, and we hope that a general NET solution can evolve from these ideas.

This paper is currently for AHI communication, but we would like to pass it out at the next NET meeting in Utah, so we need responses in time to resolve difficulties.

General Notions

Aim of Paper

This paper gives a fairly detailed explanation of an IMP-940 Monitor implementation scheme that we feel would make the whole mess work.

It attacks the single problem area of interactive display terminals, and does not get into complex interactive programming running. Interaction between user programs can easily be accomodated within our proposal.

This is a two-pass paper. One may not expect all new concepts to be introduced before they are needed.

Comments on Other Papers

TRAC

S. Carr's memo on number of translators

Network Software Interfaces

GULP

GS -- Graphics System

Elmer's memos

Letter dated July 5 1968

Letter dated May 28 1968

Letter dated August 12 1968

Memo dated 12 August 1968

Memo dated 15 August 1968

Memo dated 26 August 1968

ARPA Specifications for the IMPs

Results from Meetings

Familiarity with each others' consoles

Not much else, a lot of hardware talk

NET vs. Data Communications

Make full use of host computer

Real-time on-line consoles need it

Large number of different machines

Using Your Console

Expect no console degeneracy

Even expect extra features at times

Division of Labor

Host shoud not be aware of actual console at user site execpt in very general ways,

User's own computer can tackle most of the formatting and translation problem.

Extra Thoughts on General Notions

AHI MONITOR - IMP Communication

INTRODUCTION

AHI NET Controler

The AHI 940 has a running link to the Project Genie 940 at Berkeley. Viewed from the AHI machine, the link is a special, home-built controller connected to two 2400 bit/s duplex data lines. The duplex feature is currently ignored. During the FJCC these lines will be used for the transmisson of all the information necessary to run an AHI console, except the video. This controller will be the one used for the IMP control as well as the GENIE link. It operates much like the other special-purpose hardware devices.

In the 940 resident, writable, core memory there is a user reference cell (URC) reserved for the NET controller. This cell contains the absolute core address, using the full core memory space and no fancy relabeling, of the NET command table.

The command table contains a contiguous list of double word entries. Each entry does the following:

Points to another absolute core address

Specifies a transfer size in words.

There is a single 940 instruction (EOM) associated with the NET controller. When the instruction is executed the controller goes to the URC, gets its address, and begins stepping through the list of transfers in the command table. The single interrupt which the NET controller can issue is used to signal END OF MESSAGE when the orders in the command

table have been completed.

Kinds of Control Needed

From many programming points of view, the IMP may be considered a peripheral input/output device. It sends and receives control information and blocks of data, much like a tape drive. The expected transmission of a huge number of single-character messages, however, requires special consideration. If an excessive "call and response" syndrome develops between the IMP and the 940, the time used for interrupt interpretation, message preparation and acknowledgment, and other kinds of overhead will severely restrict the service of the TSS. There are only three kinds of information flow between the IMP and and 940 Monitor.

The first is control information for the IMP itself. This may be timing informaton for the block "race problem," dynamic status of the 940 or the IMP, or table indices. This should not be a large volume of information and should not require particularly high transfer rates.

The user-host and host-user control information and single-character messages, although they may be only a few bits, possibly require the bulk of service from the 940 Monitor. They should, therefore, be sent and received with a minimum of interrupts and translation.

Large, bulk, bit transfers such as files and data bases make up the third kind of data transfer. The 940 Monitor is unconcerned with the bits themselves. User programs will receive the blocks directly from the IMP, and must do any

translation or sorting they see fit.

NET Users as Ordinary Users

At some of the meetings and in the memos Elmer has written there appears to be apprehension about the number and kind of users that will be serviced by a host at one time.

If each individual person at a remote console is treated as a standard user within the 940 TSS, nothing special is needed to service as many different users as the system will normally bear.

Making each NET user a regular 940 user, with an internal job number and fork structure, has all the added benefits of interjob protection, file storage, and minimization of programming effort.

At first, user names may be site names to avoid overflowing our job tables.

Using Teletype Lines

Interrupts and Character Transfer

The following hardware configuration for connections between the 940 and the IMP fits well into the above considerations.

The IMP would have eight Teletype lines connected directly to our general Teletype patch-panel. These lines will operate exactly like standard Teletypes as far as the 940 Monitor is concerned. The figure eight is determined by the expected Monitor size and the number of total Teletypes that the system will be assembled to be plugged into.

The Imp would go through the already existing 940 NET controller for its bulk transfers. The control and

synchronization of the transfers is discussed below. The bits transferred are ultimately controlled by the URC and command table.

Teletype Line Usage

The eight Teletype lines would be divided into two sets: one of seven devoted to user use, and the last line devoted exclusively to IMP-940 Monitor communication.

To the 940 Monitor, the seven user lines would appear exactly like standard Teletype lines. Characters coming over the lines would be packed in the Teletype buffers in the normal way. They would then be passed off to the Executive or the user program. This character stream could contain information about bulk transfers for a user program, but this is of no concern to the Monitor.

The last Teletype line is used exclusively by the Monitor for IMP control. It would have a fixed number, say 16. No user could ever log in on Teletype 16.

If a line were not devoted to control, characters would have to be taken out of the set used for user messages. This approach opens a Pandora's box which should stay closed. With the entire range of characters open to user host, the standard set will never be changed, programs will not be affected by changes in the Monitor-IMP relationship, and everyone will be happy.

This line is currently conceived of as serving two purposes.

Suppose that there are a number of users currently running under the TSS from the IMP. They may all be in

the process of receiving bulk transfers.

When a user program is ready to accept a packet of bits it calls on the monitor with a BRS. The monitor must first prepare the URC and command table, after making sure they are not in use.

It must then notify the IMP that it can send the message, but it must be sure that it gets the message for the right user. This information is sent over the reserved Teletype line.

When the transfer is complete the IMP, through the controller, interrupts the 940. The monitor notes the fact, and the user may be fired up again when he comes up in the queue.

All the information about the IMP, the NET, and the 940 is also sent on this line. This includes information about the loads, up status, and anything else not related to, requested by, or sent by users. Everything on a user line goes to a user, and nothing is gobbled up in a secret way by the Monitor, IMP, or anything else.

Initial Sign-On

When someone wants to log onto the 940 system, a request of some kind will eventually reach the local IMP. This request would probably be initiated on the user's machine and sent to the remote IMP via the control line.

When the request finally reaches the local IMP, it (the local IMP) must seek a Teletype line to connect the remote user with the local host. Once the line is found, it

remains constant and may not be changed or used by another user until LOGOUT.

After the line has been found, the IMP may send acknowledgement back to the user site. It must also send a rubout to the 940 on the newly established line.

Log-In

When the 940 gets a rubout from a Teletype line, whatever it is attached to it goes into its normal log-in procedure.

It first sends back a short line of text with the current system name, date and time, a few carriage returns and line feeds, and the word ENTER.

The user must then enter a user name, and he is off and running on the EXEC level in the TSS.

Rubout Conventions

The 940 Monitor has some very special conventions about rubout characters that come in on Teletype lines.

A user program that is allowed executivity can request rubout control if it wishes, or leave the control up to the monitor.

If it requests control, then rubouts are sent to the program just as any normal character would be.

If control is not requested, and a rubout comes in, the Monitor wipes out the lowest-level fork in the structure connected to the Teletype. This provides a way out of loops.

In any event, if two rubouts come through in less than half a second, the Monitor wipes out the lowest fork, no matter what the user requested. This is the ultimate breakout, and a necessary

feature in the TSS.

Somehow, this feature must be handled by the IMPs in the NET. Two different solutions are immediately obvious.

The IMP could keep track of the time between characters and not lose the timings during the transfer. This does not appear to be a good approach.

Alternatively, the IMPs could send double rubouts as special control information between themselves and decode this information to local conventions at the host sites. In our case it would be two fast rubouts; somewhere else it might be a special interrupt.

AHI User Program - NET Communication

Definition of a User Program

For our purposes a user program is a piece of code that runs in user memory, protected from others, from itself, and from executing I/O instructions.

User programs make their requests for I/O, as well as other system operations, through a special system call known as a BRS.

The Monitor in the 940 will do a minimum amount of work to run the NET. All it will do is buffer enough characters to bridge the gap between swaps and provide control for bulk transfers. Everything else is done by user programs.

NLS is a user program. Special NET work for NLS will be done by NLS.

Utility routines, such as file transfers, will also operate on the user level when they are written. They will be swapped like everything else, and should not put any more burden on the Monitor that other kinds of Executive request.

Teletype Input

A user program running from the IMP would not normally know that it originated on a device other than a Teletype. The commands to read and write characters will operate in exactly the same way.

BRS's Needed

The BRS's necessary to operate the IMP from a user program are similar to those necessary to read and write files. In fact a user program can think of the IMP as two files.

A BRS would be needed to open the IMP for reading, and another for writing. These would work like the file-opening BRS'S and return

an associated file number that would be used in all future data transfers.

In addition, there would be a standard block read/write BRS. One would put the file number, the starting location, and the number of words to transfer in the three registers and issue the BRS. The file number determines the read or write mode.

At this point the monitor would go through all the usual routines for block transfers. If the starting location at and the number of words match page boundaries, the page could be frozen and the transfer done directly into user memory. If things are at odd locations a buffer could be used, as it is for drum transfers.

The data would not be sorted by the IMP, and so each user program must do its own sorting. A "read" to the IMP means to read the next N words as they were received by the IMP.

It is expected that some control information about bulk transfers will be sent over the Teletype lines. For example, the Teletype lines would send a character stream that the user program would interpret as a signal that a block of a certain size was being sent. The user program would then issue the read to the Monitor. The Monitor would synchronize with the IMP and get the message. The user program would then be started up again. It is likely that for files, this first bulk message will be conrol information about a coming series of even larger bulk messages.

NET Standard Translators

Universal Hardware Language

To mininize the numer of translators needed to nap any facility's user codes to any other facility, there is a universal hardware language.

This language is simply a large, syntactically simple language in which statements can be made about any hardware device in the NET. The language may never be used in transmission, however. Its primary use is as a set of specifications for the coding of the NSTs at each user site.

Introduction to the NST

Suppose that a user at a remote site, say UCSB, is entered in the AHI system and wants to run NLS.

The first step is to enter NLS in the normal way. At that time the UCSB system will request a symbolic program from NLS.

This program is written in an algebraic, parsing-oriented hybrid language. It is called the NLS Encode Translation Program.

The program accepts input in the NET hardware language and translates it to a form usable by NLS.

It may pack characters in a buffer, and also do some local feedback.

When the program is received at UCSB it is compiled and loaded to be run in conjunction with a standard library.

All input from the UCSB console first goes to the NST. It is processed, parsed, blocked, translated, etc. When the NST receives a character appropriate to its state it may finally

initiate transfers to the 940. The bits transferred are in a form acceptable to the 940, and maybe in a standard form so that the NLS need not differentiate between UCSB and other NET users. Information sent to UCSB from NLS must go through a similar process.

NLS will request a symbolic program from UCSB. This program maps standard NET hardware language into packs specifically made up for the UCSB machine. It is known as the NLS Decode Translation Program.

This program is compiled and run along with NLS. It operates out of user memory just as NLS does, and does not affect the Monitor.

Function of NST

Advantages of Dual Translation

After each node has implemented the library part of the NST, it need only write one program for each subsystem, namely the symbolic file it sends to each user that maps the NET hardware language into its own special language. Each user must also write a decode translation program for the new subsystem. The decode program should normally be rather small compared to the rest of the worries of using a remote system.

This is the minimum programming that can be expected if each console is used to its fullest extent.

Since the NST which runs the encode translation is coded at the user site, it can take advantage of hardware at its consoles to the fullest extent. It can also add or remove hardware features without requiring new or different

translation tables from the host.

Local users are also kept up to date on any changes in the system offered at the host site. As new features are added, the host programmers change the symbolic encode program. When this new program is compiled and used at the user site, the new features are automatically included.

The advantages of having the encode translation programs transferred symbolically should be obvious. Each site can translate any way it sees fit. Thus machine code for each site can be produced to fit that site; faster run times and greater code density will be the result.

It is expected that when there is matching hardware, the symbolic programs will take this into account and avoid any unnecessary computing.

(nst) The NST library is the set of programs necessary to mesh efficiently with the code compiled at the user sites from the encode translation it receives.

Universal Hunk of Bits

The Universal Station

AHI NLS - User Console Communication

Block Diagram

The right side of the picture represents functions done at the user's main computer; the left side represents those done at the host computer.

Each label in the picture corresponds to a statement with the same name.

There are four trails associated with this picture. The first links (in a forward direction) the labels which are concerned only with network information. The second links the total information flow (again in a forward direction). The last two are equivalent to the first two but in a backward direction. They may be set with pointers t1 through t4 respectively.

[">tif"] OR [">nif"]; ["<tif"] OR ["<nif"];

User-to-Host Transmission

(keyboard) is the set of input devices at the user's console. Input bits from stations, after drifting through levels of monitor and interrupt handlers, eventually come to the encode translator. [>nif(encode)]

(encode) translator maps the semi-raw input bits into an input stream in a form suited to the host subsystem which will process the input. [>nif(hrt)<nif(keyboard)]

The Encode program was supplied by the host subsystem when the subsystem was first requested. It is sent to the user machine in symbolic form and is compiled at the user machine into code

particularly suited to that machine.

It may pack to break characters, map multiple characters to single characters and vice versa, do character translation, and give immediate feedback to the user.

(ldm) Immediate feedback from the encode translator first goes to local display management, where it is mapped from the NET standard to the local display hardware.

A wide range of echo output may come from the encode translator. Simple character echoes would be a minimum, while command and machine-state feedback will be common.

It is reasonable to expect control and feedback functions not even done at the host user stations to be done in local display control. For example, people with high-speed displays may want to selectively clear curves on a Culler display, a function which is impossible on a storage tube.

Output from the encode translator for the host goes to the invisible IMP, is broken into appropriate sizes and labeled by the encode translator, and then goes to the NET-to-host translator.

Output from the user may be more than on-line input. It may be larger items such as computer-generated data, or files generated and used exclusively at the host site but stored at the user site.

Information of this kind may avoid translation, if it is already in host format, or it may undergo yet another kind of translation if it is a block of data.

(hrt) It finally gets to the host, and must then go through the host reception translator. This maps and reorders the standard

transmission-style packets of bits sent by various Encode programs into codes acceptable by the host subsystem in use. This part may not even exist if things work out.[>tif(net mode)<nif(encode)]

Host-to-User Transmission

(decode) Output from the host initially goes through decode, a translation map similar to, and perhaps more complicated than, the encode map.[>nif(urt)>tif(imp ctrl)<tif(net mode)]

This map at least formats display output into a simplified logical-entity output stream, of which meaningful pieces may be dealt with in various ways at the user site.

The Decode program was sent to the host machine at the same time that the Encode program was sent to the user machine. The program is initially in symbolic form and is compiled for efficient running at the host machine.

Lines of characters should be logically identified so that different line widths can be handled at the user site.

Some form of logical line identification must also me made. For example, if a straight line is to be drawn across the display this fact should be transmitted, rather than a series of 500 short vectors.

As things firm up, more and more complicated structural display information (in the manner of LEAP) should be sent and accomodated at user sites so that the responsibility for real-time display manipulation may shift closer to the user.

(imp ctrl) The host may also want to send control information to IMPs. Formatting of this information is done by the host

decoder. [>tif(urt) <tif(decode)]

The other control information supplied by the host decoder is message break up and identification so that proper assembly and sorting can be done at the user site.

From the host decoder, information goes to the invisible IMP, and directly to the NET-to-user translator. The only operation done on the messages is that they may be shuffled.

(urt) The user reception translator accepts messages from the user-site IMP and fixes them up for user-site display. [>nif(d ctrl)>tif(prgm ctrl)<tif(imp ctrl)<nif(decode)]

The minimal action is a reordering of the message pieces.

(d ctrl) For display output, however, more needs be done. The NET logical display information must be put in the format of the user site. Dispay control does this job. Since it coordinates between (encode) and (decode) it is able to offer features of display management local to the user site.[>nif(display)<nif(urt)]

(prgm ctrl) Another action may be the selective translation and routing of information to particular user-site subsystems. [>tif(d ctrl)<tif(urt)]

For example, blocks of floating-point information may be converted to user-style words and sent, in block form, to a subsystem for processing or storage.

The styles and translation of this information may well be a compact binary format suitable for quick translation, rather than a print-image-oriented format.

(display) is the output to the user. [<nif(d ctrl)]

User-to-Host Indirect Transmission

(net mode) This is the mode where a remote user can link to a node indirectly through another node.[>tif(decode)<tif(hrt)]

SRI Usage of Other Stations

UCSB

UCLA

Utah

Implementation Procedure

NST

Universal Hardware Language

The first step is the specification of the universal hardware language. This should not be too difficult. The language can be syntactically straightforward; probably a left-liner grammar will suffice. The only worry is providing for expansion in the language so that any new device may be easily accomodated.

Work should begin as soon as possible. The man-hours required should not be excessive, say a week from each site. However, the elapsed time could easily grow because of the communication problems.

Specifications and Compiler for Encode/Decode Language

The only way to design the Encode/Decode Language is to try and write an Encode program. Maybe DIA and JFR shoul try this for NLS. After an initial guess is made at the syntax, it could be written up other sites could undertake a similar task. After everyone has tried once, we could all get together and work out the details.

NST Library

The compiler and the library must be worked out at the same time, but after the syntax has settled down a bit.

TSS

As specified above, changes to TSS are not very great. It may be wise to wait until more is known about the IMPs before much serious work is done on TSS.

NLS

Fix Up Display

Encode Package

High-Speed Tubes

Storage Tubes

AHI DATA STRUCTURE

A.M. 276 CLASS NOTES

1B Compliments of the

HYPERTEXT EDITING SYSTEM

1B1 CENTER FOR

COMPUTER & INFORMATION SCIENCES

1B2 BROWN UNIVERSITY

1B3 PROVIDENCE, RHODE ISLAND

1B4 30 March, 1969

```
:MMM. 08/16/69  0952:25  JFR  ;  .PLO=1; .RTJ=0; .PGN=0; .ROM=1; .DSN=1;
.LSP=0; .NED="MULTI-MINI-MACHINE"; .DPR=0;
```

ABSTRACT

This working document describes a method of extending the AHI-NLS facility on the SDS 940 from 12 to 32 consoles.

FOREWORD

Purpose

This memo discusses the usefulness of small computers as special-purpose hardware device controllers and as complex, programmed feedback mechanisms.

Timings and simulations indicate that the current level of SDS 940 response and display-oriented feedback can be maintained, and maybe improved, while the number of consoles is tripled.

The centralization of the display buffers and the use of small computers as display controllers allows experimentation with ring-structuring techniques for inter-console collaboration.

Similarly, the use of small computers as input device controllers contributes to collaboration by providing an elegant means for linking an arbitrary collection of input devices to any console.

Finally, the small machines can quickly react to most of the single-character NLS interactions, reducing the frequency of 940 computation per user and thus reducing swapping.

History

This memo is the result of discussions held in late December 1968 and early January 1969.

The main contributors have been Roger Bates in hardware and Don Andrews, Bill Paxton, and Jeff Rulifson in software.

MULTI-MINI-MACHINE

GOALS

The initial study of 940-NLS operating characteristics centered around the goal of operating 12 work stations with response equivalent to the 6 currently operating.

Bottlenecks in the 940 system are connected with core space and swapping.

Currently, displays are stored in 940 core. Each user has his own buffers, and a new 940 core page is frozen as each user enters the system. For 5 or 6 users this is not too bad, but for more than 7 or 8 it is totally unreasonable. see(LOADS,Conclusions)

The immediate feedback that NLS gives to each user is significantly more complicated than that which can be offered through conventional echo tables.

Consequently, each NLS user must be immediately activated for each character or button push.

Statistics on NLS usage indicate that if the display buffers are removed from 940 memory, approximately 10 to 12 work stations can be serviced before the swapping reaches a critical point, and response significantly declines. see(LOADS,Conclusions)

The most direct solution for the 12-station system is simply a bulk core for display buffers. This approach has already been discussed in a current proposal to RADC for system expansion.

Our statistical observations indicate that the 940 could serve approximately 32 NLS users, if users interacted with the 940 on a work-request basis instead of a single-character basis. see (LOADS,Usage)

When more than 12 work stations are considered, the single-character interaction with the time-sharing system and user programs completely overloads the system.

The first criterion for any expansion is that it be extendable to the limits of the 940.

The second is that the system closely resemble more that may be expandable to even larger service, say 100 consoles.

If such expansion is flexible and modular in its design and hardware coupling, then alternative programming, scheduling, and queueing techniques may be tried. The problems of much larger systems may be incisively studied and the solutions tested through implementation.

MULTI-MINI-MACHINE

It has become apparent that collaboration, in its many forms, must be implemented in NLS.

Three possible modes are:

Display-frame sharing, where different logical parts of a user's current display picture may be viewed by a set of users

Input-device flexibility, where many people may sit around a table, each with his own handset and mouse; or users remote from each other can drive one another's displays

Audio communication, which takes two forms:

- Complete voice switching, so that any number of "conference calls" could be established between the work-station users

- Computer-generated sound, used as feedback to a single user or a group of users in collaboration.

Many of these features can eventually be implemented totally entirely within the framework of the 940. However, the current hardware/software approach imposes severe limitations in two ways.

The MONITOR in the 940 time-sharing system is cramped for space. This means that the extensive tables necessary for input-device interchangeability will not fit, and limited, costly schemes would have to be implemented.

The addition of hardware devices is governed by the availability of special-purpose controller and polling devices.

- Each of these operates a fixed number of each kind of cevice.

- We would prefer a scheme that allows new devices to be added and interchanged freely and in small increments.

Other Devices

What about the printer?

We might want to move our printer to a small machine.

What about micro-film output?

We might want to drive a microfilm machine from a small machine.

What about Model 37 TTYs?

What about the NET?

The idea is that the big core might serve as a buffer for

messages from the IMP.

However, we will wait until more is known about the IMP before we do any planning.

What about the NIC?

The idea here is that small machine could handle many NIC users if they ran a dedicated information-retrieval/editing system. The small machines could be a single user on the 940, again reducing the swapping effort while maintaining high-speed response.

What about storage tubes?

What about an on-line Dura?

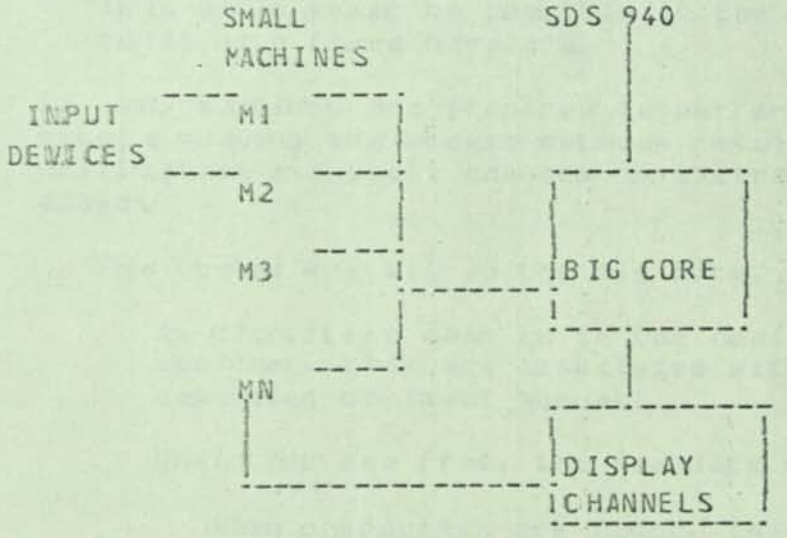
THE MULTI-MINI-MACHINE APPROACH

General Idea

There is a big core of 16-bit words. Most of it is used to store the ring of all the display buffers. Some is used to store queues of tasks, tables for associating users and input devices, and reentrant interpretive code.

Connected to the core is a bank of small machines (4K 16-bit words, 4 usec instruction times). They are used as input-device controllers, interpretive code executers for complex feedback, and channel drivers for the display channels.

Gross View



The small computers more than meet the minimal time requirements.

Each small machine has its own small core memory. If it is operating on the ring structure and determining feedback actions, it contains an interpreter for reentrant code which resides in the big core.

This code is close to the code which is compiled from the Special-Purpose Languages (SPLs) for the current NLS.

The code is not algorithmically complicated. Each input character involves only a few logical decision and table changes.

The decisons, however, may involve reference to the user's ring of display buffers, and the few hundred words necessary to define his current status.

It is this large amount of information that causes the swapping overload in the 940.

By having all of this information immediately accessible (in core rather than on drum), a small slow machine is more than adequate.

Many small machines were chosen over fewer faster ones because they could be manipulated and added in small increments.

Input devices, along with the controlling processors, can be added a few at a time as different experiments arise.

Since the logic for the feedback is in the big core, it may be expanded to two or even four time the orginal estimate. Thus the small machines will not be cramped for space.

The structure of the display rings can be modified as information is gained on the nature of display collaboration. This would never be possible if the display controller were built into fixed hardware.

If many machines are prepared to perform the same task, some simple queuing and access methods result in better machine utilization and small changes in system performarce as users are added.

The queues are all in the big core. There are two basic kinds:

As characters come in to the dedicated input-controller machines, they are associated with logical users and deposited on input queues.

When they are free, the feedback machines poll these queues.

When characters are found, they are processed.

Sometimes the 940 must be requested to intercede and complete a task. When this is so, the logical user is put in a state of limbo as far as the feedback processors are concerneds and the task is queued for the 940.

A simple implementation of the "Dijkstra Flag" keeps two processors from simultaneously modifying a data structure.

All the small machines share a single access line to the big core. Thus only one can be doing a read or write on a single cycle.

There are, however, three modes of memory access: read, write, and Dijkstra read. In the latter a word is read from memory and sent to the small machine. During the write-back in the big core, however, the top bit is cleared.

Suppose that a word in the big core is a pointer to a queue and that if the top bit is on, the queue is free; otherwise a processor is working on the quque.

If a processor Dijkstra-reads the pointer and the queue is free, it then has control of the queue as well as the pointer to it.

If the queue is in use, however, the processor is informed of this.

The 940 must have two kinds of control over the small machines.

There is a two-way interrupt line.

This permits the small machine to notify the 940 that there is something on a queue.

It also permits the 940 to interrupt the small machine and request, from a small resident program, sufficient information to run a real-time debugging package.

It is also necessary for the 940 to exercise complete control at certain times (e.g. system startup or a looping small machine). To do this the 940 must be able to operate the console switches, and read most of the consdle lignts.

There are two alternative methods of driving the displays on the system.

A small machine could drive the display directly. This would require the machine to output a word to the display every 15 usec. Only the fastest (and most expensive) of the small computers are capable of doing this. Even then they barely make it, and cutting it this close seems inadvisable.

The other approach is to have a data channel which can be loaded and activated by the small machine. In this way the small machine still interprets the ring structure and thus maintains the flexibility there. Using channels, the small machine can be relatively slow and still drive the displays at full capacity.

Two auxilary ideas are necessary to make this clear.

A channel normally contains an address register and a word count. These channels also contain an address-register buffer and a word-count buffer.

As the channel is driving the display from the working registers, the small machine can be loacing the buffer registers.

When the channel is done, it issues an interrupt to the small machine. As soon as the machine has figured out

where the interrupt came from, it can pulse the channel. The channel can then transfer the buffer registers to the working register and begin driving the displays again.

This cuts the time between buffers from 60 usec to under 20 usec.

The small machine can still have problems keeping up with the display channel if the buffers are displayed faster than the ring can be searched in the big core.

To overcome this, the small machine only searches a ring when it is changed. As it searches a simple list of necessary buffers in made it the small machine local memory. Thus the next channel address and word count are readily available within the immediate addressing structure of the small machine.

These machines can keep close watch over their time allocation. Single overloading of a display causes flicker only on that console; an extensive ring structure affects only the consoles displaying from it; and in general, bad side effects which now propagate from console to console are confined.

Input Devices

Special Input Devices

Each device will be read as a source of 16-bit code. However, the full 16 bits will not be used. Each device will look the same to the input machine; its existence and type will be defined through tables which can be set up and modified by the 940.

The devices planned for are:

- A keyboard which produces an 8-bit code
- Up to 16 pushbuttons, although we will start with only 8
- The mouse, which requires two 10-bit numbers as coordinates, and will thus take two words (one coordinate in each).

Each group of 24 input devices will have a special polling mechanism which works in the following way.

- The polling may be started and stopped by the small machine.
- When the polling is in operation, it operates continuously at its own rate. This will probably be set so that each device is sampled at least once every 5 to 15 ms.
- As each device is sampled, its contents are read into a core

location in the small machine.

If the device is a keyboard, however, the core location is changed only if the strobe is on.

The input machine may then inspect the list in core at a slower rate, say every 30 ms, and notice any changes.

The following are the maximum input rates for each device. While it is unlikely that all devices will operate at maximum rate for an extended period of time (e.g. one second), we must plan for at least half-second periods of maximum transfer rates.

When a mouse is in motion, it is usually sufficient to update the tracking display buffer every 30 ms. The coordinates must be observed at this rate, but they can be compared in the input machine, and if the mouse has not changed position the coordinates are not associated with a logical user nor are the display buffers changed.

The maximum rate for a keyboard is slightly under 100 ms per character. The keyboards are polled as fast as the mouse. However, the character in the core of the small machine is changed only when the strobe is on. Thus the small machine has at least 50 ms. to notice and process any character. Every character must be associated with a logical user and deposited on the appropriate queue in the big core, along with the time and the mouse position.

The pushbuttons are polled just like the mouse. Every 30 ms, the contents must be observed by the input machine. When any change is noted, extra action is required.

If the buttons are on the mouse, the up or down action must be associated with a user and deposited on a queue in big core along with the time and the mouse position.

If the buttons are on the handset, changes are ORed together until all switches return to the UP status. The chord is then associated with the user and put on the queue along with the time and the mouse position.

Output Devices

There will be only two high-speed output devices.

Each display channel is loaded from a small machine and transfers words from the big core to 6 displays. Each channel has a 20-bit address register, a 16-bit word count, and an interrupt line back to the small machine. The interrupt is raised when the word count goes to zero.

The audio switching system is viewed as a big switching matrix from the small machine.

- The inputs are all the voice lines and all the tone generators. Each tone is fed in as three separate lines of amplitudes 1, 2 and 4.

- The outputs from the matrix go directly to the speakers or earphones.

- The audio system is used at a ms rate when the tones are in a decay state (making them much more pleasant to listen to).

The following are the maximum rates for the two devices. These rates may be common, and plenty of leeway must be allowed.

When the display channels are running they do not effect the small machines. Thus the times which concern the small machines are the length of time it takes to load a channel and the average length of time to process a buffer once the channel has been started.

- We expect the displays to write a character in 7 usec, so the channels will transfer a word from big core to the displays every 14 usec.

- Buffers can be a minimum of 1 word long, and thus be processed in 14 usec. Normally, however, they are no shorter than 4 words, which takes 56 usec.

- To keep the displays running wide open, the small machines should make every effort to minimize the time to load a buffer. Using the scheme discussed earlier, this time can be cut to less than 20 usec.

- If the displays are storage tubes, the buffers will be the character strokes, and these will be stored in the big core.

  - In this case the small machine will have to look at the buffer generated by the 940, take it apart character by character, and drive the display with the individual character buffers.

  - Since the storage tubes take about 100 usec to write a full character, this will not overload the small machine, although it will occupy almost all of its time.

Voice switching may be a slow process. Tone decay, however, must be rapid.

Extra I/O Devices

Local Dura?

Storage Tubes?

Potter Printer?

MULTI-MINI-MACHINE

Model 37 TTYs?

Microfilm Output??

Steps for Total Input/Feedback Service

The following is an outline of the steps an input machine will go through as it gets characters from the input devices and puts them into the appropriate queues in the big core. This outline was used to estimate the total processing capacity necessary in the input machines.

Check queue for transfers to big core.

Get character from internal queue.

Make correspondence between hardware device and logical user.

Get queue control for logical user.

Put character on logical user in big core.

Make up appropriate, internal audio queue from logical user information in big core.

Give up queue control for logical user.

Get clock interrupt.

Go down input character buffer.

If same as last interrupt do nothing.

Otherwise

If keyboard, note time and mouse position, put on queue to go to big core.

If pushbuttons, decide which:

If mouse, note time and position, put on queue to go to big core.

If handset, then

If all are up, this is a character end so note time and position and despoit on queue for big core.

Otherwise, build up chord by ORing in new down buttons.

Audio Queues

Do appropriate amplitude switching for computer-generated sound decay, remove processed entries from queue.

Date/Time Queues

Some of the small machine will devote almost all of their time to searching the ring structures and driving the display channels.

Get display channel interrupt.

Use local list with update when ring changes, as discussed above.

Have all the entries in local memory, just put it up.

Changes are flagged, and require a block transfer of new pointers.

The rest of the small machines will be devoting all of their time to processing the input queues prepared by the input machines, updating display buffers, and preparing queues for the 940. The following outline of their duties was used to estimate the processing capacity necessary for the job.

Check for work to do.

Locate non-empty logical-user task queue.

Get IFM control for logical user.

Input character for processing.

Get user state.

Begin execution of interpretive code.

Get character from queue.

Get input-queue control for logical user.

Get character off queue.

Give up input-queue control.

Step through main control.

Update appropriate display buffers.

How is this done if we have long buffers? Is there a display Dijkstra flag?

There is also the problem of writing in the mouse coordinates while they are being changed.

```
        Continue until either
            Queue is empty:
                Put him to sleep.
            Or 940 is needed:
                Put him on 940 queue.
                Set flag for 940 interrupt.
    Update date and time
    Give up IFM control for logical user
    Interrupt the 940 if necessary
```

PROPOSED HARDWARE

Data Flow

Interrupt Lines

(pgb) Put-Get Box

(adr) Address register, 20 bits

(tr) Transfer register, 16 bits

(fc) Function code

Read

Write

Dijkstra test

Automatic increment

(ch) Channel

(cadr b) Channel address-register buffer, 20 bits

(cwcb) Channel word-count buffer, 16 bits

(cadr) Address register, 20 bits

(cwc) Word count, 16 bits

(cc) Core controller

Two-way channel

(cs) Console switches using the NOVA as an example

Two 16 bit NOVA readout registers--data and PC

Sixteen bits of input switches

Readout of extra minor-cycle console lights?

Single carry bit

Eighteen functions to select

PROGRAMMING TECHNIQUE

The 940 plays an important role in both the programming and the running of the small machines.

From a programming point of view, everything can be done on the 940.

The interpreter for the small machine can be coupled with good DDT to give debugging aids far beyond those that could be provided by the small machine itself.

Not only can checkout begin before the hardware is ready, but programs can be changed and debugged while the hardware is in use.

Moreover, the file system of the 940 is available to the 940 interpreter and DDT, so no new file system has to be created.

The 940 also has the ability to operate the console switches of each small machine. This will be used in at least three ways:

During initial startup the 940 can bootstrap-load the machines.

While the machines are running, they can be debugged from the 940. At any time the small machine can be stopped, examined, and restarted on a usec basis. The 940 can then take the information and, using symbol tables from assemblies and compilations, provide a monitor and debugging service.

Accurate statistics may be gathered about the operation of the machine. With control of the consoles, the 940 could even gather samplings of the instruction-counter contents, something which is normally quite difficult.

All the programming for the small machine will be done in a Machine Oriented Language similiar to the one in use on the 940 (MOL940). The compiler will be written in Tree Meta, and should present no special dificultiies.

The interpreter and DDT for the small machines will be written in MOL940, and operate on the 940. Since the DDT is executing through an interpreter, many fancy features can be added, such as operand fetch breakpoints, memory reference breakpoints, automatic timing statistics, etc. There will be two modes of operation for the DDT.

One is the normal mode of executing the code with the interpreter.

The other is the SYSdebug mode, where a live, running machine is debugged. In this case many of the fancy features may not work (like memory reference breakpoint while in run mode).

OTHER APPROACHES

Only big core

- single character interaction problem

Single large external machine

- sufficiently large core comes only with big cpu's, it fact too big.
- no fail safe features

fewer faster mini-machines

- growth comes in too large increments
- halfing the speed does not seem to double the capacity

All machines using single core

- poor addressing structure of small machines for huge memory
  - considered memory map it tradition of ATLAS--too many core cycles
  - simple relabeling cause trouble in display rings
- requires too many memory ports to keep small machine running efficiently

A.M. 276 CLASS NOTES

1. DATA STRUCTURE

The AHI data structure has as its basic unit the "statement." The statement is the smallest textual unit defined, and is simply a textual string. The file (i.e., collection of statements) is hierarchically oriented in a tree structure, each statement being a node in the tree. The reasons for this hierarchical structure will be discussed later. The file, however, can be viewed in other ways different from the sequential tree structure. For instance, associational trails can be drawn throughout the file and followed. Thus the AHI file is capable of modeling Bush's[1] network of associational trails as well as a sequential hierarchical text.

1.1 THREE TYPES OF BASIC ENTITIES

A. Statements

B. Vectors

C. Keywords

These three types of entities are stored in statement data blocks (SDB's), vector blocks, and keyword blocks, respectively. In addition, the hierarchical structure of the text is stored in ring blocks. We will only discuss the statement data blocks and ring blocks and their relation to the main file; the access and storage of vectors and keywords is very similar and so do not need to be discussed separately. (In the present version of the system, the only types of vectors that can be stored are straight lines, and no sketching facility exists other than defining the straight line by its endpoints. There is no rubberbanding. A sketching facility is planned for a future version.)

---

1. V. Bush, "As We May Think", Atlantic Monthly, July 45.

1.2 STATUS TABLES

all status tables are in the file header (block 0)

Associated with each set of blocks (there are four sets of blocks: the ring blocks, the statement data blocks, the vector blocks, and the keyword blocks) there is a small status table which has an entry for each block of its kind. Thus there is a ring status table, a statement status table, a vector status table, and a keyword status table. The entry for each block in the status table simply points to a "global" random file status table block, which gives the location of each block, whether in core or on drum. (See Fig. 1)

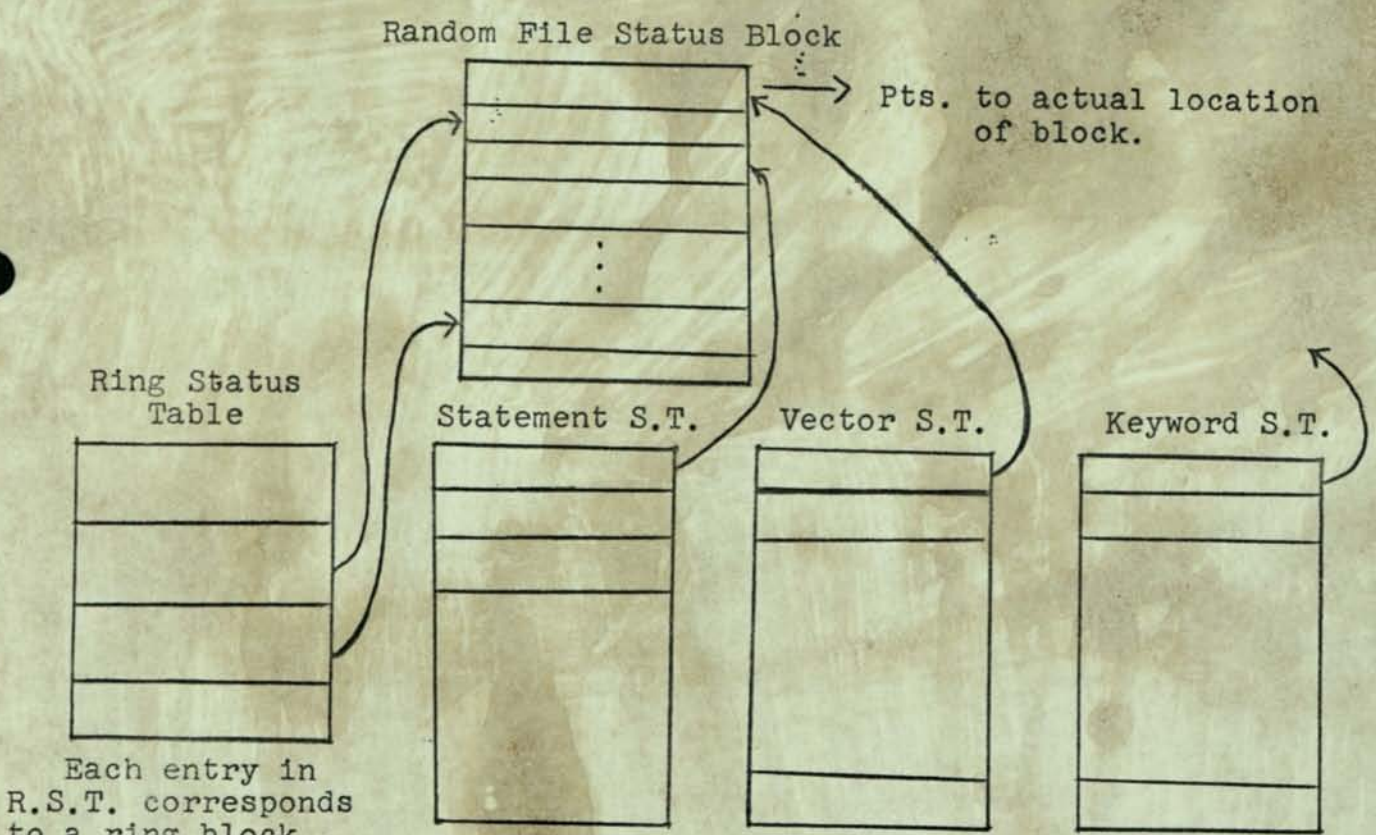


FIG. 1 Status Tables

1.3 RANDOM FILE STATUS TABLE BLOCK

The random file status table block is a block that contains an entry for every block of every type in the system (actually, there is an RFSTB for each (active) file). Each entry tells whether the block is in core or not, or whether it is unallocated (i.e., not being used at the present time and can be allocated when a file is expanded through editing). The entry also gives the information on where the block is located, on the drum or in core. It is through the RFSTB that each desired block is located by the system:

As we saw in 1.2, each ring block is mapped into an entry in the ring status table, each statement data block is mapped into an entry in the statement status table, etc. Then each entry in each status table points to an entry in the bigger RFSTB (at present there are a maximum of 64 blocks in the RFSTB), one containing pointers to the actual location of each different block in the file. This double-table method of location of each block is to facilitate control of the allocated and unallocated area on drum, and for garbage collection; furthermore, this central location mechanism allows blocks to be moved in the system, without internal pointers having to be modified.

The RFSTB contains information other than just a pointer to the block, whether in core or on drum. One area of the RFSTB, if it is less than zero, indicates the block is on the drum. If the number in this area is negative, it is the number of free words in that block. This is to prevent needless retrieval of the block for additions if there is not enough room on it for the desired update. If this area is greater than zero, the block is in core and the number is the core address of the block. There is another area in which an indicator (at present a-2) says free space is too small to consider going there. This is computed from the average lengths of the statements.

The ring status table at present has four entries, one for each of four ring blocks. This number is expandable, and is an assembly time variable. There are probably twice or three times as many SDB's and therefore the statement status table is correspondingly bigger.

~55 presently

1.4 STRUCTURE OF THE RING BLOCK

Each statement is represented in the data structure both by its associated text (see SDB's) and by a ring element, that is,

note: pointers to ring elements are called PSID's (permanent statement ID) + never changes during the existance of a statement.

by an element of a ring that contains the hierarchical tree structure of the file and points to the text associated with each node (statement) in the tree. The ring is broken into ring blocks, each of which is 1024 words long. Each ring block has a header and then is composed of ring elements, each four words long, one ring element per statement (See Fig. 2):

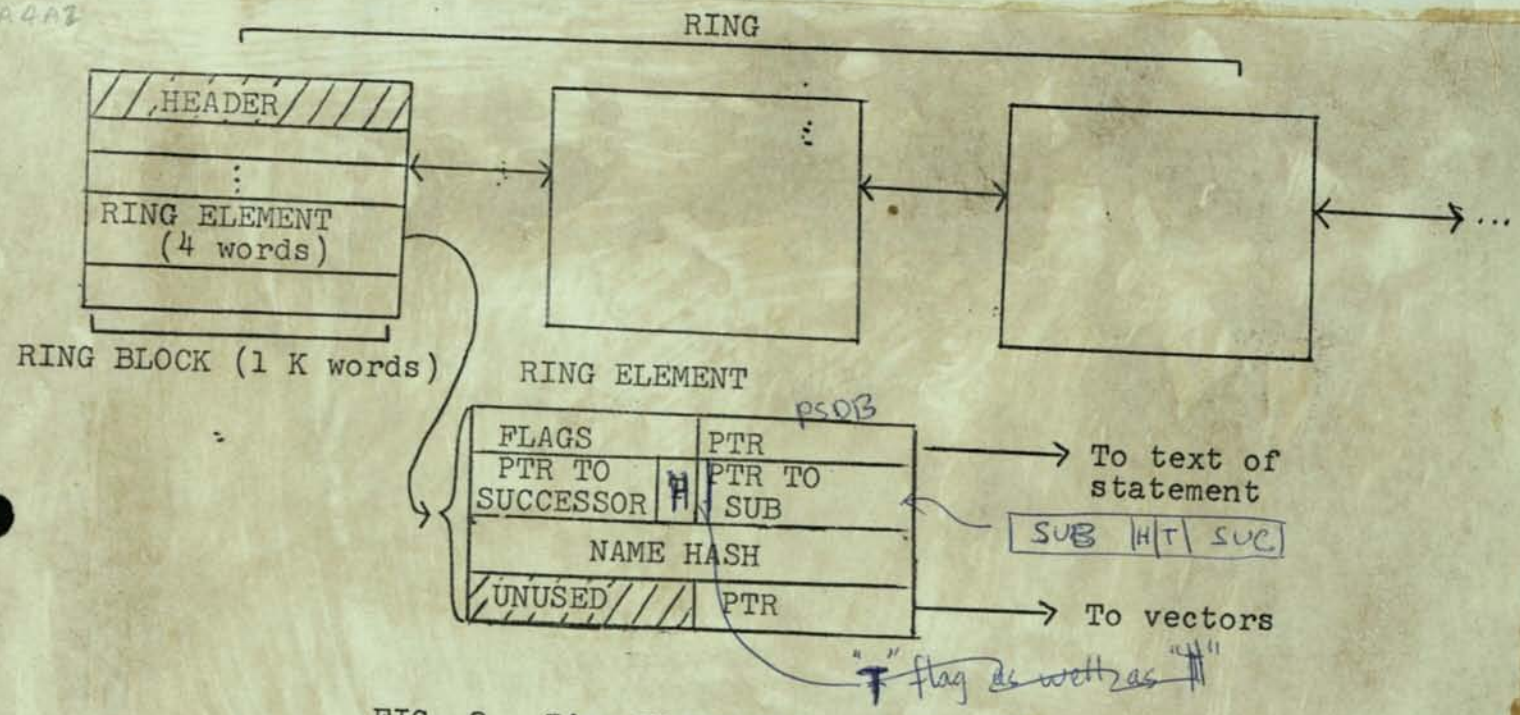


FIG. 2 Ring Block Elements

(PSDB)

Pointer (PTR): the internal pointer to the statement text in a statement data block (SDB). (The structure of an internal pointer (symbolized by 'P') will be discussed in Section 1.6.)

Flags: tell the level (in hierarchy), etc., and thus help with filtering so as to minimize drum I/O, i.e., if we want only to look at file above the third level of hierarchy and this statement is in the fourth level, there is no need to retrieve the text of this statement. (this would require updating the entire ring after every structure change).

Successor: a pointer to the ring element (anywhere in the ring) of the next succeeding statement on the same level.

Sub: a pointer to the ring element of the first statement in the level directly below the current statement.

The sub and successor pointers define the hierarchical tree structure.

SUB points to this ring element if there is no substructure

Flags 1) On if the statement has a name
2) pattern filter tested?
3) pattern filter result?

2A4A2E H: or if this statement is a head

2A4A2F T: the last successor on each level points to the ~~head~~ source (up 1 level) of that level, thus providing a back pointer. The T bit is set when it is the last successor.

2A4A2G Name hash: this is a 24-bit hash of the (optional) statement name. Thus when jumping by name, we need only scan each ring element sequentialy to get correct statement. This may be done sequentially since each file generally consists of only about 300 statements.

2A4A2G1 note: unused ring elements ~~on list~~ in a given ring block are linked together on a free list — pointed to by a pointer in the block header.

2A5 ## 1.5 STATEMENT DATA BLOCK (SDB)

2A5A The statement data blocks (SDB's) are simply areas in which to store the statement text. No structure or hierarchy is part of the SDB's since that is taken care of by the ring blocks. The system tries to put all sequential statements in the same block to save on drum I/O. The process of initial generation and placement of statements in the SDB's will be discussed in section 2. (See Fig. 3)

2A5B

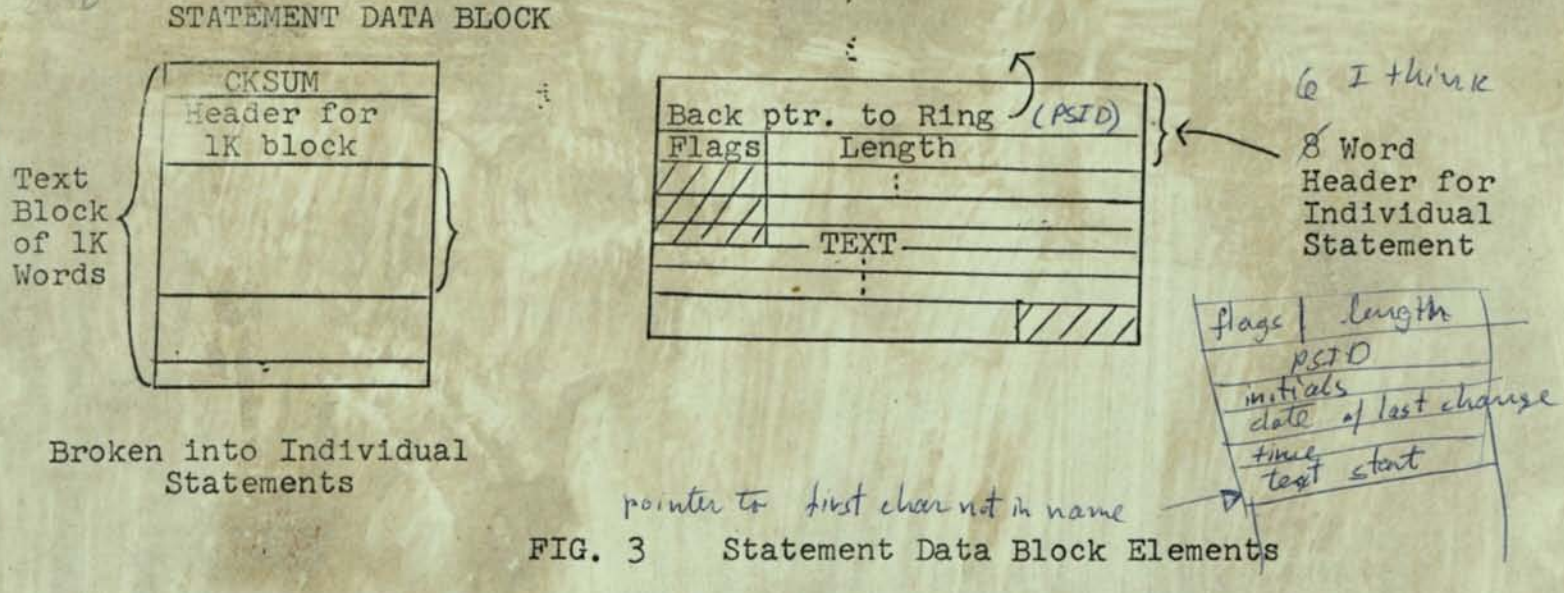


FIG. 3 Statement Data Block Elements

2A5B1 CKSUM: this is a checksum to check against hardware I/O errors in reading the statement data block from drum. Before writing out on drum, the system adds up all words in the SDB and stores the sum in CKSUM. On read-in, it re-adds the words and checks to see if the sum is the same.

All 1K file blocks are checksummed in this way

ZA5B2 Header: this contains the initials, date, and time of last user and change, fields which can be used as a later means of retrieval.

ZA5B3 Back pointer to ring: this is an internal pointer (of type 'P') to the ring element representing this statement in the text hierarchy, i.e., the ring element which points to this statement.

ZA5B4 Flags: the first bit indicates whether this SDB element is garbage, and is used when compacting the SDB. Other bits indicate whether it is difficult to format the statement on the display, that is, if the statement contains things like underlining or flicker. If it is difficult to format, the low speed scanning/formatting routine is used. Otherwise the high-speed routine is used. This saves up to 50% on time. (Other bits for other things.)

ZA5B5 Text: the text is stored in the statement data block as follows: there are two kinds of characters, (1) 8 bit character, with the high order bit off, and (2) 16 bit characters. If the high order bit is on, this signals that the character is a 16 bit character. The seven next high bits signify font, etc., of the character represented in the second eight bits. The different qualities of each character are underline, blinking, italics, boldface, etc. The user can make up his own special characters and the system will insert it. This is done by giving the special character a number. It takes less than 10 msec. to reformat a display. (I/O not included)

ZA6 1.6 LOCATING STATEMENTS IN THE DATA STRUCTURE

ZA6A 1.6.1 Mapping Statements to Ring Block Elements Through the Internal Name (Pointer) P

ZA6A1 When statements are created, they are assigned by the freelist allocator to open positions in a ring block (to a ring element) and assigned to statement data blocks according to the "garbage bits"; they are also assigned an internal (position related) name in the ring block denoted by 'P'. All ring block vacancies are kept on the freelist. The internal name P in the ring block is thus assigned by getting it off the freelist (creating a map from statement name position to internal name, and from internal name to block position) as described below:

Say that we want to retrieve statement P, an internal name (pointer) of the type found in the successor and sub fields of the ring element. It is listed in the file header where it gives the point in the ring where the file starts, i.e., it points to the ring element representing the first statement in the file.

To get statement P (10 bits) we look at P*4, which is 12 bits long (See Fig. 4). The upper 2 bits are an index on the 4-entry ring status table (RST). The entry in the RST points to an entry in the random file status table block (RFSTB). This entry in the RFSTB tells us whether the ring block containg the desired ring element is in core or not, or whether it is unallocated (in which case an error condition exists). The ring block is brought into core if necessary. The lower 10 bits of P*4 then form an index relative to the start of the ring block that bring us to the appropriate ring element. Thus from the internal name of the statement we retrieve the desired ring element.

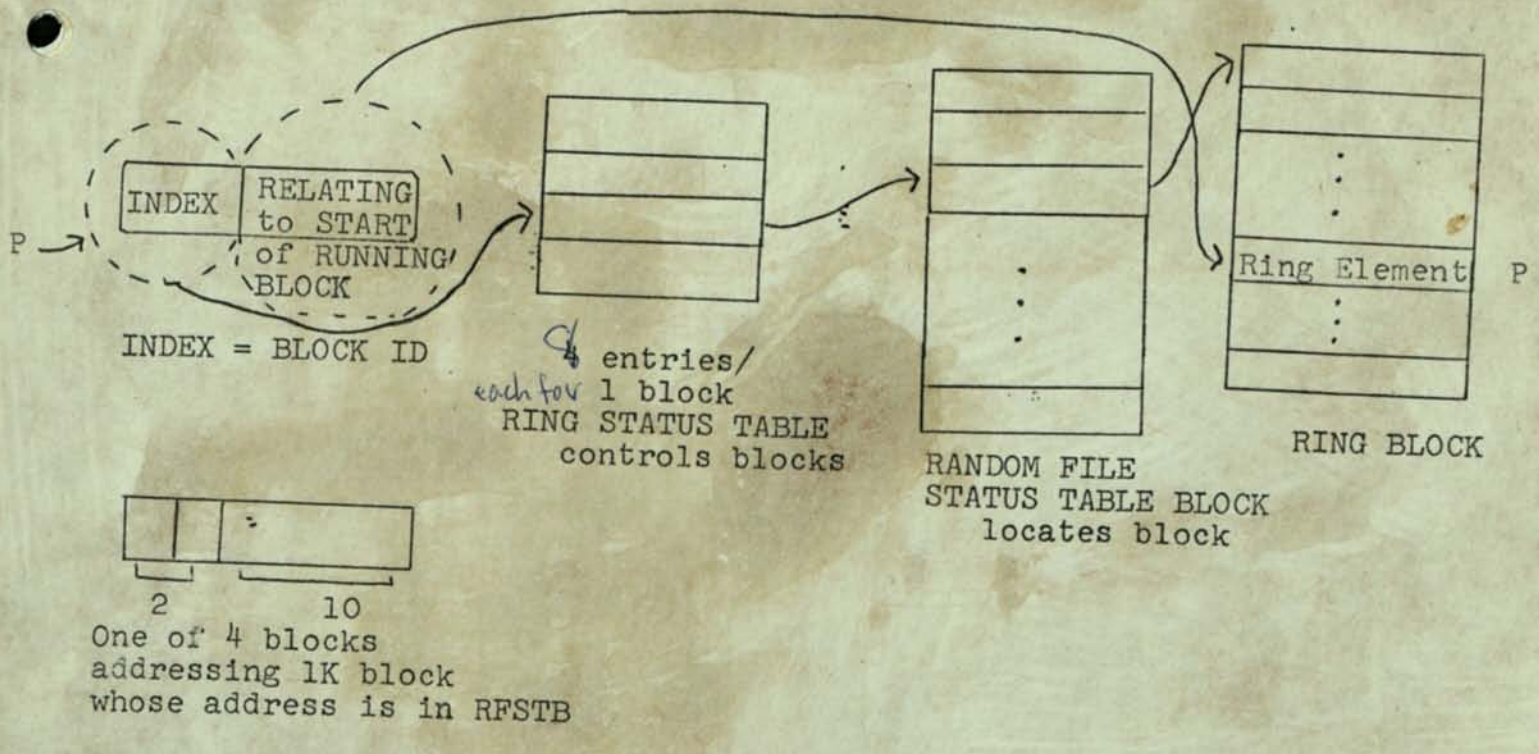


FIG. 4 Structure and Mapping of Internal Pointer 'P'

Notes on the file header: this contains pointers to all status tables and their lengths, and information on the virtual memory map. It also contains bibliographic information which may be used as a means of retrieval: last time written into, username, initials of last user, jump delimiters (these are the marks that delineate a jump, and in the present version are general parentheses), average length of statement (determined by how much activity over periods of time). This information is all contained in the first 1K words. An interesting feature is that the TS system will accept any amount extendable to 1K without using excess drum space.

The first ring element at the start is 'dummy'; and is the start of the file. When the system rewrites the file on drum after use, it searches to the first semicolon and puts in place of what is there the file description: username, initials, date and time, etc. of last use.

1.6.2 Mapping From the Ring Element to the Text of the Statement (See Fig. 5)

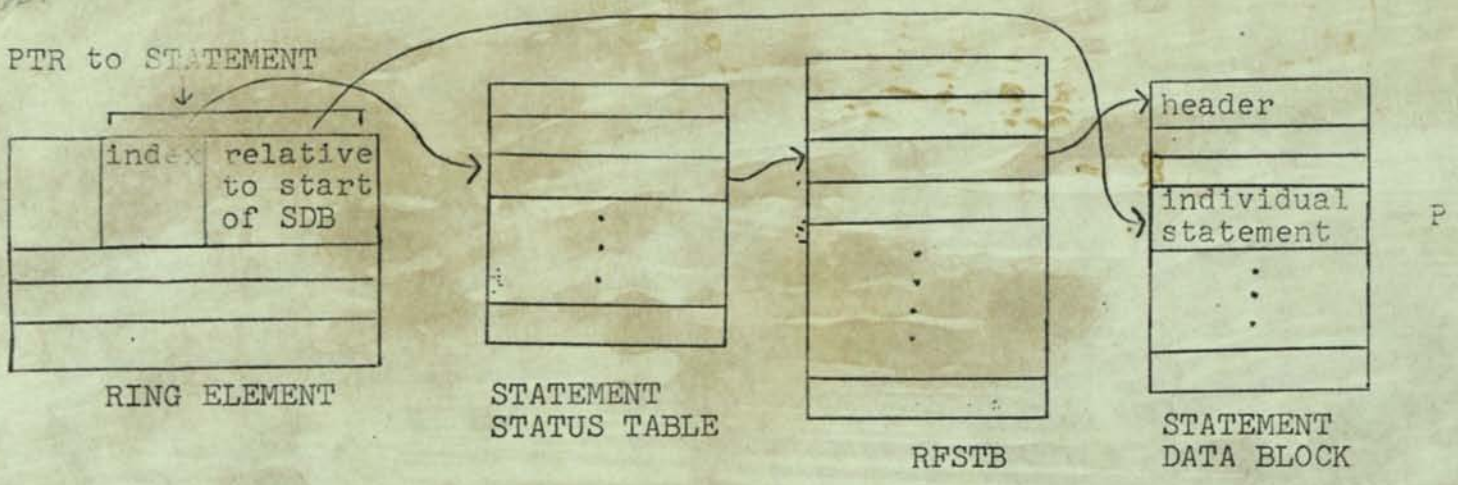


FIG. 5 Mapping from Ring Element to Text

Now that we have the appropriate ring element relating to statement P (see Fig. 2) for the structure of ring element, we can retrieve the statement-text for filter/format/display. The

system takes the "pointer to text" in the first word of the ring element. This pointer is of the same structure with respect to the statement in the statement data block as P*4 is to the ring element of the ring block. Thus the high bits are an index on the statement status table. The entry in the SST points to an entry in the RFSTB, which in turn points to the location of the appropriate SDB in which the desired statement is located. Once we have the appropriate SDB, the 10 low order bits of the original pointer point to the desired statement, relative to the start of the SDB. (See Fig. 3)

2A63 1.6.3 Generating a Sequence of Statements

2A63A Given the appropriate individual statement P (see Fig. 3 for structure of the statement entity), the sequence generator now takes the statement text for filtering/formating/display, as described in Section 3.

2A63B Which statement is taken next depends on the sequence being followed by the sequence generator. If the sequence generator is following the basic hierarchical tree structure, it will look at the ptr-to-sub field in the ring element (Fig. 2), and use that pointer as it used P above. (However, if a filter is set for a specific level and statement P was on that level, the sequence generator will ignore the sub field and take the ptr-to-successor field. ~~The sequence generator can tell the level of the statement P by the flag set the in the ring element.~~) has to find the level of the first st. & keep track of it

2A63C The sequence generator, however, may be following an associational trail. If this is the case, the content analyzer will scan the statement-text P for the appropriate trail marker. If it finds the appropriate trail marker in the statement-text, it will hash the name in the trail marker, and scan the name hashes of the ring elements until it finds the correct ring element, and continue generating statements from there. If the appropriate trail marker is not found, it will follow the tree structure as able. (- by hierarchy until a trail marker is found)

2. DATA STRUCTURE MODIFICATION

3A The data structure is modified through the basic editing commands (delete, insert, replace, move, copy, break/join) which are described briefly below in Section 4. System features and facilities are described more completely in the "NLS User's Guide" (a SRI publication).

3B We will describe how the data structure is modified for an insert; the other types of edit-modifications are all similar. If the edit is an insert, it is an insert in the middle of a statement. By system definition, all editing is based on the statement. The user types in the appropriate insert command, hits the point of insert with the mouse, and types in the insert. The insert typed appears on the screen in the literal type-in area. If the user decides the insert is complete, he hits the command-accept button. The system then makes the modification of the data structure as follows:

3C The system computes the new length of the statement by taking the old length of the statement in the statement data block and adding the length of the insert. The system then finds a free area on one of the statement data blocks of sufficient length to put the new statement. It tries to put the updated statement on the same SDB. If the edited statement does not fit in that SDB, the system tries to compact the block. If that would not give enough space, the system goes to the previous ring element and sees what SDB that statement is stored on and tries to fit the newly updated statement on that block. If it doesn't fit there, the system looks through the SDBST to find any free area and fits it in anyplace.

3D Now that the appropriate space is allocated, the updated statement is constructed. This is done by copying the header of the original statement and the text up to the insert point, adding to this the literal type-in, and copying the rest of the text of the statement. Then the "ptr-to-text" in the associated ring element is changed to point to the new statement, and the garbage bit is set in the original statement.

3E When any statement is edited, the system checks to see if there is a statement name, or label. If there is, it is rehashed and replaced in the ring element. Thus labels are always updated.

3. REDUCING THE DATA STRUCTURE TO A SCREEN DISPLAY

4A The process of scanning the data structure to retrieve and display the desired text has four basic parts: (1) the sequence generator (as discussed briefly in Section 1.6.3), (2) filtering, (3) formatting, and (4) display.

4A1 ### 3.1 SEQUENCE GENERATOR

4A1A The sequence generator is the routine that actually scans the data structure and generates the sequential text. Basically it generates a list of statements. There are three types of sequences that can be generated:

4A1A1 #### 3.1.1 Tree.

4A1A1A This is the default hierarchical structure that is generated and is simply the sequential text of the main associational trail of the text, ordered in a hierarchy of statements.

4A1A2 #### 3.1.2 Trails

4A1A2A The trail feature is used to set up statements in such a way that only a particular set of statements will be displayed and in a particular order. The set of statements is called a trail, and is an associational trail that criss-crosses the default (main) trail; it provides a manner other than the normal sequence in which to read the text. A trail marker is set up for a particular trail of statements; the pattern for this marker can be a complex syntactical form and is followed by the content analyzer (described in an SRI publication).

4A1A2B Trail markers are thus used to mark turning points from the normal sequence of statements, as a signpost to the next statement in the trail. Each time a marker appears in a statements it is followed by a statement name in parentheses that is the name of the next statement. Between trail markers

statements are displayed in normal sequence. The trails can be followed only in the forward direction; there is no capability for inverting the trail when moving backwards through the text. (SRI claims that with the complex content-analyzer, this is unnecessary.)

4A1A3 ### 3.1.3 Keywords

4A1A3A The keyword system permits a user to construct a specially formatted catalog file containing references to other files and capable of being reordered automatically according to some chosen set of weighted keywords. When reordered, the file lists references in order of relevance, according to the choice and weighting of keywords.

4A1A3B The keywords are attached to a statement. The system keeps a list of the keywords containing for each keyword a short description of the keyword, and the labels of statements tagged with this keyword. This list is visible to the user and can be changed by him. The system also keeps a list of the file-reference entries, that is, a file of any statement name tagged with a keyword, and a list of the keywords it is tagged with following it. Thus one keyword can be attached to any number of statements, and one statement may have any number of keywords attached to it.

4A1A3C The keyword system is used mainly as a retrieval-by-keyword system. The user selects desired keywords and weights them according to importance. A negative weight can also be used to blackball any keywords. According to SRI, the weights on the keywords allow more flexibility than straight Boolean retrieval functions on keywords; after the user has selected keywords and weights, the system goes to the list of keywords and picks out all statements tagged with the selected keywords. For each statement selected the system computes the weights of keywords attached to it, and displays the names of the statements in order of highest total weight. Statements with a negative total weight are not displayed. The user may then access the referenced files by using the jump command on the statement names.

4A2 ## 3.2 FILTERING

4A2A Facilities included are: level specification, branch only, subfile, content analyzer, trail flags, literals, search for trail flags and literal text, etc.

4A2B After the main structure is generated and filtered, it is formatted.

4A3 ## 3.3 FORMATTING

4A3A The formatting sets the following, and other, variables of display:

4A3B Statements numbers: the number of lines of each statement to be displayed is variable; headers, time/initials/labels can be on/off.

4A3C View change: character size, page size and dimensions, etc.

4A4 ## 3.4 DISPLAY

4A4A After the statements have been filtered out, they are displayed. The main display of the generated/filtered/formatted structure is in the file area of the screen. There are a number of one-dimensional registers used for man/machine interaction:

4A4A1 1. Echo register. This displays the last six characters typed by the user, for feedback.

4A4A2 2. Command display line. This is a line which says what command is in the process of being executed.

4A4A3 3. Name register. Displays user's name (this is on a multi-terminal system).

4A4A4 4. View specification areas. There are three view spec areas, and these are set according to the formatting variables described in Sec. 4.28.

4A4A5 5. Message area. An area for system messages to the user, such as error messages.

4A4A6 6. Literal type-in area. When the user is typing in an insert or delineating a command, the characters typed are displayed in this area.

4A4B There is, in addition to the file area, another two-dimensional area, the freeze area. This freeze area is used to "freeze" statements designated by the user so that they remain

unchanged above the file area, with the file being then displayed in the file area. The freezed statements remain unchanged despite any text manipulations or file searching that goes on in the file area. (In a future version, the freeze area will be done away with, and instead the user will be able to multi-window any number of windows. Each window will be a full file area, with all one-dimensional registers in each window. They can be any shape or size any place on the screen. With multistations, a window can be assigned to a station, giving the users at two different terminals the ability to decide who holds the chalk and who holds the eraser in each window.)

4. SYSTEM FEATURES AND FACILITIES

4.1 EDITING

The basic editing commands are delete, insert, replace, move, copy, set, and break/join. All are self explanatory, except set and break/join. The set commands allow the user to change the font on any text string. The fonts are: capital, lower case, italic, roman, boldface, no boldface, flickering, non-flickering, underline, no underline. The break and join commands allow the user to break a statement into two statements; the join command adds a text string onto another statement. The break and join commands are the only commands that operate across statement boundaries. All the other editing commands are specialized: for example, the insert commands are insert character, insert word, insert text, insert invisible, insert statement, insert branch. The specialized commands make it easier for the system to make the edits; the rationale for specialization is that since you have to type the command in you may as well specialize, and economies in data structure manipulation may be achieved (e.g., moving an entire branch of the tree).

4.2 OTHER FEATURES

4.2.1 Invisibles

When editing, invisibles such as spaces and tabs can be displayed by marks, and thus can be deleted.

4.2.2 Labels

Labels are statement names and are used for retrieval purposes by jumps, links, and keywords. They are inserted as part of the text, that is, with an insert command. A label is simply a variable-length character string that appears at the beginning of a statement in parentheses. These labels can be changed or deleted as if they were regular text.

5B2B Duplicate labels can be created. A jump to a label results in a jump to the first occurrence of that label, since the system sequentially scans the name-hash field of the ring elements. A feature contemplated for incorporation in the system is a "look for next occurrence of this label" jump to resolve duplicate labels.

5B3 ## 4.2.3 Links

5B3A A link is an association to another statement, i.e., it is a jump to another statement that can be taken at the option of the user. The link can be in the current file or in another file. There are four parameters to a link: three (the user name, filename, and label) define the point linked to. The fourth is the view specifications on the text linked to. This is an interesting feature: that view specifications can be changed on all links.

5B3B The link structure is a regular text string inserted in the text as if part of it, and is in parentheses in a syntactic form. Like labels, the link is just regular text until it is used. It can be edited at any time. When the user decides to take a link, he hits a character with the bug. The system scans forward with the content-analyzer until it picks up the nearest link structure in that statement, and jumps to the label. The link is taken by use of a jump command.

5B4 ## 4.2.4 Intrafile Return Ring

5B4A Whenever any jump is made within the file, a new entry is made in a list called the intrafile ring. Each of these entries gives a display start and a set of viewspecs. A pointer indicates the current view on the list. Each time a jump is executed, the new information is written ahead of the pointer and the pointer is moved forward. On a jump return or jump ahead, the pointer is simply moved backward or forward and no new entries are made or any deleted. The list holds a maximum of six entries, and is circular.

5B5 4.2.5 Interfile Return Stack

5B5A This works much like the intrafile stack except that it is concerned with jumps between files. The differences with the intrafile stack are: (1) the length of the list is variable, and depends on the amount of information in the links used, (2) the list is not circular, a new entry is made on the stack whenever any interfile jump is taken or whenever a new file is loaded with a load file command. (See section on multifiles for more details.)

5B5B There are no backpointers from a link, the same as with trail markers. Thus if a label that is linked to is deleted, there is no user notification that a link has been made inoperable. Also, since link structures are entered as simple text, the label in a link structure does not necessarily exist. A link or jump to a non-existent label results in an error condition.

5B6 4.2.6 Jump

5B6A The jump command brings the desired statement to the top of the display.

5B6B There are four basic types of jumps: (1) jumps to a specified label name, (2) jumps to links, (3) jumps through the tree structure, and (4) jumps among different files.

5B6B1 In case 1, the label or statement name to be jumped to can be specified by either a word-selection via the mouse or a literal entry from the keyboard.

—>5B6B2 In case 2, the statement defined by the specified link is placed at the top of the display. More detail is given in section 4.2.3.

5B6B3 The case 3 commands allow jumps to the next substatement, the next successor, the statement of which the selected statement is a substatement, the previous statement, the head of the file, the end of the file, the end of branches, and many other links on the basis of tree and file structure. For more details see the "NLS User's Guide."

5B6B4 The case 4 commands allow the user to load a number of files into the system and to jump freely among them. These will be discussed in Section 4.2.9.

5B6B5 There is one other type of jump, the jump-ahead/return. Whenever any type of jump within the current file is executed, the sytem keeps track of it, and a ring is maintained keeping a sequential track of all views that have been used. These commands allow the user to return to a previous view or to move forward after a jump return to the latest view. (See Section 4.2.3 on links for a description of this intrafile ring.)

5B6B6 A special feature of jumps is that almost all jumps allow the user to change the view specifications of the area jumped to from those of the current text. In addition, each jump saves the viewspecs of the area jumped from in the intrafile ring, so that on a jump return the text is viewed as before.

5B7 ## 4.2.7 Pointers

5B7A Pointers make it possible to select entities that are not on the display. The entity has a pointer fixed on it while it is on the screen of not more than three characters. To select the entity at any time, a mouse button is depressed and the name of the pointer is entered from the keyboard. This is exactly equivalent to making a direct bug selection of the character that has the pointer on it.

5B7B The list of pointers can be displayed and one may use it to jump to the individual pointers.

5B8 ## 4.2.8 View Specifications

5B8A The view specifications (viewspecs) are parameters that control the way in which statements are displayed. The parameters are: indenting on/off; names on/off; display file as tree/normal text; keyword reordering on/off; display of statement signatures on/off; branch-only on/off; content analyzer on/off; trail feature on/off; pointer display on/off; number of lines displayed; number of levels of statements displayed and a few others. These can be set in three ways: with the view set command, from the special keyset, or during certain commands such as jump.

5B8B These parameters are always displayed in the upper left corner of the screen with a single letter denoting each. When they are capable of being changed by the user, they are displayed with larger letters.

5B8C There is a relative level control, which allows changes to the level parameter set by the user to be interpreted relative to the level of the first statement in the display. The user can also change other viewing parameters. These include the type of mark the cursor leaves, the number of characters in a line of text, the number of spaces indented for each level, the number of lines in the text area, the spacing between lines, size of characters, etc.

5B9 ## 4.2.9 Multi-files

5B9A When a file is loaded or jumped to, it is "opened" and displayed; no copy is created, rather the file is viewed directly from the disk. For reasons of file protection, if any changes are made, it becomes impossible to continue direct viewing, so the system creates a working copy when an edit is made. In fact, this working copy is not created until all core is filled and not necessarily on the first edit. In this way the system does not make a working copy until it definitely has to. When the system creates the working copy it copies the displayed file to it, closes the displayed file, and from then on all work is done in the working copy. No working copy is created when the user is just browsing. This is done since most users just look at files and do no editing.

5B9B Files are loaded by the load command or by an interfile jump command. Entries are made in the interfile stack as files are loaded (see Section 4.2.5). The working copy and the checkpoint file are never entered in the stack.

5B9C One feature of the multi-files is that the user can create a checkpoint file at any time. This writes the present working copy out on the drum under the name checkpoint.

5B9D The interfile stack can be used like the intrafile stack to go back and forth among views on different files. Only one working copy at a time can be created, and can be looked at any time, even if a file other than the one of which a working copy was made has been currently loaded.

5B10 ## 4.2.10 Freeze

5B10A The freeze feature freezes a single statement with the present view. The frozen statement will appear at the top of the screen

whenever frozen statements are being shown, with the main text display on the under part of the screen. A fixed number of statements can be frozen, and are displayed in the freeze area in the chronological order frozen.

5B11 4.2.11 Tree-display Feature

5B11A This allows the user to see the file as a tree structure, or in the hierarchy form, instead of normal text. The tree structure shows the relationships of statements in the file to each other. This is done by indenting the differing levels of the tree to different depths, much like an outline form. This can be turned on or off by the view specifications.

5B12 4.2.12 Statement Numbering

5B12A The system numbers each statement Dewey Decimal fashion according to the tree structure. This numbering is computed at display time. The numbering can be turned off by the view specifications.

5B13 4.2.13 Vectors

5B13A The vector package allows the user to create simple line drawings, with labels for jumps. The vector is drawn by specifying the endpoints with the mouse. Either endpoint of a line can be translated, and the entire drawing and any label can be translated. These vector labels can be used as jumps to that statement name.

5. FUTURE FEATURES

5.1 MULTIWINDOWS

This may have been inspired by our multiwindows. Theirs, however, is fancier in conception. This would allow any size and shape windows to be defined, and each window to be a self-contained viewing area with all the parameters as described for the single screen display. Their multiwindow facility could also assign different windows to different users. This assignment is done by the time sharing system, though; the only programming problem is the protocol: who holds the eraser in each window.

5.2 VARIABLE SYMBOLS

This would allow the user to define a variable symbol for text, links, etc. The symbol would be filled in with text at display time, like an assembly time variable. Alternately, the variable symbol could simply be permanently defined at a later time.

5.3 WEIERSTRASS ALGORITHM

Currently the system uses a display map technique for detecting bug hits. A future plan is to use the Weierstrass Algorithm of continually subdividing the screen to find the line closest to the bug mark, which would be the line hit.

6. A FEW IMPRESSIONS

6C3 The hierarchical structure allows the text to be set out in a tree form very easily. The question of advantage of this over traditional text was discussed with Engelbart. He said that the hierarchical statement-oriented structure was selected just as a starting point and empirically has proven to be more helpful to users in terms of visualizing the text. He insisted there is no premeditated reason toward this structure, nor need it be imposed on the user.

6C4 The statement oriented quality limits the flexibility of editing somewhat. From our point of view, there is no editing across statement boundaries, for instance. Jeff said that this limitation is of no real importance since as users gain familiarity with the statement oriented system, they learn to make statements complete thoughts, and so editing across statement boundaries is not really necessary; the limitation is only on traditional thinking with traditional text. This is the same reason Engelbart stated for using hierarchy: the user quickly adapts to the structure provided him.

6C5 One advantage of the statement oriented structure is that to move a branch or a statement requires no actual movement of text, but just the changing of a few pointers.

6C6 There is great effort not to let the user hurt himself when he cannot see the entire tree structure due to filters. For example, a user cannot delete an entire statement. There might be substatements below that are filtered out that he might inadvertently delete: he must give a delete-branch command and delete the entire branch.

THE USE OF LANGUAGE
DGC 7/15/69

This file consists of notes about the nature of language as relevant to AHI's problems in communicating ideas. The emphasis is on the problems of internal communications.

Formal Technical Languages and AHI

The classical AHI approach to language problems has been an attempt to create a special subset of English which is fully controlled --- i.e. in which all definitions are clearly understood and in which there are no serious ambiguities; this subset would be created by the promulgation of "official" definitions and (this part is less clear) of rules for constructing sentences and arguments.

This approach seems to have arisen from a feeling that our language problems are problems of definition and style, which can be attacked by prescribing a uniform system of usage.

It seems very clear to me that such an approach will not work.

In my view, it is a sad fact that some ideas are simply difficult to express; one attacks the problem of expressing them by applying ingenuity, instinct, and a lot of hard work cutting and trying.

The classic approach -- establishing a special sublanguage of English -- is inspired by the practices of mathematical language and (to a varying extent) the physical sciences. These disciplines have developed very successful formal technical languages, which are almost universally understood by the people involved and without which progress and communication would be almost impossible. Before trying to adopt the methods of these disciplines, however, certain things must be noted.

The most successful formal technical languages are found in the most mathematical disciplines -- disciplines in which the relationships among concepts are very clear at least locally, disciplines which lend themselves to abstract, axiomatic methods.

In physics, for example, we have the special words "mass", "acceleration", and "force". The local relationships of these terms are completely expressed by the very simple mathematical construct f=ma.

Moreover, any one of these terms can be defined -- well enough for short-term purposes, at least -- in a fairly short English statement consisting mostly of familiar English words used in their everyday senses.

If we arrange disciplines along some sort of scale which shows

how much abstraction can be applied in each discipline, we will have mathematics at one end and something else (God knows what) at the other. astrology ?

Near mathematics we have physics, chemistry, and related "hard, quantitative" sciences; then come less quantitative, less precise ones like geology and biology; then we begin to move into areas where abstraction is more and more speculative -- psychology and the social sciences; and so forth.

Obviously this scale is not really linear; obviously there are many other things wrong with it; but we can agree that psychology, for example, allows less abstraction than physics, and furthermore that the abstractions of physics are much more generally respected than those of psychology.

We may also observe that the special language of psychology is wildly disunified, in spite of vigorous attempts by some psychologists to standardize it. It is a subject of constant disagreement among psychologists, and a source of either high frustration or low comedy to non-psychologists.

Much the same situation obtains, in greater or lesser degree, among the social sciences.

It is my feeling, shared widely by others in the sciences and the liberal arts, that it is a serious mistake to try to set up a rigorously defined special technical language unless you have a discipline that is rigorously comprehensible in abstract form.

A "jargon" might be defined as a formal technical language that doesn't make it. Jargon is bad jazz, and I want to devise ways of having good jazz. Before I go into that I will make some observations about English.

About the English Language

Millions of words of garbage have been written about the English language. I might as well contribute my own share, since everyone else does.

Most of what I have to say about English applies (as far as I know) to most other natural languages. English has one very special distinction, namely the size of its vocabulary.

I don't have the numbers on hand, but English has something like three or four times as many words as (say) Spanish.

The reason is that English is far more mixed than any other major language; for each Germanic or Latin root-word, there are closely related words taken from Celtic languages and from later developments of Germanic and Latin such as French, Norse, Flemish, etc.

There is also a second-order effect -- the huge vocabulary is an environment which favors the free coinage of still more words, by combination, by folk-etymology, by pure invention, etc.

This effect may be explained by the simple fact that the size of the vocabulary places it beyond intuitive comprehension by any normal individual. Since no one knows all of English, everyone is free to play games with vocabulary.

The same fact explains the characteristic fluidity of English definitions. English usage changes more rapidly and more unpredictably than that of any other language that is left to its own devices, and consequently the formal (dictionary) definitions are also fluid -- they lag behind usage but do follow it.

(Some languages change very rapidly by means of imposed reforms -- Danish, for example, has changed more in the last century than English has since Chaucer. But English does not respond to imposed reform, a very important point that I will come back to.)

Finally, the historically very high literacy of the English-speaking peoples has meant that English usage goes into print in vast quantities, and the print is read, saved, and re-read later on. The changes in English, therefore, do not as a rule take the form of replacement -- very little of the language gets "lost" from one century to another. Instead, it keeps growing and the relationships in the vocabulary keep getting more complex. Simultaneously, the individual word-meanings get more complex, and up at a higher level whole clusters of words (idioms) take on special meanings that evolve fluidly with time.

Thus English is a monstrously complex structure and a monstrously extensive one. Any natural language is vastly complex; English is simply more complex than most.

In any natural language, all that complexity is highly dynamic. If you poke the structure at one point (by redefining a word, for example) the entire structure may reverberate. The waves may damp out quickly, but on the other hand they may build, reflect, and come roaring back. Quite frequently the echoes will shift the whole linguistic structure surrounding the original stimulus.

In other words, if you make a local change in the language, you must be prepared for the entire language to react in such a way that other changes happen in the same locality as well as at distant points. It is a tricky business.

My own favorite metaphor for the language is that it is a living organism. Moreover it is a conscious, reactive, intelligent organism which, although it is of course a part of the people who speak it, may act as if it were quite independent of the conscious behavior of individuals or even large groups of people.

Like other organisms, it seeks to preserve its own integrity -- an integrity which individuals do not understand because of the size and complexity.

For example, dozens of careful, powerful, well-coordinated attempts have been made to reform English spelling. All have failed. They have been based on the premise that English spelling is arbitrary and illogical, when in fact it is neither.

It is possible to formulate rules which cover the spelling of 90% of English; the rules are useless to people because they are too complicated and require all sorts of esoteric input such as history of usage, etymology, and history of pronunciation, which normal people can't cope with.

But the rules do exist, they are very nearly perfect and complete, and the English language "understands" them. It vigorously resists attempts to change it from the "outside", while constantly going through its own organic changes.

It should be far easier to control some subset of the language, however, than it is to control the whole thing. Indeed, this seems to be true: the successful formal technical languages of the "hard" sciences prove that it can be done. But it seems to me that there are two special problems.

The first problem is controlling the size of your subset. You can't have a truly closed subset of English, because the associational connections between words are so strong; if you define a vocabulary, the words in it keep pulling in other words. If the process gets out of control, pretty soon your subset is so big that you might as well be trying to control all of English.

The second is knowing just what you want to express with your formal technical language, before you have a formal technical language to use for defining the problem. The obvious answer is that you start small and bootstrap, but now the first problem really gets critical.

Again, I think, we must note that the disciplines which have good formal technical languages going are ones with good, sound abstractive structures which are expressible (at least to some extent) in mathematical notation. The math notation acts as a parallel language on which to fall back when problems arise in the English expression of a concept.

I have a distinct impression that mathematical and physical and chemical English have been bootstrapped from a nucleus of intricately mixed mathematical and English expressions. Another factor has been extensive parallel use of other natural languages, notably German and French, which are radically different from English.

The point of this whole discussion about English is that a program of trying to impose special meanings upon English constructs is not going to work very well for AHI. Augmentation may someday be a discipline with the underlying structure needed to support a special sublanguage, but it isn't there yet. I am afraid that if we try to set up a formal technical language we will wind up with a jargon instead, and a jargon is a bad communication block.

Needless to say, we must have some degree of control over some part of the language. It's a question of how much, and I think the answer is "not very much".

Fortunately, there is an alternate approach. The small amount of control that we can use will be enough if we accept the idea that expression of augmentation concepts is difficult, resolve to work hard at it, and focus attention on how to talk rather than on what language to use.

Footnote on the Whorfian Hypothesis

It is certainly true within AHI that our conceptual thinking is limited by our language. But "our language" has been an embryonic formal language that meets with resistance, not the whole of Elish. I guarantee that English, if used flexibly, experimentally, cratively, has easily the resources to express the AHI ideas with a minimum of imposed definition.

Slang and "Lab Slang"

I've been discussing "straight English", formal technical languages, and jargon. There is also something which I call "lab slang".

As noted above, English is highly fluid and encourages rapid improvisation. A slang (in the general sense) is simply as special sublanguage belonging to some special group of people who are united by common concerns and common knowledge; its distinguishing features include the following:

(1) A slang uses only a small part of the substance and structure of the base language, but it always uses the full creative force of the language as a whole. English proliferates slang at a great rate because of its improvising tendency.

(2) A slang is never designed; it always evolves by direct response to immediate communication needs. In this respect it is like the base language but even more so.

(3) A slang is even more fluid than the base language. Thus a white man may study ghetto slang and write down what he learns, but by the time he tries to teach other white men the slang, his knowledge is already out of date.

(4) A slang has special communicative functions:

(a) It provides special terminology for identifying and manipulating ideas and experiences in a situation with the following characteristics:

The ideas are not yet rigorously formulated nor unambiguously related, and are subject to modification in the course of discussion.

This diffentiates slang from technical language.

The ideas are peculiar to the group.

The ideas are about equally well understood by all or most members of the group.

This diffentiates slang from jargon.

(b) It promotes group solidarity.

(c) It promotes communication in the group by making speech a strong person-to-person contact.

A slang language typically makes it difficult or impossible to separate idea-communication from personality-communication.

It constantly reaffirms the common orientation of the speakers.

It employs poetic technique to heighten emotional impact and to display the emotional content lying behind idea-communication.

It forces creative verbalization simply because it uses limited resources (small vocabulary, simplified grammar).

Creative verbalization communicates personality, because creative improvisation of any kind is largely determined by individual personality.

Lab Slang and Formal Technical Languages

A lab slang is a special case where the group using the slang consists of people working in a research discipline.

Lab slang is found most characteristically at the intellectual frontiers, where there is not enough agreed-upon structure to support a formal technical language, and in applied disciplines (such as much of engineering) where the effort involved in setting up rigorous linguistic conventions is not considered to be cost-effective.

It is worth noting that such disciplines tend to have either

slang or jargon, but not both; also that slang often exists in parallel with a formal technical language, but jargon does not (because jargon is a formal language gone bad).

Lab slang evolves because it is an excellent solution to the severe problems of communicating exploratory ideas in the absence of a lot of supporting structure.

The ideas involved are in a state of flux, so that a highly flexible language is needed; slang is the most flexible mode of verbalization.

Verbal improvisation is time-consuming if normal linguistic materials are used; slang is by nature compact in expression -- so where improvisation is necessary, slang is used.

Slang is inherently nonrigorous; it is suited to fast-moving and exploratory environments where rigor may be deceptive and certainly tends to slow the necessary intuitive thinking.

When technical language and slang coexist they do not influence each other directly because they deal with different domains of the discipline.

The cross-influences certainly exist, of course; but they seem to be subtle. There is not, for example, a process of direct incorporation of terms from slang into formal language, nor does the reverse seem to go on.

The strong relationship between the two kinds of language is an indirect and complicated one; the following is only a very rough outline of the interactions.

The slang is used for carrying on necessary detailed communication while people are working to set up enough framework to incorporate new ideas into the formal language.

At the same time, formal-language constructs are translated into slang in an attempt to discover possible extensions of existing ideas into the new domains.

When enough theoretical framework has been set up, concepts which have only been loosely expressed in slang are re-expressed (not translated) in formal lnguage.

Throughout the process, of course, use is made of any abstract symbolisms (such as math) that may be available.

Lab Slang as a Model for AHI Internal Communication

AHI doesn't yet have a real lab slang, and I am arguing that one should be encouraged to develop.

We do use the slang mode at the computer-system level, but that isn't really AHI slang -- it's computermen's slang with a local dialect superimposed.

Where we have problems is in talking about augmentation itself, and about bootstrapping and evolution. In these regions we use the slang mode only very sporadically, because we are all uptight about the ideas involved.

The approaches to language that I suggest are intended, essentially, to promote a loose, improvisational, slang mode of discourse for our own use.

Promoting such a mode is a matter of letting go -- not a negative approach at all, though it involves abandonment of some ideas that we have been treating as basics.

I would like to abandon the idea of controlling language, and push the idea of experimenting with language.

If a formal technical language depends on the existence of a commonly understood framework, and we don't have one, then we must give up on developing a technical language for the moment.

In order to keep talking while we build the framework, we need slang, which depends only on the existence of a few commonly understood concepts and experiences, but not on a commonly understood system of relationships.

This, I think, is how to bootstrap a general understanding and clear formulation of the underlying conceptual framework of AHI. It is difficult and time-consuming to proceed this way, but it works. We must have mutually intelligible conversation among ourselves on the subject of augmentation, and the only way I can see of achieving it is to let go on language.

Language and Personality

(This section is only a brief supporting note, and derives from Perls, Hefferline, and Goodman, Gestalt Therapy, Vol. II, Ch. VII, "Verbalizing and Poetry". Dave Evans has the book, and the chapter has been Xeroxed and placed on the DSS Shelf.)

An individual's linguistic habits are a primary expression of his personality; it follows that changes in linguistic patterns will affect personality (though likely not in a linear, one-to-one fashion).

Although not everyone recognizes this fact explicitly, everyone feels its impact on an intuitive level.

What this means to AHI, bluntly, is that if a program of language

control were attempted there would be no cooperation.

Language control in any domain where emotions and personalities and beliefs are involved is instinctively perceived as thought control. Language habits are so intricately involved with personality that an attempt to control them for abstract reasons is emotional dynamite.

Even the few sporadic efforts that have already been made to control AHI language have resulted in serious strains and drastic blockages of communication.

To encourage loose, improvisational, jam-session language, on the other hand, is to encourage full communication of individual personality in the "professional" domain. Community, anyone?

How to Make Good Jazz

This proposal is quite limited -- it doesn't consider Journal problems or long-range developments.

The basic idea is that if language is like an animal, you can either try to drive it or try to ride it. I've argued that it can't be driven very well, and so I think we should try to ride it. This involves three basic things:

The first is just to loosen up generally -- determine less and improvise more.

For example, the Glossary that is currently under discussion should be small and nonrigorous, with an explicit understanding that it will stay that way for the time being.

The second is to work with language at a much higher level than word-definitions. Definition of terms is the lowest level of all; we should be working with metaphor and image.

We all do this all the time in informal conversation; it's just a matter of developing the self-assurance to talk informally when we're talking in depth. If we did that, we would have the beginning of a viable, evolving lab-slang.

The third is to develop a willingness to abandon -- immediately, in real time -- verbal constructs that turn out to be uncommunicative.

This turns out to be a skill that can be developed with practice. It results in vastly improved communication and expanded understanding of one's own ideas. Translation into another language has always been one of the best ways to achieve understanding of any verbal construct.

I will be developing these ideas in detail in the future, but let me

note that the program they aim at is as follows (in no particular order):

(1) Have a glossary with a SMALL number of terms in it, all subject to ruthless modification and NOT constrained to be logically consistent.

The "definitions" might be more like encyclopedia articles than definitions.

(2) Improvise constantly.

If you throw a light beam at a three-dimensional object, you only learn a little bit about it from the shadow; but if you keep illuminating it from different angles, you eventually find out what shape it is. If the shape of the object keeps changing with time, you have to keep on taking new shadowgrams. This is hard work, but better than determining the shape from theory and leaving the object in the dark.

(3) Make stationary targets out of ideas, instead of words. Words just won't hold still.

I don't know just what it means to make a stationary target of an idea. At any rate, though, you have to have faith in the people who are shooting at the target.

(4) Eliminate semantic hangups by taking the permanence and awe and glory away from verbal constructs and lodging them in visions, where they belong.

The Zen Buddhists have a technique for this, which is just a studied disrespect for all verbal constructs. This technique may be a little too heavy for us, right now. It is very easy to overdo Zen by making a destructive game out of it.

(5) Make it impossible for people to use words as intellectual bludgeons, which many people here do all the time.

Again a matter of downgrading the importance of words.

(6) Let our working language evolve, with only a carefully limited set of controls and constraints.

I mean hands-off evolution, not guided development.

(7) Not worry too much, for now, about comprehensibility to the outside world -- internal communication is a far more urgent problem.