AD NUMBER: AD0843577

LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited.

FROM:

Distribution authorized to US Government Agencies Only; Export Control; 1 Oct 1968. Other requests shall be referred to Rome Air Development Center, Griffiss AFB, NY 13441.

AUTHORITY

RADC, USAF ltr dtd 13 Jun 1969

RADC-TR-68-250
October 1968

DEVELOPMENT OF A MULTIDISPLAY, TIME-SHARED COMPUTER FACILITY AND COMPUTER-AUGMENTED MANAGEMENT-SYSTEM RESEARCH

D. C. Engelbart
W. K. English
J. F. Rulifson

Contractor: Stanford Research Institute
Contract Number: AF30(602)-4103
Effective Date of Contract: 23 February 1966
Contract Expiration Date: 7 March 1968
Amount of Contract 659,204.00
Program Code Number: 7D30

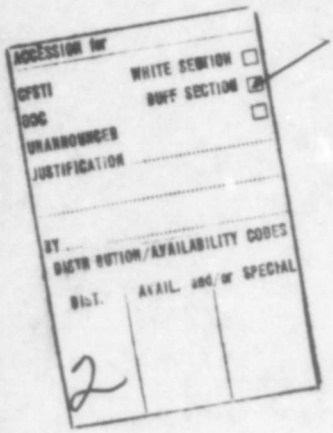
Principle Investigator: D. C. Engelbart

Project Engineer: Fred Normand, RADC
Phone: AC-315-330-3678

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 967

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, New York

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded, by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacturer, use, or sell any patented invention that may in any way be related thereto.



Do not return this copy. Retain or destroy.

ABSTRACT

This report describes work in a continuing research effort on systems for computer aid to management in the AHI Research Center of Stanford Research Institute. This research is part of an overall program aimed at significantly augmenting the intellectual abilities of humans by means of direct, interactive computer aid.

Specific applications to management problems include management-information systems, aids to conferencing and group interaction, and the ability to maintain highly flexible and dynamic records of various kinds and use them as a working framework.

Special software developments for implementation and modification of interactive user aids include Tree Meta compiler-compiler; the MOL940 machine-oriented programming language; four Special-Purpose Languages (SPL's), for high-level specification of user control functions; and a multipurpose interactive user-aid system (NLS) which integrates these components. These systems operate within a slightly modified version of the time-sharing system developed by Project GENIE at the University of California, Berkeley. The machine used is an SDS 940.

Specially developed hardware coupled to the computer provides I/O service to six CRT user consoles. A novel display system uses small CRT's located in the computer room, with television coupling to the individual user consoles. This has several advantages, including the ability to display material as black-on-white lines and symbols.

DEVELOPMENT OF A MULTIDISPLAY, TIME-SHARED COMPUTER FACILITY AND COMPUTER-AUGMENTED MANAGEMENT-SYSTEM RESEARCH

D. C. Engelbart
W. K. English
J. F. Rulifson

Stanford Research Institute

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by Fred Normand RADC (EMIIF), GAFB, N.Y. 13440 under Contract AF30(602)-4103.

AFLC, GAFB, N.Y., 6 Nov 68-106

FOREWORD

The work reported here was carried out by the Augmented Human Intellect (AHI) Research Center of Stanford Research Institute, Menlo Park, California. This Center is engaged in the exploration of the concepts of augmenting the human intellect by means of direct, interactive computer assistance to intellectual processes.

The AHI Research center operates under the multiple sponsorship of ARPA, NASA, and the U.S. Air Force, Rome Air Development Center. The specific work reported here was funded jointly by ARPA and Rome Air Development Center under Contract AF30(602)-4103. The RADC project number is 5581 and the project engineer was Fred Normand, Rome Air Develoomnet Center, EMIIF, Griffiss Air Force Base, NY 13440. The number assigned by the contractor to this report is SRI Project 5919.

This report, from the Introduction onwards, is written in a special format developed by the AHI Research Center and called "structured statement format." The reader will see that the text is broken into brief paragraphs or "Statements" which bear numbers indicating their positions in a hierarchical structure. References are given in the form (Smith3) instead of by superscript numbers, and will be found in the Bibliography at the end of the report. These conventions are part of an on-line computer text-handling system (NLTS and NLS) developed by the Center and used for all documentary work.

Distribution of this report is limited by the Mutual Security Acts of 1949.

This technical report has been reviewed and is approved.

Approved: James G. McGinnis

JAMES G. MCGINNIS, Major, USAF
Chief, Info Processing Branch
Intel & Info Processing Division

Approved: Howard Davis

HOWARD DAVIS
Acting Chief
Intel and Info Processing Division

FOR THE COMMANDER: Irving D. Gabelman

IRVING D. GABELMAN
Chief, Advanced Studies Group

CONTENTS

LIST OF ILLUSTRATIONS

1 INTRODUCTION

1A This is the final report for a contract that began in February 1966 as a one-year exploration into computer-augmented management techniques, and ended two years later, having focused mostly on the development of an advanced research facility that will serve as a laboratory for the exploration of computer aids to management.

1A1 The initial management-techniques project, as integrated into the Augmented Human Intellect (AHI) Program at SRI, was to pursue its research goals by developing computer-augmented management techniques to be used and evaluated in the management activities within the Program.

1A1A Appendix A describes the AHI Program, its "Bootstrap Community" plan, specific features of its experimental approach, its prior support history, and the status of the Program at the beginning of this project.

1A2 A major change in the plans of the AHI Program -- i.e., to establish a large computer-display research facility specially designed to support the AHI research program -- was approved early in the contract period and resulted (with the knowledge and approval of the contract monitor) in the following:

1A2A Deferment of the management-system research until the major change was implemented

1A2B Enlargement of scope and increase in funding for this project to support the new AHI research facility. This enlarged facility will provide greatly increased power for management-system research and experimentation.

1A3 The new facility is to provide continuous service to the AHI research staff, to use both in their daily work (design, study, document, program, plan, etc.) and in their system-improvement experimentation.

1A3A The new facility development is behind its initial schedule, so that the management-system research has involved a larger proportion of conceptual and methodological study, and a smaller proportion of computer programming and special-technique utilization than was initially planned.

1A4 This report includes the following:

1A4A Description of Software Developments:

1A4A1 A set of Special-Purpose Languages (SPL's) to

facilitate system programming in the areas of user-display interactive dialogue, character-string analysis, character-string construction, and ring-structure manipulation.

1A4A2 MOL940 (or simply MOL), a Machine-Oriented Language for doing detailed system-subroutine programming

1A4A3 Tree Meta, a compiler-generation system used to produce the compilers for the SPL's and to produce the syntax analyzer for the MOL compiler.

1A4A4 The On-Line System (NLS), which interactively serves the CRT work stations. It is designed for maximum ease in developing and modifying both the service functions available to a user and the particular "dialogue procedures" used to control the application of these functions. Tree Meta, the SPL's, and MOL are designed to serve the need for flexible development and evolution of the NLS).

1A4B Description of the hardware developments for the new multiconsole time-shared man-computer research system:

1A4B1 An SDS 940 time-sharing computer, using the latest time-sharing system developed by Project GENIE, University of California, Berkeley.

1A4B2 A 96-million-character disc-file system.

1A4B3 A custom-built display system, using two character-vector generators to drive twelve 5-inch CRT's, with twelve independent closed-circuit TV systems providing 17-inch displays in AHI offices and laboratories.

1A4B4 User-control input equipment, associating with each display a typewriter-like keyboard, a "mouse" (a screen-selection device developed by SRI), a five-key keyset (SRI-developed, for one-handed, chord-coded character input), and a footswitch (making visible, as special symbols, the normally non-visible space, tab, and carriage-return characters).

1A4B5 A special SRI-built interface, to service the display and input equipment with a minimum of burden on the CPU.

1A4C Description of Management-Research Developments:

1A4C1 The framework within which we view the research -- the organization to be managed, the "system" of management

to be studied and improved, etc.

1A4C2 The "ROMAN" (ROme MANagement) system of on-line aids for analyzing management data -- initial development, early in the project, on the CDC 3100.

1A4C3 Planning for the next stage of management-system research.

1A4D Appendix A describes the AHI Program, including a discussion of the bootstrap strategy and the notion of "augmented organization." An outline of existing text-manipulating systems is also presented. An account of AHI project support and a brief history of the management-system project completes Appendix A.

1A4E Appendix B is the programmer's manual for use of the special SRI-built hardware, giving memory and interrupt assignments, new computer instructions, and the means of controlling the disc, the display system, the console-input devices, and the line printer. It also lists the character set available for console keyboards, displays, and line printer, and gives the octal codes used (uniformly throughout the system) to represent them.

1A4F Appendix C describes the program used to help in the fabrication, checkout, and documentation of the SRI-built hardware. It operates on engineering specifications and generates:

1A4F1 Wire lists specifying the back-panel wiring connections to be done by fabrication technicians

1A4F2 Printout used to validate the correctness of the wiring

1A4F3 Printout structured to aid checkout and maintenance.

1A4G Appendix D gives a thorough description of the Tree Meta compiler-generation system, including syntax, program environment, a formal description, some detailed examples, a bibliography, the Tree Meta compiler written in the Tree Meta language, and supporting subroutines.

II -- SOFTWARE DEVELOPMENT

1 Special-Purpose Languages

1A The software effort for the 940 on-line system (NLS) has centered around the implementation of a set of special-purpose programming languages (SPL's). This collection of languages, originally known as the "Control Metalanguage," began as an attempt to formalize the "control dialogue" between users and our specialized system. A complete discussion of the original thoughts and motivation is given in Sec. III-3 of (Engelbart6). The SPL's are now a series of distinct languages, differentiated by radically diverse syntaxes, unusual data bases, and inherently different algorithmic-control or program-flow procedures.

1B The current thinking on final causes, standards for designing the manner of expression of languages, and functional aspects (operations on data) of the SPL's are quite different from what originally appeared in (Engelbart6). This evolution has been guided mostly by shifts of emphasis. Some changes, however, have been qualitative--they appear as major changes in the characterization of the SPL's.

1B1 Foremost in our design consideration is creating an interface between the computer (with all of its hardware and software) and individuals who are designing "user systems." To accomplish this, the SPL's must somehow embody the potential of the computer and existing software. They do this by being appropriate in their generalizations and discriminations (i.e., by having the right data and operations) and by reflecting this appropriateness in their syntax. The SPL's are designed to be clear (not necessarily obvious) to an experienced user. There has been no effort to make them manifest all of the inherent qualities of the machine and its system.

1B2 Designing the software foundations to mesh with the special-purpose languages tends to continuously influence the final organization of the system. This effect demands that the machine code supporting the system possess coherent underlying architecture. It also requires completeness (in command structure and input codes) and consistency, as the properties of any "user system."

1B3 Finally, the SPL's present us with a dimension of flexibility and user-system design experimentation never before possible. This flexibility is demonstrably unattainable without the SPL's. Even the most sophisticated macro assemblers and general-purpose languages lack the required "stylistic" properties.

1B4 It should be noted that the SPL's are neither self-documenting systems nor training systems.

1B4A As mentioned previously, the SPL's are to be clear but not obvious; it will take considerable background knowledge to even read them. The SPL's cannot be expected to resolve semantic ambiguity unless the reader has experience and familiarity almost on a level with the writer.

1B4B A listing of a user system programmed in the SPL's will not, in any sense, serve as a user's guide or a method of teaching the use of the system. An analogy can be drawn with linguistics, where one might attempt to use the theory of transformational grammars to teach the use of English. Transformational theory exhibits a philosophical structure of sentences and the potential for change in the sentences, but it says nothing about the application or effectiveness of the sentences. It is necessary first to master English. Having done so, one may gain insight through transformation theory; and having mastered a user system, one may gain insight through the SPL's and the system specifications written in the SPL's.

1C One of the SPL's is concerned primarily with the syntax and interpretation of the dialogue between a user and the computer system. The rest are concerned with the execution of the interpretation on the data bases.

1C1 The dialogue syntax and interpretation SPL began as an attempt to formally define the control-language dialogue of user systems.

1C1A In the first attempt, the control language was described as a finite state machine, and the language allowed a formal textual definition of such a machine. A program written in the control language accepted input characters from the work station, gave immediate feedback about each character and the current control state of the system via the display, and moved on to the next state, in the restricted context characteristic of such machines. A more complete discussion of this approach appears in Sec. V of (Engelbart7).

1C1B It was originally thought that such an approach was adequate for the definition of user-system control languages. But, to paraphrase John McCarthy, the model is metaphysically adequate, but epistemologically inadequate. Implementation revealed that the dialogue is a non-Markovian (nonstochastic, historically dependent) process on the part of both the machine and the user, and accurate characterization as a finite state machine results in so many states that the model is useless. A better model is a two-stack automaton with a small number of immediate-access storage registers. Such an automaton does the following things:

1C1B1 Accepts input characters from a work station

1C1B2 Stores, in its first stack, either the input characters or a variety of information about its current state

1C1B3 Stores, in its second stack, specification information derived from the characters -- e.g., precisely which character in a file the bug is pointing to

1C1B4 Places other information in its storage registers

1C1B5 Sends immediate feedback to the work station about its current control state

1C1B6 Moves to its next state on the basis of decisions influenced by either the first stack or any of the storage registers. It may even branch to a state whose address it fetches from the first stack.

1C2 The linear-content-analysis or string-analysis SPL has been called the "pattern-matcher" in previous reports. Programs written in this SPL (1) detect entities used in editing or (2) compute Boolean functions of the presence of entities used in display filtering.

1C2A The entities detectable within the SPL are characterized by the delimiting characters or strings around them. For example, a word might be defined as a string of printing characters surrounded by blanks or punctuation. Once an entity is detected, the delimiting positions can be stored for future use, and general flags may be set to indicate whether the entity has been found. The algorithmic nature of the language closely resembles the procedures used in top-down analysis with backup. The syntax of the entity specification closely resembles a single BNF rule, with extensions to permit either left or right scanning, to reposition the scan to previously stored positions or to specifications from the second stack of the automaton, and limiting quantification of an iteration operator.

1C2B The display-filter aspect of the language embeds the entity-detection constructs in the context of general Boolean functions, with the extended ability to perform unanchored or "slide" searches.

1C3 The string-construction SPL is straightforward; it permits the construction of new strings on the basis of (1) entities from the linear content-analysis pointers, (2) the contents and format

of the display through the specification stack, and (3) literal strings. Statements of the SPL are concatenation rules. The only control is an "if-then-else" construct in which the Boolean expression is a linear content-analysis statement, or an order relation on the absolute positions of previously stored pointers.

1C4 The structure-manipulation SPL is concerned with the construction of a "ring," whose elements point to the individual statements created by the string-construction SPL.

1C4A Each ring element contains, among other things, a pointer to another ring element which points to the successor statement (if any), a pointer that points to the sublist statement (if any), and flags to indicate whether this ring element is the first or last statement at a given level.

1C4B The structure-manipulation language contains a number of constructs designed for searching and testing the ring and for linking up ring elements to form the familar structured-statement format.

1D These four SPL's are not a complete system. There are portions of the system being implemented that should be handled through SPL's but are not. We intend someday to extend the SPL's as follows:

1D1 There should be an encoding language in which to specify the mapping of raw input data from the work stations into the character or macrocharacter format used in the dialogue syntax-interpretation SPL.

1D2 The format of the display and the selection of information displayed should also be handled with a special-purpose language. While the formatting of the display presents no real problems, information selection will require a lot of work. We must somehow make explicit the way in which HOP and JUMP commands will work, and turn the sequence generator into a special-purpose language.

1D3 Finally, there is the graphics SPL. The syntax of the graphics part of the system should be part of the control-dialogue SPL. The graphics SPL is in a sense equivalent to the linear content-analysis and string-construction SPL.

1E A thorough technical report on the SPL's, and detailed programming documentation on the supporting library, are being prepared.

II -- SOFTWARE DEVELOPMENT

2 Tree Meta

2A Developing the SPL's is a dynamic trial-and-error process. The development could not proceed at a viable pace without a compiler-generation system. The fact that the SPL's are unspecified, and much of the nature of their syntax is unforeseeable, means that the compiler-generation system must also be dynamic. The pliability of the Tree Meta system has amply sustained the evolution of the SPL's.

2B Tree Meta is a compiler-compiler system for context-free languages. Parsing statements of the metalanguage resemble Backus-Naur form with embedded tree-building directives. Unparsing rules include extensive tree-scanning and code-generation constructs. Appendix D describes the Tree Meta system in detail. Thorough implementation documentation for the SDS 940 appears in the discussion of the support subroutines. A history of computer metalanguages and a tutorial guide to Tree Meta are other topics of Appendix D.

3 The NLS Library

3A The unusual nature of the SPL's means that their operations and data bases are rarely reflected in the arithmetic-oriented commands of the computer. Thus, programs written in the SPL's compile mostly into subroutine calls, and the background of subroutines is called the NLS Library. The problem of choosing the appropriate subroutines is the problem of building the right SPL's.

3B The other design problem of the NLS Library is integration into the time-sharing system so that it performs smoothly, quickly, and reliably. Considerations, progress, and results in this area are discussed in (Rulifson1).

4 MOL

4A Within the SPL/Tree Meta/NLS Library system, it is a formidable task to ensure that the library will remain readable and comprehensible, while retaining fast, tight code. The solution to this problem is the Machine-Oriented Language (MOL). Developed specifically to support the NLS Library programming effort, MOL940 combines the explicitness of assembly language with the phrase structure of ALGOL-like languages. Since it allows constructs that reference the 940 registers, reflect the 940 instruction set, and smoothly interface with the TSS monitor and executive, it makes assembly-language coding unnecessary. On the other hand, the structure of the language--with blocks and if, for, while, and case clauses--significantly improves the readability of the language and drastically reduces the number of program labels.

4B A detailed justification of the development of the MOL is given in Sec. II-3-d of (Engelbart6). A technical report on the details of the language is in preparation (Hayl). The system has been received enthusiastically by other SDS 940 users and is currently being used by them. Detailed documentation on the compiler itself is available to those interested in implementing the MOL in their 940 systems.

5 SPL's and the On-Line System

5A The SPL approach to system design may now be seen by viewing the system first in terms of the functions it is to perform, and then in terms of its organizational characteristics.

5B When considered as an information-processing system, NLS 940 has the four basic levels of operation shown in Fig. II-1.

5B1 The 940 monitor receives bits, characters, and interrupts from the work station. It compiles these, along with internal clock readings, into compact words for further analysis by NLS.

5B2 The input encoder immediately translates this information into characters for future work. Simultaneously, it sends immediate feedback to the work station in the form of an echo register of characters on the display screen.

5B3 The characters, button settings, and screen-position pointers are then passed on to the dialogue-interpretation phase. Processing proceeds from this point on by normal time-sharing programs. The characters are first processed by the two-stack automaton. Feedback about the control states of the system is sent to the work station. Characters are compiled into strings, absolute screen-pointer positions are translated into character positions within the document, and subroutines of the other parts of the system are selectively executed.

5B4 Finally, textual editing and structural changes are performed on the old document. The display generator selectively chooses statements from the revised document on the basis of the following:

5B4A The new content and structure of the document

5B4B Previous status of the display screen

5B4C The command just executed

5B4D Certain global parameters (level and truncation).

5C The overlay structure of the system only vaguely resembles this

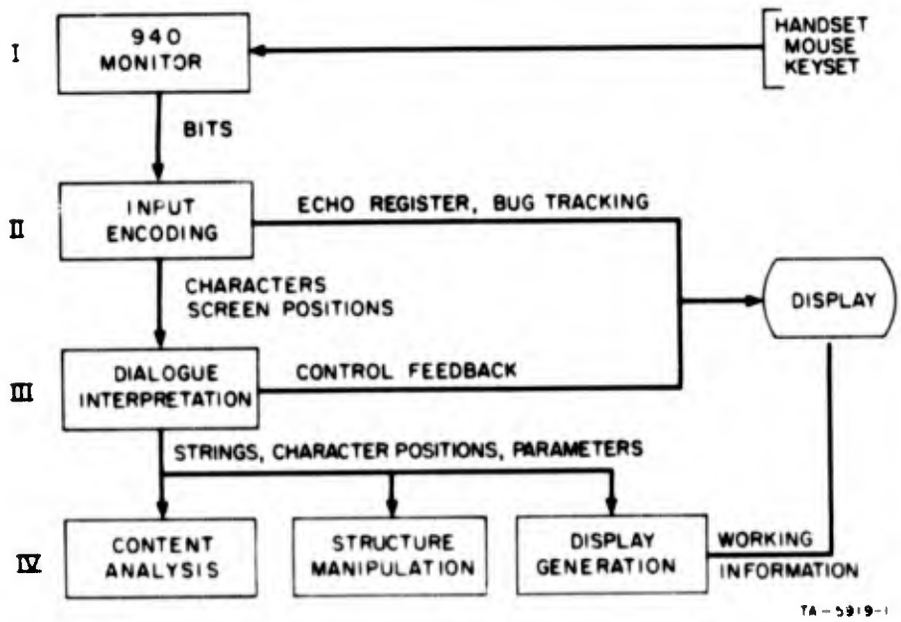


Figure II-1 Functional Organization of NLS 940

information-processing view. The actual positioning of routines in the overlay structure has been guided by a number of factors, including frequency of use. Since the entire system is in reentrant code and the overlay processor allows subroutines to overlay themselves and run properly, normal parameter and information-communication standards may be broken.

SC1 Figure II-2 is shaded to show the languages in which each overlay is written. The size of each box is proportional to the number of instructions in the corresponding overlay; the amount of shading is prcportional to the amount of code of each type.

SC2 The content analyzer is an extension of the metacompiler special-language approach. The real-time content analyzer takes statements directly from the document currently being worked with and produces immediately executable code which, when combined with the utility package, analyzes statements for display-creation routines. This small on-line compiler is written in an intermediate language specially designed for this purpose. The intermediate compiler is, of course, written in Tree Meta.

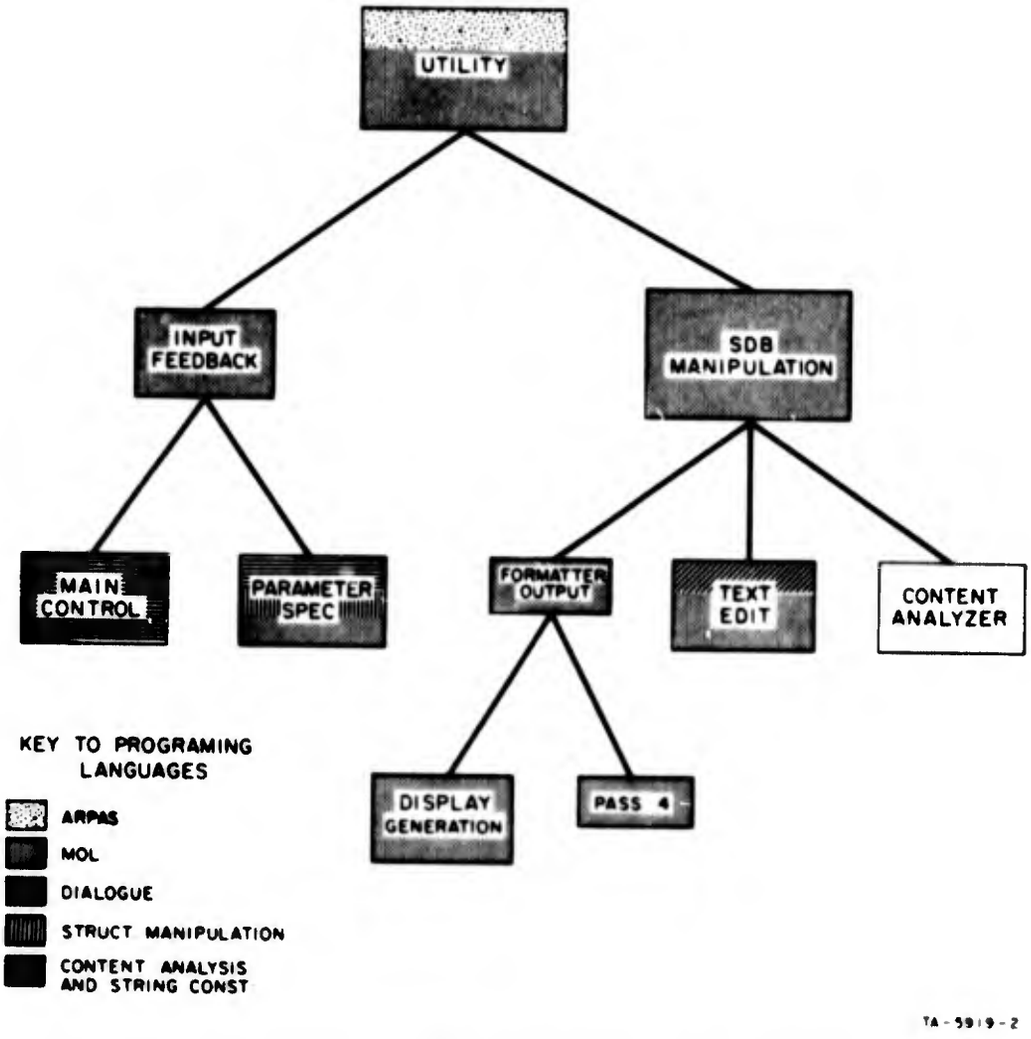


Figure II-2 Overlay Structure of NLS 940

III -- HARDWARE DEVELOPMENT: 940 Computer Facility

1 The 940 Computer Facility

1A The 940 computer was selected primarily because of the sophisticated time-sharing software that was available on it.

1A1 The CDC 3300 (with time-sharing hardware) was also carefully evaluated, since it is program-compatible with the existing text-handling system on the CDC 3100.

1A2 It became apparent very soon in our examination that for the size of the facility we were considering, the 940 was the only machine with really effective time-sharing software. For any other computer, we would have had to develop whatever software was necessary for serving multiple display consoles.

1A3 Going to the 940 involved completely reprogramming the text system, but this system was due for redesign in any case. The work on it was much more in line with the research goals of the program than expending our efforts in learning how to write time-sharing software.

1B An order for the 940 computer was placed in October 1966, contingent upon the additional funds requested for this project.

1B1 The computer was delivered in June 1967. The acceptance period began on 5 July and the computer was finally accepted on 5 August 1967. In general, the facility has operated very well, particularly when compared with some other 940 installations that we have knowledge of. Some difficulties that have been encountered since acceptance are worthy of mention.

1B1A During the first few months of operation, there was considerable difficulty with the tape drives, which were manufactured by Wang and distributed by SDS. SDS engineers did not really understand the adjustment and maintenance of these drives.

1B1B When the facility was first delivered, the circuitry for direct connection of Teletypes to the Teletype coupling equipment was not really understood.

1B1C For a period of about one month, in November, we had continuing difficulty with memory parity errors. The errors were relatively infrequent and did not occur when SDS diagnostics were used. This was very costly to us, both in lost computer time and in time spent helping SDS engineers track down the difficulties.

1B1D The Memory Interface Connection (MIC) delivered with our

facility was really a prototype and not a production unit. Documentation delivered with it was not adequate and SDS engineers did not understand its operation. We spent considerable time in checking out our equipment, debugging the MIC. and bringing its documentation up to date.

1C The present facility is somewhat changed from that originaly proposed. Fig. III-1 is a block diagram of the current configuration, and the following is a summary of its major components:

1C1 A central processor with time-sharing hardware.

1C2 Four 16K memory banks with word length of 24 bits and cycle time of 1.8 microseconds:

1C2A Each memory bank contains provisions for a second port to memory through a Memory Access Module (MAM).

1C3 Communications equipment for connecting to 16 Teletypes:

1C3A Adaptor cards designed at Bolt, Beranek and Newman are available for connecting these to dataphone terminals.

1C4 Three magnetic tape units, 75 ips, 800 bpi.

1C5 A paper-tape station with 8-level reader and punch.

1C6 Three RAD storage units:

1C6A These are drums, each with a capacity of 500,000 24-bit words.

1C6B The RADs connect to the second memory buss through a Direct Access Communication channel (DACC).

1C7 A Memory Interface Connection (MIC) through which the special peripheral equipment is interfaced.

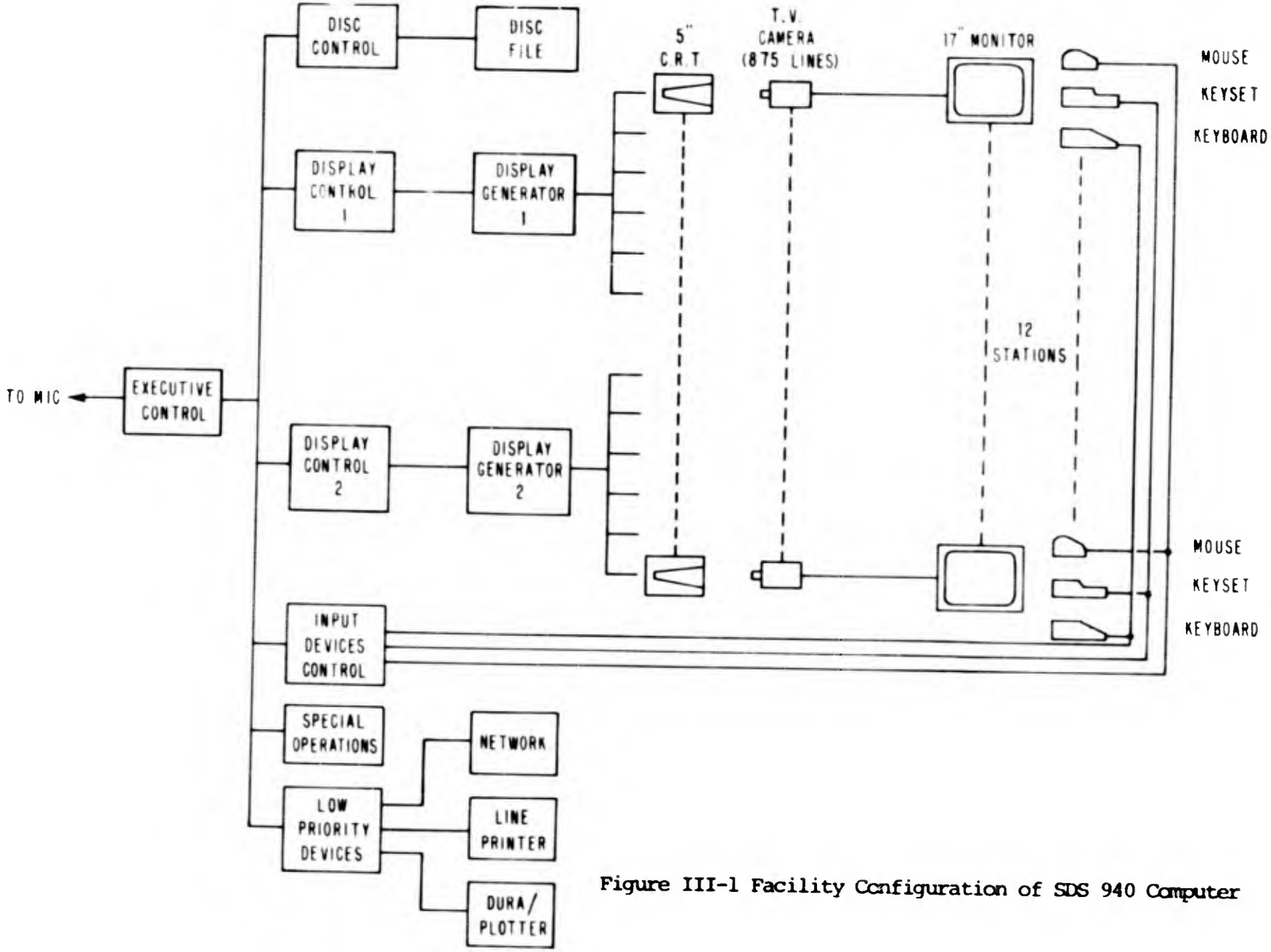


Figure III-1 Facility Configuration of SDS 940 Computer

2 Other Facility Equipment

2A In addition to the SDS equipment, the facility includes peripheral equipment of other manufacturers and a considerable amount of equipment designed and constructed by SRI.

2B All of the non-SDS equipment is interfaced to the computer through the Special Devices Channel which connects to the second memory buss through the MIC.

2B1 The decision to construct the special devices channel rather than to try to interface the equipment through regular SDS equipment was made primarily for the following reasons:

2B1A The SDS I/O channels (both the DACC and the data multiplex channel) did not offer the specific features needed.

2B1A1 It was apparent that in trying to provide display service to users the major limitation would be CPU time available for processing user requests. The Special Devices Channel (SDC) is as automatic as possible in its access to memory, and wherever possible, data is formatted so as to minimize executive time for processing to users.

2B1B In addition, the SDS channels were not thoroughly checked out as production items and might have given us considerable trouble.

2B2 The 940 computer includes a variable priority feature for memory access, which is critical in allowing the connection of this much high-speed equipment to the second memory buss.

2B2A The equipment on this buss (not including the proposed special operations equipment) will require a maximum data rate on the buss of approximately 264,000 words per second or 1 out of 2.1 memory cycles.

2B2B The variable priority scheme operates essentially as follows:

2B2B1 There is a "wired-in" priority for all devices connected to the second buss, but any device may request memory access at one of two priority levels.

2B2B2 A device requesting high-priority access takes precedence over the CPU, any device requesting low-priority access, and any device with a high-priority request which is below it in the wired-in priority. Ignoring the special operations equipment, the order of priority access is as follows:

RAD--High
Disc--High
Display 1--High
Display 2--High
CPU
RAD--Low
Disc--Low
Display 1--Low
Display 2--Low
Input Devices Controller
Low-Priority Controller.

2B2C This variable-priority scheme allows a device to request memory access at low priority when immediate access is not required, and to switch to high priority when obtaining a memory cycle is critical.

2B2C1 For example, when transferring data, the RAD needs one of every four memory cycles. The RAD controller can therefore make three memory requests at low priority. If none of these is successful, it can then request high priority access and be assured of getting the next cycle (since it is the highest priority device on the second buss).

2B2D The variable priority scheme is discussed by Melvin Pirtle (see Pirtle1). With his help, simulations were run on our particular configuration.

2B2D1 Results showed that we could expect less than 1 percent degradation in CPU performance due to the loading on the second memory buss.

2B3 A basic design feature of the channel is that all controllers operate from a fixed core addresss with no provision for transmitting addresses by EOM-POT instructions.

2B3A This fixed address is a "Unit Reference Cell" for each controller (except the Input Device Controller) that serves as the communication between the controller and the program.

2B3A1 The Unit Reference Cell serves two functions:

2B3A1A It contains the address of the command table or buffer in core that the controller is processing. This address is set by the program when controller operation is initiated (by an EOM) and is updated by the controller

as each command or buffer is successfully processed.

2B3A1B It also contains an error code, written by the controller when any error is detected in the processing of a command or buffer.

2B3A2 This method is good for system reliability, since the Unit Reference Cell will always indicate the nature of any error and the address of the command or buffer being processed when the error occurred.

2B4 Another advantage realized through the design of the hardware is a uniform character set.

2B4A The line printer, displays, and keyboards all have characters and codes matching the internal character set of the timesharing system. (The character set is described in Appendix B.)

2C The equipment connected to the Special Devices Channel is shown in block diagram in Fig. III-2. The major components and their functions are discussed below.

2C1 Executive Hardware

2C1A This is essentially an electrical interface to the SDS MIC and a multiplexer that allows asynchronous access to core by any of the six controllers connected to it.

2C1B The executive hardware decodes computer EOM instructions for program control and SKS instructions for sensing the status of the various controllers.

2C1C It accepts addresses and requests for memory access from the controller, determines relative priority among the controllers, and synchronizes the requests and passes them along to the computer via the MIC.

2C1D It contains circuits to generate parity on data transmitted to the computer, or it can pass along a parity bit generated by a controller. On read operations, parity is not checked, but is passed along to the individual controllers.

2C1E It also synchronizes interrupts from the various controllers and passes them along to the computer.

2C1F Included in the executive hardware are comprehensive debugging and monitoring aids.

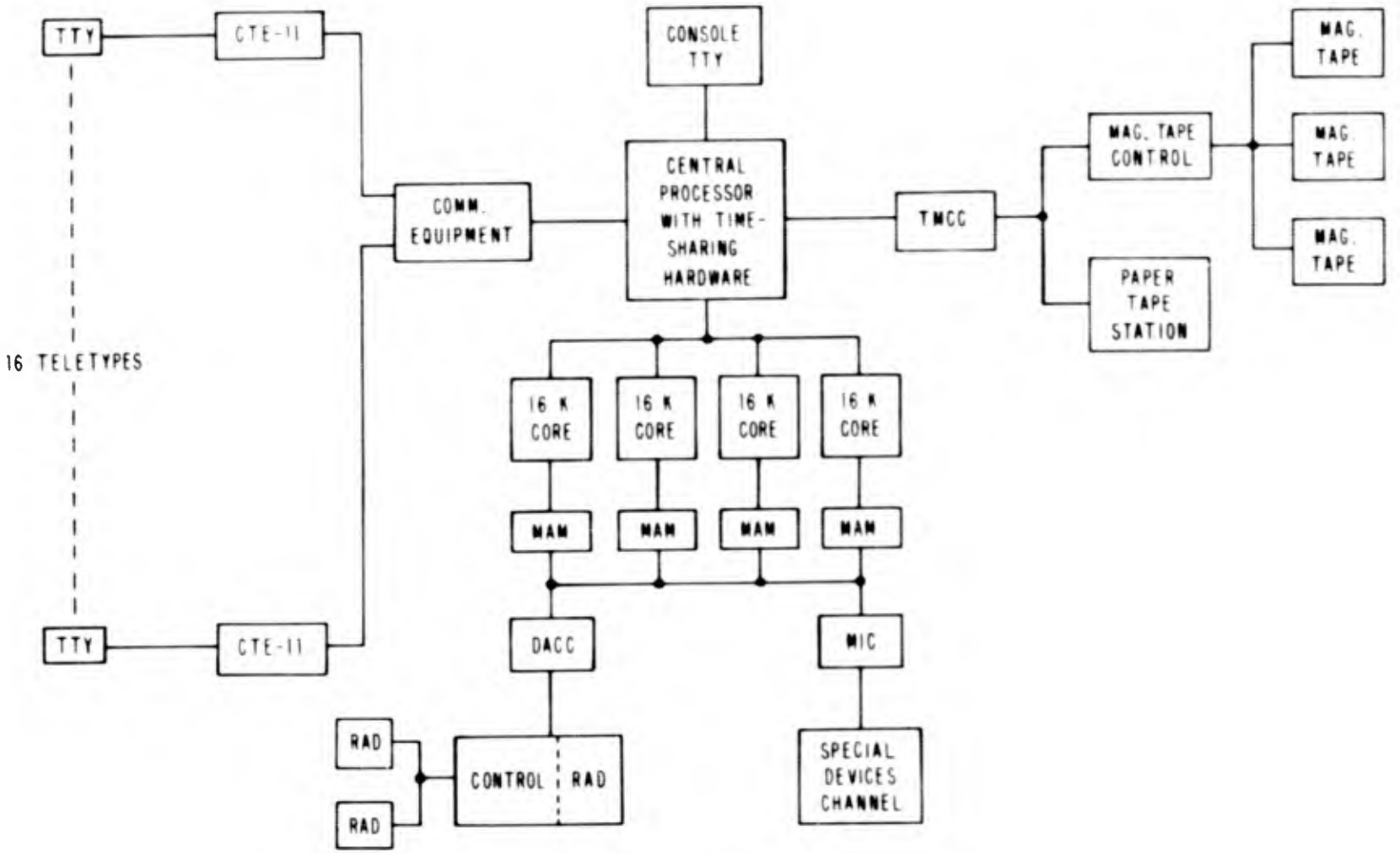


Figure III-2 Special Devices Channel

2C1F1 A debugging panel contains indicators that can be switched to any of the six controllers to monitor data, addresses, priority, and other functions (see Fig. III-3).

2C1F2 Any of the six controllers can be switched "off-line" and operated from the debugging panel with simulated data entered by hand in the monitor registers.

2C1F3 A switch is provided for each of the six controllers that will operate the controller in a step mode (one memory access per step) either on line or off line.

2C1F4 Two meters are also included on the panel which will indicate for any selected unit or the entire channel the percentage of memory cycles being requested and the percentage of memory cycles actually served.

2C2 Disc File

2C2A The disc-file system will consist of a Bryant Model 4061 disc file and a disc controller.

2C2A1 The disc file is an "A" frame with a capacity of twelve data discs. As delivered, the file will contain six data discs and will be expanded in the field as the additional capacity is needed.

2C2A1A With the initial six discs the capacity will be about 32 million 24-bit words.

2C2A2 The disc controller is being designed and built by Bryant to interface with the executive hardware. Specifications for the controller were developed jointly by Bryant, Project GENIE at Berkeley, and SRI, It includes features for reading and writing across page boundaries and extensive error-checking provisions (for details on this and other devices on the channel see Appendix A).

2C3 Display System

2C3A The display system consists of two identical subsystems each with display controller, display generator, six CRT's, and six closed-circuit television systems.

2C3B The display controllers have been designed and built at SRI. They process display "command tables" and "display lists" that are resident in core, and pass along "display buffers" containing instructions to the display generators.

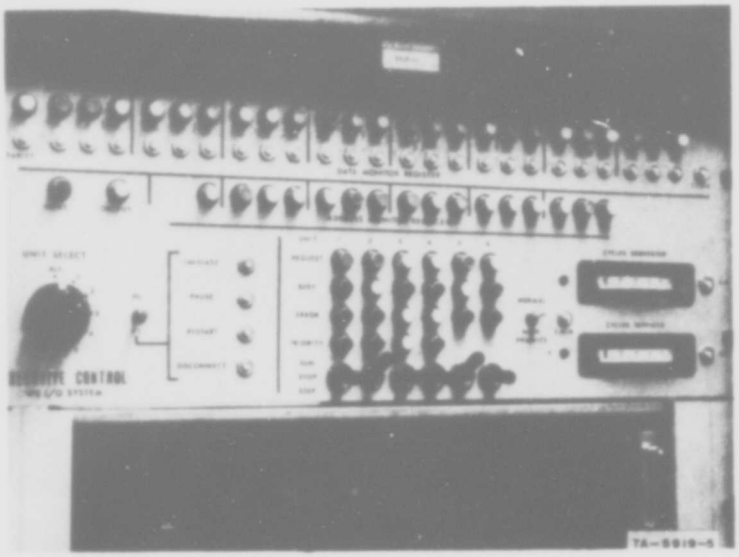


Figure III-3 Executive Control Debugging Panel

2C3C The display generator and CRT's are being purchased from Tasker to SRI's specifications. Each will have general character and vector capabilities. They will accept instructions (beam positioning, character writing, etc.) from the controller.

2C3C1 They drive in parallel six 5-inch high-resolution CRT monitors on which the display pictures are produced. Presentations for each of the CRT's are generated sequentially and unblank signals from the display controllers select one or more of the monitors at a given time.

2C3C2 Character-writing time on the displays will be approximately eight microseconds, allowing an average of 1000 characters on each of the six monitors when regenerating at 20 cycles per second. Because of image retention in the vidicons, and video reversal (see below), it is possible to regenerate at this low rate without annoying flicker.

2C3D A high-resolution (875-line) closed-circuit television system will transmit display pictures from each CRT to a television monitor at a corresponding work-station console.

2C3D1 It was originally proposed to implement a display system consisting of two 21-inch computer-refreshed computer displays and ten storage-tube display stations.

2C3D2 This system was abandoned because the Tektronix high-resolution storage tubes that were to be an integral part of it were not available. They are still not available in production quantities.

2C3D3 The present system was the most reasonable alternative for providing display service to individual offices in the building. It has both advantages and disadvantages.

2C3D3A On the positive side, refreshed displays will be provided to users with the capability of much more dynamic presentations than would have been possible with storage tubes. In addition, with the television system, it is possible to invert the video to provide a black-on-white display. This presentation is usable in higher ambient light conditions than would be possible with direct-view CRT's, and subjectively reduces the effect of flicker due to low regeneration rates.

2C3D3B The most significant disadvantage is that more computer memory must be allocated to regenerate the displays. (The television system also was somewhat more expensive to develop than was expected with the storage tubes.)

2C4 Input-Devices Controller

2C4A In addition to the television monitor, each work station console will have a keyboard, binary keyset, and mouse.

2C4B The state of these devices is read by the Input-Devices Controller (IDC) at a preset interval (about 30 milliseconds) and written into a fixed location in core.

2C4B1 Bits are added to information from the keyboards, keysets and mouse switches to indicate when a new character has been received or a switch has changed state during the sample period. A new character or switch change causes an interrupt to be issued at the end of the sample period.

2C4B2 Mouse coordinates are digitized by an A-to-D converter and formatted by the Input-Devices Controller as beam-position instructions to the display generator. provisions are made in the display controller for including an entry in the mouse position table as a display buffer. This allows the mouse position to be continuously displayed without attention from the CPU.

2C4B3 The IDC also has provisions for turning on or off up to twelve binary signals to each console for control of bits set in an output area of the fixed table.

2C4B3A These output signals may be used to control either audio feedback to the work stations or visual indicators (other than displays).

2C5 Special Operations

2C5A The box with the label "Special Operations" in Fig. III-2 is at this time only a provision in the executive hardware for the addition of a high-speed controller. Tentative plans are for adding special hardware here to provide operations not available in the 940 instruction set, such as character string moves or string pattern matching.

2C6 Low-Priority Devices Controller

2C6A This controller accommodates three devices with

relatively low transfer rates.

2C6A1 At this time only the line printer is implemented. This is a Potter Model HSP-3502 chain printer with 96 printing characters and a speed of 230 lines per minute.

2C6A2 Provisions have been made for adding an on-line selectric typewriter, a graph plotter, and a terminal for the proposed ARPA computer network.

3 Work Stations

3A Each actual work station consists of a 17-inch television monitor, keyboard, binary keyset, and mouse.

3A1 The keyboards were manufactured by Friden. They have 61 keys including spacebar, and have the "system" coding.

3A2 The keyset and mouse were developed at SRI and have been used with the 3100 system for some time (see English1).

3B Signals are transmitted to and from the work stations by cables routed through the building ceiling.

3C Experimental physical configurations have been built for various office and conference room situations. Figure III-4 shows a console in a typical office.

4 Wire List Program

4A In the development of the hardware for the facility this project contributed to the development of some computer aids for producing wire lists to direct technicians in the construction of the logic units.

4B This wire list program was also supported by other projects at SRI. It is described in detail in Appendix C.

Figure III-4 Display Console

IV -- MANAGEMENT SYSTEM RESEARCH: Introduction

1 Introduction

1A Our concept of the "system" for doing management work (the system whose study and improvement this project undertook) has undergone a steady evolution, particularly in the last few months; it became clear that we were not likely to have available our new facility for testing computerized management aids under this project, and we were led instead to do more basic studying and planning.

1A1 The kind of computer aids developed in the early part of the project for the CDC 3100 -- as described in Sec. IV-3 below -- are an important part of what we now see as the "system." But the study and considerations outlined in Secs. IV-2 and IV-4 of this section led us to plan for a somewhat different approach for the next stage of our research from that taken initially.

1A2 This research project is experimenting with the management of an organization consisting principally of "knowledge workers" (i.e., those who apply information to specific work and generate information for others) working together in an environment that includes sophisticated and highly interactive on-line aids (representative of many important organizations of tomorrow).

1A3 In the last few months we have become aware that it makes little sense to consider a "management system" as including only administrative procedures, conventions, and aids. The "governmental" aspects of an organization of "knowledge workers" are now recognized as key problems demanding our attention, within the concept of a "management system." These aspects include the processes by which goals are established and conflicts of attitude, opinion, and special interests are resolved.

1A4 These governmental problems involve a broad range of organizational questions. In the Bootstrap Community, it is clear that new and quite different forms of human organization could evolve--e.g., teams of individuals working extremely closely, requiring new procedures, protocol, methods, skills, and attitudes to provide the stable working environment that better couples human capabilities into coordinated efforts.

1A4A For instance, the monolithic pyramid of responsibility and decision control will undoubtedly change as a consequence of a higher degree of personal cooperation. Via the "instant mobility" of their individual display stations, people could rally together for a brief cooperative effort, and minutes later regroup in other ways for another set of problems.

1A4B Alternatively, they can easily keep an eye on an on-going dialogue (as developed in the common computer-held record

through which the dialogue communication is taking place) to enter into it when they feel they can contribute.

1A4C These factors make possible much more flexible means for integrating individual contributions of judgment, decision, knowledge, suggestion, etc., and provide an opportunity to develop faster, more powerful responsiveness to changes in organizational opportunities, problems, goals, etc.

1A5 To evolve towards the new organization, with its different roles and interpersonal working relationships, is now seen to be a fundamental and critical part of this project's approach.

1A5A Our current plans for launching the second stage include initial development of basic on-line aids valuable for certain management tasks.

1A5B However, we will also give consideration to the development of those aspects of the management system that will contribute to our understanding of "best" Bootstrap Community organization. This work may concentrate on the establishment of attitudes, techniques, and practices merely for "working closely" with one another.

1A5C This split suggests a logical classification of work into "administrative"-related tasks and "organization"-related tasks.

1B Contents of the Rest of this Section of the Report

1B1 A framework describing management research is presented first.

1B1A This framework introduces the notions of "task" and "activity," describes the current project activities, and discusses the concept of "experimental organization."

1B2 A detailed, technical description of a system of on-line management aids, implemented on the CDC 3100, follows this framework.

1B3 A description of general plans completes this section.

2 Framework for Management Research

2A The uniqueness of the AHI Bootstrap Community necessitated an unconventional scheme for describing work units in the project. The definition of this scheme has been a part of the development of our management system.

2A1 The AHI project has been mapped into a set of "Activities." Each Activity provides a clear scope of interest to the separate project sponsors. Actitivies are the prime work-entities in the Bootstrap Community.

2A1A Each Activity is given certain specific responsibilities, is set up as a separate cost center, and is allocated resources and authority as appropriate for its goals.

2A1B There are two basic types of Activity -- "action-oriented," and "research-oriented." The relevance of these terms will be described below in the discussion of current Bootstrap Community Activities.

2A2 Day-to-Day work can be described in terms of "tasks." For example, the programming for the SDS 940 graphics package is a "task."

2A2A Each task will tie specifically to one or more activities, with task specification, resource allocation, supervision, etc., stemming directly from the plans, resources, and responsible staff within the Activities.

2B The current Bootstrap-Community Activity breakdown is as follows:

2B1 The Executive Activity -- with responsibility for overall planning and coordinating for the Bootstrap Community.

2B2 Service-System Activity -- with responsibility for the design, implementation, and operation of the hardware-software system which provides service to the users at their respective consoles.

2B3 User-System Activity -- with responsibility for coordinating the repertoire of service functions; the control-dialogue procedures with which the user controls the execution of these functions; the terminology and conventions for specifying and documenting the functions and control-dialogue procedures; and the development and consistent application of the design principles for these user-system features.

2B4 The Network-Information-Center Activity -- with

responsibility for designing, implementing, and operating the Information Center that is to serve the forthcoming ARPA computer network.

2B5 Miscellaneous Activity -- the catch-all for miscellaneous, small activities, and for activities that are tentatively being developed towards becoming full-fledged Activities (e.g., development and operation of our X-DOC (external documentation system), etc.).

2B6 Programming-System Activity -- with responsibility for coordinating and improving the principles, conventions, techniques, computer aids, methods, etc., for doing the computer-programming tasks within the Bootstrap Community.

2B7 Engineering-System Activity -- with responsibility for coordinating and improving the principles, conventions, techniques, computer aids, methods, etc. for doing the engineering tasks within the Bootstrap Community.

2B8 Management-System Activity -- with responsibility for coordinating and improving the principles, conventions, techniques, computer aids, methods, etc., for doing the management tasks within the Bootstrap Community. This includes the principles, conventions, techniques, etc. necessary for the establishment of a stable organization of people working closely together.

2C There are two basic types of Activity

2C1 The first five Activities -- Executive, Service-System, User-System, Network Information Center, and Miscellaneous -- are "action-oriented." This means that these Activities are directly concerned with day-to-day operations within the Bootstrap Community. For instance, the Service-System must provide service, at the consoles, for each member of the Community.

2C2 The last three Activities -- Programming-System, Engineering-System, and Management-System -- are "research-oriented" in the sense that they "back up" other Activities, providing the (improved) means for performing other activities. For instance, the Management-System Activity provides the framework, principles, aids, procedures, etc. used by, say, the Executive Activity, in the actual management and government of the Bootstrap Community (noting that this Executive Activity is thus truly "action-oriented").

2C2A Resources allocated to a specific research Activity are used for the coordination of that system's evolution, for

studying, analyzing, designing, and implementing new system features, or for training Bootstrap Community personnel in the use of that system.

2C3 It is thus possible for a research Activity to sponsor a task that will contribute to the performance of either (or both) action-oriented or research-oriented Activities. It is possible for, say, techniques developed as a part of the Management-System Activity to be used by the Executive Activity to manage further technique development in the Management-System Activity. The circularity in this procedure is at the heart of the Bootstrap strategy.

2C4 "Tasks" cut across Activity boundaries, and may be simultaneously within more than one Activity designation. For instance, programming for the development of a graphics package for the SDS 940 is a task that may contribute simultaneously to the Programming System Activity (e.g., by suggesting a new programming technique), the Engineering-System Activity (e.g., the graphics package may be useful as a part of an on-line hardware debugging system).

2C5 The two-dimensional scheme of Activities and Tasks does not perfectly describe work within the Bootstrap Community. Part of the task structure is the redefinition of this conceptual scheme, and this work will proceed as a part of the Management-System research.

2D General Characteristics of the Management-System Activity

2D1 It is to be assumed that management tasks will be executed continually throughout the various levels and pursuits of each Activity. The following are examples with administrative, rather than organizational (i.e., "governmental") significance.

2D1A The planning, scheduling, assignment, and supervising of money and personnel resources comprise a large and important set of these tasks.

2D1A1 As indicated by the breakdown of Activities, engineering and programming comprise very important pursuits within the Community, and each has an Activity charged with coordinating and improving the system of doing its types of tasks.

2D1A2 But consider that, in the execution of any reasonably large engineering or programming task, there will actually be a mixture of management and technical (i.e., engineering or programming) tasks involved.

2D1A3 It is not clear at this point what practical differentiation should be made between what is considered to be a management task and what is a technical task. Consider, for example, the principles employed in planning, assigning, and reviewing the allocation of resources as involved in (obvious) management tasks. There is a good reason to assume that these principles will have meaningful and fruitful application to the way a programmer (for instance) sets up his time and tasks for the week ahead.

2D1B Recording and accountant-type analysis of resource expenditures represent another set of common and important type of management tasks.

2D1B1 Here again, we would find these tasks widely distributed throughout the Community, and potentially applicable even for fragments of an individual's activity.

2D1B2 For instance, it is an appealing possibility to develop flexible and accurate means for monitoring and recording the expenditures of a person's time towards different tasks. This would be valuable not only for regular operational analysis, but for supporting the Management-System research by helping to study better the dynamics of organizational activity.

2D1C We expect continual clarification and differentiation of categories of "management tasks" throughout the project.

2D2 The responsibilities of the Management-System Activity will be:

2D2A Management-System Development

2D2A1 Establishing and keeping current a description (perhaps "model") of the existing management system.

2D2A2 Collecting and analyzing needs and possibilities for improvement

2D2A3 Developing a framework and methodology for evaluating and selecting needs and possibilities for investing next-available resources (or for recommending special-allocation attention)

2D2A4 Implementing modification/additions to the Management system to fill selected needs or to implement selected possibilities.

2D2A5 Convincing Bootstrap-Community staff that the system or its new changes are worth using

2D2A6 Developing tests or performance-evaluation means to ascertain a user's proficiency with the system

2D2A7 Setting standards or proficiency in system usage -- perhaps with "levels" of proficiency according to types of usage

2D2A8 Educating and coaching users, to develop their proficiency.

2D2B Activity Self-Improvement

2D2B1 Provisionally to develop systematic aids for doing its own work better, in areas not within another Activity's responsibility.

2D2B1A Each such pursuit will require explicit approval of the Executive Activity -- because the Community as a whole normally has the "right" to benefit directly from all products of this activity, and also because any given new user aid might be better specified and developed in cooperation with other Activities.

2E The "Experimental Organization"

2E1 There is a subtle distinction between administrative aspects of the Management-System Activity, and questions of organization and "government."

2E2 The Bootstrap Community will begin to evolve its own form of Management System (including both "administration" and "organization") via an "empirical" approach.

2E2A A fundamental requirement is to have the "experimental organization" aware of and be practiced users of the best general "management practices" currently known -- whether or not these practices involve computer aids.

2E3 We feel that an early emphasis on "good management" will soon give way (in importance) to the notion of "good government" as the augmented community grows in complexity.

2E4 One of the most pressing problems is that of changing attitudes of Community members from wholly scientific orientation towards a more general concern for human effectiveness. Such ingredients as attitude, habit, skill, and experience

fundamentally affect the effectiveness of an organization, and it seems clear that we have to pay attention to these in the "experimental organization."

2E4A For instance, to allocate resources effectively requires serious concern for the relative value of the various results associated with the alternatives.

2E4B For this one particular practice to become integrated into one's working life as an effective feature of management (whether for managing one's personal endeavor or that of a larger organizational unit), there has to exist both the attitude-motivation set that brings a person to give this working technique a real try, and the experience and feedback that develops intuition, judgment, and skill in its application.

2E5 The broader concept of the management system (i.e., including the organizational issues) embraces many other problems. These are basically concerned with the establishment of a stable configuration for the Bootstrap Community -- a form that is "best" able to allow the Community to grow in problem-solving capability.

IV -- MANAGEMENT SYSTEM RESEARCH: The ROMAN System

3 The ROMAN System for the CDC 3100

3A General

3A1 The ROMAN management-aid system was implemented on the 3100 with two objectives in mind. These were to gain experience in using an on-line system for actual management data and to uncover some of the problems in implementing such a system and in expanding NLTS to handle this kind of data.

3A2 In selecting a test area for implementation, we chose the analysis of the status-report records of our resource expenditures as a reasonably specific and well-defined problem that would meet these needs.

3A2A The objectives were as follows:

3A2A1 Generate a data base in computer-readable form that would contain updated records regarding resource status for projects in the area of our management concern.

3A2A2 Provide the manager with general ways of calling for summary analysis of these data in tabular or graphic form.

3A2A3 Allow simple mathematical operations such as summing, averaging, or converting man-hours to dollars.

3A3 While not all of the objectives were achieved, due to the changes described under 1b in the introduction, three systems were developed:

3A3A A system operating with NLTS to generate analytic summaries of specially structured record files.

3A3B A system for paper-tape logging of computer usage.

3A3C A data structure for personnel time-reporting records involving specified jobs.

3B The ROMAN System

3B1 Note: The remainder of Sec. IV-3 is increasingly detailed and technical. The more general discussion is resumed in Sec. IV-4.

3B2 This system provides for the on-line examination and analysis of accounting-type data or any data records that can be stored in the data formats described below.

3B2A The system is operated from NLTS, which serves as an

executive.

3B2A1 An instruction calling for a given analytic action on the record files is written in an NLTS file as part of a normal text statement, with special first-character(s) "tags" identifying the beginning of special instruction words.

3B2A2 A special "execute" command may be designated in NLTS, with a given analysis-instruction statement as a command operand (designated by direct selection, by entering as a literal either the name or the location number, or by use of indirect selection via a marker call).

3B2A3 After execution, any summary results from the analysis instruction are inserted back into NLTS as a new text statement (some instructions merely generate a new record file).

3B3 In addition to the normal text files, two special file types are associated with the system.

3B3A The original data base is contained in a packed source file of four computer words per entry, made up of eight "fields" defined as follows:

project	class	type
name	job	
hour	day	
charge		

3B3A1 Generic names have been given to the fields with a particular application in mind, but any data consistent with the format of the field could be used.

3B3A2 Data within each field is stored either as a literal or as a binary code requiring table lookup. For the present, applications of the fields are defined as follows:

3B3A2A Project--the Bootstrap-Community (BC) cost center against which this charge is made. This corresponds to the SRI term "project" and to the BC term "Activity."

3B3A2B Class--an activity class such as supervisory,

technician, etc. associated with each charge. This was not actually used in the 3100 system.

3B3A2C Type--the type of charge such as computer time, labor, purchase order, etc.

3B3A2D Name--the person originating the charge.

3B3A2E Job--a code for a particular defined task against which charges are to be accumulated.

3B3A2F Hour and Day--the time the charge is incurred.

3B3A2G Charge--the actual charge in hours or dollars depending on the type.

3B3B The second file type unique to the system is the "get" file.

3B3B1 This file is a subset of a source file and contains exactly the same data for each record as the source file.

3B3B2 The "get" file is generated by a command to the system specifying the source file, a name for the "get" file, and parameters identifying the subset of records to be transferred from the source file.

3B3B2A The "get" file may be generated from another "get" file or from a source file.

3B3B2B The "get" file was incorporated in the system so that the user could generate a small file containing the data in which he is interested.

3B4 Instruction-Syntax Description Conventions

3B4A In the following description of the system, a modified Backus-Naur form is used to describe instruction syntax.

3B4A1 A dollar sign ($) designates "any number of" the following syntactic entity. Apart from this convention, the relation to Backus-Naur form is quite direct.

3B4A2 The defined term, in a definition equation, is enclosed in parentheses to make it a "statement name" in NLTS.

3B4A2A This convention greatly facilitates on-line study of such descriptive documentation -- using the Hop Name

command, a user can select the name of a syntactic entity, as found anywhere in the text (without adjacent printing characters), and immediately have his display view transferred to the definition of that term.

3B4B A semicolon terminates the definition equation of a given syntactic entity, and our convention is to follow the semicolon with the names of those terms in whose definition the given entity appears as a right-hand term.

3B4B1 This also facilitates on-line study of this type of descriptive documentation. The Hop Name command may thus be used to quickly and easily examine the different descriptive usages of any of these syntactic entities.

3B5 (instruction) = displaycommand searchcommandstring

3B5A (displaycommand) = "**" operation filenames ; instruction

3B5A1 (operation) = listop listspec / getop / plotop plotspec ; displaycommand

3B5A1A (listop) = "list"; operation. This causes a summary tabulation to be produced and inserted as a text statement following the instruction statement.

3B5A1B (listspec) = "(" searchparameter "," searchparameter ")" ; operation

3B5A1B1 The LISTSPEC designates how the summary tabulation is to be ordered.

3B5A1B2 (searchparameter) = "T" / "P" / "I" / "C" / "N" / "H" / "D" / "A" ; listspec searchcommand T = Type of record P = Job I = Project (or funding center) C = Charge N = Name H = Hour D = Day A = Activity class

3B5A1C (getop) = "get"; operation . This causes a new data file to be produced, containing a specified summary of designated data files.

3B5A1D (plotop) = "plot"; operation

3B5A1D1 The plotspec was to designate which data-field names would be respectively the X and Y axes of the plot, i.e.,

3B5A1E (plotspec) = "(" xfieldname "," yfieldname ")" ; operation

3B5A1E1 (xfieldname) = fieldname ; plotspec

3B5A1E2 (yfieldname) = fieldname ; plotspec

3B5A1E3 (fieldname) = "project" / "class" / "type" / "name" / "job" / "hour" / "day" / "charge" ; xfieldname yfieldname

3B5A1F Only LIST and GET were implemented. PLOT was to generate a graphical portrayal of the summary analysis.

3B5A2 (filenames) = "(" ((sourcefile "," getfile) / sourcefile) ")" ; displaycommand

3B5A2A (sourcefile) = the file upon which the INSTRUCTION operates; filenames

3B5A2B (getfile) = the file created as a result of the operation of the INSTRUCTION on the SOURCEFILE ; filenames

3B5B (searchcommandstring) = $ searchcommand ; instruction

3B5B1 (searchcommand) = (space/comma) "*" searchparameter logicalcondition searchdesignator ; searchcommandstring

3B5B1A (logicalcondition) = "." / "," / ">not" ; searchcommand

3B5B1A1 .not was not implemented.

3B5B1B (searchdesignator) = specific value of data in field specified by the SEARCHPARAMETER; searchcommand

3B5B1C If a given SEARCHPARAMETER does not appear in any SEARCHCOMMAND, then the data field represented by the search parameter) is summed over.

3B5B1D If a given SEARCHPARAMETER appears in a SEARCHCOMMAND without any associated LOGICALCONDITION or SEARCHDESIGNATOR, then all distinct data entries in the data field represented by that SEARCHPARAMETER are identified; i.e., there is no summing over this field.

3B5B1E Any LOGICALCONDITION may be used with a given SEARCHPARAMETER in the SEARCHCOMMAND.

3B5B1F The use of LOGICALCONDITIONs "." and "," implies some ordering among the SEARCHDESIGNATORs for each type of SEARCHPARAMETER. If "." and "," are used, they must be used as a pair, with the SEARCHCOMMAND containing "." appearing before the SEARCHCOMMAND containing ",". No other SEARCHCOMMAND may appear between the two.

3B5B1G The legal SEARCHDESIGNATORS for all SEARCHPARAMETERS, except Type itself, are determined by the type of record concerned. The list of legal Type SEARCHDESIGNATORS is as follows:

L = Labor (i.e., personnel) charges, in hours
F = Facility charges for computer use in hours
M = L + 1 = Labor (i.e., personnel) charges, in dollars
G = F + 1 = Facility charges, in dollars

3B6 Examples of DISPLAYCOMMAND

3B6A **LIST(N,P)(ROMEFILE)

3B6A1 Under the control of the associated SEARCHCOMMANDSTRING, this DISPLAYCOMMAND will display as a list of up to 8 columns, an abstract taken from the file named ROMEFILE that is sorted by Project within Name.

3B6B **GET(ROMEFILE,TEMP)

3B6B1 Under the control of the associated SEARCHCOMMANDSTRING, this DISPLAYCOMMAND will take SUBSET of ROMEFILE and store it as a GETFILE named TEMP. This will not be displayed in any form.

3B7 Examples of SEARCHCOMMAND

3B7A *p=csf1 .... searches for Project (job mnemonic) CSFL

3B7B *n ....searches for all Names

3B7C *d.660315 *d,660811 searches for all Dates between 15 March 1966 and 11 August 1966.

3B8 Example of SEARCHCOMMANDSTRING : *T=f *D.660315 *D,660701 *N=wke *N=dce *N=jfr *I=5919 *P=csnl *P=rad

3B8A This SEARCHCOMMANDSTRING will search for Type F records for all Charges under Projects CSNL and RAD incurred by WKE, DCE, and JFR between 15 March 1966 and 1 July 1966.

3B9 Example of INSTRUCTION

3B9A **list(n,i)(romefile) *n=dce *n=wke *d660401 *d660501 .
dce 40.26 wke 10.10 total 50.36

3B9A1 The result of executing this INSTRUCTION is shown above, listing all charges incurred by DCE and WKE on use of the computer facility between 1 April and 1 May. The list is sorted by name. The list is derived from the SOURCEFILE named ROMEFILE and written as a regular NLTS statement at the end of the text buffer which is subsequently moved to follow the statement containing the INSTRUCTION with conventional NLTS operations.

3B10 Source files for the ROMAN system can be generated in two ways:

3B10A Text files could be generated and updated by the NLTS and then through an off-line ROMAN routine translated into the packed source file.

3B10B Data could also be read directly from punched cards and translated into packed source files through an off-line routine.

3B10C Additional off-line processes were provided for generating and updating the look-up tables required in connection with the name and job fields of the data files.

3C Computer Logging System

3C1 To get computer logging information in computer-readable form for the ROMAN system, we designed and installed a logging system for the 3100, completely external to the computer.

3C1A Since the computer is used by many people in the laboratory in a "hands on" mode, resident programs cannot be used for logging.

3C2 The logging system consists of a card reader, a clock, and a tape-punching Teletype.

3C2A The card reader was adapted from an automatic tube tester. It accepts plastic cards with up to nine alphanumeric characters. Each card contains the initials of the user and a six-character code identifying the project or job to which the computer time is to be charged.

3C2B A simple clock was constructed using a one-minute timer

and stepping switches to provide a readout of time (hours and minutes) in ASCII code.

3C2C When a card is inserted in the reader, the Teletype prints the time followed by the initials of the user and the job code. When the card is removed from the reader, the Teletype prints the time off followed by a carriage return and two line feeds.

3C2D An interlock to the computer "Master Clear" prevents any computer operations unless a card is in place in the reader.

3C3 This system produces a punched paper tape as well as the printed record. The punched paper tape can then be read into the NLTS and edited for any errors and then, through file generation routines, entered into the ROMAN source file.

3D Job Structure and Time Reporting Procedures

3D1 We developed a structure of "jobs" that defines the activity in our area of managerial concern.

3D1A A "job" is any uniquely identifiable task or activity and is the smallest unit for which accounting records are kept.

3D1B In the structure the importance of a changing environment has been emphasized. There is flexibility in that jobs may be broken down into subjobs or grouped into larger jobs at any time.

3D1C Procedures for updating a file of job definitions and information on cost centers to which the jobs will be charged have been worked out using NLTS.

3D2 On the basis of the job structure, an experimental system of personnel time reporting was established.

3D2A Individuals taking part in this system were asked to report all of the time that they spent on any of the defined jobs, without regard for conventional accounting constraints (such as reporting 8 hours each day and 40 hours each week).

3D2A1 To make this reporting easier, we made provisions for individuals to define jobs themselves when the previously defined jobs did not adequately identify their work.

3D2B A program was written to analyze the data collected in this system.

3D2B1 Phase 1 of this program analyzes the raw data collected from individuals and produces a summary of actual hours worked on defined jobs.

3D2B2 Phase 2 produces two files to be used as input to the status-reporting programs.

3D2B2A The first file is a copy of the summary produced in Phase 1.

3D2B2B The second file is compatible with the external accounting system, producing daily and weekly totals corrected for 8 hours per day and 40 hours per week. This is essentially a file on financial status.

IV -- MANAGEMENT-SYSTEM RESEARCH: Augmented Administration

4 Augmented Administration: Some Immediate Possibilities

4A The text-manipulation facility initially built into NLS (the On-Line System on the 940 -- see Sec. II) provides in itself considerable power for aiding general processes of management, such as planning, coordinating, review, etc. However, we plan certain extensions to cover other management data-processing needs (such as were provided by the ROMAN aids on the 3100).

4A1 An "expected" type of extension, for the Management-System Research Activity, is of course that which operates upon management-information files to produce analyses of interest to various management operations.

4A2 These will be important to us, and a rather general approach is described below that will include within it those capabilities developed in the ROMAN system, but in addition should allow many different tools to be developed, experimented with, and integrated into the management-system tool kit.

4B "Ready-Made" features

4B1 The NLS "Chain Generator"

4B1A For the requirements of flexible display-creation and printout-generation, we set up this special processor entity to search in a general way through NLS files, locating in a specified sequence those statements that satisfy given requirements.

4B1A1 For instance, the chain generator could be asked to generate a "chain" of all those statements found in Files AA, BB, and CC, down to Level 4, whose content satisfied certain complex specifications.

4B1A2 Alternatively, the generator could be set to follow cross-reference links of specified type, so that the generated chain would represent an "associative trail" (to borrow an old term from Vannevar Bush) through the files.

4B1B This facility will have considerable general value, as indicated by the value of its more primitive counterpart on the CDC 3100 NLTS -- i.e., the Level-Truncation Viewing and the Pattern-Match Filter features.

4B2 CAL

4B2A This is the basic on-line (Teletype) calculator package, developed by Dr. Butler Lampson of Project Genie at U.C.

Berkeley, and furnished as a standard software subsystem with the SDS 940 Time-Sharing System.

4B2B Besides offering flexible, easy-to-use, interactive calculator programming, CAL has a "graphic" feature providing for the construction of graphs, etc. on the CRT.

4B2B1 We have experimented with this feature on our new displays (before the vector generator was really working well enough to make decent drawings), and we see immediate and relevant applicability to our needs.

4B2B2 For example, we had it generate "pie" charts, calculating and displaying the percentage distribution among an arbitrary list of units.

4B2C We have conjectured as follows upon the possibility of modifying CAL:

4B2C1 It now uses a two-level hierarchical structure in its source code (Parts and Steps), and refers "TO" and "DO" commands to exact location numbers. It tolerates gaps in the numbering, as well as interpolative numbering -- but one does not "renumber" (in a cleaning-up operation) without considerable need, because all of these references must be updated individually.

4B2C2 We would like to have CAL accept the arbitrary-depth structure as used in NLS, and to accept the Name-Link references used there, too. This would let us use the full power of NLS in composing, studying, and modifying our calculator programs, as well as providing general compatibility with our working environment.

4B2C3 We would like also to provide an interface on the operand-gathering operations of CAL so that it would work in conjunction with the Chain Generator. We want to be able to reach into any sequence of statements, in any of our NLS files, to use as input to a CAL operation any terms whose identity may be specified by location (as in fixed-field formatting) or by context (e.g., identified by descriptors), or by some combination thereof.

4B2C4 We would similarly want to develop an output interface between CAL and NLS, to allow CAL output to be inserted in an arbitrary place in an NLS file (in specified format, including graphical), or to replace an existing term or construct with an updated one, or indeed to generate a new NLS file if desired.

4C General On-Line Graphic Package

4C1 We have been developing the specifications for extending the data structure, and the Control Metalanguage syntax and translator, to enable us to compose, study, and modify graphical portrayals as an integral part of our NLS file structure.

4C1A We will concentrate initially upon the "integration" feature, i.e., upon integrating graphs, line drawings, etc. into the structured-text files so that graphics can be used to support the expository or descriptive purpose of the text, or vice versa.

4C1A1 There are several ways of achieving this.

4C1A1A One way is to employ a free mixture of both forms on the display.

4C1A1B Another means is the use of compatible cross-reference links embedded in text to refer to graphic entities, and vice versa.

4C1A1C A third way is to let the product of the chain generator be an arbitrary mixture of text and graphics, and also let the pattern-match filtering work on the graphic entities.

4C1B The control-dialogue procedures for working with the graphic portrayals are also to be harmonious with the rest of our system. This promises to be relatively easy to accomplish, because of the flexibility of the Control Metalanguage approach and the conceptual order of our basic control-language approach.

4C2 This graphics package will serve many needs within the Bootstrap Community, but its early developments will be aimed to support two principal needs:

4C2A Documentation Diagrams -- for what is needed to record thoroughly and accurately the goals, plans, implementation, operation, etc. of all of our activities. This documentation effort is very important to many aspects of getting our "bootstrapping" effectively operative -- including, as noted below, the Management-System Research Activity.

4C2B Management Analysis Portrayals ---for what is needed to portray information of special relevance to the communication, analysis, study, etc. activities of the Management System. POL=1;

IV -- MANAGEMENT-SYSTEM RESEARCH: Augmented Organization

5 Augmented Organization

5A The "environment" will be strongly influenced by the following factors:

5A1 Slow but steady development of the full NLS system on the 940 -- getting the disc system going to provide file capacity, getting the display system up to quality and quantity, getting NLS and the Time-Sharing System to give reasonably good response to all the displays, getting system reliability up past the tolerable and into the dependable, etc. For a system of its complexity, we must allow for many months of this development work.

5A2 An unknown degree of distraction, from such new development pursuits as the management aids discussed above, caused by crises in this more fundamental pursuit of getting the basic system dependable.

5A3 New staffing, to make up for some loss during the winter, and for an increase in contract budget over last year, is not likely to mature until early summer, when the university graduates become available. We will be understaffed for a while, and then will be faced with indoctrinating new people -- i.e., there will be more burden on old staff before the new staff becomes productive.

5B To reach significant results in the Management-System research and to facilitate the "Bootstrap-Community" approach, we need to devote early and serious effort to developing better organizational structure and activity within the Community and instilling much more extensive awareness and practice of (and dialogue about) "good" management principles.

5B1 Launching this particular pursuit is not dependent upon the facility development, and this fact, together with the basic importance of the pursuit, led us to predict extra emphasis on this "organizational" topic during the first six months.

5B2 We will specify and steadily pursue the specific data-manipulation developments described above, but once they are specified their development rate will hinge upon the evironmental factors listed above.

5C There are some interesting possibilities for computer aids to the "organization" problem, particularly in the area of intercommunication and conferencing.

5C1 We anticipate that individual researchers will take part in team efforts, with each researcher working in his own office and connected to others by the computer system.

5C1A A system to augment intercommunication is thus necessary to enhance the capabilities of this team.

5C2 This problem is distinct from that of augmented conferencing where the concern is with people working together at the same location.

5D Augmented Intercommunication

5D1 What will exist in this regard within the initial NLS is roughly as follows:

5D1A The creator of a file can designate who else may have access to this file (i.e., who may get a copy to study), as well as who else (not necessarily the same group) may modify the file (i.e., replace one version with another).

5D1B As soon as one person has replaced a modified file into disk storage, another can pick up a copy to learn what has been changed.

5D1C If two users may each modify a given file, they may use it to communicate between themselves, more or less as if they were taking turns writing on a blackboard visible to both. The difference is that only one at a time may write on the file, and the other cannot see the new change until it is finished.

5D1D Of course, they may use several communication files, so that each can be composing a new message simultaneously (on different files); also, any long message can be sent in parts so that the recipient need not wait through the entire composition before seeing any of it.

5D2 Generally, this basic facility, by virtue of the instantaneous availability of a new version of a given file to everyone concerned, is in itself a very significant improvement in the means of organizational intercommunication.

5D2A However, putting this improvement to work raises immediate questions and problems. Consider, for instance, that two users will be in disagreement relatively often. For such an instance, there should be a period of dialogue, arriving at a resolution of some sort.

5D2A1 The dialogue would presumably proceed via interchanges written into a communication file -- probably the file containing a passage over which the disagreement arose.

5D2A2 When the dialogue is over, one user must clean up the residue and incorporate the outcome of the dialogue into the original passage. Possibly the other user would later need help in remembering the original version so as to see how the passage had been changed; he would need to find the record of the dialogue.

5D2A2A There would need to be conventions as to the recording of the dialogue, etc.

5D2A3 A second disagreement could occur during the course of this dialogue, and a third during the dialogue to resolve the second disagreement, etc.

5D3 Special aids to human intercommunication are bound to evolve ultimately. The following are some possibilities:

5D3A Annunciators (User-Interrupt System)

5D3A1 This refers to special signals indicating to a user that there is a message (or a requested operation) to be brought to his attention.

5D3A1A These signals function like the annunciators in certain communication equipment -- more or less raising a flag at the edge of the user's consciousness so that he can interrupt his current activity when it suits the occasion. The analogy to the interrupt system in computer hardware is quite direct.

5D3A1B Each individual could probably establish his own way of having these annunciators implemented -- e.g. blinking marks at one edge of the screen, etc.

5D3A2 In responding to an annunciator signal, a user may be taken automatically to a particular view of some passage which others are working on and would like his help, or there may of course just be an independent message. In the latter case, an automatic message-answering facility could be useful, setting up in turn an annunciator call upon the originator.

5D3A3 An announcement could be sent simultaneously to a specified set of people, with code terms possible for specifying special sets of people to whom group messages are frequently sent.

5D3A3A A user could have a message circulated in a specified order through a given set of people, and

perhaps could even set up an algorithm for dynamic determination of the sequencing (with looping possible) depending upon the additions made by the successive people.

5D3A4 Different levels of urgency could be signalled.

5D3A5 One typical signal could be an "alarm clock," set by the user himself for a given time or event.

5D3B Voting

5D3B1 A logical extension to the annunciator system is a set of aids for automatically calling for votes, for increasing the urgency of an ignored annunciator signal, for presenting the issue to each voter, and for recording the votes and registering the results.

5D3B2 To initiate such a procedure, a user would need to designate the particular issue, the group over which the vote is required, the priority level, etc. Perhaps he would include a request to be interrupted when the voting was complete, or by a certain time.

5D3C Dialogue structure

5D3C1 The interactive discussion is likely to take nonlogical paths in its course toward understanding and compromise, and it could produce a messy file unless special means are provided for either accommodating this disorderly development or else cleaning up as it proceeds.

5D3C1A Initially, the parties to the dialogue would have to do this cleaning up, but it could be very useful to develop a technique for structuring the successive contributions to reflect the course of the argument and provide help in clarifying its overall status at a given moment.

5D3C2 This might entail developing a node-link structuring, with ways of labeling the branching of dead-end paths, superseded arguments, alternatives yet unexamined, etc., and various ways of displaying different views of this to help see where the argument has been, where it is incomplete, what the net result currently is, whose "turn" it is now on a given branch, etc.

5D3D Signatures

5D3D1 A new dialogue entry should include the identification of the person making it, as well as a record of the entry time.

5D3D2 The computer can automatically install this "signature" in each entry.

5D3E Concurrent access

5D3E1 The course of a dialogue would be faster and less frustrating to all parties if simultaneous access were not limitated to "one at a time making additions," as will be the initial case.

5D3E2 With the special dialogue structuring, or even in an ordinary structure (with special algorithms regulating the operations allowed each participant), simultaneous access could be much improved without undue difficulty.

5D3E2A It would probably be necessary to enter a special operating (dialogue) mode, in which many operations are disallowed, and special ones provided; all users could then reference the same file.

5D3E2B For instance, if participants were limited only to the attachment of new statements (dialogue entries), as seems a reasonable possibility, then it is logically feasible to let them both be working on the same file.

5D3E2C User A could see where User B is attaching the entry he is currently composing; if desired, he could even see the current state of that entry (although he could not do anything about it until the entry was finished).

5D3F Injunctions

5D3F1 Some sections of a dialogue file could be open for modification, to sharpen up an argument, etc. But if some passage is under attack, it could create great heat and confusion if someone changed it before another party had had his say.

5D3F2 To control this, there could be an "injunction" feature, where User A could affix an injunction upon a specified passage, and the computer system would permit no subsequent changes until "due process" had lifted that injunction.

5D3F2A "Due process" would mean release either by User A or by some higher authority who overruled him.

5D3G Journal records

5D3G1 Allowing changes to be made in sections of the dialogue record, during the course of the dialogue, has definite advantage toward providing a better current "statement" of interaction.

5D3G1A However, it loses the "history" of the development of that current state.

5D3G2 A possible way to save the history would be for the computer system to automatically keep a log (a journal record) of each action by each participant, complete enough so that from this record it could make a dynamic rerun of the dialogue as it actually occurred.

5D3G3 For research purposes it would be very useful to be able to study the dynamics of these dialogues.

5D3G3A Among the consequent possibilities is the ability to evaluate the relative effectiveness of different individuals in a new and useful way. This possibility would also be valuable for operational management systems.

5D3G3B This facility would also enable analyses to trace the source of important contributions.

5E Augmented Conferencing

5E1 Dialogue from distributed consoles is important, but we are assuming that personal-confrontation dialogue is still necessary and very important.

5E1A It is not clear how much difference there might be in the nature of the computer aids required to "augment" a conference as compared with those for the distributed dialogue, but at least the physical arrangements will be different.

5E1B The following is a description of our initial experience with augmented conferencing.

5E2 The On-Line Sponsors' Progress-Review Meeting

5E2A On 12 and 13 October 1967 the following Sponsors' representatives visited us for a progress-review meeting:

5E2A1 Robert Taylor and Barry Wessler, ARPA

5E2A2 A. E. Gribble, NASA-Langley

5E2A3 Fred Dion and Dean Bergstrom, RADC

5E2B For this meeting we built a special square table, seating five on a side, with the center area open. We arranged six of our television monitors in the center area, so that each of the 20 persons was conveniently near at least one of them. Fig. IV-1 shows this arrangement.

5E2B1 These TV monitors were placed low enough to give each participant an unobstructed view of all other participants around the table.

5E2B2 At one location at the table we set up the mouse, keyboard, and keyset terminal equipment to remotely control NLTS on the CDC 3100.

5E2B3 A TV camera captured the image generated on the large display scope in the computer room, and the video was wired to the six monitors in the conference room.

5E2B3A The video signals were inverted, so that the displays in the conference room showed black characters on a white background.

5E2B4 Six auxiliary mice were located around the table, wired so that depressing the control button on any one of them would pick up a relay to connect that mouse into a special input channel to the computer.

5E2B4A This channel controlled the position on the screen of a second, extra-large tracking arrow.

5E2B4B This setup allowed any participant easily to take a nearby mouse, hold the button down to establish his control of the extra tracking pointer, and move this pointer about on the screen as a means of pointing out items on the display about which he wished to talk.

5E2C We collected about 150 of the files that have been developed in the program, and put them on one disc pack to be available for access and study during the conference.

5E2C1 This included all of the working specifications and some of the symbolic source code of our 940 system design, vhe documentation and user guides for the 3100 and 940

subsystems, all of our recent reports and proposals, etc.

5E2C2 We also included some files specially prepared to show the activity, framework, and candidate conference topics, and a framework from which to generate the working agenda.

5E2C3 Figure IV-1 shows the specially prepared agenda file, with the view parameters set to show only first-level statements, displaying one line of each.

5E2D As a means for presenting a large quantity of complex material, in a manner flexibly adjustable to the course of the discussion and to the special needs and questions of the participants, this setup proved very valuable.

5E2D1 The full power of NLTS for moving nimbly between files, and within a given file for scanning, freezing, cross-reference hopping, content filtering, etc., gives to a practiced user an unusual capability for presenting his material.

5E3 Conferencing plans and possibilities

5E3A We were all quite enthusiastic about the experience, and we are looking forward not only to regularly running our own program meetings and design sessions in this manner, but perhaps to holding more on-line meetings with sponsors and people outside the program.

5E3B Besides this 20-man conference setup, we shall need to develop means appropriate for 2- to 5-man groups, perhaps using the office consoles.

5E3B1 One basic need would be for independent control devices (mouse, keyset, etc.) for each participant.

5E3B2 Also, provision so that whatever display surface(s) are being used can be arbitrarily divided up for independent control. (This feature is also valuable for single-person use.)

5F The challenge of adapting to not only a unique working environment (i.e. the Bootstrapping Community, where a person is a specialist-contributor researcher as well as a subject), but to a rapidly evolving environment, is quite critical here.

5G This thrust in our general plans represents a shift in focus to a broader concept of "management system," with emphasis on

organizational problems. However, in parallel with efforts to understand these issues, we will continue to develop software management (administrative) aids, based on the SDS 940.

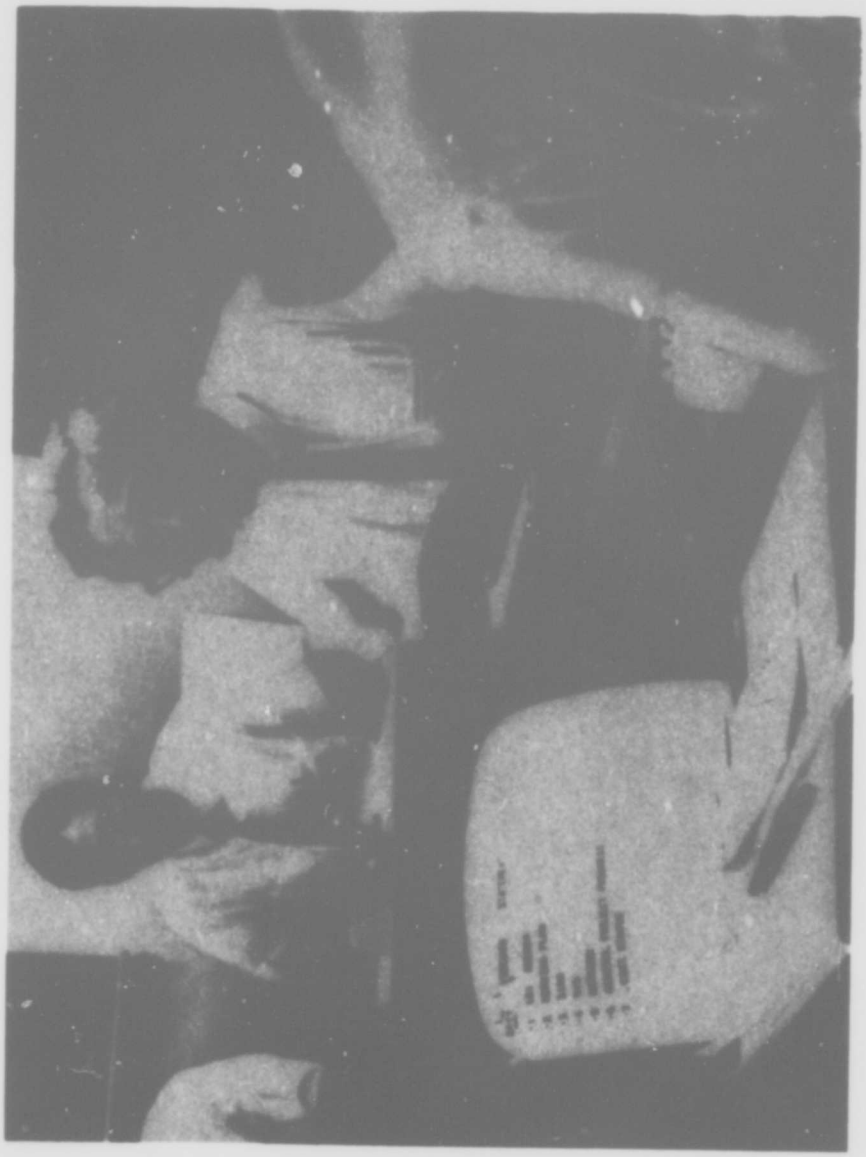


Figure IV-1 AHI Sponsors' On-Line Conference

BLANK PAGE

APPENDIX A -- General Background: General Features of AHI Research

1 General Features of AHI Research

1A Increasing the value of plentiful and immediate computer service requires a "system engineering" approach which, besides concerning itself with the development of the various computer-aid techniques and operations, treats in a coordinated fashion other significant aspects of the "user system" that are amenable to study and improvement. Some of these aspects are as follows:

1A1 The nature and structure of concepts with which the user approaches his problems. These concepts include "task," "constraint," "resource," "deadline," "progress review," "responsibility," etc. Specific instances of these common and important concepts will appear in a project environment in relationships forming a structure.

1A2 The nature and structure of the symbolic representations of these working concepts -- such as formatting, footnoting, cross-referencing, special tagging, charting, etc.

1A3 The way in which the user "encodes" his requests for service from the computer system, considering both the mental and the physical actions that are required to control the computer.

1A4 The procedures and methods with which he pursues his goals, at every level of activity where the system designer can practically make changes (e.g., how he changes a sentence, adds a reference, cleans out his files, gets a trial design approved, or sets up a project.)

1B This system-oriented approach is also an experimental one, in which all of the varied developments are put to work in a coordinated real-life working environment to be tested and evaluated.

1C The value ascribed to new developments, and the system of priorities used to select the next implementation from among many possibilities, are both based upon the associated improvements to the system's capability -- i.e., to the working effectiveness of the people in this experimental environment.

1D The users of this experimental system -- i.e., the "subjects" of experimentation -- are the staff of the AHI Research Center. We call this the "bootstrap" approach, and the group of users is called the "Bootstrap Community."

1D1 Thus, the aids developed and experimented with are those that promise to the Bootstrap Community the best payoff either in direct improvement of working abilities or in new understanding toward that end.

1E Implicit in the above, but deserving explicit comment, is the evolutionary nature of the system growth that results from this approach. Developments of various facets of this system, as well as our means to study, analyze, design, and implement them, must all evolve together in a coordinated fashion.

APPENDIX A -- GENERAL BACKGROUND: Bootstrap Approach

2 Major Considerations Associated With the Bootstrap Approach

2A The Whole-System Concept of Augmenting Human Intellect

2A1 We are concerned with the effectiveness of an "augmented human." Our concept of "augmented human" includes the human plus the language, methodology, and artifacts he uses.

2A2 It is our concern to improve the effectiveness of the whole system -- i.e., augment the human by evolving better language, methodology and artifacts, and the means for learning how best to use them in the pursuit of comprehension of our complex environment.

2A3 There is no sense in isolating a part of this system for study to the exclusion of other parts. The AHI Research Center is concerned with study, design, experimentation, and evaluation of all aspects of this system.

2A4 Limited problems can be isolated and worked on at any given time. But the evolutionary nature of our strategy allows us to continually redirect our concern as new problems are identified.

2B Empirical Evaluation of System Features Within a Working System

2B1 Considering that this is a whole-system approach and that the whole system is very complex, we cannot analytically determine the mutual-interaction effects that a change in one system component has throughout the rest of the system. The net worth of a change in one component must be evaluated by its effect on whole-system performance.

2B2 These considerations argue strongly for evaluating each feature by observing its effect within a whole system that is not only in realistic use but has had time to be shaken down. Only when the system has come to some sort of equilibrium after implementation of a new feature can the feature be evaluated. The resulting growth is by a series of small, evolutionary steps.

2C Evolutionary Approach

2C1 An important operational question is how large an evolutionary step should be to maximize experimental progress.

2C2 The "size" of a given step can be measured in several ways:

2C2A By its development cost in talent, dollars, and computer demand

2C2B By the magnitude of the changes required in the operational environment -- i.e., in hardware, conventions, skills, procedures, data formats, operating roles, etc.

2C2C By the time required to implement and check out the change, and to shake down the new whole-system operations.

2C3 At the present state of our development, most "steps" cannot be quantitatively evaluated. Evaluation and decision thus become a community problem, demanding new rules, procedures, methods and organization to enable a "consensus" to be attained. These issues are discussed as "organization" problems in Sec. IV.

2C4 Because of the great uncertainties in our research environment, the evolutionary approach means we make small steps in preference to large ones. Less uncertainty is involved with small pragmatic steps into the "unknown." Small steps can be accomplished without major equipment changes.

2C4A For example, we do not try to develop automatic speech-recognition capability, because it is a large-step change and does not at all promise corresponding value in terms of our goals.

2D The Bootstrap Strategy

2D1 The assumption that the whole-system, empirical, and evolutionary principles are all to be followed in an exploratory program leads to certain requirements:

2D1A The first requirement is a an experimental service system capable of providing the necessary service functions.

2D1B The second requirement is a set of subjects to use the system.

2D1B1 The subjects are trained in the use of current language, methods, and artifacts.

2D1B2 They use these augmentation means exclusively (where applicable) in the pursuit of significant intellectual work in which they have full professional motivations and pressures.

2D1B3 They not only become comfortable and smoothly skillful in the use of each innovation, but also pay attention to the new needs, possibilities, and complaints associated with the interaction of each innovation with the rest of the working environment.

2D1B4 This often calls for a succession of changes until the dynamic "reverberation" throughout the system caused by the introduction of the original innovation has died out, and the system assumes a certain condition of integrated smoothness in its operation.

2D1C The third requirement is a set of research tasks -- for example:

2D1C1 Studying the subjects and the subject-augmentation system

2D1C2 Analyzing and evaluating needs and possibilities

2D1C3 Designing new parts of the system or redesigning established parts

2D1C4 Implementing changes and training subjects to take advantage of new features

2D1C5 Conducting other normal research support activities including literature work, reference keeping, professional communication, planning, coordination, documentation, project management, etc.

2D2 The "bootstrap" principle, as applied here, results in one set of people playing the roles of both subjects and researchers.

2D2A This means that the researchers are constantly studying, analyzing, evaluating, designing, implementing, training, and learning over the whole range of recognizable, understandable, and designable portions of their own professional working environment -- within a set of objectives, principles, and concepts that it is their responsibility constantly to improve.

2D2B The areas in which they seek to augment their ability to comprehend and solve problems will be these activities themselves.

2D2C In that the researchers' daily work might be said to be "augmentation-system development," the sum of the developments stemming from their work will be an "augmentation-system development system."

2D2D The principles and techniques of analysis, design, implementation, evaluation, and training that they develop will be communicable to others who wish to work on specific augmentation systems.

2E The "Augmented Organization"

2E1 In our Bootstrap Community, we have an organization of "augmented humans," interactively cooperating and communicating with the aid of real-time cross-coupling through their computer system.

2E1A Such an organization is promised a direct gain in effectiveness because its individual members can do more complex tasks and do them better and quicker.

2E1B We expect a source of marked improvement in organizational effectiveness to be found in the new abilities to intercommunicate, to follow sophisticated cooperative procedures, to work simultaneously on common data, to have semi-automatic interpersonal "interrupt" protocol leading to approval or voting actions, etc.

2E1C The product of "augmented organizational coordination" of "augmented human" members presents a truly exciting potential.

2F The Management-System research is viewed as an important part in pursuing this potential within the SRI AHI Program.

APPENDIX A -- GENERAL BACKGROUND: Experimental Environment

3 Specific Features of Our Experimental Environment

3A The foregoing has been an attempt to give the reader some insight into the conceptual framework and research strategy with which the AHI Program pursues the potential of "augmenting human intellect." The following is an introduction to the actual work that has been carried out by the Augmented Human Intellect Research Center.

3B Our initial focus has been on computer-aided text manipulation. (By "text" we mean generally information represented by strings of characters. This could include mathematical equations, programming statements, etc.) There are several reasons for this:

3B1 Text is representative of our speech and much of our conscious reasoning about nontextual records; it is the basic fabric in which most of the collaboration in system development work such as ours takes place.

3B2 Text is applicable as a representation of our thoughts and actions at all levels of our working system (e.g., from coding for the computer up to long-range planning for the research program). This makes it widely applicable to our own work.

3B3 A coordinated working system for manipulating text is relatively easy to implement. With equivalent resources, a wider collection of useful working aids may be implemented for text than for graphics, for instance.

3B4 An effective system for handling the text of working records (planning, design, reference, etc.) will provide a sound structure in which later to embed manipulation techniques for other symbols -- e.g., graphics.

3C The Use of Structured Text

3C1 A very early feature of our conceptual framework regarding means for augmenting human intellect was the introduction of more explicit "structuring" of working information. We took the view that the symbols one works with are supposed to represent a mapping of one's associated concepts, and further that one's concepts exist in a "network" of relationships as opposed to the essentially linear form of actual printed records. Thus it was decided that the concept-manipulation aids derivable from real-time computer support could be appreciably enhanced by structuring conventions that would make explicit (for both the user and the computer) the various types of network relationships among concepts.

3C1A As an early experiment with this concept, we adopted some

years ago the convention of organizing all information into hierarchical structures, with provisions for arbitrary cross-referencing among the elements of a hierarchy. The principal manifestation of this hierarchical structure is simply the breaking up of text into short paragraphs called "statements," each of which bears a number showing its serial location in the text and its "level" in an "outline" of the text. For some material, the structured statement form may be undesirable. In these cases, there are means for suppressing the special formatting in the final printout of the structured text.

3C1B The ability to name individual statements and to form arbitrary cross-reference links between any two statements, when added to the basic hierarchical form, yields a general structuring capability that is quite flexible. These structuring conventions are expected to evolve relatively rapidly as our research progresses.

3C2 The basic validity of this approach has been well established by our subsequent experience. We have found that in both off-line and on-line computer aids, the conception, stipulation, and execution of significant manipulations is made much easier by the structuring conventions. Also, in working on line at a CRT console, not only is manipulation made much easier and more powerful by the structure, but a user's ability to get about very quickly within his data, and to have special "views" of it generated to suit his need, are significantly aided by the structure.

3C3 We have come to write all of our documentation, notes, reports, and proposals according to these conventions, because of the resulting increase in our ability to study and manipulate them during composition, modification, and usage. Our programming systems also incorporate the conventions. We have found it to be fairly common that after an initial period of negative reaction in reading explicitly structured material, one comes to prefer it to material printed in the normal form.

3D Means of Manipulating Working Text

3D1 Two coordinated systems of text manipulation are used within the AHI Research Center. The purpose of these systems is to aid their users in composing, modifying, and studying the text of their working information, with special attention to the ease, flexibility, and power that make working records flexible enough to be kept up to date with current thinking and developments. All of our text-manipulation techniques will work to some extent with normal "free" text, but are much more effective and powerful when

used on structured text. Output text from either system can be manipulated further with the other system. The two systems are described below.

3D2 FLTS

3D2A One system is the Off-Line Text Manipulation System (FLTS), which provides a means for harnessing computer aid for a user sitting off-line at any writing device that produces punched paper tape.

3D2A1 The paper tape is later batch-processed by a computer program that operates upon the new text input, as well as upon any specified "old" or previously processed text, according to user-specified directives (commands) embedded in the text. These directives may be deleted from the text on final output.

3D2A2 FLTS may be used for composing and modifying new files, for adding to or modifying old files, and for merging and reorganizing new and old files.

3D3 NLTS

3D3A The other system is the On-Line Text Manipulation System (NLTS), with which a user sitting in front of a CRT display gets immediate response to his key-stroke and pointing actions in terms of modification to the displayed text, or the place in text he wishes to view, or the form of a view he wishes to see.

3D4 Printout

3D5 Special printing-control features have been developed to facilitate output printing for either system. Special control directives are embedded in the text, to be carried and manipulated as part of a file's regular content. These are recognized and interpreted during output. By these means margins may be set, page headings may be established, etc.

3D5A As an example, with the exception of pages containing illustrations, every page in any of our current reports was typed on its mat directly from computer output. The system, in response to directives, automatically leaves page space for the pages that contain photographs. One directive permits the printing of the directives to be suppressed, so that they will not appear in the final copy.

3E Note: These systems are being transferred to the SDS 940 time-sharing computer.

3E1 "The 940 display-oriented on-line system is called NLS, and will include graphic- as well as text-manipulation capabilities. Its initial version will provide essentially the the same user features as the 3100 NLTS. An improved printout system is being implemented.

3E2 The equivalent of FLTS is scheduled for implementation after NLS is running well, and will be a system usable for either off-line (via paper-tape) or on-line typewriter-controlled manipulation of files containing mixed text and graphic information.

APPENDIX A -- GENERAL BACKGROUND: Prior Support History

4 Prior Support History

4A A project for the Air Force Office of Scientific Research (Contract AF 19(638)-1021, under which the basic conceptual work was done, as well as the first off-line text-manipulation work

4B An internally sponsored project at Stanford Research Institute, under which an intermediate-state off-line system was developed

4C A project for the Advanced Research Projects Agency (Contract SD-269), under which work on information structuring, basic working methodology, and the higher-level manipulation processes in the on-line system were done

4D A project for the Electronic Systems Division of the Air Force (Contract AF 19(628)-1088) which studied structuring and manipulating techniques for managing information (specifically, system-program design documentation)

4E A contract with NASA (Contract NAS 1-3988) in which we studied and developed the display-control techniques that represent the operational foundation of the on-line system.

4F A contract jointly sponsored by ARPA and NASA (Contract NAS 1-5904), begun just before this management-system research project and running in parallel since then, which has supported the basic developments of the AHI Program (see Engelbart6 and Engelbart7).

5 Status at the Beginning of the Project

5A At the time the project began, we had just replaced our earlier CDC 160A with the present CDC 3100. Our new facility comprised:

5A1 The CDC 3100 computer, in the following configuration:

5A1A Memory: 16,000 words, 24 bit, 1.75 microseconds

5A1B Three I/O channels, one of which is compatible with the interface previously used on some of our equipment for coupling to the CDC 160A

5A1C Paper-tape I/O

5A1D Three magnetic-tape transports

5A1E One IBM 1311 disk file (2,000,000 character capacity)

5A1F One 150-line/minute printer

5A1G Punched-card reader, 1200 cards per minute.

5A2 Work Station

5A2A Special (SRI) interface equipment couples the 3100 to a display generator, and to the various keyboards and selection devices of the work station.

5A2B Before the project began, SRI had installed a Straza character generator, with a repertoire of 63 characters and a generation rate of the order of 100.000 characters per second.

5A2C Early in the project, SRI added a vector generator to our display-driving equipment.

5A3 Software for the 3100

5A3A CDC provides a FORTRAN IV compiler, an assembler for their COMPASS symbolic machine language, and the SCOPE operating system, under which programs in either language may be run. (There are, of course, other CDC software systems, but these were the only ones of concern to us.)

5B An early version of NLTS had been programmed for the 3100, with essentially the same internal organization and external functional features that had been implemented on the 160A (see English1).

5B1 This implementation was programmed in COMPASS.

5B2 The 3100 had considerably more room in core for working data -- so we now could contain the whole of a 30,000-character working file in core -- whereas on the 160A only some 3000 characters out of a 17,000-character working file could be held in core.

5B3 The response to many commands, when working on a large file, was thus noticeably faster.

5B4 Quite a few improvements in detailed software design were made in this reprogramming, but it was not a "redesign," since our basic need was to get a system operating as quickly as possible on the 3100. The 160A was to be removed, and we did not want to be without a usable on-line text manipulation system.

5C The structure-manipulation part of a new FLTS had been specified for the 3100, and was being implemented in FORTRAN IV.

5C1 The manipulation functions specified were essentially the same as in the system previously implemented in ALGOL on our B5500 (see Engelbart5).

5D The basic assembly-debugging facilities within the COPE system had essentially been put into operation. This version of the system was still written in COMPASS.

5E A SNOBOL3 compiler had been designed for the 3100, and was essentially programmed (in COPE) but not checked out.

5F XDOC: External-Document System

5F1 Since 1959, under various successive sponsors, we have been accumulating a file of "external document" (XDOC) citations and reprints.

5F2 The citations had been punched on paper tape as they accumulated, and we had about 2000.

5F3 Of the actual referenced papers, at least 75 percent existed as reprints or copies, stored in our files under their XDOC accession numbers.

5F4 The only index we had to this was a card file arranged alphabetically by author.

5G The further developments of these 3100 systems are described in (Engelbart6) and (Engelbart7).

APPENDIX A -- General Background: History of Management Project

6 History of Management-System Project

6A The project began in February 1966, and an initial set of computer aids was specified and implemented (as described earlier in this report) to support the management of our manpower and facility resources. This was as initially planned for the research project, making use of the CDC 3100 computer-display system.

6B At the ARPA Contractors' meeting at MIT on 7 and 8 April 1966, in a discussion between D. C. Engelbart of SRI and Robert Taylor of ARPA, it was agreed to study the possibilities for a significant expansion in the computer-display facilities for the AHI Program -- an expansion for which ARPA was tentatively interested in receiving a proposal.

6B1 The basis for the initial exploration was to consider establishing on the order of twelve CRT consoles, to be distributed among the offices and work areas of the AHI Research Program, to be served with full-time availability by a time-shared computer system able to provide the needed level of response and file storage.

6C Knowing that any such expansion would significantly alter the planned activity for the management-system project, Engelbart visited RADC on 9 April to discuss the situation with Mr. A. R. Barnum and his staff. The consensus reached during this discussion was as follows:

6C1 If such a change were to be made, it would be better (for the AHI program as a whole) to do it as soon as possible, since the much-increased research progress yielded by the proposed new system would thus have longer to produce results.

6C2 Once a firm decision to change facilities was reached, it would be wasteful to continue to invest time and money in developing management aids on the CDC 3100 -- since we would not carry on with the use of the old facility, and thus would not be able to use, evaluate, and improve these aids.

6C3 Thus, it was realized that, compared with the initial expectations on this management-system research project, significantly less results would be obtained during the project period if this new facility development were to take place -- i.e., results represented by explicit new computer-augmented management practices.

6C4 This project had been viewed, by both sponsor and contractor, as an initial stage of at least a two-stage effort, and while the contemplated facility changeover would diminish the results from

the first stage, it would tremendously enhance the productivity of a second stage.

6C4A Part of the tentative new-facility plan being studied was that ARPA would support the entire computer facility, allowing free use of it by other projects (such as this management-system project) that fitted into the coordinated "bootstrap" research pursuit. This would free the fairly significant portion of the subsequent RADC support money otherwise required for CDC 3100 computer charges, to be applied towards increased manpower effort on the management-system research.

6C4B Also, the near 20-fold increase in console availability, plus the "augmented organizational-coordination" feature, would produce a very much richer environment in which this increased management-system research staff would operate. This environment not only would permit a much wider range of experimental techniques to be studied, but would also represent much more realistically the working environment for the manager of the future.

6C5 In the long run, then, RADC's return in research results would be increased by shifting the nature of this first stage to coordinate with such a facility expansion, if the latter were made possible.

6C6 It was agreed then that SRI would proceed with a study as to the feasibility, costs, etc. associated with such an expansion, and if appropriate, submit a proposal to ARPA for its support. Concurrently, the initial plans for the management-system research would be pursued; and if the proposed expansion became significantly probable, discussion of the associated rearrangements for the RADC contract would be taken up.

6D Through the Summer of 1966, SRI conducted its study and prepared a proposal -- developed around acquiring an SDS 940 computer, and building special interface and display hardware. At a discussion with personnel representing ARPA and NASA (who were also involved in support of concurrent AHI research), held at NASA's Langley Research Center on 21 and 22 September 1966, a consensus was reached regarding both the value of such an approach to the whole AHI research program, and the interest of both NASA and ARPA in proceeding with negotiations. This established a high enough probability of the new facility becoming a reality that effort under the ARPA-NASA contract was shifted entirely to the planning and specifications for the new system.

6E In accordance with prior arrangement, a telephone discussion was

then held with Mr. Barnum in which it was agreed that SRI should request an extension of the contract without funds for this project. Accordingly SRI requested that the conract termination date be so extended, from its original date of 22 March 1967 to a new date of 1 September 1967.

6F It had been initially assumed that the added ARPA funds would be attached to the existing ARPA-NASA contract. It was found preferable, however, to route this added funding to SRI through RADC. Formal proposals were submitted by SRI on 1 February 1967 (ESU 67-10).

6G On 5 May 1967, the modifications to Contract AF 30(602)-4103 were finalized and money became available for ordering equipment and implementing the new system.

6G1 Concurrently, the termination date of the contract was adjusted to 7 March 1968; thus this "first stage" of management-system research became but a part of the larger project, and its termination date was made concurrent to that of the facility support portion -- the latter being an inflexible date, since a large segment of the funding supports the computer lease, which the AHI program has no flexibility to adjust.

6H At the time of this report -- i.e., at the above-mentioned termination date, the new facility is not quite functional, and no special aids have been programmed for management-system support. However, there have been developed special features in the data structuring, in the Control Metalanguage, etc. (see Sec. III) to enable the ready addition of these contemplated special computer aids.

APPENDIX B -- HARDWARE REFERENCE MANUAL

1 Introduction

1A This document is a preliminary reference manual for programming the peripheral equipment connected to the Special-Devices Channel (SDC).

1B The Special-Devices Channel is an I/O channel designed to interface a group of nonstandard (non-SDS) equipment to the SDS 940 computer. It consists of an executive control and six independent units.

1B1 The channel provides direct access to memory for each of the units.

1B1A Memory addresses, direction of transfer, and priority are supplied independently by the units.

1B1B The units operate asynchronously. The executive control standardizes timing and determines priority among the units.

1B2 It responds to EOM and SKS instructions for program control of the devices.

1B2A Each unit operates from a fixed address in core, with no provision for transfer of addresses with the EOM instructions.

1B3 Interrupts are transmitted to the computer under certain conditions in the various units. Nine interrupt lines are provided.

1C The units and associated devices handled by the channel are as follows:

1C1 A disc file with approximately 32 million words of storage. The average access time is 95 ms and the data rate is 42,000 words per second.

1C2 Two independent display systems each with six displays.

1C3 An Input-Device Controller (IDC), which reads the input devices associated with the CRT work stations (keyboards, binary keysets, and mice) and also provides certain output signals for use at the consoles.

1C4 A terminal to accommodate special hardware for time-sharing system improvements.

1C5 A Low-Priority Controller accommodating three low-priority devices:

1C5A A line printer with printing speed of 230 lines/min and 96 characters, including upper- and lower-case alphabets.

1C5B An on-line Selectric typewriter, with provisions to accommodate a plotter at some future date.

1C5C A terminal to interface the proposed ARPA computer network.

2 The following terms are used in describing the operation of the SDC:

2A Advance sector word -- A word in core that is written by the disc controller to indicate the sector in each zone that will be available next.

2B Command table -- A contiguous buffer in core containing commands to a disc or display controller.

2C Console status table -- A bank of 56 words that contain data on the input devices and output signals for each console.

2D Display buffer -- A contiguous buffer in the display pages containing display instructions.

2D1 The display buffer may cross the page boundary and has a maximum length of 1023 words.

2E Display controller -- The system hardware that interprets computer instructions (EOM and SKS) and controls the command-table processing.

2F Display generator -- The display-system hardware that accepts display instructions and from them produces a picture on a CRT.

2G Display instructions -- Instructions to the display generator (beam motion, character writing, etc.) that cause the actual generation of a display.

2H Display list -- A contiguous buffer in the display pages containing pointers to display buffers.

2H1 The display list, by definition, begins in the first display page and may extend into the second page (if it exists). Within this restriction it may be of any length.

2I Display pages -- One or two pages in user core that may contain a display list and display buffers.

2I1 The first display page is the page in which the display list begins.

2I2 If a second page is available it is the next consecutive (to the user) page.

2J Error word -- A fixed core location in which the disc controller writes bits indicating error conditions in the disc system.

2K Input Devices Controller -- The system hardware that controls the

input devices (keyboard, keyset, mouse and switches) for each console and the output signals (other than display) to each console.

2L Unit reference cell (URC) -- A fixed location in core for each unit that contains the address of the command (or print buffer) being processed by that unit and the error code associated with that unit when an error condition is detected.

2M Print buffer -- A contiguous buffer in core containing characters (control and data) for the line printer.

3 Core and Interrupt Assignments

3A The following fixed octal core locations are assigned to the devices on the channel:

10	URC - line printer
11	URC - network
12	URC - typewriter/plotter
20	URC - first display system
30	URC - second display system
50	URC - disc system
51	Advance sector word - disc system
52	Error code - disc system
260 - 350	Console Status Table

3B The following are interrupt assignments:

205	Input Device Controller
206	Disc - normal
207	Disc - error
210	Special Operations
211	Line Printer
212	Network
213	Typewriter/Plotter
214	Display 1
215	Display 2

4 Computer Instructions

4A Program control of the Special Devices Channel is through EOM instructions.

4A1 The EOM code is 20230YXX

4A1A The Y digit refers to the EOM type. The EOM actions vary for the units and are described along with the respective units.

4A1B The X digits refer to the unit. These are

1	Disc file
2	First display system
3	Second display system
4	Special operations
5	Input device controller
16	Line printer
26	Network
36	Typewriter/plotter

4B SKS instructions are used to sense status in the system.

4B1 The SKS code is 04030YXX

4B1A The Y digit refers to the SKS type. The conditions sensed are described along with the respective units.

4B1B The X digits refer to the unit with the same coding as for EOM instructions.

5 Disc File System

5A General

5A1 The subsystem described here consists of a Bryant Disc File, Series 4000, Mod A2A, and a control unit. The present 7-disc system is capable of storing approximately 32 million 24 bit words.

5B EOM and SKS Instructions

5B1 Four EOM instructions are defined for the disc subsystem.

5B1A The EOM codes are

20230101	Go chain
20230201	Go no-chain
20230301	Disconnect
20230401	Reset

5B1B The EOM actions are:

5B1B1 Go-Chain -- This EOM causes the controller to start command processing.

5B1B1A Processing always starts with the command addressed by the URC when the EOM is executed.

5B1B1B If a disconnect request has previously been stored by a Disconnect EOM and the system is still busy (processing commands), a Go-Chain EOM cancels the disconnect request.

5B1B1C A Go-Chain EOM issued while the system is busy and no disconnect request is stored results in a command error.

5B1B2 Go-No Chain -- This EOM causes the controller to process the single command table entry pointed to by the URC.

5B1B2A A Go-No Chain EOM received while the controller is processing commands results in a command error.

5B1B3 Disconnect -- This EOM causes the controller to disconnect at the next normal interrupt condition.

5B1B3A The execution of a Go-Chain EOM before the next

normal interrupt condition is reached cancels the disconnect request.

5B1B4 Reset ---This EOM immediately terminates any disc operation in process when the EOM is received, and returns the system to the disconnect state.

5B2 Two SKS instructions are defined.

5B2A The SKS codes are

04030101	Skip if Busy
04030201	Skip on No Error Conditions

5B2B The conditions sensed are;

5B2B1 Skip if Busy -- This instruction causes the next instruction to be skipped if the disc system is busy.

5B2B2 Skip on No Error Condition -- This instruction causes the next instruction to be skipped if no outstanding error conditions exist on the disc subsystem. Execution of this instruction does not reset any error conditions.

5C Command-Table Processing

5C1 After either Go EOM the system begins processing commands with the command addressed by the URC.

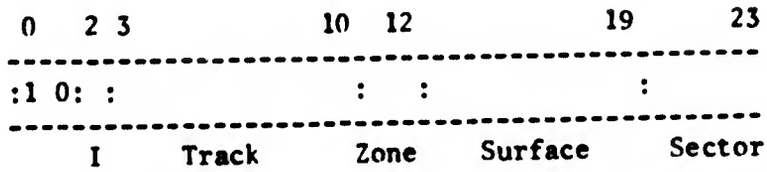
5C1A The URC always points to the current command being processed.

5C1A1 In a Go-Chain or Go-No Chain operation, after the successful completion of the command, the URC is updated (incremented by 3) to point to the first word of the next command.

5C2 There are three types of commands in the command table.

5C2A Data Transfer Command -- This command consists of three command words located in contiguous memory locations.

5C2A1 The first word contains the disc address. It consists of concatenated binary address fields. Not all combinations in certain address fields are used; the unused combinations form invalid addresses. The address word has the following format:



5C2A1A Interrupt bit -- If Bit 2 is a 1, a normal interrupt is given after successful completion of the command.

5C2A1B Track Address Field (8 bits) -- This field is used to select one of 256 head array positions. All bit combinations in this field are valid.

5C2A1C Zone Address Field (2 bits) -- This field is used to select one of the three disc frequency zones as follows:

00	Zone 1
01	Zone 2
10	Zone 3
11	Invalid

5C2A1D Surface Address Field (7 bits) -- This field is used to select one of the 12 data surfaces and two heads per zone.

5C2A1D1 Surfaces are numbered 0 to 11, with the low order bit selecting the head.

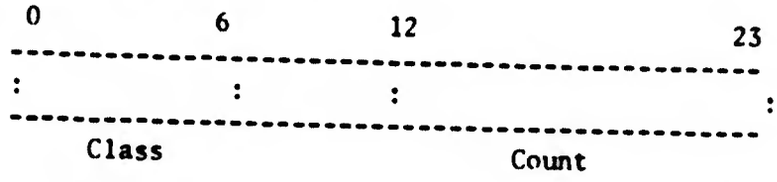
5C2A1D2 The valid addresses for the 6 disc system are 0000000 through 0010111.

5C2A1E Sector Address Field (4 bits) -- This field is used to select the proper sector on a track.

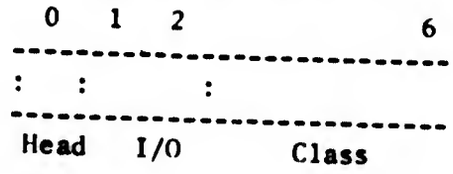
5C2A1E1 The valid combinations for this field depend on the zone selected. Sectors are numbered zero to k, where k is one less than the number of sectors in the zone. The following combinations for each zone are valid.

Zone	Address Field	Sectors
1	0000-0010	3
2	0000-0111	8
3	0000-1010	11

5C2A2 The second word contains the class and word count. Its format is as follows:



5C2A2A Class Field contains the Direction-of-Transfer Bit (Read/Write) and information on headers. It is subdivided as follows:



5C2A2A1 Head -- If this bit is a 1, header fields are written with the record.

5C2A2A2 I/O -- These bits determine the direction of transfer and the use of the class field as follows:

00	Read - No compare with class
01	Read - Compare with class
10	Write record and class field
11	Write if class compares equal

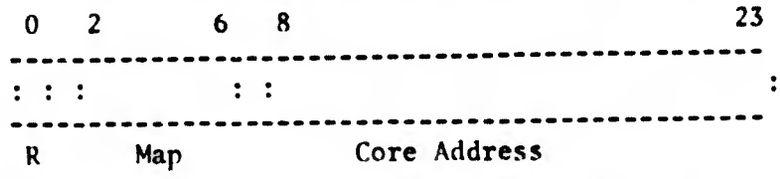
5C2A2A3 Class -- This 4-bit field appears in each record defining a class to which the record belongs. If class comparison is called for and fails, an error interrupt is given.

5C2A2B Count Field -- This field defines the number of 24-bit words to be transferred.

5C2A2B1 The maximum word count is 2048. Exceeding this count in the command word results in an illegal word count error.

5C2A2B2 If the field is zero the command serves to position the head array only. (Headers may be written with a word count of zero).

5C2A3 The third word contains the core memory address at which the transfer is to begin and mapping information to be used in crossing page boundaries. The word format is:



5C2A3A The R bit indicates whether mapping is to be used.

5C2A3A1 A 1 in the R position indicates mapping, and a 0 indicates no mapping.

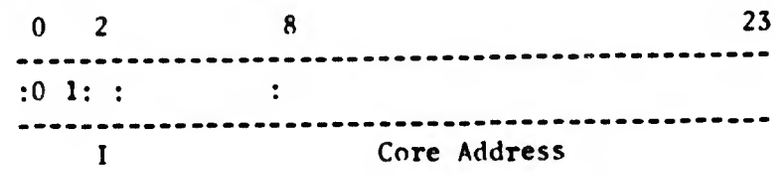
5C2A3B The "map" bits (Bits 2 - 6) are used to remap the 16-bit address when a page boundary is crossed.

5C2A3B1 When the page boundary is crossed (lower 11 address bits are zero) and mapping is used, the upper 5 bits of the address are replaced by the map bits.

5C2A3B2 If no mapping is used (R=0), crossing a page boundary results in a map error.

5C2A3C Core Address -- This field contains the absolute core address at which the information transfer is to begin.

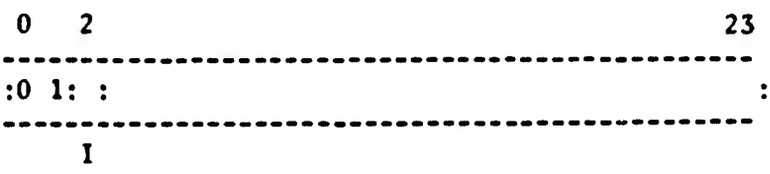
5C2B Branch Command -- This command causes the next command word to be taken from the core location given in the branch command word rather than in sequence in the command table. The core address is absolute and no remapping takes place. The word format is:



5C2B1 If the interrupt bit is set a normal interrupt will be generated after the command is executed.

5C2B2 Note: After a branch command the URC is written with the entire contents of the branch command word.

5C2C Disconnect Command Word -- This word causes the disc controller to disconnect. The word format is:



5C2C1 If the interrupt bit is set a normal interrupt will be generated after the command is executed.

5D Disc File Formats

5D1 Disc Format: Each of the twelve data surfaces is divided into three zones, with a pair of heads for each zone. Each of the three zones has a separate clock frequency and bit density optimized for the zone.

5D2 Zone Format: A zone is divided into 512 tracks, corresponding to each of two heads at 256 positions of the head array.

5D3 Track Format: A track is divided into sectors by prerecorded sector pulses. The number of sectors per track is a function of the zone.

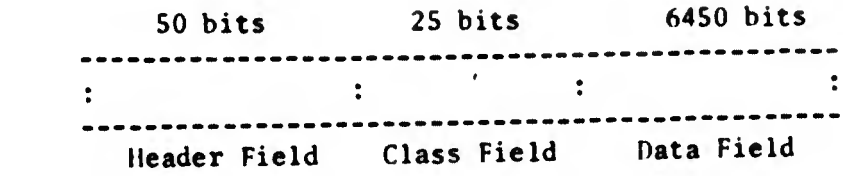
Zone 1	3 sectors/track	Inner Zone
Zone 2	8 sectors/track	Middle Zone
Zone 3	11 sectors/track	Outer Zone

5D4 Sector Format: There is one fixed-length record per sector, with a data field of 256 24-bit words. Associated with each record is a header field used to identify the record and ensure that head and zone selection are correct before writing or reading a record, and a class field used to grant access to records by class.

5D4A In all subfields of the sector a preamble and postamble are used to ensure reliable reading of the first and last bits of the subfield.

5D4A1 These bits are all "ones," generated by the controller and never transferred to the computer.

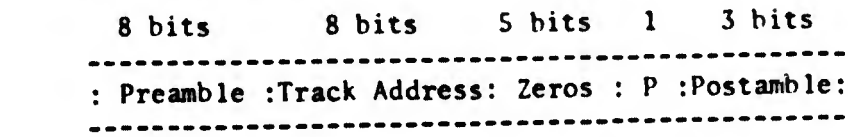
5D4B The overall format of the sector is



5D4B1 The header field consists of two header words generated by the control unit and is not transferred to the Central Processor.

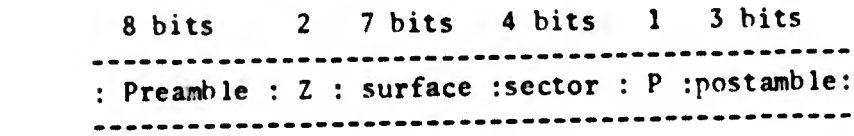
5D4B1A These words are written only when special key switches (one for each header word) are on and a 1 appears in the 0 bit of the class and count word.

5D4B1B Header word 1



5D4B1B1 This word is written by the disc controller and is used for track verification.

5D4B1C Header word 2



5D4B1C1 Zone Subfield (2 bits) -- These two bits correspond to the zone address and are used to insure proper selection of the zone.

5D4B1C2 Surface Subfield (7 bits) -- These seven bits are used to ensure correct selection of the head and recording surface.

5D4B1C3 Sector Subfield (4 bits) -- This subfield is used to identify the sector or record and is unique on each track.

5D4B1C4 Parity Subfield (1 bit) -- Odd parity is generated for each header word and is checked whenever the header is read.

5D4B2 Class Field Format -- The format of the class field is:

8 bits	4 bits	9 bits	1	3 bits
Preamble	Class	Zeros	P	Postamble

5D4B2A Class Subfield -- This is a 4-bit field defining the class to which a record belongs. Normally the class field is read and compared with that appearing in the command word; if they are equal the operation proceeds.

5D4B2B Parity Subfield (1 bit) -- Odd parity

5D4B3 Data Field Format

8 bits	6400 bits	8 bits	3 bits
Preamble	Data	Check Bits	Postamble

5D4B3A Data Subfield (6400 bits) -- This subfield consists of 256 24-bit machine words. An odd parity bit is inserted every 24 bits by the control unit. It is transferred in its entirety on a read operation with odd parity generated for each word. If less than 256 words are transferred on a write, the control unit generates the necessary zeros to fill out the data sub field.

5D4B3B Check Subfield (8 bits) -- This subfield is used for error checking over the data record. It is generated by the control unit on a read or write operation and is never transferred to the central processor.

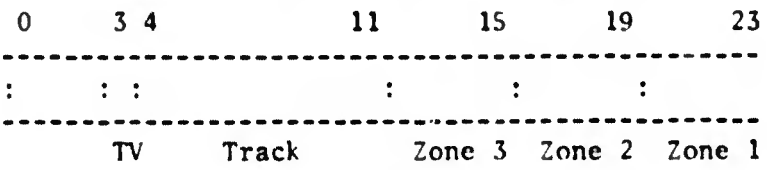
5D4B4 Gap Format -- A gap of 75 bit times is allowed between each alterable segment of the sector format and the next. This allows sufficient time for the recovery of the read amplifiers after writing a segment of the sector field.

5E Clocking

5E1 Clock tracks are prerecorded on a separate disc with its own set of heads that do not move.

5E1A Each zone has a separate heads for write clock and sector/index pulse.

5E1B When the system is busy, the advance sector word is updated by the controller to indicate the next available sector in each zone. This word has the following format.



5E1B1 "TV" is the track verification bit. When this bit is a 1 the heads have settled on the addressed track.

5E1B2 The "track" code indicates the head array position if TV is 1 and head array destination if TV is 0.

5F Error Conditions

5F1 Whenever an abnormal condition is detected by the controller the following actions occur:

5F1A Any data transfer operation in process is terminated.

5F1A1 A disc read operation is terminated immediately on detection of the error.

5F1A2 On a disc write operation the remainder of the current sector is filled with zeros and the operation is terminated.

5F1B Bits indicating the error conditions are written in the disc error word.

5F1C An abnormal interrupt is generated.

5F1D The controller goes to the disconnect state.

5F2 The disc error word contains a 1 for every abnormal condition that has occurred. At least one bit will always be set and more than one can be set.

5F2A The format of this word is

Bit	
12	Illegal Word Count
13	Map Error
14	Control Unit Error

15	Class Not Equal
16	Not Ready
17	Angular Position Error
18	Head Position Error
19	Invalid Address
20	Command Error
21	Data Transfer Error
22	Check Field Error
23	Word Parity Error

5F3 Data and Command Errors

5F3A Word Parity Error (Bit 23) -- This condition is set whenever the parity is incorrect on a 24-bit sequence in the data field of a record during a read operation.

5F3B Check Field Error (Bit 22) -- This bit is set whenever the check bits at the end of the record indicate that an error has been made in reading the record.

5F3C Data Transfer Error (bit 21) -- This bit is set when data being transferred from the Central Processing Unit to the Control Unit has incorrect parity.

5F3D Command Error (bit 20) -- This bit is set for the following conditions:

5F3D1 Incorrect parity for a command word transferred from the comuter.

5F3D2 Invalid command code.

5F3D3 A Go-No Chain EOM received while busy.

5F3D4 Go-Chain EOM receive while busy and no disconnect request waiting.

5F4 Addressing and Positioning Errors

5F4A Invalid Address (Bit 19) -- This bit is set when the disc address specified in a transfer command is invalid or a data transfer exceeds the cylinder.

5F4A1 A cylinder consists of all tracks on all surfaces that can be accessed from a single head position.

5F4B Head Position Error (Bit 18) -- This bit is set if the head array is not correctly positioned as determined by failure

to get track verification after 7 revolutions or incorrect track address in header word 1.

5F4C Angular Position Error (Bit 17) -- This bit is set when the angular position specified in the address does not match that read from header word 2, or if a parity error is detected in header word 2.

5F4D Illegal Word Count (Bit 12) -- This bit is set when the word count in a data transfer command exceeds 2048.

5F5 Miscellaneous Errors

5F5A Not Ready (bit 16) -- This bit is set if the control unit receives an information transfer command and the disc is not ready.

5F5B Class Compare Not Equal (bit 15) -- This bit is set if a class compare is requested and the record has a different class from the Information Transfer Command.

5F5C Control Unit Error (bit 14) -- This bit is set when timing or sequencing errors in the control unit prevent completion of the operation.

5F5D Map Error (bit 13) -- This bit is set when a data transfer crosses a page boundary and mapping is not legal.

6 The Display System

6A General Characteristics

6A1 The display system has general character, vector, and plotting capabilities.

6A1A Screen position is specified by 10 bits for horizontal and 10 bits for vertical, with a resolution of at least 500 line pairs in each dimension.

6A1A1 The time required for full screen deflection and settling to within 0.1 percent is approximately 15 microseconds.

6A1B The character generator can produce 128 characters but at present only 96 characters are implemented. The lower case codes act as nulls.

6A1B1 Characters and codes are given in Sec. 9.

6A1B2 All characters are written relative to a beam position at the lower left corner of the allotted character space.

6A1B3 Four character sizes are available with a range of sizes allowing from 32 to 128 characters per line.

6A1B4 In the print mode, the average time per character for a full screen of characters is approximately 12 microseconds.

6A1C The line generator is capable of drawing straight lines of varying type between specified endpoints.

6A1C1 In the line mode, the time required to draw a full screen line is approximately 25 microseconds; shorter lines take proportionally less time.

6B Display Controller Operation

6B1 EOM instructions for the display controller are:

1 Initiate
2 Pause
3 Restart
4 Reset

6B1A The EOM actions are:

6B1A1 The Initiate EOM starts command processing.

6B1A1A Processing will always start with the command addressed by the Unit Reference Cell at the time the EOM is executed.

6B1A1B An Initiate EOM directed to a busy unit will be ignored by the system.

6B1A2 The Pause EOM will cause the system to stop processing commands after the command in process at the time the EOM is received.

6B1A2A The system may be started again by the Restart EOM.

6B1A3 The Restart EOM will cause the system to continue processing (with the next command) after a pause or a failure interrupt.

6B1A3A A Restart EOM executed following a Pause EOM will cancel the pause request.

6B1A3B A Restart EOM directed to a unit that is disconnected (reset) will be ignored.

6B1A4 A Reset EOM will immediately stop all display processing and return the system to the reset state.

6B2 SKS codes for the display controller are .

1 Command Table Active
2 Unit Busy

6B2A The conditions sensed are as follows:

6B2A1 Command Table Active

6B2A1A The unit is in the process of interpreting a command.

6B2A1A1 When this status is detected the command table should not be changed by the program.

6B2A2 Unit Busy

6B2A2A The unit is processing commands.

6B2A2A1 The unit is not busy when in either the pause or disconnect state.

6B3 Command Table Processing

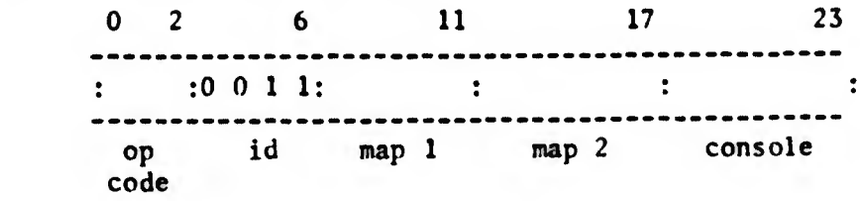
6B3A After an Initiate EOM, the display system will process the command table without further action by the program.

6B3B The address of the first word of the command being processed is always contained in the Unit Reference Cell.

6B3B1 This cell is updated on the successful completion of a command.

6B3C Each entry in the command table consists of two words with the following format:

6B3C1 First word:



6B3C1A Bits 0 - 2 contain the op code. Op-codes are

0 Disconnect
1 Pause
2 Start Display, Normal
3 Start Display, Step Mode
4 Branch
5 Branch Timed
6 NOP
7 Reset

6B3C1B Bits 3 - 6 contain a code to identify this as the first word of a command.

6B3C1C Bits 7 - 11 contain the relocation map for the first display page.

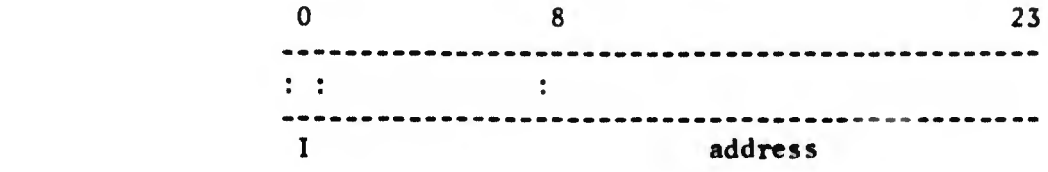
6B3C1D Bits 12 - 17 contain the relocation map for the second display page.

6B3C1D1 The highest order bit (Bit 12) of this map

indicates a legal map. If this bit is 0 the map will not be used and only one display page is available to the user.

6B3C1E The 6 bits (18 - 23) indicate which of the 6 CRTs (one or more) will be unblanked for this display list.

6B3C2 Second word:



6B3C2A Bit 0 is the interrupt bit.

6B3C2B Bits 1 - 7 are not used.

6B3C2C Bits 8 - 23 contain an address but may be ignored depending on the op-code.

6B3D Processing of op-codes is as follows:

6B3D1 Disconnect -- Command processing will immediately stop and the system will be reset. An Initiate EOM must be issued to initiate further processing.

6B3D2 Pause -- The system will stop processing commands until a Restart or Initiate EOM is received.

6B3D3 Start Display, Normal -- The system will display the identified display list on the selected console.

6B3D3A A 14-bit display-list address in user core is contained in the command address field. Relocation map 1 is always used to map this into an absolute 16-bit address.

6B3D4 Start Display, Step Mode -- The system will display only the single display list entry pointed to by the address field.

6B3D5 Branch -- The display system will jump to the location specified in the address field and continue to process commands.

6B3D6 Branch Timed -- The same as the Branch command,

except that the branch will not be executed unless the regeneration timer has run out.

6B3D6A For synchronous operation the branch will be executed on the first sync pulse after the timer has run out.

6B3D6B The regeneration timer is set for the desired regeneration rate and will be reset whenever a Branch Timed command is executed.

6B3D7 NOP -- Processing will go immediately to the next command.

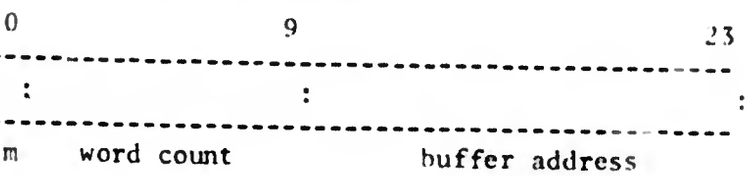
6B3D7A The interrupt bit is ignored. No interrupt will be given.

6B3D8 Reset -- This sends a reset signal to the display generator.

6B3E If the interrupt bit is set in any command (except NOP) a display interrupt is given after successful processing of that command.

6B4 Display-List Processing

6B4A The display list format is as follows:



6B4A1 The buffer address is the 14-bit address in user core of the first word of the display buffer.

6B4A2 The word count gives the length of the buffer.

6B4A3 The M bit indicate that mapping is not to be used for this buffer.

6B4B Each entry in the display list is processed in the following manner:

6B4B1 If the word count is not zero, the display buffer will be transmitted to the display generator.

6B4B1A If two display pages are available the buffer may begin in either page and may extend over the page boundary. The appropriate relocation map will be used.

6B4B1B If the M bit is set mapping is not applied to this buffer. The address field refers to an absolute address in lower core and only one display page is available.

6B4B1C If the buffer begins outside the display pages or extends beyond the display pages, an error condition will result.

6B4B2 If the word count is zero, this entry in the display list will be ignored and the system will proceed immediately to the next display list entry.

6B4B3 A display list entry containing a zero address terminates the display list.

6B4B3A If the display list extends beyond the display pages an error will result.

6B5 Errors in display commands will be detected and reported by the system.

6B5A These error conditions are as follows:

6B5A1 Illegal Command

6B5A1A The command identifier field of a command table entry is not correct, or the op-code field is not a legal op-code.

6B5A2 Display List Overflow

6B5A2A Display list extends beyond display pages.

6B5A3 Illegal Buffer Address

6B5A3A The buffer address given by the display list is not within the display pages.

6B5A4 Buffer Overflow

6B5A4A Display buffer extends beyond display pages.

6B5A5 Excessive Time

6B5A5A The processing of a command has exceeded the

maximum time allowed.

6B5A5A1 Maximum time will be adjustable and will normally be set to allow a frame of about 1200 characters for each command (about 6 milliseconds).

6B5B Action on error conditions is as follows:

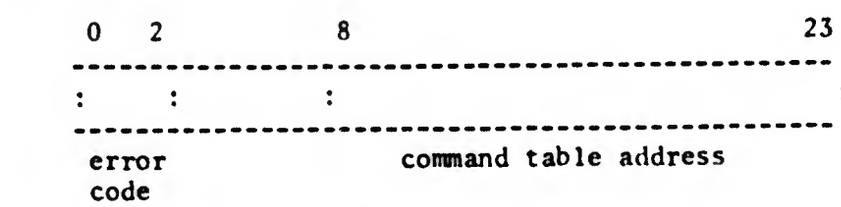
6B5B1 Processing action depends on the type of error.

6B5B1A For transmission parity, the processing will continue without interruption.

6B5B1B For all other errors command processing will terminate at the command causing the error, and may be restarted by a Restart EOM.

6B5B2 An error code indicating the type of error will be entered into the Unit Reference Cell.

6B5B2A The format for the Unit Reference Cell is as follows:



6B5B2A1 Error codes are

000	No Error (normal interrupt)
001	List Overflow
010	Buffer Overflow
011	Illegal Buffer Address
100	Illegal Command
101	Excessive Time

6B5B2A2 Bits 3 - 7 are unused.

6B5B2A3 Bits 8 - 23 contain the address of the command that resulted in the error.

6B5B3 An interrupt will be generated.

6C Display Instructions

6C1 The display system operates in two general states -- the control state and the normal state.

6C2 A 1 in the most significant bit position always indicates a control word. There are two types of control word:

6C2A Parameter and Mode Specification Words

6C2A1 The parameter specification words set parameters that are retained in the system until changed by a further parameter specification or by a system reset.

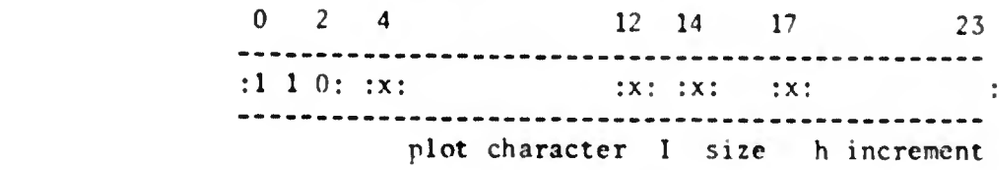
6C2A2 There are two parameter specification words: certain parameters can be set by either word.

6C2A3 For both words the "X" bit preceding the parameter designates whether that parameter is to be changed.

6C2A3A When this bit is set the associated parameter specification will be used.

6C2A3B When this bit is a 0 the parameter existing in the system will not be changed.

6C2A4 First Parameter Specification Word:



6C2A4A Bits 5 through 11 specify the plotting character to be used in plot modes.

6C2A4B Bit 13 specifies the intensity for all writing modes.

6C2A4B1 A 1 indicates high intensity and a 0 low intensity.

6C2A4C Bits 15 and 16 specify the character size used in print and plotting modes.

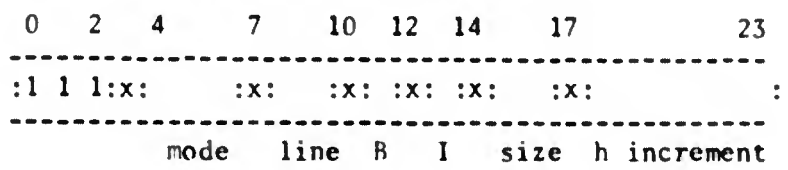
6C2A4C1 Size codes and corresponding approximate numbers of characters per line are

Code	Characters/Line

00	128
01	80
10	64
11	42

6C2A4D Bits 18 through 23 specify the horizontal increment to be used in the incremental plot mode and print mode.

6C2A5 Second Parameter Specification Word:



6C2A5A Bits 4 through 6 specify the modes for the normal state.

6C2A5A1 Codes and associated modes are

Code	Mode
000	Print
001	Print Italics
010	Incremental Plot, Absolute
011	Incremental Plot, Relative
100	Random Plot, Absolute
101	Random Plot, Relative
110	Line
111	Line Grid

6C2A5B Bits 8 and 9 specify the line type used in line writing modes.

6C2A5B1 Line types and codes are

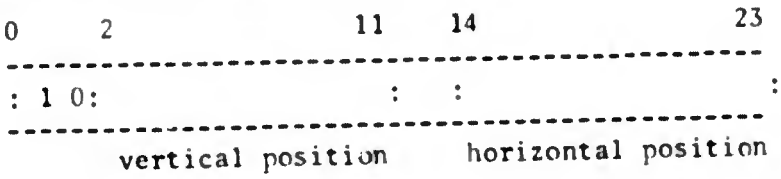
Code	Line Type
00	Solid Line
01	Dotted Line
10	Dashed Line
11	Dot-Dash Line

6C2A5C Bit 11 specifies blinking.

6C2A5C1 When blink is called for the unblank will be gated on and off at a manually adjustable rate (about 1 cycle/second) until the blink is reset by a parameter specification word or by a reset.

6C2A5D Other bits are as described for the first parameter specification word.

6C2B Position Specification Word:



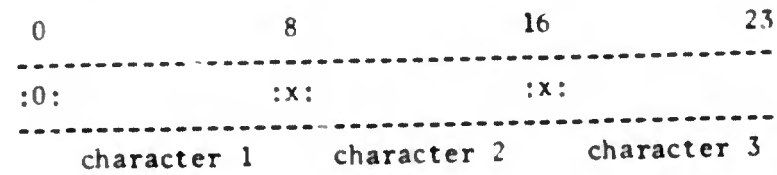
6C2B1 Bits 2 through 11 specify the absolute vertical position.

6C2B2 Bits 14 through 23 specify the absolute horizontal position.

6C2B3 Bits 12 and 13 are not used.

6C3 In the normal state, there are eight possible modes. The mode is changed by a parameter specification word or by a reset.

6C3A Print Mode:



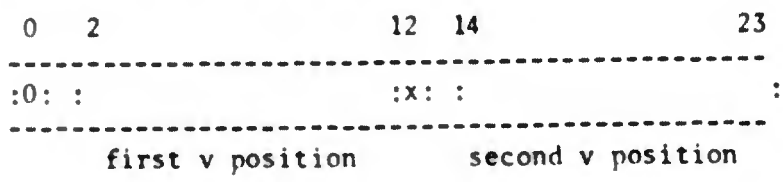
6C3A1 This mode is used for writing lines of text on the display in a typewriter-like fashion.

6C3A2 Characters are packed three to a word with automatic horizontal incrementing after each character is written. The horizontal increment is specified in the parameter specification word.

6C3A2A Exceeding the maximum number of characters in a line will cause the characters to "wrap around" and write over the first part of the line.

6C3A3 A 1 in bit positions 8 or 16 indicates that the remaining characters in the word are to be ignored.

6C3B Incremental Plot, Absolute:



6C3B1 In this mode, two plotting characters may be written for each 24-bit word.

6C3B1A Bits 2 through 11 specify the absolute vertical position of the first character to be plotted.

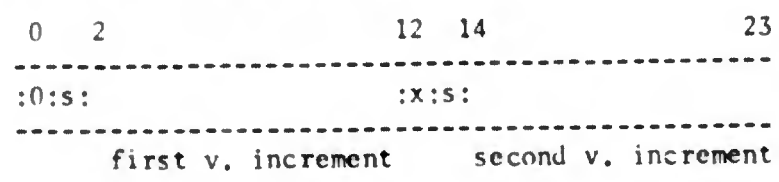
6C3B1B Bits 14 through 22 specify the absolute vertical position of the second character to be plotted.

6C3B2 The horizontal position is incremented after each character is plotted, by an amount specified in the parameter specification word.

6C3B3 A 1 in bit position 12 indicates that the second half of this word is to be ignored. The horizontal position will not be incremented for the second word half.

6C3B4 Bits 1 and 13 are not used.

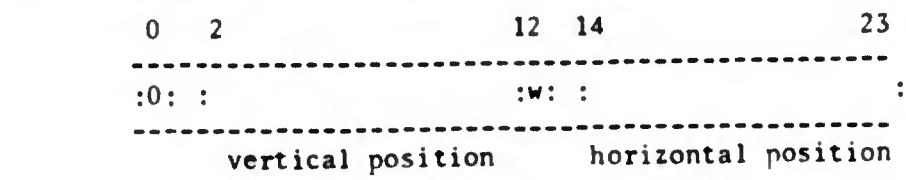
6C3C Incremental Plot, Relative:



6C3C1 This mode is identical to Incremental Plot, Absolute, except that vertical positions are specified relative to the previous beam position. Bits 1 and 13 are the sign bits.

6C3C2 In this and all other relative modes negative numbers are given in two's-complement form. That is, a negative change of one unit is called for by the binary number 11 111 111 111. 6c3c3 Relative positioning that exceeds the screen limits will result in "end-around" operation.

6C3D Random Plot, Absolute:



6C3D1 In this mode one plotting character may be written for each 24-bit word.

6C3D1A Bits 2 through 11 specify the absolute vertical position.

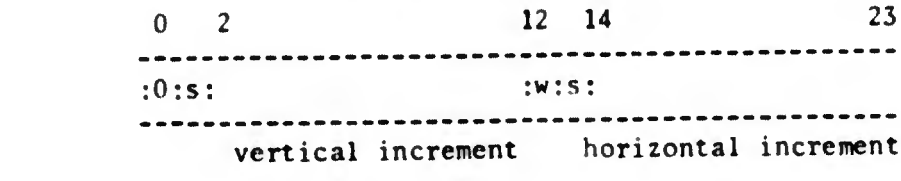
6C3D1B Bits 14 through 23 specify the absolute horizontal position.

6C3D2 Bit 12 indicates whether or not a character is to be written.

6C3D2A If Bit 12 is a 1, the designated plotting character will be written at the specified position.

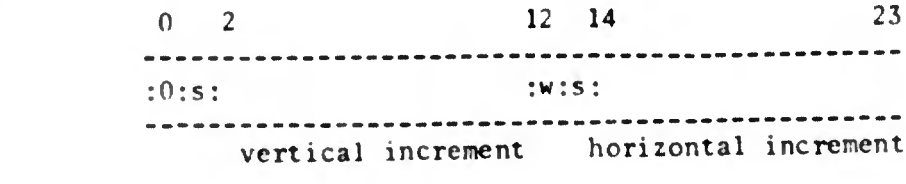
6C3D2B If Bit 12 is a 0, the beam position will be changed but no character will be written.

6C3E Random Plot, Relative:



6C3E1 This mode is identical to Random Plot, Absolute, except that vertical and horizontal positions are specified relative to the previous beam position. Bits 1 and 13 are the sign bits.

6C3F Line:



6C3F1 In this mode a line is written from the previous beam position to a new point specified relative to that beam position.

6C3F1A When a line is specified that exceeds the screen limits it is drawn in the direction and length indicated. (Some distortion may of course result outside of the normal working area.) Beam position is then updated in the normal end-around manner.

6C3F1B Bits 2 through 11 specify the change in vertical position, with Bit 1 indicating the sign.

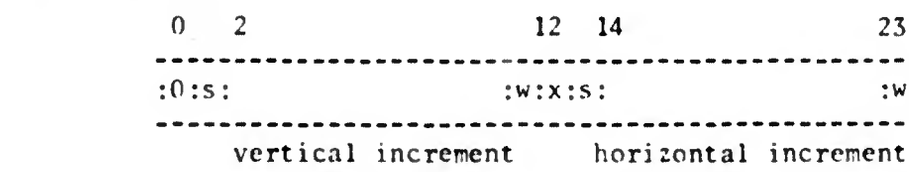
6C3F1C Bits 14 through 23 specify the change in horizontal position, with Bit 13 indicating the sign.

6C3F2 Bit 12 specifies whether or not a line is to be written.

6C3F2A If Bit 12 is a 1, a line of the type designated by the parameter specification word will be drawn and the beam position changed to correspond to the end of the line.

6C3F2B If Bit 12 is a 0, no line will be drawn but the beam position will be changed.

6C3G Line Grid:



6C3G1 This mode is provided to facilitate writing horizontal and vertical lines on the screen. One vertical and one horizontal line may be written for each 24-bit word. The accuracy has been reduced to 9 bits to accommodate this feature.

6C3G1A Bits 2 through 10 specify the change in vertical position for the first line, with Bit 1 indicating the sign.

6C3G1B Bit 11 indicates whether or not the line is to be drawn as a result of this vertical change.

6C3G1B1 If Bit 11 is a 1, a line with length and direction specified by the vertical change will be drawn from the previous beam position. The vertical beam position will then be changed to correspond to the end of the line.

6C3G1B2 If Bit 11 is a 0, no line will be drawn, but the vertical position will be changed.

6C3G1C In a similar manner, Bits 14 through 22 and Bit 13 specify the change in horizontal position with Bit 23 indicating whether or not a line is to be written.

6C3G1D The first word half or vertical line is always executed before the horizontal line.

7 Input Devices Controller (IDC)

7A The EOM instructions for the IDC are

1	Initiate
4	Disconnect

7A1 No URC is associated with the IDC. The 16-bit address of the Console Status Table (CST) will be wired into the IDC.

7A2 No error conditions are detected by the IDC.

7B Communication between the IDC and the software is through the CST.

7B1 At a regular sampling interval (determined by the IDC) the input devices are read and information relative to their state written into the CST. A block of "output" words is also read from memory by the IDC to control signals to the various consoles.

7B2 The format of the CST is

```
--------
1      1
1      1  Block 0, outputs, 16 words.
1      1
--------
1      1
1      1  Block 1, keyboard and keysets, 12 words.
--------
1      1
1      1  Block 2, switches, 12 words.
--------
1      1
1      1  Block 3, A-D converter (mice), 16 words.
1      1
--------
```

7B2A The order within each block corresponds in the obvious way to the numbering of the console stations.

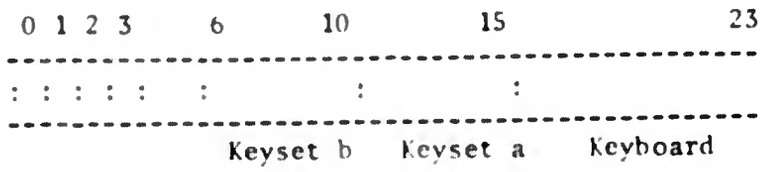
7B2B The word format within the blocks is as follows:

7B2B1 Outputs -- Bits 19-21 correspond to six possible output signals the software may send to the console.

7B2B1A An output signal is of a "one-shot" nature. To

control a signal that has a duration in time, such as a light, two signals are required: one to turn the light on, another to turn it off.

7B2B2 Keyboard-Keyset



7B2B2A Bits 16-23 contain the code of the character most recently received from the keyboard. Characters and codes are shown in Sec. 9.

7B2B2B Bits 11-15 contain the code of the character most recently received from Keyset a.

7B2B2C Bits 6-10 contain the code most recently recieved from Keyset b.

7B2B2D Bits 0-3 indicate when valid characters have been written.

7B2B2D1 Bit 3 is set to one by the hardware whenever a new keyboard character is written.

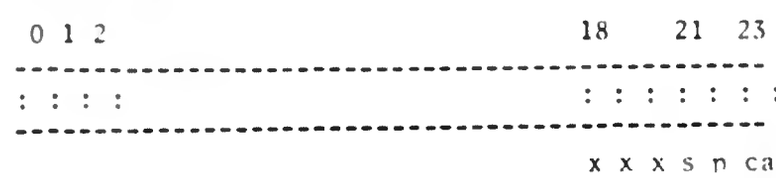
7B2B2D2 Bit 2 is set to one whenever a new character is written for Keyset a.

7B2B2D3 Bit 1 is set to one whenever a new Keyset b character is written.

7B2B2D4 Bit 0 is set to one whenever a character is written in any position.

7B2B2E The valid code remains in the CST for the sampling interval (approximatly 30 ms). During any sampling interval when a "valid character" bit is not set, the corresponding character location may contain garbage.

7B2B3 switches



7B2B3A The state of the console switches at each sampling period is indicated by Bits 21 - 23, and changes in the switches between two successive sampling periods are indicated by bits 18 - 20.

7B2B3A1 Bit 23 is the CA switch on the mouse and Bit 20 indicates a change in this switch.

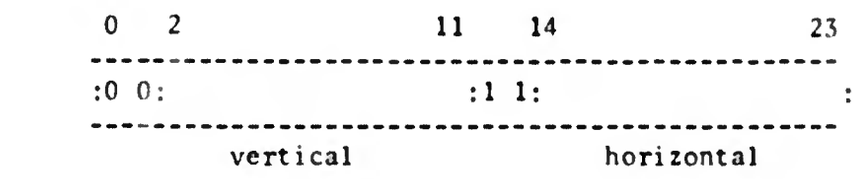
7B2B3A2 Bit 22 is the pointer switch on the mouse, with Bit 19 indicating a change.

7B2B3A3 Bit 21 is the spare button on the mouse, with Bit 18 indicating a change.

7B2B3A4 For each switchword, Bit 0 is set to one whenever any switch has changed.

7B2B3A5 In the last word of Block 2, Bit 0 is set if Block 1 contains any valid characters or if any of the console switches (Block 2) have changed state.

7B2B4 Mice:



7B2B4A Bits 2-11 contain the vertical coordinate from the A/D converter

7B2B4B Bits 14-23 contain the horizontal coordinate from the A/D converter

7B2B4C The 10-bit coordinates are positive, with origin at the lower left corner of the display.

7B2B4D Bits 0 and 12 are specified to eliminate the necessity for reformatting the coordinates before sending them to the display.

7B2B4E Bits 1 and 13 are "don't cares" to the display; however, Bit 1 is set to 0 and Bit 13 is set to 1.

7B3 After the last word of Block 2 has been written, if there have been any valid characters written in Block 1 or if any switch changes have been indicated in Block 2, a 205 interrupt is issued.

8 Line Printer

8A General Information

8A1 The printer is a Potter HSP-3502 chain printer with 96 printing characters and a printing speed of about 230 lines per minute.

8A2 The printer will accomodate paper widths from 2-1/2 to 18-1/2 inches. Character spacing is 10 per inch and line spacing is program selectable at 6 per inch or 10 per inch. The maximum number of characters per line is 132.

8A3 Characters and codes are shown in Sec. 9.

8B EOM and SKS Codes

8B1 The EOM codes are

20230106	Initiate
20230406	Reset

8B1A The Initiate EOM starts the printer with the word and character designated by the contents of the URC at the time the EOM is given.

8B1A1 The printer controller will continue to process the print buffer until an illegal character or end-of-buffer code is read, or until a Reset EOM is issued.

8B1A2 An Initiate EOM given while the printer is busy will be ignored.

8B1B The reset EOM will immediately terminate all printing and return the system to a reset state.

8B1B1 A Reset EOM given while the printer is disconnected will be ignored.

8B2 One SKS code is provided for the printer. The code is

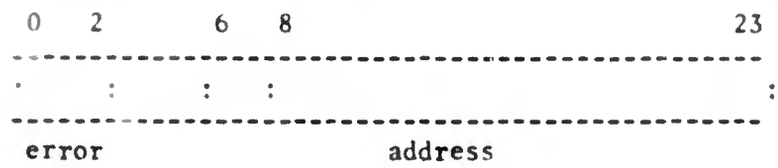
04030106	Skip on Ready

8B2A This SKS will skip if the printer is ready to begin operation. If the printer is not ready, the following actions occur:

8B2A1 The indicator lights on the printer begin to blink, indicating that attention is required.

8B2A2 An interrupt is issued when the printer is made ready.

8C The Unit Reference Cell associated with the printer system has the following format:



8C1 Bits 6 - 23 contain the absolute address of the first character of the line in the print buffer currently being printed.

8C1A Bits 8 - 23 denote the absolute word address.

8C1B Bits 6 - 7 indicate the character in the word.

8C1B1 A 00 code is the leftmost character. The 11 code is not used but will be interpreted as the leftmost character.

8C2 When error conditions are detected the error code is contained in Bits 0-3.

8C3 After a line has been successfuly printed the address in the unit reference is updated to point to the first character of the next line.

8D The print buffer is a contiguous sequence of words in core that is interpreted by the printer controller as three 8-bit characters per word.

8D1 Characters in the print buffer may be either data characters or control characters.

8D1A The control characters are

370	(SC1) Space on Channel 1 (6 lines/inch)
371	(SC4) Space on Channel 4 (10 lines/inch)
373	(NOP) No Operation
372	(NSP) No Spacing
374	(EJT) Top of Form
375	(EOB) End of Print Buffer
376	(EOL) End of Line

377 (NOP) No Operation

8D1A1 Control codes contained anywhere in a line control the spacing after that line.

8D1A1A The spacing option (SC1 or SC4) is stored by the controller and need not be included in each line.

8D1A1A1 When the controller is initiated (by an EOM) the option will be SC1.

8D1A1A2 Channel 1 will normally be set for single space and Channel 4 for double space.

8D1A1B The NSP code inhibits spacing only on the line in which it occurs.

8D1A1C Only one EJT code in a line will be recognized.

8D1A1C1 For a page eject operation the EJT must be followed by an EOL or EOB.

8D1A2 An EOL or EOB code will cause the current line to be printed with any characters already in the line left justified.

8D1A2A The maximum number of characters in a line is 132. When that number of characters have been read the current line will automatically be printed and the next character in the print buffer will be the first character in the next line.

8D1A3 An EOB code will generate an interrupt to the computer after the line is printed and any spacing action has been completed.

8D1A4 The number of lines on a page is normally set to 60 (controlled by the format tape). When the last line has been printed, an automatic page eject will position the paper at the top line of the next page.

8D1B Data characters are either printing characters or space. Characters and codes are given in Sec. 9.

8D1C Any character codes in the print buffer other than data characters or control characters are considered illegal codes and will result in an error.

8D2 Print buffers may be as large as desired, but no relocation mapping is provided. If a buffer is to extend across a page boundary the software system must ensure that the two pages are consecutive in memory.

8E Error Conditions

8E1 On the detection of any error, an interrupt is issued and the error code is written in the Unit Reference Cell.

8E2 The error conditions detected and p error codes are:

000	No Error
101	Illegal Character Code
110	Printer Not Ready
111	Excessive Time

8E2A Zeros in the error-code bits of the Unit Reference Cell after an interrupt indicate a normal interrupt (printer made ready or EOB).

8E2B The 101 code indicates that an illegal character has been detected in the print buffer.

8E2C The 110 code indicates printer off-line, paper out, or ribbon failure.

8E2D The 111 code indicates that in a normal printing operation excessive time has been required for printing a line.

8E2D1 The timer is normally set for 2.5 seconds. This error will indicate printer failures not detected by other printer error circuits.

9 Character Set

9A The following is the universal character set used with the AHI 940 and peripheral devices. Variations for particular devices are noted and explained below.

000	(a)	␣	(space)	100		•	(center dot)
001		!		101		a	
002		"		102		b	
003		#		102		c	
004		$		104		d	
005		%		105		e	
006		&		106		f	
007		'		107		g	
010		(		110		h	
011		)		111		i	
012		*		112		j	
013		+		113		k	
014		,		114		l	
015		-		115		m	
016		.		116		n	
017		/		117		o	
020		0		120		p	
021		1		121		q	
022		2		122		r	
023		3		123		s	
024		4		124		t	
025		5		125		u	
026		6		126		v	
027		7		127		w	
030		8		130		x	
031		9		131		y	
032		:		132		z	
033		;		133	(b)	‾	(overbar)
034		<		134	(b)	_	(underline)
035		=		135	(c)	(alternate mode)	
036		>		136	(c)	(command delete)	
037		?		137	(c)	(rubout)	
040		@		140			
041		A		141	(c)	(backspace)	
042		B		142			
043		C		143			
044		D		144	(c)	(command accept)	
045		E		145			
046		F		146			
047		G		147			
050		H		150			
051		I		151	(a)	→	(tab)
052		J		152			

(d) ! (d) ◆ (d) ␣→

053	K	153		
054	L	154		
055	M	155	(a)	↵ (carriage return)
056	N	156		
057	O	157		
060	P	160		
061	Q	161		
062	R	162		
063	S	163		
064	T	164		
065	U	165		
066	V	166		
067	W	167	(c)	(backspace word)
070	X	170		
071	Y	171		
072	Z	172		
073	[	173		
074	\	174		
075	]	175		
076	↑	176		
077	←	177	(e)	null
		200	(c)	(shift II)

9A1 (a) These symbols for spacing characters apply to the display only. They are optionally displayed under control of a separate input signal to the display system but cause horizontal incrementing at all times.

9A1A For the line printer the 000 code is a space, and codes 151 and 155 are illegal.

9A2 (b) These two characters are displayed and printed outside the normal character space and do not cause horizontal incrementing.

9A3 (c) Keyboard only. These characters do not display or print.

9A4 (d) Plotting character on line printer only

9A5 (e) Null character--no character displayed or printed, and no incrementing.

BLANK PAGE

APPENDIX C -- WIRELIST GENERATOR PROGRAM

1 Introduction

1A This appendix describes a computer program that supports the design and construction of digital equipment.

1A1 The program automates several of the clerical tasks associated with logical design, and performs diagnostic operations that are difficult without computer aid.

1A2 The program produces a set of listings that document a digital design. Some of the listings are structured to aid the maintenance and checkout operations, others are structured to aid the process of wiring the equipment, and another is used to validate the correctness of the wiring.

1B Experience in the use of this program has so far been as follows:

1B1 There is a significant reduction in number of errors in comparison with hand-prepared wirelists.

1B2 Initial checkout of equipment with the listings is relatively simple.

1B3 The program provides an easy means of recording changes and producing up-to-date documentation on the equipment.

1C An important aspect of the program is that it permits one to prepare input data with the on-line text-editing facility (NLTS).

1C1 The form of program input conforms to present high-level language standards; statements are free-field, and delimiters and key words are used to control translation. Since record boundaries are not significant, the input is essentially a stream of text, and thus compatible with text files that are prepared with NLTS.

1D The remainder of this appendix describes the characteristics of the program in greater detail.

1D1 Section 2 describes the background of the development of the program.

1D2 Section 3 describes the input language and the role of NLTS for maintaining design descriptions in this language.

1D3 Section 4 describes the functions of the program and the documentation that it produces.

1D4 Section 5 is a summary of experience in the use of the

program.

2 Background of Program Development

2A The construction of digital equipment requires a document known as a "wirelist." This is an ordered list of the wire connections that are to be made.

2A1 Since the standard document for the design of a piece of equipment is the logic schematic, and not the wirelist, it is necessary to translate the schematic into a wirelist.

2A2 When this translation is done by hand, it is normal for errors to appear in the wirelist, even after laborious cross-checking. Moreover, the hand-produced wirelist is not easily organized to facilitate the assembly of the unit, and if assembly is a hand process, additional errors are inevitable in spite of additional cross-checking.

2B The process of translating logic schematics to wirelists is purely mechanical, and easily automated.

2B1 A moderate-sized piece of equipment contains several thousand connections, so that hand translation of logic schematics to wirelists can take days or weeks of technician time.

2B2 When a wirelist generator program is used for the translation, the technician instead spends his time translating the schematic into machine-readable form.

2B2A This effort is comparable to the effort required to write the wirelist itself, or even somewhat less.

2B3 Once the design is in machine-readable form, many different processes can be carried out on it over and above the generation of the wirelist.

2B3A Among these processes are error-checking, generation of several different types of documentation, and first-order optimization of the ordering of connections in the wirelist.

2B4 The cost of computer processing to produce these results is far smaller than the saving in both time and expense.

2C Wirelist programs are actually only part of the story. Within the current state of the art, it is possible to produce diagnostic test schedules, fault directories, and logic schematic drawings, and to automate the layout of components in the backplane or on circuit cards.

2D Wirelist programs have been used for many years, but are not available in the public domain because of proprietary status. For that reason, and because of a desire to use the wirelist program in conjunction with NLTS, a program development effort was undertaken at SRI.

2D1 Many of the characteristics of the resulting program are similar to those of its predecessors (e.g., the THULE system used by Lockheed Missiles and Space Company, and the ADD system used by Philco-Ford), but the program has more advanced characteristics than earlier wirelist programs. Some of these features are described in the following sections.

3 The Input Language

3A Input data to the program is oriented to the description of logic gates instead of the description of pins on the gates.

3A1 This results in a reduction of 50 percent or more in the number of characters required to describe a design.

3A2 The free-field form of input is ideally suited to on-line editing and much less error-prone than conventional fixed-field input data.

3A3 Finally, the input language is such that a logic designer can use it with ease, instead of adapting to conventions that are machine-oriented rather than design-oriented.

3B There are four types of information that the designer must give to the wirelist program. They are as follows:

3B1 The backplane, and the conventions used for addressing pins in the backplane

3B2 The catalog of pluggable logic units

3B3 The catalog of logic gates that are mounted on the pluggable units

3B4 The logic schematic of the design.

3C In most production environments, the first three items remain relatively constant from design to design.

3C1 The purpose of using catalogs and backplane descriptions is to save the designer the effort of retrieving data from the catalogs. Since the program can perform the retrieval when the catalog is supplied, the designer need not specify the information

in the design description. This function of the catalog in the program will become more clear as the language is described.

3D The backplane description is a single statement that gives the levels of interconnection in the backplane and the scheme for specifying pin addresses on the backplane. For example:

```
CHASSIS CAGE[1:10](CARD[A:Z](PIN[1:20]));
```

3D1 This statement states that a backplane has ten cages, labeled 1 through 10, each of which has card slots labeled A through Z, and each card slot has 20 pins, labeled 1 through 20.

3D1A "CHASSIS" is a key word that the program recognizes.

3D1B The names "CAGE," CARD," and "PIN" are names supplied by the designer.

3D1C A typical pin in the backplane described by the statement above is specified by giving the cage, card, and pin designators in that order. Hence an address might be (5,C,10) which specifies Cage 5, Card C, Pin 10.

3E The catalog of logic gates is specified by giving the number of inputs, the loading of each input, and the fanout capability of the output. For example, a two-input NAND is specified by the statement:

```
GATE N2(A,B), LOAD=1, FANOUT=8;
```

3E1 This states that there are two inputs, "A" and "B," that each input draws one unit load, and that an N2 gate can drive eight unit loads.

3F This is the simplest type of gate description. In the general case, a gate is an iterative collection of stages, with some additional inputs and outputs that do not fit the iterative structure. To facilitate the description of these gates, a prototype stage is described just once and the number of stages is given. The special inputs and outputs are described individually.

3G The description of pluggable units follows the format of gate descriptions.

3G1 Each gate on a pluggable unit is described separately, and the pins to which the gate inputs and outputs are connected are specified.

3G2 A unique index is given to each gate, and that index serves

to locate signals on cards. Thus, the designer need only specify that a signal comes from a given gate on a particular card and the program will supply the proper pin connections.

3G3 An example of a pluggable unit specification is:

```
LOGICAL CARD C100=
1 :=N2<1> (2,3),
4 :=N2<2> (5,6);
```

3G3A Here we have specified that a C100 card has two N2 gates on it; the first has output on Pin 1 and inputs on Pins 2 and 3, while the second has output on Pin 4, with inputs on Pins 5 and 6.

3H The logic schematic can be specified without designating the absolute physical locations of the gates and pins. This allows the program to process schematics and perform diagnostic functions before the design has been laid out physically on a backplane.

3H1 The form of specification is to give each gate a name, and to specify the names of the gates that are connected to the input of each gate. A typical statement of this type is:

```
YC :=N2(YA,YB);
```

3H1A The output signal name appears to the left of the symbol ":=", and this is followed by the name of the gate.

3H1B The input signals appear in parentheses and the semicolon terminates the statement. This is similar in appearance to the gate declaration for N2, except that the reserved word "GATE" replaces the output signal name and the ":=". The symbol ":=" is interpreted to mean "is produced by."

3I There are several means for specifying the physical location of signals in the backplane.

3I1 One such way is to merge a set of location specifications with a set of signal specifications of the type shown above.

3I2 Another method is to give the physical address of a gate in the same statement that gives the names of the inputs to the gate. Thus, the statement

```
UC ON 2 AT (2,A)
```

can be merged with the signal-definition statement above to specify that UC is the second gate on the card plugged into

location (2,A). Alternatively, we could have written

```
YC := N2(YA,YB) ON 2 AT (2,A);
```

3I3 The two methods are equivalent. In either event, it is also necessary to specify that a card is to be plugged into location (2,A), and this is done by the statement

```
C100 AT (2,A);
```

3J This covers essentially all the main features of the input language structure. It is important to mention that the program is insensitive to the order of appearance of statements, except that the statements that describe the card catalog must appear before the statements that describe a schematic diagram. Thus, if the medium for source input is a punched-card deck, the deck can be shuffled without affecting the program's behavior.

3K Earlier we remarked that a considerable saving in input characters is achieved by using a gate-oriented description instead of a pin-oriented description.

3K1 In practice, even higher-level descriptions are useful and powerful to have in a language.

3K2 One such tool is the macro, which functions for wirelist programs in much the same way that a macro functions in assembly languages.

3K3 Although the language was designed so that macros could be incorporated into it, they have not yet been implemented. The reason for this is that much of the power of macros has been realized through the use of an on-line text-editing facility.

3L This facility has been used extensively as a powerful aid for generating and maintaining the files of input data.

3L1 The rapid on-line search and edit facilities make it easy to find and modify statements that need to be changed.

3L2 Through the use of the "pattern matcher," extensive checking and correcting of the syntax can be done on line before the input data is submitted to the wirelist program.

3L3 Furthermore, the text-structuring features are a convenient way to organize the input data into sections and include comment statements where needed.

4 The Wirelist Program

4A The similarity of the input language to ALGOL is no accident. The input compiler is an adaptation of an ALGOL compiler, is written in ALGOL, and is compiled by the same compiler from which it is adapted. The machine on which it runs is the Burroughs B-5500.

4B The wirelist program is composed of two functional blocks--an input compiler and a file processor. The interface between the blocks is an intermediate file that contains the design information.

4B1 The input compiler creates a symbol table of the catalog description as it is scanned, thereby taking advantage of the symbol table processing capability that is characteristic of compilers.

4B1A As signals are processed, information from the symbol table is retrieved and inserted in records that are output to an intermediate disc file.

4B1B The disc records bear a striking similarity to the conventional fixed-field input to wire-list programs, with the major difference that each record contains more information than is usually input from punched cards. One record is written for each pin, so that one input statement may cause several records to be generated.

4B1C Each record gives an internal name to the pin it describes and contains the name of the signal at the pin and the name of the signal on the output of the gate to which the pin is connected. It also contains the name of the gate, the name of the pluggable card, the symbolic address, and other information that is valuable for design documentation.

4B2 The file is processed by a succession of sorting operations. At the close of each sort, some processing is done on the file and a listing of the results is produced. The listings that can be produced are given below.

4B2A A card-placement listing--This lists the slots in the backplane and the cards associated with each slot, arranged lexicographically by card-slot location.

4B2B A load-check listing--This lists every gate output and all of the inputs driven by the output, ordered lexicographically by signal name. Allowable fanout is given for the output of each gate, and the loading of each input is given. The system automatically checks for overloads, absent sources, absent loads, and multiply-defined names.

4B2C A signal input list--This lists every signal output and

all signals that are inputs to the gate that produces each signal, ordered lexicographically by signal-output name. In addition, this listing includes the pin assignment for each output signal.

4B2D A master-term list--This lists the wire chains to be made, with each chain arranged lexicographically by backplane address, and the chains arranged lexicographically by signal name.

4B2E A pin listing--This lists the signal name associated with each pin on the backplane, arranged lexicographically by pin location. The listing also gives the number of wire wraps on each pin. This is used to check a backplane after wiring.

4B2F Multilevel wire lists--These list portions of the wire chains ordered lexicographically by signal name and backplane location to facilitate orderly wiring and assembly of the backplane. The lowest-level listing gives the interpin connections for each card, ordered lexicographically by card slot. The next level gives the intercard chains for each cage, ordered lexicographically by cage and then by signal name. The next level is an intercage list, and further levels are possible up to five levels.

4B2G Tallies of gates used--The system can give a tally of the number of gates of each type that are used. Tallies are kept for the entire design and for submodules that are identified by the user.

4C The set of listings that can be produced by the program resembles the output of other wirelist programs. A simple set of controls allows the user to request particular listings, so that the program does not have to perform all processing on every run. Some of the listings can be produced when the file is incomplete. An important example of this capability is the ability to perform load checks when the addresses of the signals are not specified.

5 User Experience

5A Three different modules of digital equipment have been built and checked out using the wirelist program, and a fourth unit is under development currently. The experience to date has shown the value of the program in two important areas.

5A1 Wiring errors were reduced from 5 percent to 0.1 percent as a result of the computer processing. This accounted for a great reduction in the time required to check out a unit.

5A1A These figures are based on a comparison of similar logic systems hand-wired by various technicians. Although the technicians had similar experience and capabilities, individual differences may have accounted for some of the dfference in error rates.

5A2 The effort of documentation was reduced. The output listings that show the wiring chains and the inputs to gates provide sufficient documentation for capturing the state of the design. These listings together with logic sketches are adequate for design debugging. The expense of obtaining accurately drafted, finished logic schematics was avoided without compromising either the accuracy of the documentation or its utility for maintenance and checkout.

5B The program language is extremely easy to learn, and was mastered in a few hours by the logic designers who used it. The wirelist program was in a state of flux during the processing of the first few designs, which accounted for a few minor problems. By the time the most recent design was processed, the program operation was extremely smooth and the user was well acquainted with the use of all of its facilities.

5C The computer cost for program operation is roughly 8 to 10 cents per conductor, with up to five iterations through the program to eliminate all errors. Dollar costs for designs run from $100 to $300 depending on the size of the design. The saving in checkout time is many times this cost.

5D In summary, the wirelist program functions in several capacities in the support of digital design. It is used to improve production techniques, quality control, and documentation, and for maintenance and checkout assistance. Not to be overlooked in importance is the ability to capture a design in a text file that can be manipulated in the on-line interactive environment of NLTS.

BLANK PAGE

APPENDIX D -- TREE META: Introduction

1 Terms such as "metalanguage" and "metacompiler" have a variety of meanings. Their usage within this report, however, is well defined.

1A "Language," without the prefix "meta," means any formal computer language. These are generally languages like ALGOL or FORTRAN. Any metalangauge is also a language.

1B A compiler is a computer program that reads a formal-language program as input and translates that program into instructions that may be executed by a computer. The term "compiler" also means a listing of the instructions of the compiler.

1C A language that can be used to describe other languages is a metalanguage. English is an informal, general metalanguage that can describe any formal language. Backus-Naur Form or BNF (Nurl) is a formal metalanguage used to define ALGOL. BNF is weak, for it describes only the syntax of ALGOL, and says nothing about the semantics or meaning. English, on the other hand, is powerful, yet its informality prohibits its translation into computer programs.

1D A metacompiler, in the most general sense of the term, is a program that reads a metalanguage program as input and translates that program into a set of instructions. If the input program is a complete description of a formal language, the translation is a compiler for the language.

2 The broad meaning of the word "metacompiler," the strong, divergent views of many people in the field, and our restricted use of the word necessitate a formal statement of the design standards and scope of Tree Meta.

2A Tree Meta is built to deal with a specific set of languages and an even more specific set of users. This project, therefore, adds to the ever-increasing problem of the proliferation of machines and languages, rather than attempting to reduce it. There is no attempt to design universal languages, or machine independent languages, or any of the other goals of many compiler-compiler systems.

2B Compiler-compiler systems may be rated on two almost independent features: the syntax they can handle and the features within the system that ease the compiler-building process.

2B1 Tree Meta is intended to parse context-free laguages using limited backup. There is no intent or desire on the part of the users to deal with such problems as the FORTRAN "continue" statement, the PL/I "enough ends to match," or the ALGOL "is it procedure or is it a variable" question. Tree Meta is only one part of a system-building technique. There is flexibility at all levels of the system and the design philosophy has been to take

the easy way out rather than fight old problems.

2B2 Many of the features considered necessary for a compiler-compiler system are absent in Tree Mea. Such things as symbol-tables that handle ALGOL-style blocks and variable types are not included. Neither are there features for multidimensional subscripts or higher level macros. These features are not present because the users have not yet needed them. None, however, would be difficult to add.

2B3 Tree Meta translates directly from a high-level language to machine code. This is not for the faint of heart. There is a very small number of users (approximately 3); all are machine-language coders of about the same high level of proficiency. The nature of the special-purpose languages dealt with is such that general formal systems will not work. The data structures and operations are too diverse to produce appropriate code with current state-of-the-art formal compiling techniques.

3 There are two classes of formal-definition compiler-writing schemes.

3A In terms of usage, the productive or synthetic approach to language definition is the most common. A productive grammar consists primarily of a set of rules that describe a method of generating all the possible strings of the language.

3B The reductive or analytic technique states a set of rules that describe a method of analyzing any string of characters and deciding whether that string is in the language. This approach simultaneously produces a structure for the input string so that code may be compiled.

3C The metacompilers are a combination of both schemes. They are neither purely productive nor purely reductive, but merge both techniques into a powerful working system.

4 The metacompiler class of compiler-compiler systems may be characterized by a common top-down parsing algorithm and a common syntax. These compilers are expressible in their own language, whence the prefix "meta."

4A The following is a formal discussion of top-down parsing algorithms. It relies heavily on definitions and formalisms which are standard in the literature and may be skipped by the lay reader. For a language L, with vocabulary V, nonterminal vocabulary N, productions P, and head S, the top-down parse of a string u in L starts with S and looks for a sequence of productions such that S⇒u (S produces u).

APPENDIX D -- TREE META: Introduction

4A1 Let

```
V = [E, T, F, +, *, (, ); X ]
N = [E, T, F]
P = [E ::= T / T + F
     T ::= F / F * T
     F ::= X / ( E )
L = (V,N,P,E)
```

4A2 The following intentionally incomplete ALGOL procedures will perform a top-down analysis of strings in L.

4A2A boolean procedure E; E := if T then (if issymbol('+') then E else true) else false; comment issymbol (arg) is a Boolean procedure that compares the next symbol in the input string with its argument, arg. If there is a match the input stream is advanced;

4A2B boolean procedure T; T := if F then (if issymbol('*') then T else true) else false;

4A2C boolean procedure F; F := if issymbol('X') then true else if issymbol('(') then (if E then (if issymbol(')') then true else false) else false) else false;

4A3 The left-recursion problem can readily be seen by a slight modification of L. Change the first production to

```
E ::= T / E + T
```

and the procedure for E in the corresponding way to

```
E := if T then true else if E ....
```

4A3A Parsing the string "X+X", the procedure E will call T, which calls F, which tests for "X" and gives the result "true." E is then true but only the first element of the string is in the analysis, and the parse stops before completion. If the input string is not a member of the language, T is false and E loops infinitely.

4A3B The solution to the problem used in Tree Meta is the arbitrary number operator. In Tree Meta the first production could be

```
E ::= T$( "+" T)
```

where the dollar sign and the parentheses indicate that the quantity can be repeated any number of times, including 0.

4A3C Tree Meta makes no check to ensure that the compiler it is producing lacks syntax rules containing left recursion. This problem is one of the more common mistakes made by

inexperienced metalanguage programmers.

4B The input language to the metacompiler closely resembles BNF. The primary difference between a BNF rule

<go to> ::= go to <label>

and a metalanguage rule

GOTO = "GO" "TO" .ID;

is that the metalanguage has been designed to use a computer-oriented character set and simply delimited basic entities. The arbitrary-number operator and parenthesis construction of the metalanguage are lacking in BNF. For example:

TERM = FACTOR $(("*" / "/" / "") FACTOR);

is a metalanguage rule that would replace 3 BNF rules.

4C The ability of the compilers to be expressed in their own language has resulted in the proliferation of metacompiler systems. Each one is easily bootstrapped from a more primitive version, and complex compilers are built with little programming or debugging effort.

5 The early history of metacompilers is closely tied to the history of SIG/PLAN Working Group 1 on Syntax Driven Compilers. The group was started in the Los Angles area primarily through the effort of Howard Metcalfe (Schmidt1).

5A In the fall of 1962, he designed two compiler-writing interpreters (Metcalf1). One used a bottom-to-top analysis technique based on a method described by Ledley and Wilson (Ledley1). The other used a top-to-bottom approach based on a work by Glennie (Glennie1) to generate random English sentences from a context-free grammar.

5B At the same time, Val Schorre described two "metamachines"--one generative and one analytic. The generative machine was implemented, and produced random algebraic expressions. Schorre implemented Meta I the first metacompiler, on an IBM 1401 at UCLA in January 1963 (Schorre1). His original interpreters and metamachines were written directly in a pseudo-machine language. Meta I, however, was written in a higher-level syntax language able to describe its own compilation into the pseudo-machine language. Meta I is described in an unavailable paper given at the 1963 Colorado ACM conference.

5C Lee Schmidt at Bolt, Beranek, and Newman wrote a metacompiler in March 1963 that utilized a CRT display on the time-sharing PDP-1 (Schmidt2). This compiler produced actual machine code rather than interpretive code and was partially bootstrapped from Meta I.

6 Schorre bootstrapped Meta II from Meta I during the Spring of 1963 (Schorre2). The paper on the refined metacompiler system presented at

the 1964 Philadelphia ACM conference is the first paper on a metacompiler available as a general reference. The syntax and implementation technique of Schorre's system laid the foundation for most of the systems that followed. Again the system was implemented on a small 1401, and was used to implement a small ALGOL-like language.

7 Many similar systems immediately followed.

7A Roger Rutman of A. C. Sparkplug developed and implemented LOGIK, a language for logical design simulation, on the IBM 7090 in January 1964 (Rutman1). This compiler used an algorithm that produced efficient code for Boolean expressions.

7B Another paper in the 1964 ACM proceedings describes Meta III, developed by Schneider and Johnson at UCLA for the IBM 7090 (Schneider1). Meta III represents an attempt to produce efficient machine code for a large class of languages. It was implemented completely in assembly language. Two compilers were written in Meta III--CODOL, a compiler-writing demonstration compiler, and PUREGOL, a dialect of ALGOL 60. (It was pure gall to call it ALGOL). The rumored METAFORE, able to compile full ALGOL, has never been announced.

7C Late in 1964, Lee Schmidt bootstrapped a metacompiler from the PDP-1 to the Beckman 420 (Schmidt3). It was a logic equation generating language known as EQGEN.

8 Since 1964, System Development Corporation has supported a major effort in the development of metacompilers. This effort includes powerful metacompilers written in LISP which have extensive tree-searching and backup capability (Book1) (Book2).

9 An outgrowth of one of the Q-32 systems at SDC is Meta 5 (Oppenheim1) (Schaffer1). This system has been successfully released to a wide number of users and has had many string-manipulation applications other than compiling. The Meta 5 system incorporates backup of the input stream and enough other facilities to parse any context-sensitive language. It has many elaborate push-down stacks, attribute setting and testing facilities, and output mechanisms. The fact that Meta 5 successfully translates JOVIAL programs to PL/1 programs clearly demonstrates its power and flexibility.

10 The LOT system was developed during 1966 at Stanford Research Institute and was modeled very closely after Meta II (Kirkley1). It had new special-purpose constructs allowing it to generate a compiler which would in turn be able to compile a subset of PL/1. This system had extensive statistic-gathering facilities and was used to study the characteristics of top-down analysis. It also embedded system control, normally relegated to control cards, in the metalanguage.

APPENDIX D -- TREE META: Introduction

11 The concept of the metamachine originally put forth by Glennie is so simple that three hardware versions have been designed and one actually implemented. The latter at Washington University in St. Louis. This machine was built from macromodular components and has for instructions the codes described by Schorre (Schorre2).

12 A metaprogram is a set of metalanguages rules. Each rule has the form of a BNF rule, with output instructions embedded in the syntactic description.

12A The Tree Meta compiler converts each of the rules to a set of instructions for the computer.

12B As the rules (acting as instructions) compile a program, they read an input stream of characters one character at a time. Each new character is subjected to a series of tests until an appropriate syntactic description is found for that character. The next character is then read and the rule testing moves forward through the input.

13 The following four rules illustrate the basic constructs in the system. They will be referred to later by the reference numbers R1A through R4A.

```
R1A     EXP = TERM ("+" EXP / "-" EXP / .EMPTY);

R2A     TERM = FACTOR $("*" FACTOR / "/" FACTOR);

R3A     FACTOR = "-" FACTOR / PRIM;

R4A     PRIM = .ID / .NUM / "(" EXP ")";
```

13A The identifier to the left of the initial equal sign names the rule. This name is used to refer to the rule from other rules. The name of rule R1A is EXP.

13B The right part of the rule--everything between the initial equal sign and the trailing semicolon--is the part of the rule which effects the scanning of the input. Five basic types of entities may occur in a right part. Each of the entities represents some sort of a test which results in setting a general flag to either "true" or "false".

13B1 A string of characters between quotation marks (") represents a literal string. These literal strings are tested against the input stream as characters are read.

13B2 Rule names may also occur in a right part. If a rule is processing input and a name is reached, the named rule is invoked. R3A defines a FACTOR as being either a minus sign followed by a FACTOR, or just a PRIM.

13B3 The right part of the rule FACTOR has just been defined as "a string of elements," "or" "another string of elements." The

"or's" are indicated by slash marks (/) and each individual string is called an alternative. Thus, in the above example, the minus sign and the rule name FACTOR are two elements in R3A. These two elements make up an alternative of the rule.

13B4 The dollar sign is the arbitrary number operator in the metalanguage. A dollar sign must be followed by a single element, and it indicates that this element may occur an arbitrary number of times (including zero). Parentheses may be used to group a set of elements into a single element as in R1A and R2A

13B5 The final basic entities may be seen in rule R4A. These represent the basic recognizers of the metacompiler system. A basic recognizer is a program in Tree Meta that may be called upon to test the input stream for an occurrence of a particular entity. In Tree Meta the three recognizers are "identifier" as .ID, "number" as .NUM, and "string" as .SR. There is another basic entity tha is treated as a recognizer but does not look for anything. It is .EMPTY and it always returns a value of "true."

14 Suppose that the input stream contains the string X+Y when the rule EXP is invoked during a compilation.

14A EXP first calls rule TERM, that calls FACTOR, that tests for a minus sign. This test fails and FACTOR then tests for a plus sign and fails again. Finally FACTOR calls PRIM, that tests for an identifier. The character X is an identifier; it is recognized and the input stream advances one character.

14B PRIM returns a value of "true" to FACTOR, which in turn returns to TERM. TERM tests for an asterisk and fails. It then tests for a slash and fails. The dollar sign in front of the parenthesized group in TERM, however, means that the rule has succeeded because TERM has found a FACTOR followed by zero occurrences of "asterisk FACTOR" or "slash FACTOR." Thus TERM returns a "true" value to EXP. EXP now tests for a plus sign and finds it. The input stream advances another character.

14C EXP now calls on itself. All necessary information is saved so that the return may be made to the right place. In calling on itself, it goes through the sequence just described until it recognizes the Y.

14D Thinking of the rules in this way is confusing and tedious. It is best to think of each rule separately. For example: one should think of R2A as defining a TERM to be a series of FACTORs separated by asterisks and slashes and not attempt to think of all the possible things a FACTOR could be.

APPENDIX D -- TREE META: Basic Syntax

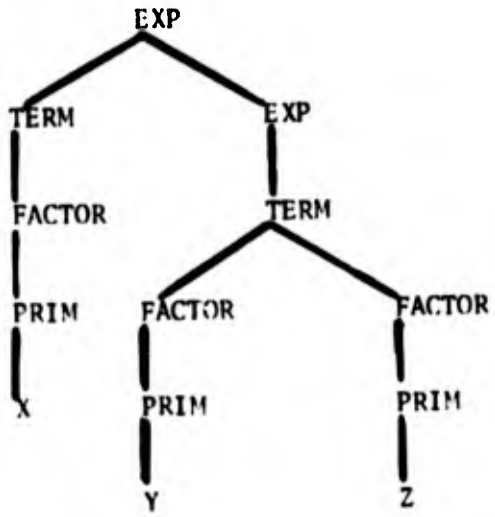
15 Tree Meta is different from most metacompiler systems in that it builds a parse tree of the input stream before producing any output. Before we describe the syntax of node generation, let us first discuss parse trees.

15A A parse tree is a structural description of the input stream in terms of the given grammar.

15A1 Using the four rules above, the input stream

X+Y*Z

has the following parse tree

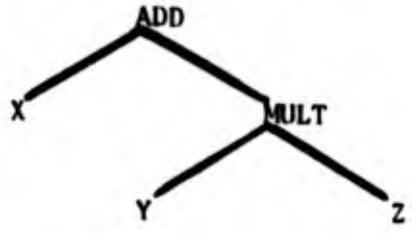


15A2 In this tree each node is either the name of a rule or one of the primary entities recognized by the basic recognizer routines.

15A3 In this tree there is a great deal of subcategorization. For example, Y is a PRIM, which is a FACTOR, which is the left member of a TERM. This degree of subcategorization is generally undesirable.

15B The tree produced by the metacompiler program is simpler than the one above, yet it contains sufficient information to complete the compilation. .

15B1 The parse tree actually produced is



15B2 In this tree the names of the nodes are not the rule names of the syntactic definitions, but rather the names of rules that will be used to generate the code from the tree.

15B3 The rules that produce the above tree are the same as the four previous rules with new syntax additions to perform the appropriate node generation. The complete rules are:

```
R1B     EXP = TERM ("+" EXP :ADD/ "-" EXP :SUB) [2] .EMPTY);

R2B     TERM = FACTOR $(("*" FACTOR :MULT/ "/" FACTOR :DIVD)
        [2]);

R3B     FACTOR = "-" FACTOR :MINUS[1] / PRIM;

R4B     PRIM = .ID / .NUM / "(" EXP ")";
```

15C As these rules scan an input stream, they perform just like the first set. As the entities are recognized, however, they are stored on a push-down stack until the node-generation elements remove them to make trees. We will step through these rules with the same sample input stream:

```
X+Y*Z
```

15C1 EXP calls TERM, which calls FACTOR, which calls PRIM, which recognizes the X The input stream moves forward and the X is put on a stack.

15C2 PRIM returns to FACTOR, which returns to TERM, which returns to EXP. The plus sign is recognized and EXP is again called. Again EXP calls TERM, which calls FACTOR, which calls PRIM< which recognizes the Y. The input stream is advanced, and Y is put on the push-down stack. The stack now contains Y X, and the next character on the input stream is the asterisk.

15C3 PRIM returns to FACTOR, which returns to TERM. The asterisk is recognized and the input is advanced another character.

15C4 The rule TERM now calls FACTOR, which calls PRIM, which recognizes the Z, advances the input stream, and puts the Z on the push-down stack.

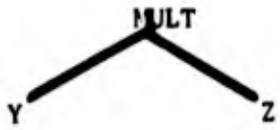
15C5 The :MULT in now processed. This names the next node to be put in the tree. Later we will see that in a complete metacompiler program there will be a rule named MULT which will be processed when the time comes to produce code from the tree. Next, the [2] in the rule TERM is processed. This tells the system to construct a portion of a tree. The branch is to have two nodes, and they are to be the last two entities recognized (they are on the stack). The name of the branch is to be MULT, since that was the last name given. The branch is constructed and the top two items of the stack are replaced by the new node of the tree.

15C5A The stack now contains

MULT

X

15C5B The parse tree is now



15C5C Notice that the nodes are assembled in a left-to-right order, and that the original order of recognition is retained.

15C6 Rule TERM now returns to EXP which names the next node by executing the :ADD -- i.e., names the next node for the tree. The [2] in rule EXP is now executed. A branch of the tree is generated that contains the top two items of the stack and whose name is ADD. The top two items of the stack are removed, leaving it as it was initially, empty. The tree is now complete, as first shown, and all the input has been passed over.

16 The unparsing rules have two functions: they produce output and they test the tree in much the same way as the parsing rules test the input stream. This testing of the tree alows the output to be based on the deep structure of the input, and hence better output may be produced.

APPENDIX D -- TREE META: Basic Syntax

16A Before we discuss the node-testing features, let us first describe the various types of output that may be produced. The following list of output-generation features in the metacompiler system is enough for most examples.

16A1 The output is line-oriented, and the end of a line is determined by a carriage return. To instruct the system to produce a carriage return, one writes a backslash (upper-case L on a Teletype) as an element of an unparse rule.

16A2 To make the output more readable, there is a tab feature. To put a tab character into the output stream, one writes a comma as an element of an output rule.

16A3 A literal string can be inserted in the output stream by merely writing the literal string in the unparse rule. Notice that in the unparse rule a literal string becomes output, while in the parse rules it becomes an entity to be tested for in the input stream. To output a line of code which has L as a label, ADD as an operation code, and SYS as an address, one would write the following string of elements in an unparse rule:

```
"L" , "ADD" , "SYS"
```

16A4 As can be seen in the last example of a tree, a node of the tree may be either the name of an unparse rule, such as ADD, or one of the basic entities recognized during the parse, such as the identifier X.

16A4A Suppose that the expression X+Y*Z has been parsed and the program is in the ADD unparse rule processing the ADD node (later we will see how this state is reached). To put the identifier X into the output stream, one writes "*1" (meaning "the first node below") as an element. For example, to generate a line of code with the operation code ADA and the operand field X, one would write:

```
, "ADA", *1
```

16A4B To generate the code for the left-hand node of the tree one merely mentions "*1" as an element of the unparse rule. Caution must be taken to ensure that no attempt is made to append a nonterminal node to the output stream; each node must be tested to be sure that it is the right type before it can be evaluated or output.

16A5 Generated labels are handled automatically. As each unparse

rule is entered, a new set of labels is generated. A label is referred to by a number sign (upper-case 3 on a Teletype) followed by a number. Every time a label is mentioned during the execution of a rule, the label is appended to the output stream. If another rule is invoked in the middle of a rule, all the labels are saved and new ones generated. When a return is made the previous labels are restored.

17 As trees are being built during the parse phase, a time comes when it is necessary to generate code from the tree. To do this one writes an asterisk as an element of a parse rule -- for example

```
R5B    PROGRAM = ".PROGRAM" $(ST *) ".END";
```

which generates code for each statement after it has been entirely parsed. When the asterisk is executed, control of the program is transferred to the rule whose name is the root (top node or last generated node) of the tree. When return is finally made to the rule which initiated the output, the entire tree is cleared and the generation process begins anew.

17A An unparse rule is a rule name followed by a series of output rules. Each output rule begins with a test of nodes. The series of output rules make up a set of highest-level alternatives. When an unparse rule is called, the test for the first output rule is made. If it is satisfied, the remainder of the alternative is executed; if it is false, the next alternative output rule test is made. This process continues until either a successful test is made or all the alternatives have been tried. If a test is successful, the alternative is executed and a return is made from the unparse rule with the general flag set "true." If no test is successful, a return is made with the general flag "false."

17B The simplest test that can be made is the test to ensure that the correct number of nodes emanate from the node being processed. The ADD rule may begin

```
ADD[-,-] =>
```

The string within the brackets is known as an out-test. The hyphens are individual items of the out-test. Each item is a test for a node. All that the hyphen requires is that a node be present. The name of a rule need not match the name of the node being processed.

17B1 If one wishes to eliminate the test at the head of the out-rule, one may write a slash instead of the bracketed string of items. The slash, then, takes the place of the test and is always true. Thus, a rule which begins with a slash immediately after the rule name may have only one out-rule. The rule

```
MT / => .EMPTY;
```

is frequently used to flag the absence of an optional item in a list of items. It may be tested in other unparse rules but it itself always sets the general flag true and returns.

17B2 The nodes emanating from the node being evaluated are referred to as *1, *2, etc., counting from left to right. To test for equality between nodes, one merely writes *i for some i as the desired item in an out-test. For example, to see if node 2 is the same as node 1, one could write either [-,*1] or [*2,-]. To see if the third node is the same as the first, one could write [-,*2,*1]. In this case, the *2 could be replaced by a hyphen.

17B3 One may test to see if a node is an element which was generated by one of the basic recognizers by mentioning the name of the recognizer. Thus to see if the node is an identifier one writes .ID; to test for a number one writes .NUM. To test whether the first node emanating from the ADD is an identifier and if the second node exists, one writes [.ID,-].

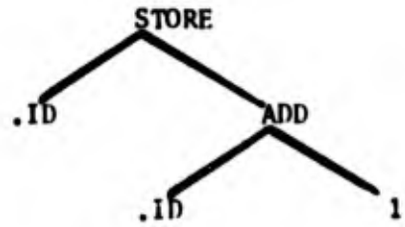
17B4 To check for a literal string on a node one may write a string as an item in an out-test. The construct [-,"1"] tests to be sure that there are two nodes and that the second node is a 1. The second node will have been recognized by the .NUM basic recognizer during the parse phase.

17B5 A generated label may be inserted into the tree by using it in a call to an unparse rule in another unparse rule. This process will be explained later. To see if a node is a previously generated label one writes a number sign followed by a number. If the node is not a generated label the test fails. If it is a generated label the test is successful and the label is associated with the number following the number sign. To refer to the label in the unparse rule, one writes the number sign followed by the number.

17B6 Finally, one may test to see if the name matches a specified name. Suppose that one had generated a node named STORE. The left node emanating from it is the name of a variable and on the right is the tree for an expression. An unparse rule may begin as follows:

```
STORE[-,ADD[*1,"1"]] => , "MIN " *1
```

The *1 as an item of the ADD refers to the left node of the STORE. Only a tree such as



would satisfy the test, where the two identifiers must be the same or the test fails. An expression such as X ← X + 1 meets all the requirements. The code generated (for the SDS 940) would be the single instruction MIN X, which increments the cell X by one.

17C Each out-rule, or highest-level alternative, in an unparse rule is also made up of alternatives. These alternatives are separated by slashes, as are the alternatives in the parse rules.

17C1 The alternatives of the out-rule are called "out-exprs." The out-expr may begin with a test, or it may begin with instructions to output characters. If it begins with a test, the test is made. If it fails the next out-expr in the out-rule is tried. If the test is successful, control proceeds to the next element of the out-expr. When the out-expr is done, a return is made from the unparse rule.

17C2 The test in an out-expr resembles the test for the out-rule. There are two types of these tests.

17C2A Any nonterminal node in the tree may be transferred to by its position in the tree rather than its name. For example, *2 would invoke the second node from the right. This operation not only transfers control to the specific node, but it makes that node the one from which the next set of nodes tested emanate. After control is returned to the position immediately following the *2, the general flag is tested. If it is "true" the out-expr proceedes to the next element. If it is "false" and the *2 is the first element of the out-expr the next alternative of the out-expr is tried. If the flag is "false" and the *2 is not the first element of the out-expr, a compiler error is indicated and the system stops.

17C2B The other type of test is made by invoking another unparse rule by name and testing the flag on the completion of the rule. To call another unparse rule from an out-expr, one writes the name of the rule followed by an argument list enclosed in brackets. The argument list is a list of nodes in

the tree. These nodes are put on the node stack, and when the call is made the rule being called sees the argument list as its set of nodes to analyze. For example:

```
ADD[MINUS[-],-] => SUB[*2,*1:*1]
```

17C2B1 Only nodes and generated labels can be written as arguments. Nodes are written as *1, *2, etc. To reach other nodes of the tree one may write such things as *1:*2, which means "the second node emanating from the first node emanating from the node being evaluated." Referring to the tree for the expression X+Y*Z if ADD is being evaluated, *2:*1 is Y.To go up the tree one may write an "uparrow"(↑) followed by a number before the asterisk-number-colon sequence. The uparrow means to go up that many levels before the search is made down the tree. If MULT were being evaluated,↑1*1 would be the X.

17C2B2 If a generated label is written as an argument, it is generated at that time and passed to the called unparse rule so that that rule may use it or pass it on to other rules. The generated label is written just as it is in an output element--a number sign followed by a number.

17C3 The calls on other unparse rules may occur anywhere in an out-expr. If they occur in a place other than the first element they are executed in the same way, except that after the return the flag is tested; if it is false a compiler error is indicated. This use of extra rules helps in making the output rules more concise.

17C4 The rest of an out-expr is made up of output elements appended to the output stream, as discussed above.

17D Sometimes it is necessary to set the general flag in an out-expr, just as it is sometimes necessary in the parse rules. .EMPTY may be used as an element in an out-expr at any place.

17E Out-exprs may be nested, using parentheses, in the same way as the alternatives of the parse rules.

18 There are a few features of Tree Meta which are not essential but do make programming easier for the user.

18A If a literal string is only one character long, one may write an apostrophe followed by the character rather than writing a quotation mark, the character, and another quotation mark. For example: 'S and "S" are interchangeable in either a parse rule or an

unparse rule.

18B As the parse rules proceed through the input stream they may come to a point where they are in the middle of a parse alternative and there is a failure. This may happen for two reasons: backup is necessary to parse the input, or there is a syntax error in the input. Backup will not be covered in this introductory chapter. If a syntax error occurs the system prints out the line in error with an arrow pointing to the character which cannot be parsed. The system then stops. To eliminate this, one may write a question mark followed by a number followed by a rule name after any test except the first in the parse equations. For example:

```
ST = .ID '= question 2 E EXP question 3 E ';

question 4 E :STORE[2] ;
```

Suppose this rule is executing and has called rule EXP, and EXP returns with the flag false. Instead of stopping Tree Meta prints the line in error, the arrow, and an error comment which contains the number 3, and transfers control to the parse rule E.

18C Comments may be inserted anywhere in a metalanguage program where blanks may occur. A comment begins and ends with a percent sign, and may contain any character -- except, of course, a percent sign.

18D In addition to the three basic recognizers .ID, .NUM, and .SR, there are two others which are occasionally very useful.

18D1 The symbol .LET indicates a single letter. It could be thought of as a one-character identifier.

18D2 The symbol .CHR indicates any character. In the parse rules, .CHR causes the next character on the input stream to be taken as input regardless of what it is. Leading blanks are not discarded as for .ID, .NUM, etc. The character is stored in a special way, and hence references to it are not exactly the same as for the other basic recognizers. In node testing, if one wishes to check for the occurrence of a particular character that was recognized by a .CHR, one uses the single quote-character construct. When outputting a node item which is a character recognized by a .CHR, one adds a :C to the node indicator. For example, *1:C.

18E Occasionally some parts of a compilation are very simple and it is cumbersome to build a parse tree and then output from it. For this reason the ability to output directly from parse rules has been added.

18E1 The syntax for outputting from parse rules is generally the same as for unparse rules. The output expression is written within square brackets, however. The items from the input stream that normally are put in the parse tree may be copied to the output stream by referencing them in the output expression. The most recent item recognized is referenced as * or *S0. Items recognized previous to that are *S1, *S2, etc., counting in reverse order--that is, counting down from the top of the stack they are kept in.

18E2 Normally the items are removed from the stack and put into the tree. However, if they are copied directly to the output stream, they remain in the stack. They are removed by writing an ampersand at the end of the parse rule (just before the semicolon). This causes all input items added to the stack by that rule to be removed. The input stack is thus the same as it was when the rule was called.

APPENDIX D -- TREE META: Program Environment

19 When a Tree Meta program is compiled by the metacompiler, a machine-language version of the program is generated. However, it is not a complete program since several routines are missing. All Tree Meta programs have common functions such as reading input, generating output, and manipulating stacks. It would be cumbersome to have the metacompiler duplicate these routines for each program, so they are contained in a library package for all Tree Meta programs. The library of routines must be loaded with the machine-language version of the Tree Meta program to make it complete.

19A The environment of the Tree Meta program, as it is running, is the library of routines plus the various data areas.

19B This section describes the environment in its three logical parts: input, stack organization, and output.

19B1 This is a description of the current working version, with some indications of planned improvements.

20 Input Machinery

20A The input stream of text is broken into lines and put into an input buffer. Carriage returns in the text are used to determine the ends of lines. Any line longer than 80 characters is broken into two lines. This line orientation is necessary for the following:

20A1 Syntax-error reporting

20A2 A possible anchor mode (so the compiler can sense the end of a line)

20A3 An interlinear listing option.

20A4 In the future, characters for the input buffer will be obtained from another input buffer of arbitrary block size, but at present they are obtained from the system with a Character I/O command.

20B It is the job of routine RLINE to fill the input line buffer. If the listing flag is on, RLINE copies the new line to the output file (prefixed with a comment character--an asterisk for our assembler). It also checks for an End-of-File, and for a multiple blank character, which is a system feature built into our text files. There is a buffer pointer that indicates which character is to be read from the line buffer next, and RLINE resets that pointer to the first character of the line.

20C Input characters for the Tree Meta program are not obtained from the input line buffer, but from an input window, which is actually a

character ring buffer. Such a buffer is necessary for backup. There are three pointers into the input window. A program-character counter (PCC) points to the next character to be read by the program. This may be moved back by the program to effect backup. A library-character counter (LCC) is never changed except by a library routine when a new character is stored in the input window. PCC is used to compute the third pointer, the input-window pointer (IWP). Actually, PCC and LCC are counters, and only IWP points into the array RING which is the character ring buffer. LCC is never backed up and always indicates the next position in the window where a new character must be obtained from the input line buffer. Backup is registered in BACK, and is simply the difference between PCC and LCC. BACK is always negative or zero.

20D There are several routines that deal directly with the input window.

20D1 The routine PUTIN takes the next character from the input line buffer and stores it at the input-window position indicated by IWP. This involves incrementing the input-buffer pointer, or calling RLINE if the buffer is empty. PUTIN does not change IWP.

20D2 The routine INC is used to put a character into the input window. It increases IWP by one by calling a routine, UPIWP, which makes IWP wrap around the ring buffer correctly. If there is backup (i.e., if BACK is less than 0), BACK is increased by one and INC returns, since the next character is in the window already. Otherwise, LCC is increased by one, and PUTIN is called to store the new character.

20D3 A routine called INCS is similar to INC except that it deletes all blanks or comments that may be at the current point in the input stream. This routine implements the comment and blank deletion for .ID, .NUM, .SR, and other basic recognizers. INCS first calls INC to get the next character and increment IWP. From then on, PUTIN is called to store succeeding characters in the input window in the same slot. As long as the current character (at IWP) is a blank, INCS calls PUTIN to replace it with the next character. The nonblank character is then compared with a comment character. INCS returns if the comparison fails, but otherwise skips to the next comment character. When the end of the comment is located, INCS returns to its blank-checking loop.

20D3A Note that comments do not get into the input window. For this reason, BACK should be zero when a comment is found in the loop described above, and this provides a good opportunity for an error check.

20D4 Before beginning any input operation, the IWP pointer must

be reset, since the program may have set PCC back. The routine WPREP computes the value of BACK from PCC-LCC. This value must be between 0 and the negative of the window size. IWP is then computed from PCC modulo the window size.

20D5 The program-library interface for inputting items from the input stream consists of the routines ID, NUM, SR, LET, and CHR. The first four are quite similar. ID is typical of them, and works as follows: First MFLAG is set false. WPREP is called to set up IWP, then INCS is called to get the first character. If the character at IWP is not a letter, ID returns (MFLAG is still false); otherwise a loop to input over letter-digits is executed. When the letter-digit test fails the flag is set true, and the identifier is stored in the string storage area. The class of characters is determined by an array (indexed by the character itself) of integers indicating the class. Before returning, ID calls the routine GOBL which updates PCC to the last character read in (which was not part of the identifier). That is, PCC is set to LCC+BACK-1.

20D6 The occurrence of a given literal string in the input stream is tested for by calling routine TST. The character count and the string follow the call instruction. TST deletes leading blanks and inputs characters, comparing them one at a time with the characters of the literal string. If at any point the match fails, TST returns false. Upon reaching the end of the string, TST sets the flag true, sets PCC to LCC+BACK, and returns. In addition to TST, there is a simple routine to test for a single character string (TCH). It inputs one character (deleting blanks), compares it to the given character and returns false, or adjusts PCC and returns true.

21 Stacks and Internal Organization

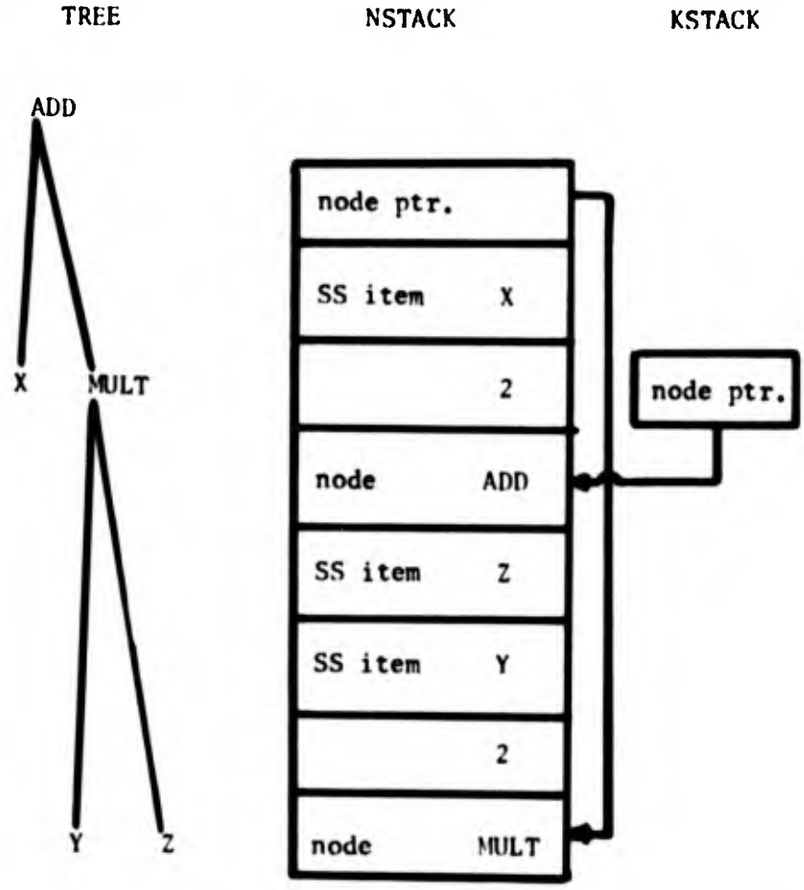
21A Three stacks are available to the program. A stack called MSTACK is used to hold return locations and generated labels for the program's recursive routines. Another stack, called KSTACK, contains references to input items. When a basic recognizer is executed, the reference to that input item is pushed into KSTACK. The third stack is called NSTACK, and contains the actual tree. The three stacks are declared in the Tree Meta program rather than the library: the program determines the size of each.

21A1 The operation of MSTACK is very simple. At the beginning of each routine, the current generated labels and the location that the routine was called from are put onto MSTACK. The routine is then free to use the generated labels or call other routines. The routine ends by restoring the generated labels from MSTACK and returning.

21A2 KSTACK contains single-word entries. Each entry will eventually be placed in NSTACK as a node in the tree. The format of the node words is as follows: There are two kinds of nodes, terminal and nonterminal. Terminal nodes are references to input items. Nonterminal nodes are generated by the parse rules, and have names which are names of output rules.

21A2A A terminal node is a 24-bit word with either a string-storage index or a character in the address portion of the word, and a flag in the top part of the word. The flag indicates which of the basic recognizers (ID, NUM, SR, LET, or CHR) is to read the item from the input stream.

21A2B A nonterminal node consists of a word with the address of an output rule in the address portion, and a flag in the top part which indicates that it is a nonterminal node. A node pointer is a word with an NSTACK index in the address and a pointer flag in the top part of the word. Each nonterminal node in NSTACK consists of a nonterminal node word followed by a word containing the number of subnodes on that node, followed by a terminal node word or node pointers for each subnode. For example,



21A2C KSTACK contains terminal nodes (input items) and nonterminal node pointers that point to nodes already in NSTACK. NSTACK contains nonterminal nodes.

21B String Storage is another stack-like area. All the items read from the input stream by the basic recognizers (except CHR) are stored in the string-storage area (SS). This consists of a series of character strings prefixed by their character counts. An index into SS consists of the address of the character count for a string.

Strings in SS are unique. A routine called STORE will search SS for a given string, and enter it if it is not already there, returning the SS index of that string.

21C Other routines perform housekeeping functions like packing and unpacking strings, etc. There are three error-message writing routines to write the three types of error messages (syntax, system, and compiler). The syntax error routine copies the current input line to the teletype and gives the line number. A routine called FINISH closes the files, writes the number of cells used for each of the four stack areas (KSTACK, MSTACK, NSTACK, and SS), and terminates the program.

21C1 At many points in the library routines, parameters are checked to see if they are within their bounds. The system error routine is called if there is something wrong. This routine writes a number indicating what the error is, and terminates the program. In the current version, the numbers correspond to the following errors:

21C1A (1) Class codes are illegal

21C1B (2) Backup too far

21C1C (64) Character with code greater than 63 in ring buffer

21C1D (4) Test for string longer than ring size

21C1E (5) Trying to output a string longer than maximum string length

21C1F (6) String-storage overflow

21C1G (7) Illegal character code

21C1H (8) Trying to store SS element of length zero

21C1I (11) MSTACK overflow

21C1J (12) NSTACK overflow

21C1K (13) KSTACK overflow

21D There is a set of routines used by Tree Meta that are not actually part of the library, but are loaded with the library for Tree Meta. They are not included in the library since they are not necessarily required for every Tree Meta program, but more likely only for Tree Meta. They are called "support routines." The routines perform short but frequently needed operations and serve to

increase code density in the metacompiler. Examples of the operations are generating labels, saving and restoring labels and return addresses on MSTACK, comparing flags in NSTACK, generating nodes on NSTACK, etc.

22 Output Facilities

22A The output from a Tree Meta program consists of a string of characters. In the future it might be a string of bits constituting a binary program, but at any rate it can be thought of as a stream of data. The output facilities available to the program consist of a set of routines to append characters, strings, and numbers to the output stream.

22A1 A string in SS can be written on the output stream by calling the routine OUTS with the SS index for that string. OUTS checks the SS index and generates a system-error message if it is not reasonable.

22A2 A literal string of characters is written by calling the routine LIT The literal string follows the call as for TST.

22A3 A number is written using routine OUTS. The binary representation is given, and is written as a signed decimal integer.

22A4 All of the above routines keep track of the number of characters written on the output stream (in CHNO). Based on this count, a routine called TAB will output enough spaces to advance the current output line to the next tab stop. Tabs are set at 8-character intervals. The routine CRLF will output a carriage return and a line feed and reset CHNO.

22A5 There are several routines that are convenient for debugging. One (WRSS) will print the contents of SS. Another (WRIW) will print the contents of the input window.

APPENDIX D -- TREE META: Formal Description

23 This chapter is a formal description of the complete Tree Meta language. It is designed as a reference guide.

23A For clarity, strings that would normally be delimited by quotation marks in the metalanguage are capitalized instead, in this chapter only.

23B Certain characters cannot be printed on the report-generating output media but are on the teletypes and in the metalanguage--their names, preceeded by periods, are used instead. They are .exclamation, .question, .pound, .ampersand, .backslash, and .percent.

24 Programs and Rules

24A Syntax

24A1 program = .META .id (.LIST / .empty) size / .CONTINUE $rule .END;

24A2 size = '(siz $(', siz) ') / .empty;

24A3 siz = .chr '= .num;

24A4 rule = .id ('= exp (.ampersand / .empty) / '/ "=>" gen1 / outrul) '; ;

24B Semantics

24B1 A file of symbolic Tree Meta code may be either an original main file or a continuation file. A compiler may be composed of any number of files but there may be only one main file.

24B1A The mandatory identifier following the string .META in a main file names the rule at which the parse will begin.

24B1B The optional .LIST, if present, will cause the compiler currently being generated to list input when it is compiling a program.

24B1C The size construct sets the allocation parameters for the three stacks and string storage used by the Tree Meta library. The default sizes are those used by the Tree Meta compiler. M, K, N, and S are the only valid characters; the size is something that must be determined by experience. The maximum number of cells used during each compilation is printed out at the end of the compilation.

24B2 When a file begins with .CONTINUE, no initialization or

storage-allocation code is produced.

24B3 There are three different kinds of rules in a Tree Meta program. All three begin with the identifier that names the rule.

24B3A Parse rules are distinguished by the = following the identifier. If all the elements that generate possible nodes during the execution of a parse rule are not built into the tree, they must be popped from the kstack by writing an ampersand immediately before the semicolon.

24B3B Rules with the string / => following the identifier may be composed only of elements hat produce output. There is no testing of flags within a rule of this type.

24B3C Unparse rules have a left bracket following the identifier. This signals the start of a series of node tests.

25 Expressions

25A Syntax

25A1 exp = '←suback ('/ exp / .empty) / subexp ('/ exp / .empty);

25A2 suback = ntest (suback / .empty) / stest (suback / .empty);

25A3 subexp = (ntest / stest) (noback / .empty);

25A4 noback = (ntest / stest ('.question .num (.id / '.question) / .empty)) (noback / .empty);

25B Semantics

25B1 The expressions in parse rules are composed entirely of ntest, stest, and error-recovery constructs. The four rules above, which define the allowable alternation and concatention of the test, are necessary to reduce the instructions executed when there is no backup of the input stream.

25B2 An expression is essentially a series of subexpressions separated by slashes. Each subexpression is an alternative of the expression. The alternatives are executed in a left-to-right order until a successful one is found. The rest of that alternative is then executed and the rule returns to the rule that invoked it.

25B3 The subexpressions are series of tests. Only subexpressions that begin with a leftarrow are allowed to back up the input stream and rescan it.

25B3A Without the arrow at the head of a subexpression, any test other than the first within the subexpression may be followed by an error code. If the error code is absent and the stest fails during compilation, the system prints an error comment and stops. If the error code is present and the stest fails, the system prints the number following the '.question in the error code, and if the optional identifier is given the system then transfers control to that rule; otherwise it stops.

25B3B If the test fails, the input stream is restored to the position it had when the subexpression began to test the input stream and the next alternative is tried. The input stream may never be moved back more characters than are in the ring buffer. Normally, backup is over identifiers or words and the buffer is long enough.

26 Elements of Parse Rules

26A Syntax

```
26A1  ntest = (':.id / '[ ( .num '] / genp '] ('.backslash /
.empty ) / '< genp '> ('.backslash / .empty) / (.CHR / '*) / "=>"
/ comm;

26A2  genp = genp1 / .empty;

26A3  genp1 = genp2 (genp1 / .empty);

26A4  genp2 = '* (S .num / .empty) (L / C / N / .empty) / genu;

26A5  comm = .EMPTY / '.exclamation .sr;

26A6  stest = '. .id / .id / .sr / '( exp ') / ''.chr / (.num '$ /
'$) (.num / .empty) stest / '- (.sr / ''.chr);
```

26B Semantics

26B1 The ntest elements of a parse rule cannot change the value of the general flag, and therefore need not be followed by flag-checking code in the compiler.

26B1A The : .id construct names the next node to be put into the tree. The identifier must be the name of another rule.

26B1B The [.num] constructs a node with the name used in the last : .id construct, and puts the number of nodes specified after the arrow on the new node in the tree.

26B1C The [genp] is used to write output into the normal

output stream during the parse phase of the compilation.

26B1D The < genp> is used to print output back on the user teletype instead of the normal output stream. This is generally used during long compilations to assure the user that the system is still up and running correctly.

26B1E The occurrence of a .chr causes one character to be read from the input stream into a special register which may be put into the tree just as the terminal symbols recognized by the other basic recognizers are.

26B1F An asterisk causes the rule currently in execution to perform a subroutine call to the rule named by the top of the tree.

26B1G The "=>" ntest construct causes the input stream to be moved from its current position past the first occurrence of the next stest. This may be used to skip over comments, or to move the input to a recognizable point such as a semicolon after a syntax error.

26B2 The comm elements are common to both parse and unparse rules.

26B2A The .EMPTY in any rule sets the general flag true.

26B2B The .exclamation-string construct is used to insert patches into the compiler currently being produced. The string following the .exclamation is immediately copied to the output stream as a new line. This allows the insertion of any special code at any point in a program.

26B3 Stests always test the input stream for a literal string or basic entity. If the entity is found it is removed from the input stream and stored in string storage. Its position in string storage is saved on a push-down stack so that the entity may later be added as a terminal node to the tree.

26B3A A .id construct provides a standard machine-language subroutine call to the identifier. Supplied with the Tree Meta library are subroutines for .id, .num, .sr, .chr, and .let which check for identifier, number, string, character, and letter respectively.

26B3B An identifier by itself produces a call to the rule with the name of the identifier.

26B3C A literal string merely tests the input stream for the

string. If it is found it is discarded. The apostrophe-character construct functions like the literal string, except that the test is limited to one character.

26B3D The number-$-number construct is the arbitrary-number operation of Tree Meta. m$n preceding an element in a parse rule means that there must be between m and n occurrences of the next element coming up in the input. The default options for m and n are zero and infinity respectively.

26B3E The hyphen-string and hyphen-character constructs test in the same way as the literal string and apostrophe-character constructs. After the test, however, the flag is complemented and the input-stream pointer is never moved forward. This permits a test to be sure that something does not occur.

27 Unparse Rules

27A Syntax

27A1 outrul = '[outr (outrul / .empty);

27A2 outr = items '] "=>" outexp;

27A3 items = item (', items / .empty);

27A4 ,m = '- / .id '[outest / nsimpl / '. .id / .sr / ''.chr / '.pound;

27B Semantics

27B1 The unparse rules are similar to the parse rules in that they test something and return a true or false value in the general flag. The difference is that the parse rules test the input stream, delete characters from the input stream, and build a tree, while the unparse rules test the tree, collapse sections of the tree, and write output.

27B2 There are two levels of alternation in the unparse rules. The highest level is not written in the normal style of Tree Meta as a series of expressions separated by slashes; rather, it is written in a way intended to reflect the matching of nodes and structure within the tree. Each unparse rule is a series of these highest-level alternations. The tree-matching parts of the alternations are tried in sequence until one is found that successfully matches the tree. The rest of the alternation is then executed. There may be further test within the alternation, but not complete failure as with the parse rules.

27B3 The syntax for a tree-matching pattern is a left bracket, a series of items separated by commas, and a right bracket. The items are matched against the branches emanating from the current top node. The matching is done in a left-to-right order. As soon as a match fails the next alternation is tried.

27B4 If no alternation is successful a false value is returned.

27B5 Each item of an unparse alternation test may be one of five different kinds of test.

27B5A A hyphen is merely a test to be sure that a node is there. This sets up appropriate flags and pointers so that the node may be referred to later in the unparse expression if the complete match is successful.

27B5B The name of the node may be tested by writing an identifier that is the name of a rule. The identifer must then be followed by a test on the subnodes.

27B5C A nonsimple construct, primarily an asterisk-number-colon sequence, may be used to test for node equivalence. Note that this does not test for complete substructure equivalence, but merely to see if the node being tested has the same name as the node specified by the construct.

27B5D The .id, .num, .chr, .let, or .sr checks to see if the node is terminal and was put on the tree by a .id recognizer, .num recognizer, etc. during the parse phase. This test is very simple, for it merely checks a flag in the upper part a word.

27B5E If a node is a terminal node in the tree, and if it has been recognized by one of the basic recognizers in meta, it may be tested against a literal string. This is done by writing the string as an item. The literal string does not have to be put into the tree with a .sr recognizer; it can be any string, even one put in with a .let.

27B5F If the node is terminal and was generated by the .chr recognizer it may be matched against another specific character by writing the apostrophe-character construct as an item.

27B5G Finally, the node may be tested to see if it is a generated label. The labels may be generated in the unparse expressions and then passed down to other unparse rules. The test is made writing a .pound-number construct as an item. If the node is a generated label, not only is this match

successful but the label is made available to the elements of the unparse expression as the number following the .pound.

28 Unparse Expressions

28A Syntax

28A1 outexp = subout ('/ outexp / .empty);

28A2 subout = outt (rest / .empty) / rest;

23A3 rest = outt (rest / .empty) / gen (rest / .empty);

28A4 outt = .id '[arglst '] / '(outexp ') / nsimpl (': (S / L / N / C) / empty);

28A5 arglst = argmnt (', arglst / .empty) / .empty;

28A6 argmnt = nsimp / '.pound .num;

28A7 nsimpl = '↑ nsimp / nsimp;

28A8 nsimp = '* .num (': nsimp / .empty);

28A9 genl = (out / comm) (genl / .empty);

28A10 gen = comm / genu / '< / '> ;

28B Semantics

28B1 The rest of the unparse rules follow more closely the style of the parse rules. Each expression is a series of alternations separated by slash marks.

28B2 Each alternation is a test followed by a series of output instructions, calls of other unparse rules, and parenthesized expressions. Once an unparse expression has begun executing calls on other rules, elements may not fail; if they do a compiler error is indicated and the system stops.

28B3 The first element of the expression is the test. This element is a call on another rule, which returns a true or false value. The call is made by writing the name of the rule followed by a series of nodes. The nodes are put together to appear as part of the tree, and when the call is made the unparse rule called views the nodes specified as the current part of the tree, and thus the part to match against and process.

28B3A Two kinds of things may be put in as nodes for the

calls. The simplest is a generated label. This is done by writing a .pound followed by a number. Only the numbers 1 and 2 may be used in the current system. If a label has not yet been generated, one is made up. This label is then put into the tree.

28B3B Any already constructed node also may be put into the tree in this new position. The old node is not removed--rather a copy is made. An asterisk-number construct refers to nodes in the same way as the highest-level alternation.

28B4 This process of making new structures from the already-existing tree is a very powerful way of optimizing the compiler and condensing the number of rules needed to handle compilation.

28B5 The rest of the unparse expression is made up of output commands, and more calls on unparse rules. As noted above, if any except the first call of an expression fails, a compiler error is indicated and the system stops.

28B6 Just as in the parse rules, brackets may be used to send immediate printout to the user Teletype.

28B7 The asterisk-number-colon construct is used frequently in the Tree Meta system. It appears in the node-matching syntax as well as in the form of an element in the unparse expressions. When it is in an expression it must specify a node that exists in the tree.

28B7A If the node specified is the name of another rule, then control is transferred to that node by the standard subroutine linkage.

28B7B If the node is terminal, then the terminal string associated with the node is copied onto the output stream.

28B7C The simplest form of the construct is an asterisk followed by a number, in which case the node is found by counting the appropriate number of nodes from left to right. This may be followed by a colon-number construct, which means to go down one level in the tree after performing the asterisk-number choice and count over the number of nodes specified by the number following the colon. This process may be repeated as often as desired, and one may therefore go as deep as one wishes. All of this specification may be preceded by an ↑-number construct which means to go up in the tree, through parent nodes, a specified number of times before starting down.

28B7D After the search for the node has been completed, a number of different types of output may be specified if the node is terminal. There is a compiler error if the node is not terminal.

28B7D1 :s puts out the literal string

28B7D2 :l puts out the length of the string as a decimal number

28B7D3 :n puts out the string-storage index pointer if the node is a string-storage element; otherwise it puts out the decimal code for the node if it is a .chr node.

28B7D4 :c puts out the character if the node was constructed with a .chr recognizer.

29 Output

29A Syntax

```
29A1  genu = out / '. .id '] ((.id / .num) / .empty) '] / '.pound
.num (': / .empty);

29A2  out = ('.backslash / ', /  .sr / ''.chr / "+w" / "-w" / ".w"
/ ".pound" ;
```

29B Semantics

29B1 The standard primitive output features include the following:

29B1A Write a carriage return with a backslash

29B1B Write a tab with a comma

29B1C Write a literal string by giving the literal string

29B1D Write a single character using the apostrophe-character construct

29B1E Write references to temporary storage by using a working counter. Three types of action may be performed with the counter. +W adds one to the counter and writes the current value of the counter onto the output stream. -W subtracts one from the counter and does not write anything. .W writes the current value without changing it. Finally, .pound W writes the maximum value that the counter ever reached during the compilation.

29B2 The .id [(.num/.id)] is used to generate a call (940 BRM instruction) with a single argument in the A register. It has been used mostly as a debugging tool during various bootstrap sessions with the system. For example, .CERR[5] generates a call to the subroutine CERR with a 5 in the A register.

29B3 .pound 2 means "define generated label 2 at this point in the program being compiled." It writes the generated label in the output stream followed by an EQU *. This construct is added only to save space and writing.

APPENDIX D -- Tree Meta: Conclusions and Future Plans

50 Since the work on Tree Meta is still in progress, there are few conclusions and plentiful future plans.

51 There are many research projects that could be undertaken to improve the Tree Meta system.

51A Something that has never been done, and that we feel is very important, is a complete study of the compiling characteristics of top-down analysis techniques. This would include an accurate study of where all the time goes during a compilation as well as a study of the flow of control during both parse and unparse phases for different kinds of compilers and languages. At the same time it would be worthwhile to try to get similiar statistics from other compilers. It may be possible to interest some people at Stanford in cooperating on this.

51B SDC has added an intermediate phase to their metacompiler system. They call it a bottom-up phase, and it has the effect of putting various attributes and features on the nodes of the tree. This allows one to write simpler and faster node-matching instructions in the unparse rules. We would like to investigate this scheme, for it appears to hold the potential for allowing the compiler writer to conceptualize more complex tree patterns and thus utilize the node-matching features to a fuller extent.

51C Yet another intermediate phase could be added to Tree Meta which would do transformations on the tree before the unparse rules produce the final code. In attempts to write compilers in Tree Meta to compile code for languages with complex data structures (such as algebraic languages with matrix operations or string-oriented languages with tree operations) and to make these compilers produce efficient code, we have found that tree transformations similar to those used for natural-language translation allow one to specify easily and simply the rules for tree manipulation that permit the unparse rules to produce efficient, dense code. Implementation of the tree-transformation phase into the Tree Meta system would be an extensive research project, but could add a completely new dimension to the power of Tree Meta.

51D There are a series of additions, some very small and some major, that we intend to add to Tree Meta during the next year.

51D1 Other metacompiler systems have had a construct that allows nodes to have an arbitrary number of nodes emanating from them. This requires additions in parse rules to specify such a search, additions in the node-matching syntax, and additions in the output syntax to scan and output any number of branches.

51D2 We have always felt that it would be nice to have the basic

recognizers such as "identifier" defined in the metalanguage. There have been systems with this feature, but the addition has always had very bad effects on the speed of compilation. We feel that this new freedom can be added to Tree Meta without having telling effects on the compilation speed.

31D3 The error scheme for unparse rules is rather crude--the compiler just stops. We would like to find a reasonable way of accommodating such errors and putting the recovery-procedure control in the metalanguage.

31D4 Currently the unparse rules expand into 6 times as many machine-language instructions as the parse rules. This happens because we did not choose the most appropriate set of subroutines and common procedures for the unparse rules. Without changing the syntax of Tree Meta or the way the stacks work, we feel that we can reduce the size of the unparse rules by a factor of 4. This would free a considerably larger amount of core storage for stacks and enlarge the size of programs that Tree Meta could handle. It would also make it run faster in time-sharing mode since less would have to be swapped into core to run it.

31D5 In doing some small tests on the speed of Tree Meta we found that better than 80 percent of the compilation time is spent outputting strings of characters to the system. The code that Tree Meta now produces is the simplest form of assembly code. It would be a very simple task to make Tree Meta able to directly produce binary code for the loader rather than symbolic code for the assembler. A similar change could also be made to output absolute code directly into core so that Tree Meta could be used as the compiler for systems that do incremental compilation.

31E Finally, there is the following list of minor additions or changes to be made to the Tree Meta system.

31E1 Make the library output routines do block I/O rather than character I/O. This could cut compilation times by more that 70 percent.

31E2 Fix Tree Meta so that strings can be put into the tree and passed down to other unparse rules. This would allow the unparse rules to be more useful as subroutines and thus cut down the number of unparse rules needed in a compiler.

31E3 Finally, we would like to add the ability to associate a set of attributes with each terminal entity as it is recognized, to test these attributes later, and to add more or change them if necessary. To do this we would associate a single 24-bit word with the string when it is put into string storage and add syntax

to the metalanguage to set, reset, and test the bits of the word.

APPENDIX D -- Tree Meta: Bibliography

1 (Book1) Erwin Book, "The LISP Version of the Meta Compiler," TECH MEMO TM-2710/330/00, System Development Corporation, 2500 Colorado Avenue, Santa Monica, California 90406, 2 November 1965.

2 (Book2) Erwin Book and D. V. Schorre, "A Simple Compiler Showing Features of Extended META," SP-2822, System Development Corporation, 2500 Colorado Avenue, Santa Monica, California 90406, 11 April 1967.

3 (Glennie1) A. E. Glennie, "On the Syntax Machine and the Construction of a Universal Computer," Technical Report Number 2, AD 240-512, Computation Center, Carnegie Institute of Technology, 1960.

4 (Kirkley1) Charles R. Kirkley and Johns F. Rulifson, "The LOT System of Syntax Directed Compiling," Stanford Research Institute Internal Report ISR 187531-139, 1966.

5 (Ledley1) Robert Ledley and J. B. Wilson, "Automatic Programming Language Translation Through Syntactical Analysis," Communications of the Association for Computing Machinery, Vol. 5, No. 3 pp. 145-155, March 1962.

6 (Metcalfe1) Howard Metcalfe, "A Parameterized Compiler Based on Mechanical Linguistics," Planning Research Corporation R-311, March 1, 1963, also in Annual Review in Automatic Programming, Vol. 4, 125-165.

7 (Naur1) Peter Naur et al., "Report on the Algorithmic Language ALGOL 60," Communications of the Association for Compting Machinery, Vol. 3, No. 5, pp.299-384, May 1960.

8 (Oppenheim1) D. Oppenheim and D. Haggerty, "META 5: A Tool to Manipulate Strings of Data," Proceedings of the 21st National Conference of the Association for Computing Machinery, 1966.

9 (Rutman1) Roger Rutman, "LOGIK. A Syntax Directed Compiler for Computer Bit-Time Simulation," Master Thesis, UCLA, August 1964.

10 (Schmidt1) L. O. Schmidt, "The Status Bit," Special Interest Group on Programming Languages Working Group 1 News Letter, 1964.

11 (Schmidt2) PDP-1

12 (Schmidt3) EQGEN

13 (Schnieder1) F. W. Schneider and G. D. Johnson, "A Syntax-Directed Compiler-Writing Compiler to Generate Efficient Code," Proceedings of the 19th National Conference of the Association for Computing Machinery, 1964.

14 (Schorre1) D. V. Schorre, "A Syntax-Directed SMALGOL for the 1401,"

Proceedings of the 18th National Conference of the Association for Computing Machinery, Denver, Colorado, 1963.

15 (Schorre2) D. V. Schorre, "META II, A Syntax-Directed Compiler Writing Language," Proceedings of the 19th National Conference of the Association for Computing Machinery, 1964.

APPENDIX D -- TREE META: Detailed Examples

1 This section of the report is merely the listings of compilers for two languages.

2 The first language, known as SAL for "small algebraic language," is a straightforward algebraic ALGOL-like language.

3 The second example resembles Schorre's META II. This is the original metacompiler that was used to bootstrap Tree Meta. It is a one-page compiler written in its own language (a subset of Tree Meta).

```
--0--


%TREE META SMALL ALGEBRAIC LANGUAGE - 29 SEPTEMBER 1967 %

.META PROGRAM .LIST

PROGRAM = ".PROGRAM" DEC * $( DEC *) :STARTN[0] ST * $('; ST *)
                ".FINISH" ?1E :ENDN[0] * FINISH ;

DEC = ".DECLARE" .ID $(', .ID :DO[2]) '; :DECN[1];


E = RESET => '; $(ST *) ".END" ?99E :ENDN[0] * FINISH;

ST = IFST / WHILEST / FORST / GOST / IOST / BLOCK /
       .ID ('= :LBL[1] ST :DO[2] / '← EXP :STORE[2]);

IFST = ".IF" EXP ".THEN" ST (".ELSE" ST :SIFTE[3] / .EMPTY :SIFT[2]);

WHILEST = ".WHILE" EXP ".DO" ST :WHL[2];

FORST = ".FOR" VAR '← EXP ".BY" EXP ".TO" EXP ".DO" ST :FOR[5];

GOST = ".GO"  ".TO" .ID :GO[1];

IOST = ".OPEN" ("INPUT" .ID '( .ID ') :OPNINP[2] /
                "OUTPUT" .ID '( .ID ') :OPNOUT[2]) /
       ".CLOSE" .ID :CLSFIL[1] /
       ".READ" .ID ': IDLIST :BRS38[2] /
       ".INPUT" .ID ': IDLIST :XCIO[2] /
       ".WRITE" .ID ': WLIST :OUTNUM[2] /
       ".OUTPUT" .ID ': WLIST :OUTCAR[2] ;
IDLIST = VAR (IDLIST :DO[2] / .EMPTY);

WLIST = (.ID / .NUM / .SR) (WLIST :DO[2] / .EMPTY);


BLOCK = ".BEGIN" ST $('; ST :DO[2]) ".END";


EXP = ".IF" EXP ".THEN" EXP ".ELSE" EXP :AIF[3] / UNION;

UNION = INTERSECTION ('\'/ UNION :OR[2] / .EMPTY);

INTERSECTION = NEG ('& INTERSECTION :AND[2] / .EMPTY);

NEG = "NOT " NEGNEG / RELATION;

NEGNEG = "NOT " NEG / RELATION :NOT[1];

RELATION = SUM((  "<=" SUM :LE /
                  "<"  SUM :LT /
                  ">=" SUM :GE /
                  ">"  SUM :GT /
                  "="  SUM :EQ /
                  '#   SUM :NE ) [2] / .EMPTY);
```

```
SUM = TERM (('+ SUM :ADD/ '- SUM :SUB)[2]/ .EMPTY);

TERM = FACTOR (('* TERM :MULT/'/ TERM :DIVID/'! TERM :REM)[2]/.EMPTY);

FACTOR = '- FACTOR :MINUS[1] / '+ FACTOR / PRIMARY;

PRIMARY = VARIABLE / CONSTANT / '( EXP ');

VARIABLE = .ID :VAR[1];

CONSTANT = .NUM :CON[1];

SIFTE[-,-,-] => LOPR[*1,#1,#2] BRF[*1,#2] #1,"EQU *"\ *2 SIFTE1[#2,*3];

SIFTE1[#1,-] => ,"BRU",#2\ #1,"EQU *"\ *2 #2,"EQU *"\;

SIFT[-,-] => LOPR[*1,#1,#2] BRF[*1,#2] #1,"EQU *"\ *2 #2,"EQU *"\;

WHL[-,-] => #1,"EQU *"\ WHL1[*1,#2] *2 ,"BRU",#1\ #2,"EQU *"\;

WHL1[-,#2] => LOPR[*1,#1,#2] BRF[*1,#2] #1,"EQU *"\;

GO[-] => ,"BRU",*1\;

FOR[-,-,-,-,-] => <"DO NOT USE FOR STATEMENTS">;

LBL[-] => *1,"EQU *";

AIF[-,-,-] => LOPR[*1,#1,#2] BRF[*1,#2] #1,"EQU *"\ ACC[*2] AIF1[#2,*3];

AIF1[#1,-] => ,"BRU",#2\ #1,"EQU *"\ ACC[*2] #2,"EQU *"\;

LOPR[OR[-,-],#1,-] => LOPR[*1:*1,#1,#2] BRT[*1:*1,#1]
                                        #2,"EQU *"\ LOPR[*1:*2,#1,*3]
     [AND[-,-],-,#1] => LOPR[*1:*1,#2,#1] BRF[*1:*1,#1]
                                        #2,"EQU *"\ LOPR[*1:*2,*2,#1]
     [NOT[-],#1,#2]  => LOPR[*1:*1,#2,#1]
     [-,-,-] => .EMPTY;

 BRT[OR[-,-],#1] => BRT[*1:*2,#1]
    [AND[-,-],#1] => BRT[*1:*2,#1]
    [NOT[-],#1]   => BRF[*1:*1,#1]
    [LE[-,-],#1]  => BLE[*1:*1,*1:*2,#1]
    [LT[-,-],#1]  => BLT[*1:*1,*1:*2,#1]
    [EQ[-,-],#1]  => BEQ[*1:*1,*1:*2,#1]
    [GE[-,-],#1]  => BGE[*1:*1,*1:*2,#1]
    [GT[-,-],#1]  => BLE[*1:*2,*1:*1,#1]
    [NE[-,-],#1]  => BNE[*1:*1,*1:*2,#1]
    [-,#1]        => ACC[*1] ,"SKE =0"\ ,"BRU",#1\;

 BRF[OR[-,-],#1]  => BRF[*1:*2,#1]
    [AND[-,-],#1] => BRF[*1:*2,#1]
    [NOT[-],#1]   => BRT[*1:*1,#1]
```

```
   [LE[-,-],#1]  => BLE[*1:*2,*1:*1,#1]
   [LT[-,-],#1]  => BGE[*1:*1,*1:*2,#1]
   [EQ[-,-],#1]  => BNE[*1:*1,*1:*2,#1]
   [GE[-,-],#1]  => BLT[*1:*1,*1:*2,#1]
   [GT[-,-],#1]  => BLE[*1:*1,*1:*2,#1]
   [NE[-,-],#1]  => BEQ[*.1:*1,*1:*2,#1]
   [-,#1]        => ACC[*1] ,"SKA =-1"\ ,"BRU",#1\;

BLT[-,-,#1] => (TOKEN[*1] ACC[*2] ,"SKE",*1\,"SKG",*1\ /
                WORK[*1]  ACC[*2] ,"SKE","T+".W\,"SKG","T+".W-W\ )
                     ,"BRU *+2"\ ,"BRU",#1\;

BLE[-,-,#1] => (TOKEN[*2] ACC[*1] ,"SKG",*2\ /
                TOKEN[*1] ACC[*2] ,"SKG",*1\,"BRU *+2"\ /
                WORK[*2]  ACC[*1] ,"SKG","T+".W-W\ )
                    ,"BRU",#1\;

BEQ[-,-,#1] => (TOKEN[*2] ACC[*1] ,"SKE",*2\ /
                TOKEN[*1] ACC[*2] ,"SKE",*1\ /
                WORK[*2]  ACC[*1] ,"SKE","T+".W-W\ )
                    ,"BRU *+2"\ ,"BRU",#1\;

BGE[-,-,#1] => (TOKEN[*1] ACC[*2] ,"SKE",*1\,"SKG",*1\ /
                WORK[*1]  ACC[*2] ,"SKE","T+".W\,"SKG","T+".W-W\ )
                       ,"BRU",#1\;

BNE[-,-,#1] => (TOKEN[*2] ACC[*1] ,"SKE",*2\ /
                TOKEN[*1] ACC[*2] ,"SKE",*1\ /
                WORK[*2]  ACC[*1] ,"SKE","T+".W-W\ )
                     ,"BRU",#1\;

STORE[-,VAR[*.]]           => "*ITS ALREADY THERE"\
     [-,ADD[VAR[*1],CON["1"]]] => ,"MIN",*1\
     [-,ADD[VAR[*1],-]]    => ACC[*2:*2] ,"ADM",*1\
     [-,SUB[VAR[*1],-]]    => ACC[*2:*2] ,"CNA; ADM "*1\
     [-,-]             => BREG[*2] ,"STB",*1\ /
                          ACC[*2] ,"STA",*1\;
ADD[MINUS[-],-] => SUB[*2,*1:*1]
   [-,-]        => TOKEN[*2] ACC[*1] ,"ADD",*2\ /
                   WORK[*1] ACC[*2] ,"ADD","T+".W-W\;

SUB[-,-] => TOKEN[*2] ACC[*1] ,"SUB",*2\ /
            TOKEN[*1] (BREG[*2] ,"CBA; CNA; ADD "*1\ /
                       ACC[*2] ,"CNA; ADD "*1\) /
            WORK[*2] ACC[*1] ,"SUB","T+".W-W\;

MINUS[-] => TOKEN[*1] ,"LDA",*1\ ,"CNA"\ /
            BREG[*1] ,"CBA; CNA"\ /
            ACC[*1] ,"CNA"\;

DIVID[-,-] => TOKEN[*2] (BREG[*1] ,"CBA"\ /
                         ACC[*1]) ,"RSH 23; DIV "*2\ /
              WORK[*2] (BREG[*1] ,"CBA"\ /
                        ACC[*1]) ,"RSH 23; DIV  T+".W-W\;
```

```
BREG[MULT[-,-]] => TOKEN[*1:*2] ACC[*1:*1] ,"MUL",*1:*2"; RSH 1"\ /
                   TOKEN[*1:*1] ACC[*1:*2] ,"MUL",*1:*1"; RSH 1"\ /
                   WORK[*1:*1] ACC[*1:*2] ,"MUL","T+".W-W"; RSH 1"\
    [REM[-,-]]  => TOKEN[*1:*2] (BREG[*1:*1] ,"CBA"\ /
                                 ACC[*1]) ,"RSH 23; DIV "*1:*2\ /
                   WORK[*1:*2] (BREG[*1:*1] ,"CBA"\ /
                                ACC[*1:*1]) ,"RSH 23; DIV T+"
                                    .W-W"; RSH 1"\;

ACC[-] => TOKEN[*1] ,"LDA",*1\ /
          BREG[*1] ,"CBA"\ /
          *1;

WORK[-] => BREG[*1] ,"STB","T+"+W\ /
             ACC[*1] ,"STA","T+"+W\;

TOKEN[VAR[.ID]] => .EMPTY
     [CON[.NUM]] => .EMPTY;

MULT / => .EMPTY;

REM / => .EMPTY;

AND / => .EMPTY;

OR / => .EMPTY;

NOT / => .EMPTY;

ENDN / => "T","BSS",+W\ ,"END"\;

VAR[.ID] => *1;

CON[.NUM] => '= *1;

LE / => .EMPTY;

LT / => .EMPTY;

EQ / => .EMPTY;

GE / => .EMPTY;

GT / => .EMPTY;

NE / => .EMPTY;

DO[-,-] => *1 *2;

OPNINP[-,-] => ,"CLEAR; BRS 15; BRU "*2"; BRS 16; BRU "*2"; STA "*1\;

OPNOUT[-,-] => ,"CLEAR; BRS 18; BRU "*2"; LDX =3; BRS 19; BRU "
                                *2"; STA "*1\;
```

```
CLSFIL[-] => ,"LDA "*1"; BRS 20"\;

BRS38[-,.ID] => ,"LDA "*1"; LDB =10; BRS 38; STA "*2\
     [-,-]   => BRS38[*1,*2:*1] BRS38[*1,*2:*2];

XCIO[-,.ID] => ,"CIO "*1"; STA "*2\
    [-,-]   => XCIO[*1,*2:*1] XCIO[*1,*2:*2];

OUTCAR[-,.ID] => ,"LDA "*2"; CIO "*1\
      [-,.NUM] => ,"LDA ="*2"; CIO "*1\
      [-,.SR]  => ,"LDA ="#1"; LDB ="*2:L"; LDX "*1"; BRS 36; BRU "*2\
                                  #1,"ASC "''*2''\
      [-,-]    => OUTCAR[*1,*2:*1] OUTCAR[*1,*2:*2];

OUTNUM[-,.ID] => ,"LDA "*1"; LDA =10; BRS 38;"\
      [-,.NUM] => ,"LDA ="*2"; CIO "*1\
      [-,.SR] => ,"LDA ="#1"; LDB ="*2:L"; LDX "*1"; BRS 36; BRU "*2\
                 #1,"ASC "''*2''\
      [-,-] => OUTNUM[*1,*2:*1] OUTNUM[*1,*2:*2];

STARTN / => "START","EQU","*"\;

DECN[.ID] => *1,"BSS 1"\
    [-]   => DECN[*1:*1] DECN[*1:*2] ;

.END
```

--0--

```
                .META PROGRM    %5%

PROGRM =        ".META" .ID ?1? <"META II 1.1">
        [" NOLIST EXT,NUL;$START BRM INITL"]
        ["$KSTKSZ EQU 1;$MSTKSZ EQU 100;$NSTKSZ EQU 1;$SSSIZE EQU 550"]
        (".LIST" [,"CLA; STA LISTFG"] / .EMPTY)
        [,"BRM RLINE; BRM "*"; BRM FINISH"]
        ('( SIZ $(', SIZ) ') ?17E / .EMPTY)
                $ST ".END" ?2E
        ["STAR BSS 1;SSTOP DATA SS+SSSIZE-5;$SS BSS SSSIZE"]
        ["$MSP DATA MSTK;$MSPT DATA MSTK+MSTKSZ-5;$MSTK BSS MSTKSZ"]
        ["$NSP DATA NSTK;$NSPT DATA NSTK+NSTKSZ-5;$NSTK BSS NSTKSZ"]
        ["$KSP DATA KSTK;$KSPT DATA KSTK+KSTKSZ-5;$KSTK BSS KSTKSZ"]
        [,"END"]    <"DONE">;
ST =    .ID '= ?3E <"ST"> [*,"ZRO; LDA *-1; BRM CLL"]
        EXP ?4E '; ?5E  [,"BRU RTN"];
EXP =   SUBEXP $('/  [,"LDA MFLAG; SKE =0; BRU "*1]
                SUBEXP) [*1,"EQU *"];
SUBEXP =        (GEN / ELT [,"LDA MFLAG; SKE =1; BRU "*1])
                $REST [*1,"EQU *"];
REST = GEN / ELT [,"LDA MFLAG; SKE =0; BRU *+4"]
                ('?  .NUM ?12E [,"LDA ="*"; BRM ERR"]
                                (.ID [,"BRM",*]/ '?  [,"BRS EXIT"])?13E/
                .EMPTY [,"CLA; BRM ERR; BRS EXIT"]);
ELT = '.  .ID ?6E [,"BRM",*"; STA STAR"] /
        .ID [,"BRM",*]/
        .SR [,"BRM TST; DATA "*L"; ASC "'*'"] /
        '( EXP ?7E  ') ?8E  /
        '' .CHR [,"LDA ="*N"; BRM TCH"];
GEN = '[    $OUT '] ?10E  [,"BRM CRLF"] /
        '$  [*1,"EQU *"] ELT ?9E
                [,"LDA MFLAG; SKE =0; BRU "*1"; MIN MFLAG"] /
        ".EMPTY" [,"LDA =1; STA MFLAG"] /
        ".CHR" [,"BRM WPREP; BRM INC; LDA* IWP; STA STAR; MIN NCCP"] /
    '< .SR ?12E '> ?13E [,"BRM LITT; DATA "*L"; ASC "'*'"; BRM CRLFT"]
        "=>" [*1,"EQU *"] ELT ?14E
                [,"LDA MFLAG; SKE =0; BRU *+3; MIN NCCP; BRU "*1]/
        '! .SR ?15E [,*];
OUT =   .SR [,"BRM LIT; DATA "*L"; ASC "'*'"] /
        ',  [,"BRM TAB"] /
        '*  (.NUM [,"LDA =47B; CIO FNUMO; MIN CHNO; LDA GN"
                *"; BRM GENLAB; STA GN"*"; BRM OUTN"] /
                'L  [,"LDA* STAR; BRM OUTN"] /
                'N  [,"LDA STAR; BRM OUTN"] /
                'C [,"LDA STAR; CIO FNUMO; MIN CHNO"] /
                .EMPTY [,"LDA STAR; BRM OUTS"])/
        '' .CHR [,"LDA ="*N"; CIO FNUMO; MIN CHNO"]/
        '; [,"BRM CRLF"];
E = => ';  [,"BRU RTN"] $ST ".END" ?11E [,"END"] FINISH;
SIZ =   "K=" .NUM ["$KSTKSZ EQU "*] /
        "M=" .NUM ["$MSTKSZ EQU "*] /
        "N=" .NUM ["$NSTKSZ EQU "*] /
        "S=" .NUM ["$SSSIZE EQU "*];
                .END
```

```
--0--


                .META PROGRM  %TREE 1.3%

PROGRM = (".META" .ID ?1? (".LIST" :LIST[0]/ .EMPTY :MT[0]) SIZE
        :BEGIN[3] /
        ".CONTINUE" :MT[0] ) <"TREE 1.3"> :SETUP[1] *  $( RULE * )
        ".END" ?2E :ENDN[0] * <"DONE">;

SIZE = '(  SIZ $(', SIZ :DO[2]) ') ?50E / .EMPTY :MT[0];

SIZ = .CHR '= ?54E .NUM ?55E :SIZS[2];

RULE = .ID
        ( '= EXP ?3E ('& :KPOPK[1] / .EMPTY) :OUTPT[2] /
        '/ "=>" ?3E GEN1 :SIMP[2] /
        OUTRUL :OUTPT[2]) ?5E '; ?6E ;

EXP = '- SUBACK ?7E ('/ EXP ?8E :BALTER[2] / .EMPTY :BALTER[1]) /
        SUBEXP ('/ EXP ?9E :ALTER[2]/ .EMPTY);

SUBACK = NTEST (SUBACK :DO[2] / .EMPTY) /
        STEST (SUBACK :CONCAT[2] / .EMPTY);

SUBEXP = (NTEST / STEST) (NOBACK :CONCAT[2] / .EMPTY);

NOBACK = (NTEST / STEST ('? .NUM ?10E :LOAD[1] (.ID / '? :ZRO[0]) ?11E
        :ERCOD[3] / .EMPTY :ER[1])  )
        (NOBACK :DO[2] / .EMPTY);

NTEST =  ': .ID ?12E :NDLB[1] /
        '[ ( .NUM   '] ?14E :MKNODE[1] /
                GENP '] ?52E (':/.EMPTY :OUTCR[0] :DO[2]) ) /
        '< GENP '> ?53E (': /.EMPTY :OUTCR[0] :DO[2]) :TTY[1] /
        (".CHR" :GCHR /
        '* :GO) [0] /
        "=>" STEST ?15E :SCAN[1] /
        COMM;

GENP = GENP1 / .EMPTY :MT[0];

GENP1 = GENP2 (GENP1 :DO[2] / .EMPTY);

GENP2 = '* ('S .NUM ?51E :PAROUT[1] / .EMPTY :ZRO[0] :PAROUT[1])
        ('L :OL / 'C :OC / 'N :ON / .EMPTY :OS)[0] :NOPT[2]/ GENU;

COMM =  ".EMPTY" :SET[0] /
        '! .SR ?18E :IMED[1];

STEST = '. .ID ?19E :PRIM[1] /
       .ID :CALL[1]/
        .SR :STST[1] /
        '( EXP ?20E ') ?21E /
        '' .CHR :CTST[1]/
    (.NUM '$ ?23E /'$ :ZRO[0]) (.NUM /.EMPTY :MT[0]) STEST ?24E :ARB[3]/
        '- (.SR :NSR[1] / '' .CHR  :NCHR[1]) ?26E :NIST[1];
```

--1--

```
OUTRUL = '[ OUTR ?27E (OUTRUL :ALTER[2] / .EMPTY) :OSET[1];

OUTR = OUTEST  "=>" ?29E OUTEXP ?30E :CONCAT[2];

OUTEST = ( ('] :MT / "-]" :ONE / "-,-]" :TWO / "-,-,-]" :THRE) [0] /
                ITEMS '] )  :CNTCK[1];

ITEMS = ITEM (', ITEMS ?32E :ITMSTR[2] / .EMPTY :LITEM[1]) ;

ITEM = '- :MT[0] /
       .ID '[ ?33E OUTEST ?34E :RITEM[2]/
       NSIMP1 :NITEM[1] /
       '. .ID ?35E :FITEM[1] /
       .SR :TTST[1] /
       '' .CHR  :CHTST[1] /
       '# .NUM ?37E :GNITEM[1];

OUTEXP = SUBOUT ('/ OUTEXP :ALTER[2] / .EMPTY);

SUBOUT = OUTT (REST :CONCAT[2] / .EMPTY) / REST;

REST = OUTT (REST :OER[2]/ .EMPTY) / GEN (REST :DO[2]/ .EMPTY);

OUTT = .ID '[ ?39E ARGLST '] ?40E :OUTCLL[2] / '( OUTEXP ') ?41E /
       NSIMP1 (': ('S :OS / 'L :OL / 'N :ON/ 'C :OC)[0] :NOPT[2] /
                .EMPTY :DOIT[1]);

ARGLST = ARGMNT :ARG[1] (', ARGLST :DO[2] / .EMPTY) / .EMPTY :MT[0];

ARGMNT = NSIMP :ARGLD[1] / '# .NUM :GENARG[1];

NSIMP1 = <- ': NSIMP :UP[2] / NSIMP :LKT[1];

NSIMP = '* .NUM (<- ': NSIMP :CHASE[2] / .EMPTY :LCHASE[1]);

GEN1 = (OUT/COMM) (GEN1 :DO[2] / .EMPTY);

GEN = COMM / GENU / '< :TTY[0] / '> :FIL[0];

GENU =  OUT /
       '. .ID ?42E '[ ?43E ((.ID / .NUM) :LOAD[1] :CALL[2] /
                            .EMPTY :CALL[1]) '] /
       '# .NUM :GNLBL[1] (':  :DEF[1] / .EMPTY) ;

OUT = ('\ :OUTCR / ', :OUTAB) [0] /
       .SR :OUTSR[1] /
       '' .CHR :OUTCH[1] /
       "+W" :UPWRK[0] :OUTWRK[1] /
       "-W" :DWNWRK[0] /
       ".W" :MT[0] :OUTWRK /
       '?'W  :MAXWRK[0];

E = .EMPTY RESET  => ';  $( RULE * ) ".END" ?99E FINISH;
```

```
%OUT RULES%

SETUP [-] => ,"NOLIST NUL,EXT;GEN OPD 101B5,1,1;BF OPD 102B5,1,1"\
       "BT OPD 103B5,1,1;PSHN OPD 104B5,1,1;PSHK OPD 105B5,1,1"\
       "MKND OPD 106B5,1,1;NDLBL OPD 107B5,1,1;GET OPD 110B5,1,1"\
       "BPTR OPD 111B5,1,1;BNPTR OPD 112B5,1,1;RI1 OPD 113B5,1,1"\
       "RI2 OPD 114B5,2;FLGT OPD 115B5,1,1;BE OPD 116B5,1,1"\
       "LAB OPD 117B5,1,1;CE OPD 120B5,1,1;LDKA OPD 121B5,1,1"\
    "$KSTKSZ EQU 100;$MSTKSZ EQU 130;$NSTKSZ EQU 1300;$SSTKSZ EQU 1400"\
       *1;

BEGIN[-,-,-] => "$START BRM INITL; CLA; STA WRK; STA XWRK"\ *3 *2
       ,"BRM RLINE; BRM "*1"; BRM FINISH"\;

LIST / => " CLA; STA LISTFG;";


OUTPT[-,-] => *1:S ,"ZRO; LDA *-1; BRM CLLO"\ *2 ,"BRU RTNO"\;

SIMP[-,-] => *1 ,"ZRO"\ *2 ,"BRR "*1\;

BALTER[-] => ,"BRM SAV"\ *1 ,"BRM RSTR"\
       [-,-] => ,"BRM SAV"\ *1 ,"BRM RSTR; BT "#1\ *2 #1.D[];

D / => ,"EQU *"\;

ALTER[-,SET[]] => *1 *2
       [CONCAT[-,-],-] =>PMT[*1:*1,#1] *1:*2 ,"BRU "#2\ #1.D[] *2 #2.D[]
       [-,-] => *1 ,"BT "#1\  *2  #1.D[];

PMT[PRIM[-],#1] => ,"BRM "*1:*1:S"; BF "#1"; MRG "*1:*1:S"FLG; PSHK =0"\
       [-,-] => *1 ,"BF "#1\;

ER[ALTER[-,SET[]]] => *1
       [-] => *1 ,"BE =-1"\;

DO[-,-] => *1 *2;

CONCAT[-,-] => *1 ,"BF "#1\ *2  #1.D[];

LOAD[.NUM] => ,"LDA ="*1:S\
       [.ID] => ,"LDA "*1:S\;

CALL[-] => ,"BRM "*1\
       [-,-] => *2 ,"BRM "*1\;


MT / => .EMPTY;

CLA / => "CLA";

ZRO / => "0";
```

```
ERCOD[-,-,-] => *1 *2 ,"BE "*3\;

NDLB[-] => ,"NDLBL ="*1\;

MKNODE[-] => ,"MKND ="*1\;

ARB[ZRO[],MT[],-] => #1.D[] *3,"BT "#1"; MIN MFLAG"\
       [.NUM,MT[],-] => ARB1[*1] #1.D[]  *3
            ,"SKR* MSP; BT "#1"; SKN* MSP; BRU *+3; BT "#1"; MIN MFLAG"\
                .ARB3[]
       [-,.NUM,-] => ARB1[*2] #1.D[] *3
                ,"SKR* MSP; BT "#1"; SKN* MSP"\ ARB2[*1,*2];

ARB1[-] => ,"BRM SAV; LDA ="*1;S"+1; MIN MSP; STA* MSP"\;

ARB2[-,.NUM] => ,"BRU *+4; CLA; STA MFLAG; BRU *+4; LDA* MSP; SKG ="*2
                "-"*1"; MIN MFLAG"\ .ARB3[]
       [-] => ,"BRU *+3; CLA ; STA MFLAG"\ .ARB3[];

ARB3 / => ,"LDA =-1; ADM MSP; BRM RSTR"\;

GCHR /=> ,"BRM WPREP; BRM INC; LDA* IWP; MRG CHRFLG; MIN NCCP; PSHK =0"\

GO / => ,"BRM OUTREE; BT *+3; LDA =2; BRM CERR"\;

SET / => ,"LDA =1; STA MFLAG"\;

TTY[-] => TTY[] *1 FIL[]
   [] => ,"LDA =1; STA FNUMO"\ XCHCH[];

FIL[] => ,"LDA XFNUMO; STA FNUMO"\;

XCHCH/ => ,"LDA TCHNO; XMA CHNO; STA TCHNO"\;

STRING[-] => " DATA "*1;L"; ASC "''*1''"\;

OSET[-] => ,"BRM BEGN"\ *1;

CNTCK[-] => *1 ,"CLB; SKE NCNT; STB MFLAG"\;

ONE / => ,"LDA =1"\;

TWO / => ,"LDA =2"\;

THRE / => ,"LDA =3"\;

ITMSTR [-,-] => *1 ,"MIN CNT; EAX -1,2"\  *2;

LITEM [-] => *1 ,"MIN CNT; LDA CNT"\;

RITEM[-,-] => ,"RI1 ="*1"; BRU "#1\ *2 ,"RI2"\ #1.D[];

OER[-,-] => *1, "CE =1"\ *2;
```

```
OUTCLL [-,-] => ,"LDA NSP; STA SNSP; NDLBL ="*1"; CLA; STA CNT"\
                ,"LDA KT; STA ME"\ *2
       ,"MKND CNT; PSHN SNSP; LDX KT; BRM* 0,2; BRM POPK"\
       ,"LDA* NSP; STA NSP"\;

ARGLD[-] => ,"LDA ME"\ *1;

ARG [-] =>  *1 ,"PSHK =0; MIN CNT"\;

CHASE [-,-] => ,"GET ="*1"; BPTR *+3; LDA =3; BRM CERR"\ *2;

LCHASE [-] => ,"GET ="*1\;

DOIT [-] =>  *1  ,"BNPTR "#1
       "; CAX; PSHK =0; BRM* 0,2; BRM POPK; BRU *+2"\
       #1.D[] ,"BRM OUTS"\;

NOPT [-,-] => *1 ,"BNPTR *+3; LDA =4; BRM CERR;" *2;

SCAN [-] => #1.D[] *1 ,"BT *+3; MIN NCCP; BRU "#1\;

PRIM [-] => ,"BRM "*1"; BF *+3; MRG "*1"FLG; PSHK =0"\;

STST [-] => ,"BRM TST;" STRING[*1];

CTST [-] => ,"LDA ="*1:N"; BRM TCH"\;

OS / => " BRM OUTS"\;

ON / => " ETR =77777B; BRM OUTN"\;

OL / => " CAX; LDA 0,2; BRM OUTN"\;

OC / => " ETR =377B; CIO FNUM0; MIN CHNO"\;

GNLBL [-] => ,"GEN GNLB"*1\;

DEF [-] => *1 ,"BRM LIT; DATA 6; ASC "'"" EQU *"'"\;

OUTCR / => ,"BRM CRLF"\;

OUTAB / => ,"BRM TAB"\;

OUTSR [-] => ,"BRM LIT; " STRING[*1];

OUTCH [-] => ,"LDA ="*1:N"; CIO FNUM0; MIN CHNO"\;

FNDN / => "SSTOP DATA SS+SSTKSZ-5;$SS BSS SSTKSZ"\
       "MSP DATA MSTK;$MSPT DATA MSTK+MSTKSZ-5;$MSTK BSS MSTKSZ"\
       "NSP DATA NSTK;$NSPT DATA NSTK+NSTKSZ-5;$NSTK BSS NSTKSZ"\
       "KSP DATA KSTK;$KSPT DATA KSTK+KSTKSZ-5;$KSTK BSS KSTKSZ"\
       "WRK BSS 1;XWRK BSS 1; END"\;

SAVG [-] => ,"BRM SAVGN"\ *1 ,"BRM RSTGN"\;
```

```
IMED [-] => ,*1\;

NITEM[-] => ,"STX INDX; LDA KT"\ *1
        ,"CLB; LDX INDX; SKE 0,2; STB MFLAG"\;

FITEM[-] => ,"FLGT "*1:S"FLG"\;

TTST[-] => ,"BRM SSTEST;" STRING[*1];

CHTST[-] => ,"CLB; LDA ="*1:N"; MRG CHRFLG; SKE 0,2; STB MFLAG"\;

GNITEM[-] => ,"FLGT GENFLG; ETR =77777B; STA GNLB"*1:S\;

GENARG[-] => ,"LAB GNLB"*1:S"; MRG GENFLG"\;

NTST[-] => ,"LDA NCCP; STA SNCCP"\ *1
        ,"LDA =1; SKR MFLAG; BRU *+2; STA MFLAG; LDA SNCCP; STA NCCP"\;

NCHR[-] => ,"LDA ="*1:N"; BRM TCH"\;

NSR[-] => ,"BRM TST; "STRING[*1];

UP["1",-] => ,"LDA* KSP"\ *2
        [-,-] => ,"LDX KSP; LDA 1-"*1:S",2"\ *2;

LKT[-] => ,"LDA KT"\ *1;

UPWRK / => ,"MIN WRK; LDA WRK; SKG XWRK; LDA XWRK; STA XWRK"\;
DWNWRK / => ,"LDA =-1; ADM WRK"\;
OUTWRK[-] => *1, "LDA WRK; BRM OUTN"\;
MAXWRK / => ,"LDA XWRK; BRM OUTN"\;
SIZS[.CHR,-] => *1:C"STKSZ EQU "*2:S\;

KPOPK[-] => ,"MIN MSP; LDA KT; STA* MSP; MIN MSP; LDA KSP; STA* MSP"\
   *1 ,"LDX MSP; LDA 0,2; STA KSP; LDA -1,2; STA KT; LDA =-2; ADM MSP"\;

PAROUT[ZRO[]] => ,"LDA KT"\
        ["0"] => ,"LDA KT"\
        [-] => ,"LDKA ="*1\;

                .END
```

BLANK PAGE

BIBLIOGRAPHY

1 (Engelbart1) D. C. Engelbart, "Special Considerations of the Individual as a User, Generator, and Retriever of Information," Paper presented at Annual Meeting of American Documentation Institute, Berkeley, California (23-27 October 1960).

2 (Engelbart2) D. C. Engelbart, "Augmenting Human Intellect: A Conceptual Framework," Summary Report, Contract AF 49(638)-1024, SRI Project 3578, Stanford Research Institute, Menlo Park, California (October 1962), AD289565.

3 (Engelbart3) D. C. Engelbart, "A Conceptual Framework for the Augmentation of Man's Intellect," vistas in information handling, Vol. 1, D. W. Howerton and David C. Weeks, eds. (Spartan Books, Washington, D.C., 1963).

4 (Engelbart4) D. C. Engelbart, "Augmenting Human Intellect: Experiments, Concepts, and Possibilities," Summary Report, Contract AF 49(638)-1024, SRI Project 3578, Stanford Research Institute, Menlo Park, California (March 1965), AD640989.

5 (Engelbart5) D. C. Engelbart and B. Huddart, "Research on Computer-Augmented Information Management," Technical Report ESD-TDR-65-168, Contract AF 19(628)-4088, Stanford Research Institute, Menlo Park, California (March 1965), AD622520.

6 (English1) W. K. English, D. C. Engelbart, and B. Huddart, "Computer-Aided Display Control," Final Report, Contract NAS 1-3988, SRI Project 5061, Stanford Research Institute, Menlo Park, California (July 1965).

7 (Engelbart6) D. C. Engelbart, W. K. English, and J. F. Rulifson, "Study for the Development of Human Intellect Augmentation Techniques," Interim Progress Report, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (March 1967).

8 (Engelbart7) D. C. Engelbart, "Study for the Development of Human Intellect Augmentation Techniques," Final Report, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (March 1968).

9 (Hopper1) J. D. Hopper and L. P. Deutsch, "COPE: An Assembler and On-Line-CRT Debugging System for the CDC 3100," Technical Report 1, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (March 1968).

10 (Hay1) R. E. Hay and J. F. Rulifson, "MOL940: A Machine-Oriented ALGOL-Like Language for the SDS 940," Technical Report 2, Contract NAS 1-5904, SRI Project 5890, Stanford Research Institute, Menlo Park, California (March 1968).

BIBLIOGRAPHY

11 (Pirtle1) M. Pirtle, "Intercommunication of Processors and Memory," Proceedings of Fall Joint Computer Conference (November 1967).

12 (Rulifson1) J. F. Rulifson, "Aspects of Reliability and Response in a Display-Oriented Time-Sharing System," Stanford Research Institute, Menlo Park, California (April 1968).

13 Note: Reports with AD numbers are available from Defense Documentation Center, Building 5, Cameron Station, Alexandria, Virginia 22314. Reference Nos. 4 and 6 may be obtained from CFSTI, Sills Building, 5825 Port Royal Road, Springfield, Virginia 22151; cost $3.00 per copy or 75 cents for microfilm.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Stanford Research Institute
333 Ravenswood Avenue
Menlo Park, California 94025

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP
N/A

3. REPORT TITLE
DEVELOPMENT OF A MULTIDISPLAY, TIME-SHARED COMPUTER FACILITY AND COMPUTER-AUGMENTED MANAGEMENT-SYSTEM RESEARCH

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Final Report

5. AUTHOR(S) (First name, middle initial, last name)
D. C. Engelbart
W. K. English
J. F. Rulifson

6. REPORT DATE
October 1968

7a. TOTAL NO. OF PAGES
185

7b. NO. OF REFS
13

8a. CONTRACT OR GRANT NO.
AF30(602)-4103

b. PROJECT NO.
5581

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
SRI Project 5919

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
RADC-TR-68-250

10. DISTRIBUTION STATEMENT
This document is subject to special export controls and each transmittal to foreign governments, foreign nationals or representatives thereto may be made only with prior approval of RADC (EMIIF), GAFB, N.Y. 13440.

11. SUPPLEMENTARY NOTES
RADC PROJECT ENGINEER
Fred Normand
AC-315-330-3678

12. SPONSORING MILITARY ACTIVITY
Rome Air Development Center (EMIIF)
Griffiss Air Force Base, New York 13440

13. ABSTRACT

Special software developments, to facilitate easy implementation and modification of powerful interactive user aids, include the Tree Meta compiler generator, the MOL940 Machine-Oriented systems programming Language, four Special-Purpose Languages (SPL's) for high-level specification of user-control dialogue and interactive functions, and an associated On-Line System (NLS) integrating these components

DD FORM 1 NOV 65 1473

UNCLASSIFIED
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Multidisplay, Time-Shared Computer Facility						
Interactive Man/Machine System						
Computer Aids to Management						
Computer Display Generator						
Syntax-Driven Compiler-Compiler						
Special-Purpose Languages						