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)*

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

2a. REPORT SECURITY CLASSIFICATION

UNCLASSIFIED

2b. GROUP

3. REPORT TITLE

A DESCRIPTION OF THE ILLIAC IV OPERATING SYSTEM

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Research Report

5. AUTHOR(S) *(First name, middle initial, last name)*

P. A. Alsberg, J. L. Gaffney, G. R. Grossman, T. W. Mason, G. A. Westlund

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
November 1, 1968	89	

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
46-26-15-305	ILLIAC IV Document No. 212
b. PROJECT NO. USAF 30(602)4144	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* DCS File No. 791
d.	

10. DISTRIBUTION STATEMENT

Qualified requesters may obtain copies of this report from DCS.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
NONE	Rome Air Development Center Griffiss Air Force Base Rome, New York 13440

13. ABSTRACT

The operating system for ILLIAC IV is described. Residing principally on the B6500 computer, the operating system includes a job parser, a program collector, a disk file allocator, a data processor, a loader, an execution monitor, job partners, a hardware supervisor and a subsystem resident in ILLIAC IV. A brief description of the ILLIAC IV hardware and discussions of file security and interactive uses of ILLIAC IV are included.

DD FORM 1 NOV 65 1473

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
job parser						
program collector						
disk file allocator						
data processor						
loader						
execution monitor						
job partner						
hardware supervisor						
subsystem						
file security						

ILLIAC IV Document No. 169

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

A DESCRIPTION OF THE ILLIAC IV OPERATING SYSTEM

by

P. A. Alsberg

J. L. Gaffney

G. R. Grossman

T. W. Mason

G. A. Westlund

File No. 791
November 1, 1968

ABSTRACT

The operating system for ILLIAC IV is described. Residing principally on the B6500 computer, the operating system includes a job parser, a program collector, a disk file allocator, a data processor, a loader, an execution monitor, job partners, a hardware supervisor and a subsystem resident in ILLIAC IV. A brief description of the ILLIAC IV hardware and discussions of file security and interactive uses of ILLIAC IV are included.

ACKNOWLEDGMENT

Much of the organization of the operating system is due to the efforts of N. Saville, as recorded in his ILLIAC IV document no. 181, "A Description of the System Specification of OSK: An Operating System for ILLIAC IV". Acknowledgment is also extended to D. McIntyre for writing Appendix C, "Interactive Uses of ILLIAC IV".

TABLE OF CONTENTS

I. INTRODUCTION

The operating system provides several services. Each of these services is implemented by one or more program modules within the system. A diagram of the interaction of the modules is given in Figure 1-1.

The user interface to the operating system is through the Job Parser. The job parser can be driven by such available input media as tapes, card decks, user operated consoles, or other running B6500 programs. It scans a user's job control language and provides him with access, some of which are conversational for console users, to a variety of utility routines and parts of the operating system. The utility routines perform data reformatting, file editing, compiling, file collecting, and assembling functions. Virtually any utility can be added. For example, the job parser has a conversational interface to an education utility that teaches a user how to use the system and provides him with system documentation.

The collection of programs and some of the data needed on the ILLIAC IV is performed within the B6500 by the program collector module.

The scheduling of ILLIAC IV time is performed by three system modules: the disk file allocator, the data pre/post processor, and the execution monitor. These three programs queue up requests for ILLIAC IV use, assign Disk IV space to them, move any files that are needed onto the disk before a run, save any files by removing them from the disk after a run, schedule the use of BIOM space, and allocate particular ILLIAC IV quadrants to the individual jobs (see Appendix A for a description of the ILLIAC IV hardware).

There are two modules resident in the ILLIAC IV, the loader and OS4. The loader loads program files from Disk IV into the array memory. OS4 provides standard monitor functions and I/O execution coordination routines needed by all users.

The running ILLIAC IV job requires at least two intercommunicating programs, the ILLIAC IV resident program and a B6500 resident job partner that coordinates B6500 actions with the ILLIAC IV program and provides all I/O support. Each job must have at least one job partner for all of its

quadrants. Standard job partners are furnished by the operating system; however, the user has the option of supplying his own job partner(s).

The building of I/O descriptors, the initial recognition of interrupts, and the actual issuing of I/O commands to the hardware are functions of a set of routines in the B6500 Master Control Program (MCP) which are collectively known as the hardware supervisor. Since a user may choose not to use the system-supplied job partner with the execution of his ILLIAC IV program, but may write his own, the hardware supervisor also checks all of the I/O requests made by a job partner for validity before passing them onto the hardware.

Most of the modules named above are separately running B6500 ALGOL programs. They communicate with each other through the use of the in-core file facility provided by the B6500 MCP. The hardware supervisor routines (called "MCP intrinsics") are coded in ESPOL to facilitate the issuing of I/O descriptors and the passing of interrupts to the correct job partner. The ILLIAC IV loader and OS4 are coded in the ILLIAC IV assembly language.

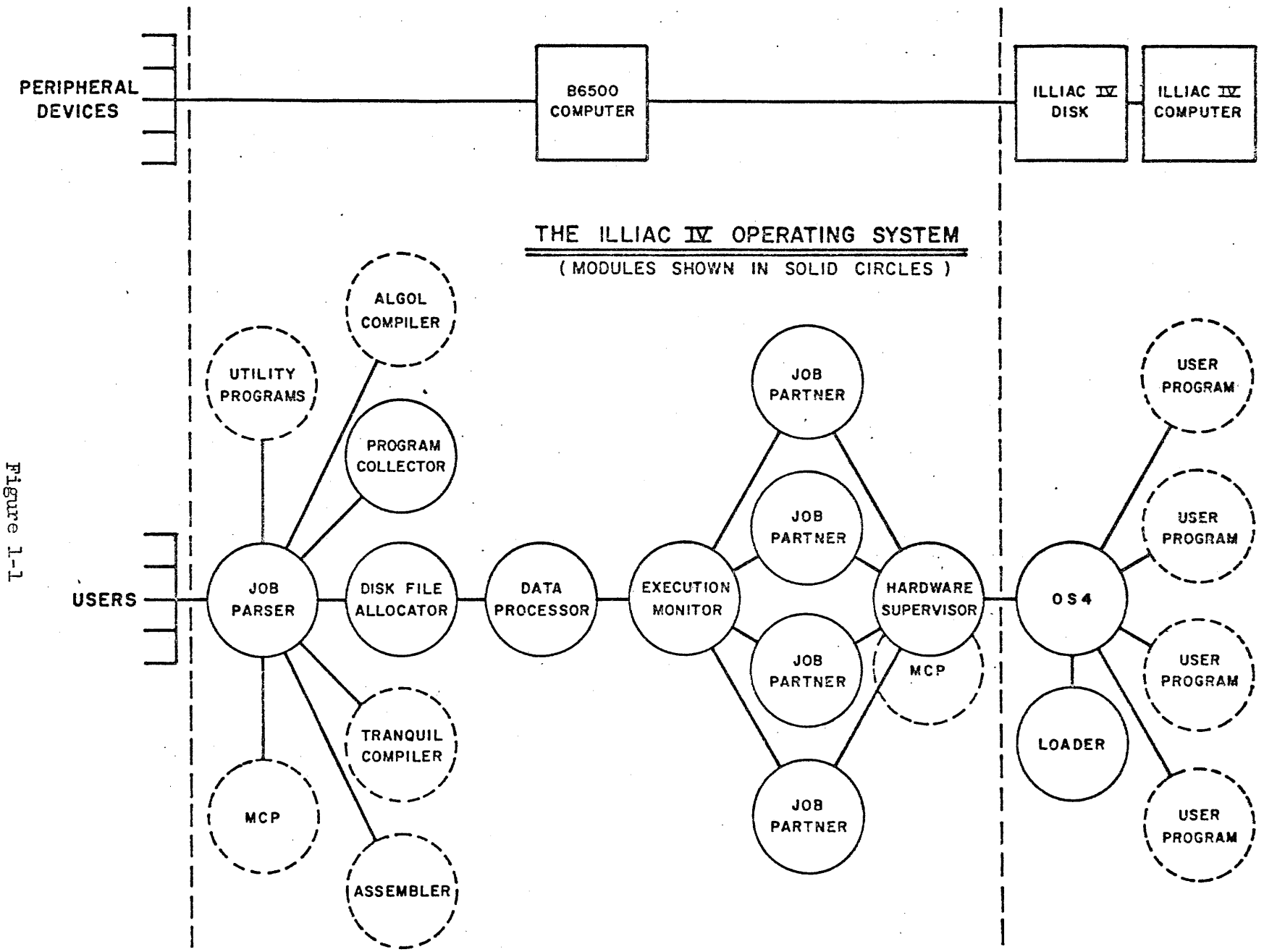

PERIPHERAL DEVICES
B6500 COMPUTER
ILLIAC IV DISK
ILLIAC IV COMPUTER
THE ILLIAC IV OPERATING SYSTEM
(MODULES SHOWN IN SOLID CIRCLES)
ALGOL COMPILER
UTILITY PROGRAMS
PROGRAM COLLECTOR
USERS
JOB PARSER
DISK FILE ALLOCATOR
DATA PROCESSOR
EXECUTION MONITOR
JOB PARTNER
JOB PARTNER
JOB PARTNER
JOB PARTNER
HARDWARE SUPERVISOR
MCP
OS4
USER PROGRAM
USER PROGRAM
USER PROGRAM
USER PROGRAM
LOADER
TRANQUIL COMPILER
MCP
ASSEMBLER

Figure 1-1

II. THE JOB PARSER

The job parser is the user's and operator's interface to the operating system. It is driven by any available input medium, such as tapes, card decks, and user operated consoles. Interfaces are provided to a variety of utility routines and to parts of the operating system.

Regardless of the input medium, all users submit log-in statements and satisfy file security requirements (project name and individual user code). Once admitted to the system, a variety of control language statements are at their disposal. Various compilers and assemblers are available, such as Tranquil, a higher-level language, and ASK, an assembly language. In addition, the users at conversational devices have access to utility routines, such as TEACH, FILE/SECURITY, and REMOTE/EDIT.

TEACH is an education utility that instructs the user in the use of the system and provides him with levels of system documentation. Step-by-step instructions for getting a program run on ILLIAC IV are provided in a conversational manner. A listing of the available job control language statements and a description of their functions is also given.

FILE/SECURITY is a utility program used to admit new projects and users to the system. It is also used to declare file access for projects and users. File Security on ILLIAC IV is tree-structured by projects and sub-projects. The organization of the security system is available for inspection by any valid user, with the exception that user authentication codes are never revealed. A description of the file security system is contained in Appendix B.

REMOTE/EDIT is a file editing utility used to create or modify data or source program files. The utility provides features similar to those provided by the B5500 program REMOTE/CARD.

The job that the user submits to the parser is composed of one or more B6500 and/or ILLIAC IV programs called elements. An example of an element would be an ASK assembly, a Tranquil compilation, an ILLIAC IV execution, a data reformatting utility program, etc.

The job parser is responsible for executing these elements in the proper order for the user. The user specifies this order by breaking the job

up into serial and parallel job steps where each job step may be an element or can be composed of one or more other serial and parallel job steps nested to any depth. Steps that are serial must be run sequentially while steps that are parallel are independent of each other and may be, but not necessarily will be, run simultaneously on different processors in the B6500 or ILLIAC IV. The user is allowed to set time limits for the whole job and for the individual steps.

A job is then defined to be a job step with its associated accounting information. For example, consider the user who wishes to compile a Tranquil main program and two ASK subroutines. Following the compilation and assemblies, the user wishes to collect his main program and necessary subroutines prior to his ILLIAC IV run. At the same time the above steps are being taken, the user would like some of his data files reformatted before his ILLIAC IV execution. Once all of the above is done, the ILLIAC IV program is to be run and that run followed by a printing step to process his output.

This job is broken down into elements in Figure 2-1. In terms of steps, the job consists of three serial steps: preparation, ILLIAC IV execution, and printing. The preparation step has parallel code preparation and data preparation steps. The code preparation step has serial translation and collection steps; and the translation step has parallel compilation and assembly steps.

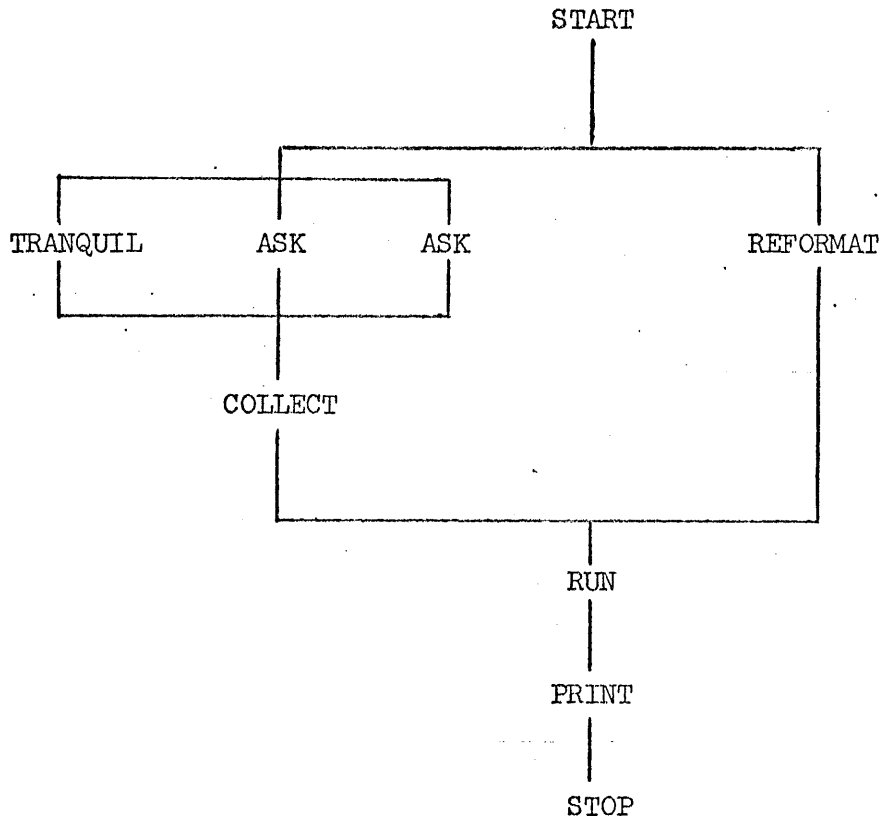


Figure 2-1. ELEMENTS OF A SAMPLE JOB

Each element can return a condition code that may be tested to determine whether succeeding job steps are to be executed or skipped. Thus, the user can specify that no steps are to be executed if his compilation were to fail; as would be indicated by the condition code returned by the compiler.

All of the above control information is specified to the job parser in the ILLIAC Control Language (ICL). ICL allows the external specification of job steps and allows parameters to be passed to these job steps by the user. Therefore, the system and the user can keep libraries of commonly used steps on the B6500 disk. For example, the job in Figure 2-1 could exist as a prepared step on disk and be called in with one statement every time a user wishes to run it. The typical compile-run-print steps that a student might use are available in the system library and have appropriate parameters for specifying input files and alternative output devices.

ICL statements are used to specify both ILLIAC IV and B6500 files. This obviates the need to switch to B6500 control cards to specify B6500 files. Some file information may be built for the parser by the user's file declarations in Tranquil or ASK. This information is stripped off the code file at collection time and prepared as an ICL statement by the collector. In steps that are elements, the program provided ICL statements, and current user ICL statements are merged to produce the complete file specifications. In case of conflict, the current user statements supersede the program statements.

The execution of a B6500 element involves the preparation of suitable B6500 control information for use by the Burroughs' MCP to initiate the program.

ILLIAC IV execution requires considerably more effort on the part of the parser. Several ILLIAC IV elements may pass files between each other. That is, the output of one step may be input to another. When this situation is recognized by the parser, all such interrelated ILLIAC IV elements are gathered together as one ILLIAC IV scheduling request. This is done so that the passed files are not moved on and off of the ILLIAC IV disk for each element's execution. An internal identification number is assigned,

and a job dossier is built on the B6500 disk for all system modules to look at. The dossier contains all of the file information, maximum limits, and step information provided by the user's and program's ICL statements. In addition, information needed by the system to schedule the request is also contained in the dossier. A short message is then sent to the disk allocator. The message identifies the parser as the sender, indicates that this is a schedule request, and passes a pointer to the dossier.

The disk allocator then proceeds to queue and finally provides disk space for the entire scheduling request. Once the request has completed ILLIAC IV execution and data files have been moved back to the B6500, the disk space assigned to that request is freed and the disk allocator notifies the job parser of step completion and passes back condition codes and other appropriate information.

The job parser also interfaces with the operator's console. The operator has privileged access to the system parameters and is therefore capable of ordering a change in the state of the system. Typically, the operator may wish to interrogate the progress of a job in the system, perhaps hold a job at a certain state in processing, or change some scheduling parameters to counter an unusual shift in the job mix. The message identifies the sender, the operation to be performed, and a parameter that typically is an internal job identification number or the new value of a scheduling parameter. For example, on receipt of a message to hold a job at its current point, the parser will prepare a hold message and insert the internal job number. If the job is presently in the parser, it is held there until further operator action. If the job is not there, the message is sent to the disk file allocator who either holds the job if he has it or reoriginates the message to the data processor. The message thus chases the job down from module to module, following the same path the job must take until it finds the job or goes to the execution monitor, back to the data processor, back to the disk file allocator, and finally to the job parser without finding it. If the complete circuit is taken without finding the job, the job is no longer in the system. The message must double back like the job because it may pass the job it is looking for on the way through the modules. If,

for example, module A does not have the job and passes the message to module B who has it, module B may finish with the job and pass it to A before it reads the message from A.

A message to change a scheduling parameter may only pass through the system in one direction. It is passed from module to module until a module recognizes the task as belonging to it and resets the appropriate parameter. For example, a request to change the category 3 time shard (time shards are explained in the disk file allocator description) from 8 to 10 would go to the job parser, then to the disk allocator, who will recognize a time shard change request as his area of jurisdiction and will take appropriate action.

III. THE PROGRAM COLLECTOR

The Collector is a B6500 program, written in Extended ALGOL, the purpose of which is the preparation of programs, as output by assemblers or compilers, for eventual execution on ILLIAC IV. The following facilities are provided:

1) gathering program segments from user files, user libraries, and/or the system library, and verifying that they are indeed ILLIAC IV program elements;

2) reformatting program segments as output by assemblers or compilers into a standard form suitable for input to the loader;

3) verifying the validity of links between program elements (i.e., the absence of duplicate or missing links);

4) constructing core-load, sub-core-load, and super-core-load program structures which may be used immediately on ILLIAC IV and/or saved for later use (see Sheaves, below);

5) mapping overlayable program elements into ILLIAC IV memory according to the directions of the user;

6) constructing and maintaining system and user program libraries;

7) communicating to the operating system, the loader, and/or the user any information necessary and/or requested for the proper disposition of the product of its actions (error conditions, memory maps, timing information, etc.).

These facilities are explained in more detail below.

CONTROL File

The Collector executes commands passed to it in a card image file named CONTROL. The command verbs are:

MAP <map file name> [<map parameters>]
causes the Collector to use the card image file denoted by <map file name> as a set of Memory Allocation Processor commands. See below.

LOAD <load file name> [<load parameters>]

causes the Collector to prepare the contents of the file denoted by <load file name> for processing by the loader, including verification of the contents of the file as a complete ILLIAC IV program.

LIBRARY <library file list>

causes the Collector to recognize the files named in the <library file list> as user-provided program libraries (see below).

CREATE <library create parameters>

causes the Collector to create a library file according to the <library create parameters>.

EDIT <library edit parameters>

causes the Collector to edit a library file according to the <library edit parameters>

In order for a job to be run on ILLIAC IV, it is necessary that the Collector be invoked in order to a) perform all operations necessary to form a complete ILLIAC IV program ready for the Loader and/or b) verify that all such operations have been performed and communicate this verification to the operating system. These functions are initiated by the LOAD statement. Only one LOAD statement is allowed per run, and all runs involving CONTROL files containing LOAD statements must be initiated through the Job Parser. Runs in which the CONTROL file does not contain a LOAD statement may be initiated directly by the user.

If a job consists of only a main program, with all external program segments (subroutines, data space allocation, etc.) residing in system or user program libraries, the CONTROL file may consist of (optional) LIBRARY statements and a LOAD statement, and no MAP statement or file is necessary. If, however, more than one user-supplied, non-library-contained program segment is present, then at least one MAP statement must appear in the CONTROL file and at least one corresponding MAP file must exist. MAP files are explained in the next section.

MAP Files

A MAP (Memory Allocation Processor) file consists of card images containing statements in a language which allows the user to specify the origins of program segments and the manner in which they are to be connected to form core-load, sub-core-load, and super-core-load (overlayable) program structures.

The basic element of a program structure is called a STALK. Stalks are program segments as output by assemblers or compilers, or sub-program structures previously created by action of the Collector.

When Stalks are combined to form program structures, the result is called a Sheaf. Sheaves may consist of one or more Stalks, each of which may itself be a Sheaf.

MAP Language Statements

MAP language statements are of two types: STALK statements and SHEAF statements. The syntax for the STALK statement is:

<stalk statement> ::= STALK <stalk name> <file part> <select part>
<stalk name> ::= <identifier>
<file part> ::= <file name>/<entry name> LIBRARY /<entry name> SYSTEM
<select part> ::= <empty>/[<entry selection criteria> <boj part>]
<entry selection criteria> ::= <entry inclusion criteria> / <entry exclusion criteria>
<entry inclusion criteria> ::= <entry list>
<entry exclusion criteria> ::= NOT <entry list>
<boj part> ::= @<entry name>
<entry list> ::= <entry name> / <entry list>,<entry name>
<entry name> ::= <identifier>

The STALK statement causes the Collector to recognize a file or part of a file as a program segment and to label it with an identifier. If the <file part> is a file name, the file of that name is assumed to contain the program segment. The <file name> may denote a library file, which need not be declared in a LIBRARY statement in the CONTROL file. If the <entry name> LIBRARY construct is used, the Collector will search all user libraries

declared in LIBRARY statements in the CONTROL file previous to the MAP statement which caused the Collector to be directed to the current MAP file of which the STALK statement is a part. The search will continue until a) a program segment is found which contains the <entry name> as a declared entry, or b) until all user files are exhausted, in which case the system library will be searched. If a segment is found in either the user or system libraries, it will be used to form the Stalk; otherwise, an error condition exists. If the <entry name> SYSTEM construct is used, only the system library will be searched as above.

If the <select part> is empty, then all entry names declared in the program segment are preserved, and any previously specified first executable location in the program segment is preserved. If the <select part> is present with the <entry inclusion criteria> construct, only those <entry name>s in the <entry list> will thereafter be recognized as entries to the Stalk. If the <entry exclusion criteria> construct is present, then all <entry name>s not in the <entry list> will thereafter be recognized. If the <boj part> is present, the first executable location in the program segment will thereafter be considered to be the location denoted by the <entry name>.

The syntax for SHEAF statement is:

<sheaf statement> ::= SHEAF <sheaf name> <save name>
[<map phrase>] <select part>
<sheaf name> ::= <identifier>
<save name> ::= <empty>/<file name>
<map phrase> ::= <contiguous segment> | <map phrase> <contiguity operator> <contiguous segment>
<contiguous segment> ::= <overlay segment> | <contiguous segment> <overlay operator> <overlay segment>
<overlay segment> ::= <stalk name> | <sheaf name> | (<map phrase>)
<contiguity operator> ::= <
<overlay operator> ::= =

The <sheaf statement> causes the Collector to combine previously created Stalks and/or Sheaves into a new program segment, in which the relative layout of memory is specified, and to label the segment with an identifier.

If the <save name> is present, a copy of the Sheaf will be created with the given <file name>. The <save name> must be present if the Sheaf is to be the file named in the LOAD statement in the CONTROL file of a Collector run.

The <map phrase> construct specifies which Stalks and Sheaves are to be combined into the new program segment and how that program segment is to be laid out in ILLIAC IV memory. The operator "<" specifies that the program segment denoted by <contiguous segment> following the operator is to be located at a higher address in ILLIAC IV memory than the program segment preceding the "<". The operator "=" specifies that the program segments bracketing the "=" are to be allocated the same memory space (i.e., they "overlay" each other). The "=" operator has precedence over the "<" operator. The use of parentheses delimits the scope of the operators in the traditional algebraic sense.

The <select part> in the <sheaf statement> has the same meaning as in the <stalk statement>.

Program Segment Files

A program segment as output by an assembler or a compiler must have the following format:

1) its physical record size must be a multiple of the disk segment size as defined by the B6500 hardware;

2) it must contain program words consisting of 16 bits of loader information followed by 32 bits of instruction syllable, data half-word, or loader instruction (see Section IX); the last three program words must consist of two ILLIAC IV HALT instruction syllables followed by a loader THEND instruction;

3) following the THEND instruction must come the ENTRY table, if any; the ENTRY table consists of the names of those locations of the program segment which have been declared as entries paired with their relative locations in the program segment;

4) following the ENTRY table must come the EXTERNAL reference list, consisting of those names declared as external to the code

segment; all reference to these external names within the program segment must be by their position in the EXTERNAL reference table;

5) the last record of a program segment file must begin on a disk segment boundary and must contain the following information in 48 bit words:

WORD	CONTENTS
0	"ILLIAC IV" in BCL, used to identify the file as a program segment file;
1	length of the loaded program segment in syllables;
2	word number in the file at which the ENTRY table begins;
3	word number in the file at which the EXTERNAL table begins;
4	number of entries in the EXTERNAL table;
5	modulus of the largest FILL in the program segment (see Section IX);
6	first executable location in the program, or zero if not specified;
7	must be zero;
8	must be zero.

Format of Sheaf Files

Sheaf files contain all information relating to the program segments which comprise them, such as length, entries, external names referenced, overlay information, etc., as well as the program segments themselves. Each program segment in a Sheaf file has the following format:

WORD	CONTENTS
0	length of the program segment in ILLIAC IV syllables;
1	length of the external reference table;
--	the external reference table, consisting of names paired with pointers to the master entry table (see below);

WORD	CONTENTS
--	a loader PNOP instruction (see Section IX);
--	program words, in a different format from those in the program segment file as output from the assembler or compiler; the 16 bits of loader information in each program word are stripped off and combined into 64 bit words, each of which contains the loader information for four ILLIAC IV 32-bit instruction syllables; the 32-bit instruction syllables are combined into 64-bit words, each of which contains two such syllables; the layout of these words is as follows:

0) 64×64 bit words, each containing two 32-bit syllables

1-7) same as 0);

8) 64×64 bit words, each containing four 16-bit loader information fields; the 0th word contains the information corresponding to the four syllables in word 0 of row 0 and word 0 of row 1, in that order; in general, word $n+64 \times 8$ of each block of program words contains the loader information for the syllables in words n and n+64;

Each program segment in a Sheaf file must end with a loader THEND instruction.

Format of Program Library Files

Program library files are B6500 disk files which contain program segments which are catalogued according to their entry names. Each program library file contains two types of records: directory records and program segment records. The directory records contain entries of the following form:

<entry name> <location in file> <length> where <entry name> is a name declared as EXTERNAL in a program segment, <location in file> is the logical record number in the file of the first record containing the program segment, and <length> is the length of the program segment in logical records.

Record 0 of a program library file must contain "LIBRARY" in BCL in word 0. The last word in each directory record points to the next logical

record which is a directory record. If the contents of this word is -1, the record is the last directory record in the file. Empty entry positions in the directory records are flagged by the character "?" in the <entry name> part of the entry.

Program library files are created and edited by the Collector according to the information in the CREATE and EDIT statements in the CONTROL file.

Symbolic Output from the Collector

The Collector provides to the user, on request, symbolic output on output devices such as teletypes or line printers. This output consists of detailed reports of the Sheaving operations, including memory maps, and library create or edit information, such as listings of library directories.

IV. THE DISK FILE ALLOCATOR

The disk file allocator queues and schedules requests for ILLIAC IV disk space. Although its primary responsibility is the allocation of disk space, the disk file allocator, because of its position in the ILLIAC IV job flow, also performs scheduling.

The scheduling of jobs is distributed over three modules: the disk file allocator, the data processor, and the execution monitor. Each of these modules schedules its own tasks independently of the others; therefore, in this operating system, there is no portion that can be called "the scheduler"--that is, there is no section of the operating system that oversees all operations or is even aware of the state of all jobs in the operating system.

A request for ILLIAC IV processing goes first to the disk file allocator to get disk space and then to the data processor to transfer the B6500 files to the ILLIAC IV disk. Once a user's data is on the disk, the request is sent to the execution monitor which schedules the use of the ILLIAC IV quadrants and the BIOM. After ILLIAC IV processing, the job moves again to the data processor where ILLIAC IV files are moved back to the B6500. The final action is to notify the disk file allocator to free the previously assigned disk space.

It is expected that the jobs submitted to ILLIAC IV will be of considerable size and thus only a few jobs can be kept on the disk. Therefore, the data processor and execution monitor are constrained to attempt a sophisticated type of scheduling that may stall one job on the disk while other jobs move past it. Instead, an attempt is made to speed the turnover of all jobs so that the occupied disk space is freed as soon as possible. In view of this, a "first come-first served" scheduling algorithm operates in those modules.

The disk file allocator, then, inherits the responsibility for selecting a particular job to enter the processing stream. The disk file allocator will schedule requests for fragments of time called time shards that are assigned to n categories. Each of the n categories contains jobs which are within some maximum specified limits and require the class of

service or turnaround time of that category. Each category has a time shard and a time slice. The time slice is initially set to the time shard. Both of these time units are expressed in quadrant minutes (240 quadrant minutes per hour). The time shard represents the portion of processing time that a category should receive relative to other categories, and the time slice represents the amount of this shard that has not yet been used.

For an example, consider the three-category system shown below. The lowest category (I) represents the best service.

	Category		
Shift	I	II	III
1	60	10	0
2	25	20	35
3	9	1	23
	limit of 20 quadrant-minutes per job	limit of 60 quadrant-minutes per job	unlimited in execution time

This shows that jobs executed in the first shift should roughly conform to a 60:10:0 mix. Shift 2, unlike shift 1, will process jobs in the lowest category of service, and shift 3 time is heavily weighted toward processing of category III jobs.

The operator adjusts the shards for each category during his shift so that he maintains some particular rate of turnaround time for each category. Of course, there will be times when everyone wants category one, in which case, higher category users will be slighted in favor of maintaining the turnaround time in the lower categories.

An adjunct to the scheduling process is the project time budget for each category. Although most projects will have identical budgets, some may have requirements for special services which will be reflected in their time budget. The project budgets are used to level the load placed on each category.

The time used by each run in a given category is decremented from the time remaining from the original allotment in that category. Once a time allotment becomes negative or zero, that category cannot be used by the

project unless it is the lowest category. Time is returned to a category by the following algorithm. Once all categories have used at least some portion, for example ten percent of their budget, then that portion is added to each time allotment. This allows a user to buffer his periods of heavy demand for rapid service by making up for them with lower category runs before or after the heavy demand times. However, over periods of weeks, his use of all categories will conform to the mix prescribed by his project budgets.

This scheduling process is initiated under one of two conditions. If disk space is freed by a completed request, or if a new job enters a category with a positive time slice and is the only job in that category and all higher categories have empty queues or non-positive time slices, then the scheduling algorithm is executed.

Once a job is picked for scheduling, the disk file allocator attempts to allocate the disk space needed for the selected job. If the file contiguity, size, and phasing restrictions cannot be met, no space is allocated and no attempt is made to schedule another job. If the files can be allocated, then the allocation procedure maps them onto the disk so that checkerboarding of the disk will be minimized with respect to the probable requirements of future requests. At the same time a file map table is constructed and placed in the job dossier. The file map table is used by the hardware supervisor to build I/O descriptors that point to physical storage unit, track, and segment addresses corresponding to the logical file and record numbers passed to it by the job partner. Although the table is originally generated as a singly dimensioned array by the disk file allocator, it must eventually be changed into the special doubly dimensioned array described below. Due to the varying array row lengths, ESPOL code in the hardware supervisor is required to build the final two-dimensional array.

The file map table consists of n rows, usually of unequal size, where n is the number of files. The first word in each row contains three 16-bit fields. The first field specifies the size of a logical record in terms of 256-word segments (a record consists of an integral number of physical segments). The second field contains the total number of segments

in the file. The third field specifies the number of supplementary mapping entries in the row.

After the first word, the rest of the row contains mapping entries in ascending order of virtual segment numbers. Each mapping entry consists of four fields. The first is a 20-bit virtual segment number. If m is a record number in file x, and file x has n segments per record, then the virtual segment corresponding to m is virtual segment m × n. That is, a virtual segment specifies the number of 256-word segments from the beginning of the file but is not a physical segment address. The second, third, and fourth entries are sixteen, two, and eleven bit fields specifying on which storage unit, track, and segment this virtual segment resides. All virtual segments from the one specified in this mapping entry up to but not including the virtual segment specified in the next mapping entry follow contiguously on the same storage unit and track.

Example:

An ILLIAC IV program has three files.

Array Row 0

3	102	3	
0	12	1	0
47	12	0	1153
48	12	2	461

Array Row 1

4	12	7	
0	1	0	341
1	1	2	341
3	0	2	117
8	3	2	236
9	14	1	0
10	0	1	11
11	1	2	1021

Array Row 2

1	256	1	
0	3	1	731

File zero has 34 logical records each 768 words (three segments) long. One hundred two segments are required for the file and they are broken into three areas on storage unit 12. Records 0 through 14 appear

contiguously starting at segment zero on track one. The first 512 words of logical record 15 follow immediately after record 14, but the last 256 words of the record are on track zero at segment 1153. The last 19 logical records are on track 2 in segments 461 through 513.

File one is badly fragmented. Each record occupies four segments. There are three records in file one. Record one has its first virtual segment on storage unit one, track zero at segment 341; the next two virtual segments are on track two at segment 341 and 342; the last 256 words are on storage unit zero, track two in segment 117. The second record (virtual segments 4, 5, 6, and 7) is on storage unit zero, track two in segments 118, 119, 120, and 121. The last record is spread over storage units three, fourteen, zero and one.

File two is on storage unit three, track one and consists of 256 segments--each containing one logical record in segments 731 through 986.

The above example illustrates the flexibility the file map table provides to the disk allocator. If there is any free space on the disk, no matter how badly fragmented the disk is, the allocator can use it if necessary. Furthermore, the table is quick and easy to scan and does not waste B6500 core. The amount of saved core needed is small. If four jobs were running on ILLIAC IV each with ten files in four fragments, the total amount of core use would only be 200 words. Hopefully, most jobs will not have ten files and will not be badly fragmented. If the typical job has about four files with each file in about two fragments, less than 50 words of core would be required.

Once a job has been allocated disk space and the appropriate tables are in the dossier, it is removed from the category queue and a message is sent to the data processor. This message identifies the disk allocator as the sender, specifies that data preprocessing is to be done, and points to the job dossier which contains the preprocessing requests for this job.

When the request has been completely processed--that is, ILLIAC IV execution is complete and all necessary files have been moved to the B6500, the disk allocator will receive a completion message from the data processor. The disk allocator will then free the disk space used by the job by updating its disk space tables and a job complete message will be sent to the job parser. If a queue exists, the disk file allocator will attempt to schedule another job.

V. THE DATA PROCESSOR

The data processor is responsible for moving files onto the ILLIAC IV disk before execution and for removing these files after execution.

Preprocessing requests are received from the disk file allocator. At that time, the disk space necessary for the job has been allocated and the specification of which B6500 files are to be moved to which ILLIAC IV disk files are in the job dossier. In addition, the job dossier indicates the standard reformatting procedures to be performed on the data while it is being transferred. Once the data is moved, the data processor sends a message to the execution monitor that points to the job dossier and tells the execution monitor that it is ready for ILLIAC IV processing.

At the completion of ILLIAC IV execution, the execution monitor sends a similar post-processing message to the data processor. The files to be moved from the ILLIAC IV disk to the B6500 as well as the standard reformatting algorithms to use for post-processing are indicated in the job dossier.

Once the data is moved back to the B6500, the data processor sends a message to the job parser notifying it of the job's completion. The message also points to the job dossier which may be needed by the job parser for the calculation of condition codes and other housekeeping tasks.

The data processor splits each pre- or post-processing request into short canonical units and places these units onto the appropriate pre- or post-process queue. Since the amount of data to be moved between B6500 files and ILLIAC IV files is known from the B6500 or ILLIAC IV file tables or an explicit ICL statement, and since all procedures used to reformat that data are well-timed system routines, it is possible to approximate the execution time of each request. The data processor uses this approximation to split a total request into request units that will tie up a block of BIOM space for only a short, known period of time.

BIOM space is allocated in dynamic and static blocks. Static blocks are assigned to running ILLIAC IV programs for the duration of the run. Dynamic blocks are assigned only when requested by an ILLIAC IV job partner or the data processor. Blocks of BIOM space are made available to

the data processor only when all requests from running ILLIAC IV programs have been honored.

When BIOM space is made available, the data processor first tries to schedule as many post-process units as possible. If any BIOM space is left, then preprocess units are scheduled. Therefore, the order of access to BIOM space is ILLIAC IV program, then post-process, and finally preprocess requests.

Within a preprocess or post-process queue, units are ordered first by job so that the oldest jobs are scanned first; then units are ordered within a job so that the units with the greatest BIOM space requirements are scanned first; and finally, within unit requests which require the same amount of space, the requests requiring the greatest time are placed first.

The scheduling of requests on one of the process queues is performed as follows. The process queue is scanned for the first request in the next job that can use the space. The space is then allocated to that request and the data processing is initiated. The queue is then scanned again until all BIOM space is allocated from the oldest job on the processing queue. If all the BIOM cannot be exhausted from the next job, the following action is taken. If all preprocess requests are initiated for the oldest job, the allocation algorithm is performed on the next oldest job. If all preprocess requests are not initiated for the oldest job, then the earliest possible time for processing the next request on the queue is calculated from the estimated finish times of post-process and preprocess requests, and the maximum times of dynamic and static ILLIAC IV program requests. After the earliest time is calculated, the rest of the process queue is scanned without regard to job boundaries. Those requests that can be completed before the calculated time are initiated. This scanning process continues until BIOM is full or the process queue is exhausted--in which case no further BIOM allocation is attempted from that queue.

The execution of these requests is performed by a slave program belonging to the data processor. This program is similar to the ILLIAC IV job partners in the sense that it has access to the hardware supervisor.

However, its function is only that of an I/O monitor. That is, it initiates as many I/O requests as it can at one time and then sleeps until they are completed. When awakened, it either performs a transformation on some data it has just read, writes it on the B6500 or ILLIAC IV disk and goes to sleep, or it may have just completed a write operation and must now initiate more reads to get the next data for reformatting.

VI. THE EXECUTION MONITOR

The execution monitor schedules the use of the BIOM and the ILLIAC IV quadrants. It also assigns job partners to the various quadrants.

BIOM space is allocated on a static (i.e., for the duration of a run) or dynamic (i.e., short term) basis. Static BIOM requests are contained in the job dossiers, are assigned at the time a program is initiated, and are released at program termination. Dynamic BIOM space is allocated only when requested by a job partner or the data processor. It is released upon job partner or data processor notification or at job termination.

The execution monitor maintains a BIOM map which identifies static, dynamic, and unassigned BIOM space as well as the users of assigned portions, and the time at which they will be relinquished. The relinquish time is the approximate time specified by the data processor or the maximum time specified by a job partner or a job dossier. This map is used by the data processor to schedule pre- and post-processing and by the execution monitor to schedule the use of the ILLIAC IV quadrants.

BIOM requests from job partners are honored on a "first come-first served" basis. BIOM requests from the data processor are not honored unless there are no BIOM requests pending from the job partners. All changes in the status of the BIOM map are also communicated to the data processor so that it may do appropriate pre- and post-processing scheduling.

When the execution monitor receives a request for ILLIAC IV execution from the data processor, it is placed in the ready queue. All jobs in the ready queue have completed their data preprocessing phase and are now ready to use one or more ILLIAC IV quadrants. Some of these requests are multi-element steps that were originally built by the job parser. The execution monitor is responsible for executing such requests in the proper order indicated in the job dossier.

When quadrant space becomes available, the next job on the ready queue is initiated if: a) it fits into the number of quadrants available and, b) its static BIOM requirements can be met without reducing the current amount of dynamic BIOM space available to less than is required by the programs currently executing or the program that is being scheduled. If the proposed job does not fit the above requirements, then a time is calculated at which the job can be run. This time is the latest guaranteed time of execution as calculated from the maximum time specifications of jobs currently executing on ILLIAC IV and the time specifications in the BIOM map.

The ready queue is then scanned for a job that fits into the available quadrant and BIOM space and also has a run time that will allow completion before the latest guaranteed time just calculated. If such a job is not formed, then no program is initiated. If a program is formed that meets the above requirements, then it is immediately initiated before any older jobs in the ready queue. If quadrant space is still available after initiating the job, new BIOM and quadrant requirements are calculated, and the scan of the ready queue is continued for another program that meets the new BIOM and quadrant requirements and the latest time guarantee.

Both the execution monitor and the job partner are involved in ILLIAC IV program initiation. First the execution monitor halts the allocated control units and sets the configuration control register. Then file map tables for the job are sent to the hardware supervisor along with BIOM allocation tables. These tables will be used by the hardware supervisor to check all disk and BIOM I/O requests for validity before executing them. Once the control units are set up and the hardware supervisor notified of the new program, the execution monitor initiates the job partner and passes file information to it. The job partner is then responsible for further initialization of the quadrants. This usually involves the loading of OS4, the loader, and the main program from the disk. Then the job partner will start the control unit and continue processing only when an I/O request is received from the ILLIAC IV program. When the job partner goes to end of job, the B6500 operating system notifies the execution monitor.

The user is charged for ILLIAC IV time from the time his job partner starts to the time it reached the end of the job. If he wishes to return a condition code, then it must be passed to the execution monitor by the job partner before the job partner terminates.

When all the programs for a given ILLIAC IV request have reached the end of the job, the execution monitor sends a message to the data processor. This message points to the job dossier and tells the data processor to begin post processing of the user's disk files.

VII. THE JOB PARTNERS

Corresponding to each job to be run on ILLIAC IV is an ALGOL-written job partner residing in the B6500. This is a standard (although user-modifiable) program which is responsible for seeing that the ILLIAC IV job gets loaded, initialized, initiated and monitored. Control is passed to the job partner by the execution monitor after the module clears the ILLIAC IV job for execution.

Each job partner communicates with its active ILLIAC IV job through OS4, the ILLIAC IV resident operating system. Necessary communications are established via the TRO and TRI registers in the CU. OS4 loads the TRO when it wishes to send information to its associated job partner, thereby causing a B6500 interrupt which is given to the job partner. The job partner then reads the TRO register, interprets the contents, and performs the indicated action (I/O, end of job, release a file, etc.). Likewise, the job partner communicates with OS4 by loading the TRI, which causes an ILLIAC IV CU interrupt and an entry into OS4. OS4 reads the TRI and takes the appropriate action (mark an I/O complete, halt because time has expired, etc.).

The job partners are written using a modified Burroughs ALGOL compiler which allows access to programs controlling the ILLIAC IV input/output. These programs, collectively called the Hardware Supervisor, are embedded in the B6500 MCP (Master Control Program). Since each job partner is an independent B6500 program and is associated with only one active ILLIAC IV job, some means of supervision must be provided. This supervision is the task of the Execution Monitor module. The execution monitor schedules jobs as they are released by the data processor and creates tables for the job partner. These tables define the areas of disk, the BIOM space, and the ILLIAC IV quadrants available to the active job. The job partner cannot change these tables and the MCP routines will not issue I/O commands which violate the data in the tables. Thus the system is protected from an errant job partner.

The job partners are coded to perform the following six functions:

1. to initiate the ILLIAC IV job;
2. to process I/O interrupts;

3. to respond to communication requests from the job in execution;

4. to issue commands to OS4, the ILLIAC IV resident operating system;

5. to recover from ILLIAC IV errors;

6. to terminate the ILLIAC IV job when necessary.

These six functions are described in more detail below.

1. ILLIAC IV job initiation is performed by the job partner upon clearance from the execution monitor. The job partner is told the quadrant configuration for the configuration control registers--MC0, MC1, and MC2--as well as the location on the ILLIAC IV disk of the first segment of the user program that must be loaded into memory. The job partner then performs three tasks. First, it issues commands to stop the ILLIAC IV quadrant(s) and to initialize the control unit registers to their starting values. Second, it loads OS4 from the disk into ILLIAC IV memory. Third, it commands OS4 to load the user program from the disk to the ILLIAC IV quadrant(s) and start execution.

2. I/O requests from ILLIAC IV deal only with transfer of information from the ILLIAC IV disk to the ILLIAC IV memory (READ) and from the memory to the ILLIAC IV disk (WRITE). Upon receiving an interrupt signal from ILLIAC IV, the execution monitor uses hardware supervisor programs to "scan in" a descriptor. This descriptor, which is generated by the ILLIAC IV I/O controller, indicates which control unit requires attention and thus determines which job partner is to take action. The appropriate job partner is alerted and in turn requests a hardware supervisor routine to read the control unit's TRO register. From the TRO bit configuration, the job partner determines the type of request submitted by the ILLIAC IV job.

The above describes a standard I/O request. However, in order to complement the standard I/O handling, resident within each quadrant's job partner is a special I/O monitor designed to service special input/output needs. This monitor is invoked by an interrupt formatted as a

communication request, but the content of the request and the actions resulting from it give the connotation of an I/O request.

The monitor's primary function is to perform sophisticated data retrieval and transfers upon request from the ILLIAC IV job in execution. It is evident that very large data files on the ILLIAC IV Disk will be broken down into smaller logical "blocks" for transfer to and from PE memory. In addition, these blocks may be skewed across the disk segments from one track to another in a "checkerboard" fashion in order to minimize disk latency. It is the responsibility of the I/O monitor to locate particular blocks and initiate I/O. The I/O monitor's function is thus that of a liason between the system and the user who wants to do something "fancy" with the disk.

The information passed from the ILLIAC IV to the B6500 is the same as a standard I/O request except that the logical record number and number of words to transmit are not involved. Instead, the block coordinates are given. The I/O monitor will "map" the block coordinates into a logical record number and obtain the number of words to transmit from a File Characteristics Table containing:

a. format specifications for mapping the file on disk;

b. block sizes (in terms of segments);

c. block skewing displacements;

d. block counts and bounds.

Each file involving special mapping has such a table associated with it.

In addition to this scheme involving the block coordinates, it is also possible for the user to specify his own run-time block location algorithm in the job partner. This is a procedure available to the I/O monitor during run-time. Its operation is as follows:

a. The user issues a Communications Request as before, omitting the block-coordinate parameters.

b. The I/O monitor passes control to the user-written block-computation procedure.

c. The block-computation procedure has already computed the block coordinates for the present I/O transaction from the previous call and immediately provides the proper logical record number and word count for I/O initiation. The user's procedure then computes the next block that will be called for, computes the logical record number and word counts, and places them in an I/O queue within the I/O monitor to await the next signal from ILLIAC IV.

The user's algorithm thus determines what block will be needed next and therefore saves the block-coordinate mapping time at each interrupt. While the ILLIAC IV is in execution, the procedure in the I/O monitor is calculating the next move.

File blocking appears to be the general solution to non-core-contained data structures. Tranquil, of course, is compiling in terms of 256 × 256 word blocks. A single array of dimension greater than this canonical size is blocked 256 × 256 and coordinates or indices assigned to the blocks. The disk mapping under these conditions is straightforward, and I/O requests for certain blocks fit readily into the I/O monitor's functions.

Although large mesh structures (e.g., those involved in PDE problems) may call for rather complicated mapping onto the disk files, it is envisioned that a large class of problems to be run on ILLIAC IV will not require very sophisticated mapping schemes and thus can be accommodated on ILLIAC IV disk easily using the standard job partner operations.

3. The following Communication Requests are generated by OS4 upon command of the ILLIAC IV program, and are described more completely in Section VIII:

MARKEOF — Mark the end of a file on the ILLIAC IV Disk. Close the file and release it to the disk file maintenance portion of the operating system.

STATUS — Determine the status of a file on the ILLIAC IV disk.

LIMIT — Set one of four interval timers. The request includes an interrupt address. When the time limit is reached, control is transferred to the interrupt address.

FREE — Cancel a previous LIMIT request for a specified timer.

TIME — Obtain and communicate to the user the current time of day and the time remaining until the end-of job. It is assumed that the end-of-job time has been specified in the job control language. The TIME request permits the user to anticipate his scheduled end-of-job and to provide for necessary outputs in the event he might otherwise over-run his scheduled time.

DATE — Obtain and communicate to the user the current date. The days Sunday through Saturday are represented as integers one through seven. The date is represented as mm/dd/yy.

OPEN — Open a file. This request must precede the first attempt to read or write a file on the ILLIAC IV disk if the logical file number is not known.

CLOSE — Close a file. This request frees the file to the system and specifies the action to be taken with respect to the file, e.g., purging, transmitting to back-up memory, printing, punching, displaying, etc.

ERROR — Faulty data in the ILLIAC IV job. The job partner has access to a table of error messages and actions to be taken. This table is indexed on the error codes transmitted through the request. Some codes cause immediate program termination, others permit continuation after recording the message, and others cause termination after the particular code has been received a suitable number of times.

4. Commands to OS4 are issued by the job partner to alter normal ILLIAC IV execution via the BREAK, RESTART, TRAP, and LOAD commands. The BREAK request saves and the RESTART request restores the entire program status for the quadrants involved. All CU and PE register contents, as well as PE memory, are transferred to or from the ILLIAC IV Disk when those commands are

are issued. The TRAP command causes temporary suspension of ILLIAC IV job execution with a transfer to an interrupt address given in the command parameters. The LOAD command causes OS4 to load program segments from the ILLIAC IV Disk into PE memory. The specific TRI register configurations for these commands are given in the discussion of OS4.

5. Recovery from ILLIAC IV errors. The recovery procedures for I/O error interrupts include the option of specifying an interrupt address to which control is transferred when the procedures are unsuccessful. Such recovery procedures involve one or more attempts to read or write with subsequent job termination if the operation fails and if no interrupt address is given. Other interrupts may be non-recoverable, thus giving the job partner the alternative of terminating the job or branching to the interrupt address given. In the instance of a power failure interrupt, a BREAK command is executed for OS4 to save the status of the job partner's quadrants.

6. ILLIAC IV job termination. In the instance of fatal ILLIAC IV hardware and software errors requiring job termination, the job partner notifies the execution monitor to initiate a "halt CU".

It should be noted that job overtime termination is not effected by the job partner, but rather by the MCP. ILLIAC IV job overtime is noted by the MCP which then sends the job partner to end-of-job, in turn causing the ILLIAC IV quadrant(s) to halt.

VIII. OS4--THE ILLIAC IV RESIDENT SUBSYSTEM

OS4 performs three functions. It monitors ADVAST interrupts; it processes ILLIAC IV program requests; and it responds to communications from the B6500. Each of these three functions are described in detail below.

Interrupts in ILLIAC IV are caused by the setting of bits in the interrupt register (AIN), and the setting of corresponding bits in the interrupt mask register (AMR). The interrupts are of four types. Bits 0-3 control hardware failure interrupts; bits 4-8 and 13 control software failure interrupts; bits 9-12 and 14 control program dependent interrupts; and bit 15 controls the B6500 to ILLIAC IV communication interrupt:

0 Power failure
1 Instruction parity error
2 Undefined instruction
3 CU stalled

4 Improper MC1 or MC2
5 Improper local address
6 ADB wrap-around
7 Execute loop
8 Improper skip distance
13 Protected write

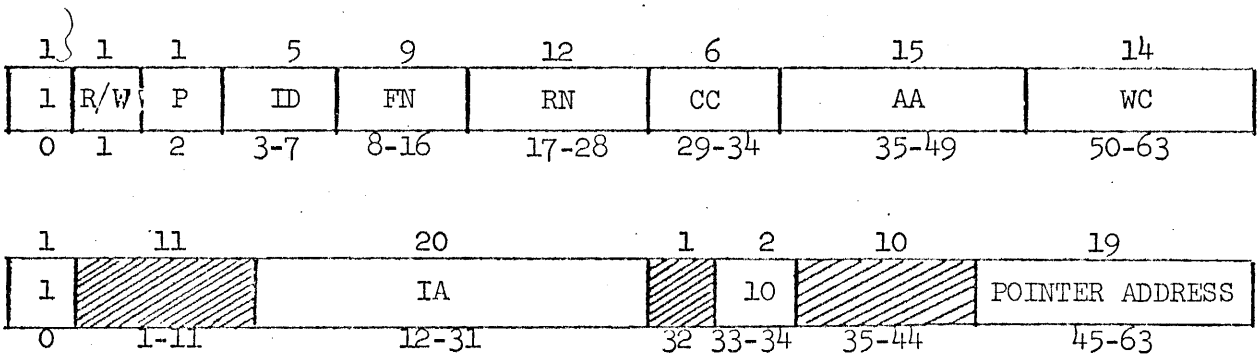
9 Program request
10 PE mode
11 PE overflow
12 Real-time input
14 Branch trace

15 TRI loaded

Unless modified by a program request, the hardware failure and software failure interrupt bits are always set in the interrupt mask register. An interrupt of these types causes OS4 to halt the program and interrupt the B6500. The program dependent interrupts are treated individually as follows: Bit 9 is always set; this bit controls all requests from user programs running on ILLIAC IV. Specific responses are dictated by the type of request. Bits 10, 12, and 14 are reset. Unless these bits are set in the interrupt mask register by suitable program requests, the corresponding interrupts do not occur. Bit 11 is always set. Unless modified by a program request, a PE overflow interrupt causes OS4 to halt the program and interrupt the B6500. The B6500 to ILLIAC IV communication interrupt bit (TRI loaded) is always set. The actions taken by OS4 are determined by the contents of the TMU input register (TRI).

Program requests are macro statements in a user program running on ILLIAC IV. They may be written in the ILLIAC IV assembler language or in any fashion which generates a correct calling sequence to OS4 as described below. They each include the ADVAST instruction INR, the execution of which causes a program request interrupt (AIN bit 9). There are three types of program requests: requests for input or output, requests for communication with the B6500 job partner, and requests for use of ILLIAC IV resident utility routines. All three types of requests employ the use of the INR instruction to cause the interrupt, and employ the use of the ADVAST accumulator register no. three (AC3) to contain the information necessary to identify the request type and action to be taken. While in the interrupt mode, several ADVAST registers are used, but all except AC3 are returned to their original state upon leaving the interrupt mode and returning to the user program. AC3 is returned with information relative to the response for the request.

Input/output requests deal only with transfer of information from the ILLIAC IV disk to the PE array memory (READ), and from the PE array memory to the ILLIAC IV disk (WRITE). I/O requests are distinguished from communicate and utility requests by means of the most significant (left-most) bit in AC3. For I/O requests, AC3 bit 0 is set (contains a one). The remainder of AC3 contains either all the parameters necessary for the request or contains an address which points to a table of the parameters in PE memory. If bits 33-34 contain 10, bits 45-63 contain a pointer address. If not, bits 1-63 contain the parameters. The AC3 configurations and parameters used are as follows:



R/W Read/write indicator, one bit. If bit 1 is a zero, a read is requested. If bit 1 is a one, a write is requested.

P — Priority indicator, one bit. If bit 2 is a one, the variant field of the I/O descriptor fetched by the input-output control will be set to indicate priority. The descriptor will then override all others in the queue for disk access. Since the disk queuer can admit only one priority descriptor, the B6500 maintains its own queue of priority descriptors. Hence, care should be used in the setting of this bit by the user program on ILLIAC IV. Too many unserviced priority requests can provide slower service than a balanced mix between priority and non-priority requests.

ID — Identification number of the I/O request, five bits. OS4 maintains a table for 32 entries of pending I/O transactions. The identification number provides the index to this table and identifies the request to the B6500. Subsequent program requests can be used to advise the ILLIAC IV user program of the status of the particular transaction, given the identification number.

FN — File number, nine bits. Up to 512 logical file numbers may be used to reference files on the disk. A correspondence is maintained by the B6500 between the logical file number (FN), its physical location on the disk and its file name.

RN — Record number, twelve bits. Up to 4096 logical records may be contained in each disk file. The length of a logical record is some multiple of 256 words (a disk segment), and is chosen by the user before execution of the program. Lengths of logical records may vary from file to file, but must remain constant within a file.

CC — Configuration control, six bits. The left-most four bits are treated in a manner similar to the hardware's use of the configuration control registers MC1 and MC2. That is, specific quadrants of a united configuration are to transmit or receive the data. The right-most two bits denote the arrangement of the data within the quadrant(s): 00 indicates that the data is to be read into or written from full rows across the quadrant(s); 01 specifies half-rows; and 11 specifies only a quarter of the quadrant rows to be involved. The data address specifies which half or quarter, and of course in the half-row mode, the address must correspond with the first or third quarter. 10 is undefined.

AA Array address, fifteen bits. The data is to be read from or written into PE memory, starting at the location specified by 16 times the AA. Since the hardware limits all transmissions to blocks of "long-words" which are 1024 bits in length (16 words), these transmissions must start and stop at long-word boundaries. A four-quadrant ILLIAC IV with 2048 words per PE memory contains 32,768 long-words.

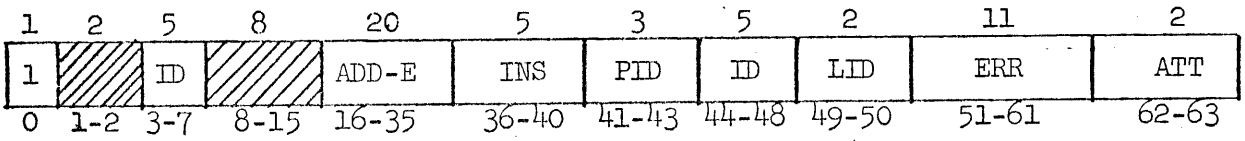
WC Word count, fourteen bits. Up to 16,384 long-words may be transmitted. This is equivalent to one-half of a four-quadrant array memory or approximately 28 percent of one disk storage unit.

If bits 33-44 of AC3 contain 10, the pointer address is a nineteen-bit location of the first 64-bit word of a table in PE memory. The table must contain four words (eight syllables), with the parameters located in the table in the same order as described above. Each parameter is right-adjusted in its 32-bit syllable. No provision is made within OS4 to distinguish between parameter values that are to be provided by the ILLIAC IV user program and those that are to be provided by the B6500 job partner. All values found in the table are sent to the B6500 by OS4. The job partner may override certain values at its option.

There are two methods of operational control exercised by OS4 over I/O requests. In the first method, the user program wishes to request I/O and not be interrupted when the transmission is complete. It signifies this by means of the parameter-packed form of register AC3, or if the pointer address form is used, bits 12-31 of AC3 must contain zeroes. In the second method, the pointer-address form must be used, and bits 12-31 of AC3 contain an interrupt address. In either method, OS4 returns control to the user program as soon as it has processed the request and sent the parameters to the B6500. In the first method, when the transaction is complete and the B6500 interrupts the user program (TRI loaded interrupt), OS4 records the status of the transaction into its 32-entry table and returns normal control to the user, leaving the ADVAST registers in the user's state. In the second method, OS4 saves the user's contents of register AC3, records the status of the completed transaction into AC3, leaves the interrupt mode, and transfers control to the

user at the location specified by the interrupt address. The interrupt address, having been provided by the user as one of the I/O parameters, was saved in the 32-entry table indexed according to the identification number of the transaction. When the user is finished in his interrupt routine, he performs a program request to return to OS4. At that point, the original contents of AC3 are restored and control is returned to the user at the point where the TRI loaded interrupt occurred.

All entries to user interrupt routines, and exits from them, are controlled by the five-bit transaction identification numbers. The 32-entry table is used to save interrupt addresses, the user's contents of AC3, and the return-from-interrupt locations. Bits 3-7 of AC3 contain the ID number when entering an interrupt routine and must also contain it upon exit. Bits 3-7 of the TMU registers TRO and TRI contain it for communications with the B6500. The format of the information sent to the B6500 in register TRO is identical to the format of the parameter-packed AC3 described above. The format of the reply information from the B6500 in register TRI is:



ID — Transaction identification number, five bits.

ADD-E — Last address used in PE memory.

INS — Instruction code, five bits. Read = 20_8, write = 21_8.

PID — Partner identification number, three bits. Used by the B6500 job partner to differentiate between quadrant transactions.

ID — Transaction identification number, five bits. Repetition of ID in bits 3-7.

LID — List ID, two bits.

ERR — Error field, eleven bits:

51 Disk time out
52 Disk not ready
53 (not used)
56 Memory access error
57 B6500 memory parity error
58 Memory address error

54	Disk parity or lock-out	59	Descriptor error
55	No memory module	60	Unit not ready
		61	Unit busy

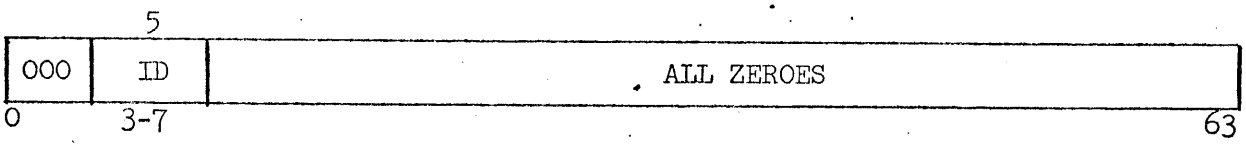
ATT Attention field, two bits:

00--No errors

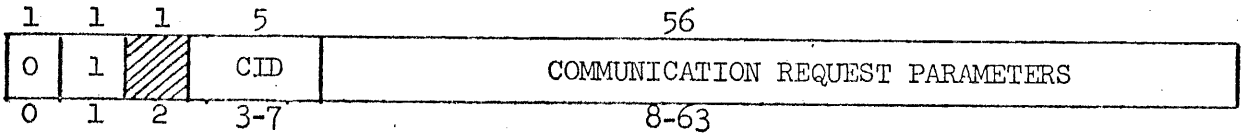
01--Hardware exception

11--Software attention

In the first method of operational control, where an interrupt address is not specified, the contents of TRI are placed in the 32-entry status table. In the second method, the contents of TRI are placed in register AC3 for the user's interrupt routine. The interrupt routine need only scan bits 62 and 63 for zeroes to ascertain that the transaction was completed with no errors. Additional I/O requests may be made from within the interrupt routine. In fact, portions of the program containing I/O requests and interrupt routines may be nested to an arbitrary level, provided that the thread of return control is never broken by failing to include the proper transaction identification numbers in bits 3-7 of AC3 each time a return is made from an interrupt routine to OS4. Returns from interrupt routines must always be done by executing an INR instruction after placing register AC3 in the following state:



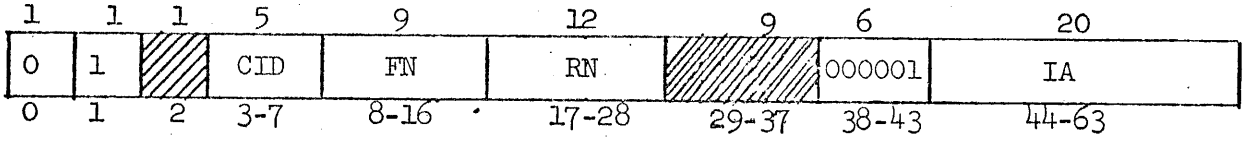
Communication requests are program requests which are used for requesting certain actions to be performed by the job partner on the B6500. Communication requests are distinguished from I/O requests by means of the most significant (left-most) bit in AC3. For communication requests, AC3 bit 0 is reset (contains a zero). Communication requests are further distinguished from utility requests by means of bit 1 in AC3. This bit, for communication requests, is set (contains a one). The remainder of AC3 contains all the parameters necessary for the request:



The communication identification number (CID) contained in bits 3-7 is treated in a manner similar to the ID of the I/O request. A 32-entry table is maintained by OS4 for communication requests. This table is separate from, but performs a similar function to, the 32-entry table for I/O requests.

Included in the first implementation of the operating system are nine communication requests: MARKEOF, STATUS, LIMIT, FREE, TIME, DATE, OPEN, CLOSE, and ERROR.

MARKEOF is a communication request used to mark the end of a file on the ILLIAC IV disk. It causes OS4 to generate a signal to the B6500 to close the file and release it to the disk file maintenance portion of the operating system. In effect, the length of the file becomes specified, and the file will not be accessed by the current job on ILLIAC IV. The disk file maintenance program is permitted to remove the file from the ILLIAC IV disk and transmit it to backup storage on the B6500. The format of AC3 for this request is:



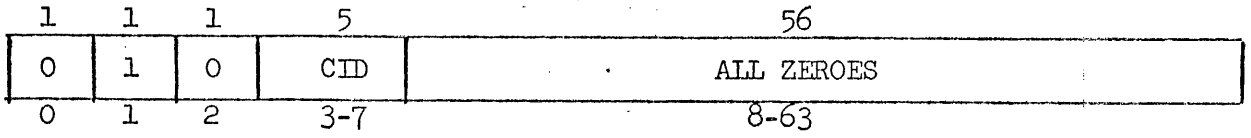
CID — Communication identification number, five bits.

FN — Logical file number, nine bits.

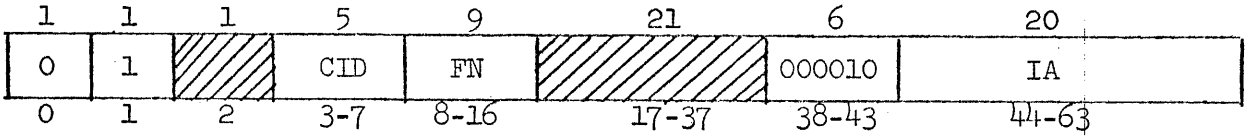
RN — Logical record number, twelve bits.

IA — Interrupt address, 20 bits. If the user program wishes the B6500 to interrupt it upon completing its action, and OS4 is to transfer control to a user interrupt routine, the IA should contain the location of the first code syllable in the routine. If the B6500 is to perform its actions and not interrupt the program, the IA should contain all zeroes.

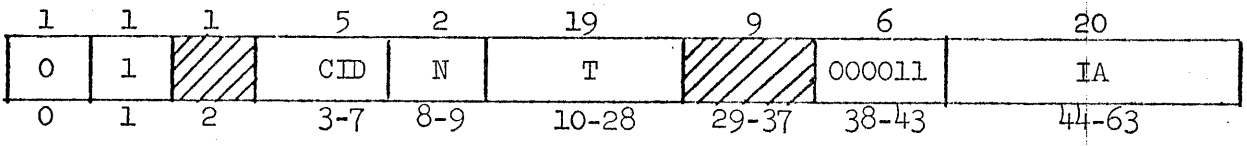
If an interrupt routine is used, the proper exit from it must include the AC3 setting:



STATUS is a communication request for determining the status of a file on the ILLIAC IV disk. The format of AC3 for this request is:



LIMIT is a communication request for obtaining time limit interrupts. It causes the B6500 to set one of four interval timers. Since the ILLIAC IV hardware does not include a real-time clock, the clock on the B6500 multiplexor is used. The request contains four parameters in AC3:



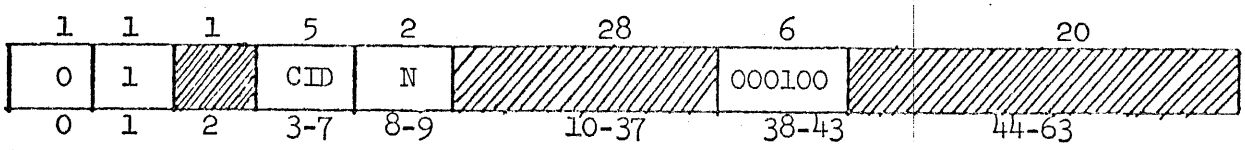
CID — Communication identification number, five bits.

N — Interval timer number, two bits. Up to four LIMIT requests may be in effect at one time. N denotes which of the four interval timers is to be set. If a LIMIT request is made with a previously specified value of N, and that previous limit has not been reached, the new limit replaces the old.

T — Time limit, nineteen bits. The user's program is to be interrupted after T sixtieths of a second have elapsed. The largest possible value for T denotes a maximum time limit of approximately 2-1/2 hours.

IA — Interrupt address, 20 bits. When the time limit is reached, control is to be transferred to the user's program at his interrupt address IA.

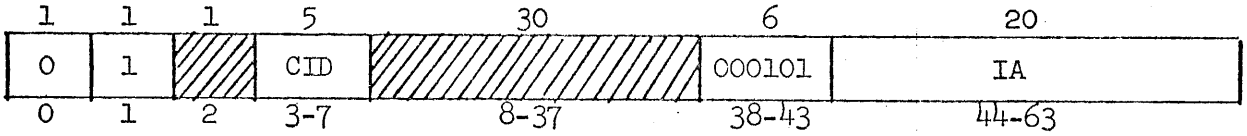
FREE is a communication request for cancelling previous LIMIT requests. The request contains two parameters in AC3:



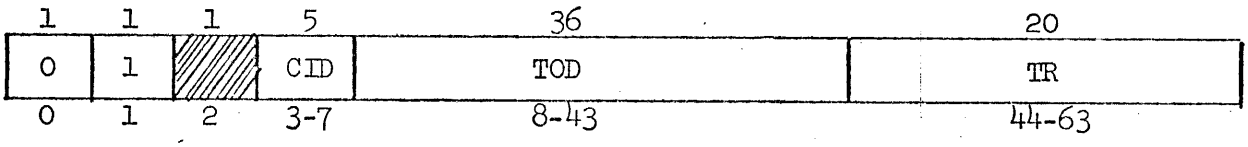
CID Communication identification number, five bits.

N Interval timer number, two bits. The FREE request cancels the effect of a previous LIMIT request on interval timer N. It does not affect settings of the other three timers.

TIME is a communication request for obtaining the current time of day and the time remaining until the end-of-job. It has two parameters in AC3:



If the interrupt address (IA) is equal to zero, execution of the user program is suspended until the B6500 responds with the information. If the IA is non-zero, control is returned to the user after OS4 sends the request to the B6500. When the B6500 responds, the user program is interrupted and control is transferred to the user at his interrupt address. In either event, the information is supplied in register AC3:

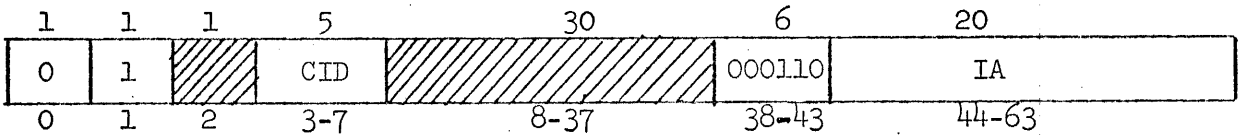


CID Communication identification number, five bits.

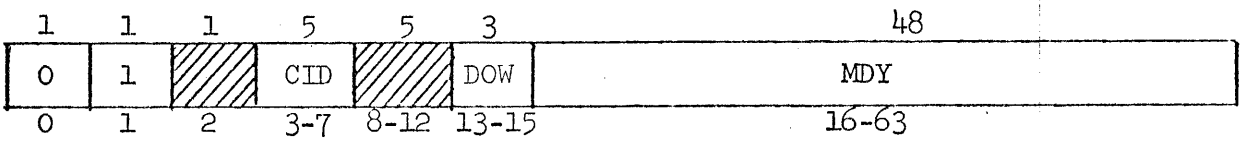
TOD Time of day, thirty-six bits. The time of day is represented as six 6-bit characters. It is based upon a 24-hour clock (one minute before midnight is 235900) and is given in hours (hh), minutes (mm), and seconds (ss).

TR Time remaining, twenty bits. The time remaining is represented as a binary integer in sixtieths of a second. It has a maximum value of approximately five hours. It is assumed that the time at which end-of-job would occur has been specified externally to the execution of the program via control card information. The use of the TIME request permits the user to anticipate his scheduled end-of-job and provide for necessary outputs in the event he might otherwise over-run his scheduled time while in the debugging stage.

DATE is a communication request for obtaining the current data. It has two parameters in AC3:



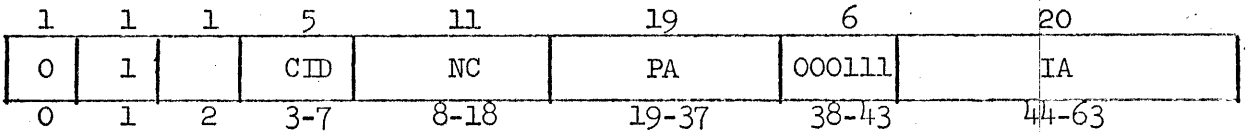
The interpretation of the IA is the same as in the TIME request described above. The information supplied to the user in AC3 has the following format:



DOW Day of the week, three bits. The days Sunday through Saturday are represented as binary integers one through seven.

MDY Month-day-year, forty-eight bits. The date is represented as eight 6-bit characters showing the month, day, and year with intervening slashes in the form mm/dd/yy.

OPEN is a communication request used to open a file. Its use must precede the first attempt to read or write a file on the ILLIAC IV disk if the logical file number is not known. It has four parameters in AC3:



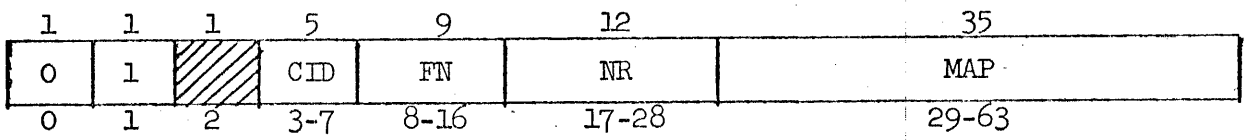
CID Communication identification number, five bits.

NC Number of characters, eleven bits. A binary integer representing the number of characters in the name of the file.

PA Pointer address, nineteen bits. The location of the first 64-bit word in PE memory which contains the name of the file.

IA Interrupt address, twenty bits. Same interpretation as in TIME request above.

The response to this request, contained in AC3, has the format:



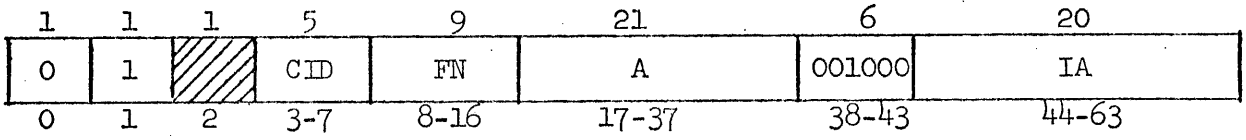
CID Communication identification number, five bits.

FN Logical file number, nine bits. May be used in subsequent I/O requests.

NR Number of records, twelve bits. Up to 4096 logical records may be contained in each disk file.

MAP Thirty-five bits. Contains such information as blocking factors, layout configuration of the file in disk storage unit areas, type of format of the file data, etc. This information is optional, with format of the MAP determined by the user or compiler.

CLOSE is a communication request used to close a file on the ILLIAC IV disk. It frees the file to the system. An input file may be purged. An output file may be transmitted from the ILLIAC IV disk to the B6500. The request contains four parameters in AC3:



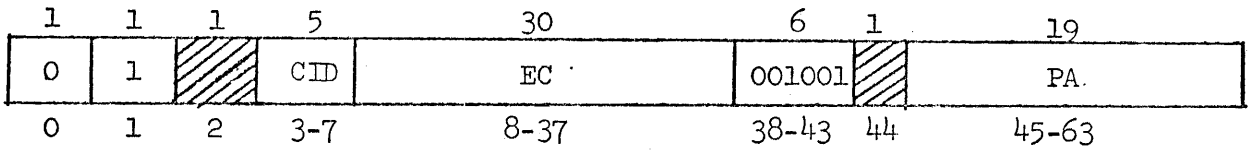
CID Communication identification number, five bits.

FN Logical file number, nine bits

A Action, twenty-one bits. Specific action to be taken by the operating system on the B6500, such as purging, transmitting to back-up memory, printing, punching, displaying, etc, is determined by the user or compiler.

IA Interrupt address, twenty bits.

ERROR is a communication request normally used by mathematical subroutines to indicate the occurance of faulty data; for example, a square root routine entered with negative arguments in one or more processing elements. The request can be used by any user program to enter information into the user's error file or cause program termination. The ERROR request has three parameters in AC3:

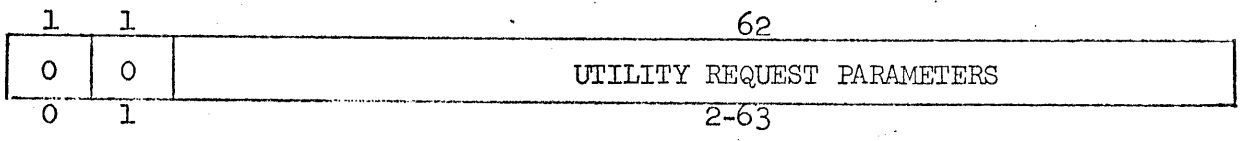


CID Communication identification number, five bits.

EC Error code, thirty bits. The job partner has access to a table of error messages and actions to be taken indexed on the error codes. Some codes cause immediate program termination, others permit continuation after recording the message, and others cause termination after a suitable number of times the particular code is received.

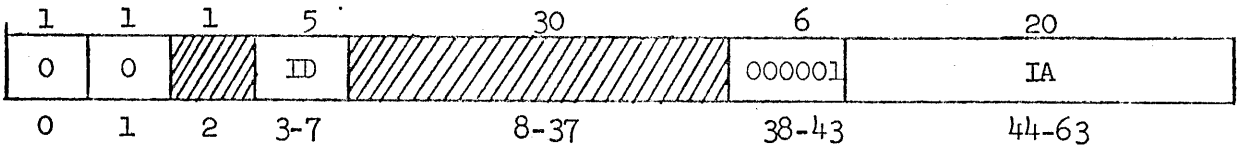
PA Pointer address, nineteen bits. Location of a table in PE memory which contains the error information to be recorded. Normally, the first word in the table contains the table length if it is not included in the error code.

Utility requests are program requests which are used for requesting certain actions to be performed by the ILLIAC IV resident operating system (OS4) and its allied utility routines. Utility requests, like communication requests, are distinguished from I/O requests by means of the most significant (left-most) bit in AC3. For utility requests (and communication requests), AC3 bit 0 is reset (contains a zero). Utility requests are further distinguished from communication requests by means of bit 1 in AC3. This bit, for utility requests, is reset (contains a zero). The remainder of AC3 contains all the parameters necessary for the request:



Utility requests are different from communication and I/O requests in that they are not queued and interrupt addresses are not used. Once a utility request is made to OS4, control is not returned to the user program until all actions necessary to satisfy the request are completed. Included in the first implementation of the operating system are nine utility requests: ADVISE, SELECT, REMOVE, BREAK, RESTART, LOAD, RECONFIGURE, TRACE, and EXIT.

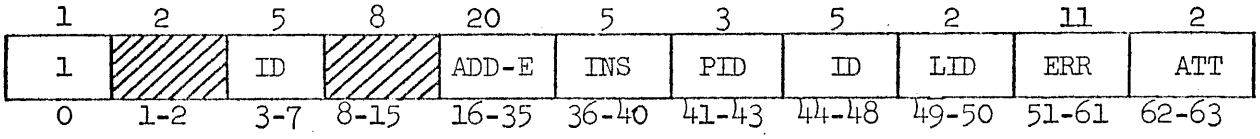
ADVISE is a utility request used for obtaining the status of a pending I/O transaction. It has two parameters in AC3:



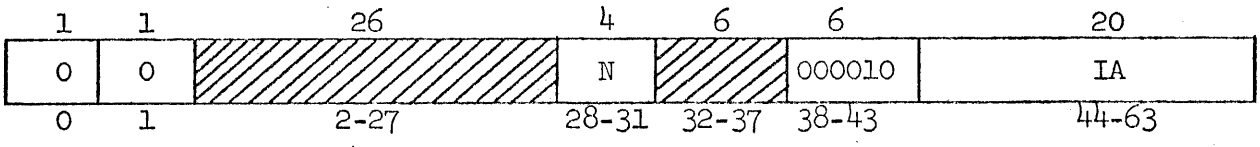
ID Identification number of the associated I/O request, five bits.

IA Interrupt address, twenty bits.

There are three methods of operational control exercised by OS4 over ADVISE requests. In the first method, the user's program is stalled until the I/O transaction is complete, whereupon OS4 returns to the user program with the status information in AC3. This method is used if the I/O request did not contain an interrupt address, and the interrupt address field of the ADVISE request contains all ones. In the second method, the latest status information is immediately returned to the user program, and no interrupt is later made when the transaction is complete. This method is used if the I/O request did not contain an interrupt address, and the interrupt address field of the ADVISE request contains all zeroes. In the third method, the latest status information is immediately returned to the user program, but an interrupt will occur later when the transaction is complete. This method is used if the I/O request did contain an interrupt address, or the ADVISE request does contain a valid interrupt address. If the interrupt address supplied in the ADVISE request disagrees with the interrupt address supplied in the I/O request, the IA supplied with the ADVISE request is used. The format of the status information supplied to the user program in AC3 is identical to that received through TRI, described above under I/O requests:

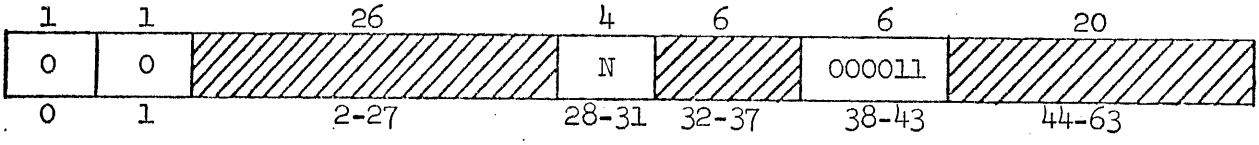


SELECT is a utility request used to modify the handling of interrupts by OS4. The request specifies a particular bit in the ADVAST interrupt mask register (AMR) to be set. When an interrupt occurs, and that corresponding bit in the ADVAST interrupt register (AIN) is set, control is to be transferred to the user's program at a specified interrupt address. The request has two parameters in AC3:



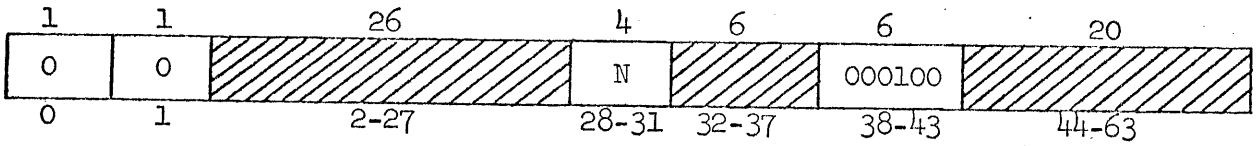
N — Bit number in AMR, four bits. AMR bits 0-8 or 10-15 may be selected. A value of N=9 does not select bit 9 of AMR (user program request) since the SELECT and REMOVE requests are handled via this interrupt. Instead, bit 4 of the ADVAST interrupt control register (ACR) is altered (alternate interrupt base in use). Normally, OS4 operates with ACR bit 4 set. That is, the alternate interrupt base is used, which causes all interrupts to start OS4 control at location 1. OS4 normally maintains a halt instruction at location 0. If a SELECT request is made with N=9, OS4 resets the ACR bit 4, and plants a jump to the specified interrupt address at location 0. This means that all subsequent interrupts bypass OS4 completely and transfer control to the user's own version of an OS4 at the specified interrupt address. It must be noted that, although the REMOVE request, described below, can be used to undo the effect of a SELECT request for bits 0-8 or 10-15, the effect of a SELECT request with N=9 cannot be undone except via special treatment in the user's version of OS4.

IA — Interrupt address, twenty bits. Once a SELECT request is used to modify control under a specific interrupt, the occurrence of that interrupt will cause entry to the user's interrupt routine at this interrupt address. The entry is made out of the interrupt mode. Register AC3 is saved by OS4 when the interrupt occurs, but it is not changed. The contents of the instruction counter are placed in TRO for the interrupt routine. In order to return via OS4 to the user program at the point of interruption, the interrupt routine must perform the INR instruction with register AC3 in the following format:



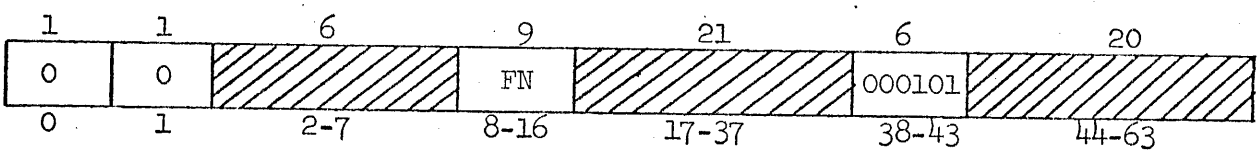
OS4 maintains a table of fifteen entries indexed on N to enable return of control from within nested interrupts. Each entry contains the location at which the interrupt occurred.

REMOVE is a utility request used to restore OS4's control over interrupts. Any previous SELECT request designating bit N of the interrupt and interrupt mask registers is to be ignored. Use of the REMOVE request returns handling of the specific interrupt to the normal action taken by OS4 as described above under the interrupt handling function of OS4. The REMOVE request has one parameter in AC3:



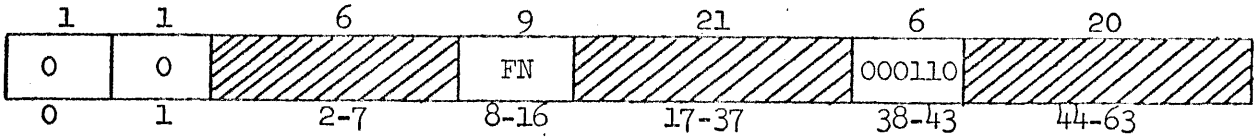
N — Bit number in AMR, four bits. A value of N=9 (user program request) is ignored since the SELECT and REMOVE requests are handled via this interrupt.

BREAK is a utility request for saving the current status of a program for later RESTART. It causes the contents of all CU and PE registers and PE memory in the quadrants united to be saved on the ILLIAC IV disk. The request contains two parameters in AC3:



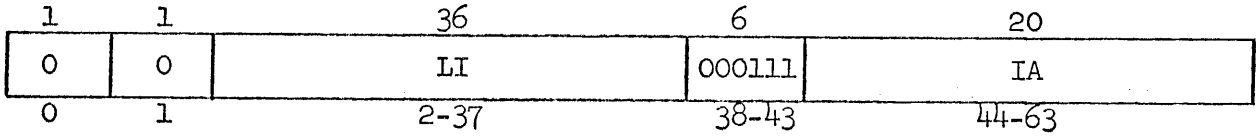
FN — Logical file number, nine bits. The request is postponed and control is not returned to the user program's main code stream until all pending I/O requests are completed.

RESTART is a utility request for restoring the status of a progam to the point at which a previous BREAK request was completed. The request contains the parameter FN which refers to the file in which the previous BREAK request caused the program status to be saved. It is anticipated that the user will be restricted to having no more than two files on the ILLIAC IV disk for BREAK-RESTART requests. The format of AC3 for the RESTART request is:



The BREAK and RESTART requests described above are made from within a running ILLIAC IV program. Other BREAK-RESTART actions are available to be initiated from the job partner on the B6500, from within the executive control section of the operation system, from the operator's console, and from remote user terminals.

LOAD is a utility request used for loading program segments into PE memory. It is normally used by the operating system, but is available to the user program. The request contains two parameters in AC3:



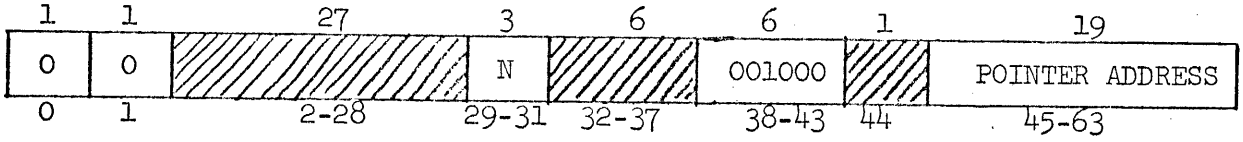
LI — Loader information, thirty-six bits. The format and meaning of these bits are dictated by the design of the high-level language compiler used, and restrictions placed on the overlay process by the loader.

IA — Interrupt address, twenty bits.

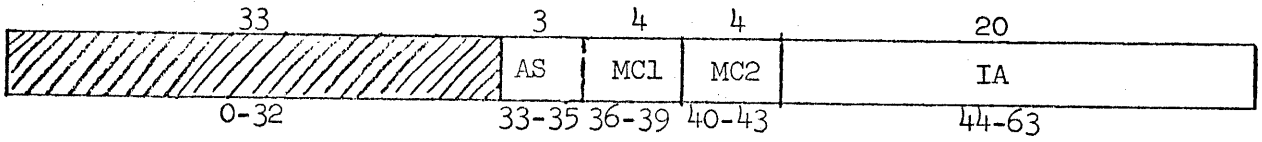
RECONFIGURE is a utility request for changing the configuration control in a multi-quadrant job. Job execution across several unitied quadrants may be split into asynchronous execution of different code streams in separate quadrants. After running asynchronously, the separate code streams in the job may be reunited.

There are three configuration control registers in each control unit: MC0, MC1, and MC2. MC0 controls the array size. MC1 controls instruction fetching; thus it determines in which quadrants the program code is located. MC2 controls instruction execution and data fetching; thus it determines in which quadrants the data is located.

There are a total of 167 valid combinations of the configuration control register settings. Within a multi-quadrant job, the user is concerned with specific settings of the MC1 and MC2 registers, but is not concerned with the absolute settings of his MC0 registers. However, he must have the capability of specifying relative MC0 settings to reflect his array sizes within the job. The RECONFIGURE request contains two parameters in AC3:



The pointer address is a nineteen-bit location of the first 64-bit word of a table in PE memory. The table contains the reconfiguration information in one to four words. The number of words is indicated by the parameter N in AC3. Each word contains sufficient information to provide configuration control for one subsequent code stream:



AS — Array size, three bits. Since the operating system controls the actual settings of register MC0, AS merely indicates the number of ones to be placed in MC0 to control the array size.

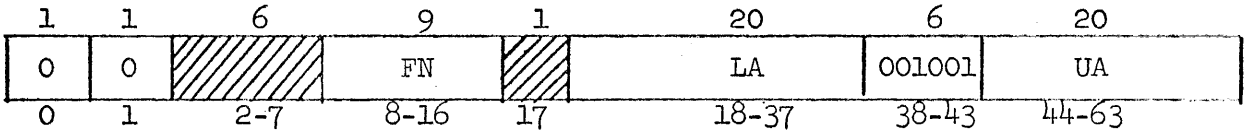
MC1 — Desired MC1 setting, four bits. Invalid settings of MC1 are three ones, or a bit position is set which is greater than the number of ones in MC0.

MC2 Desired MC2 setting, four bits. Invalid settings of MC2 are similar to those for MC1.

IA Interrupt address, twenty bits. After changing the configuration, control is to be transferred to the new code stream starting at this location.

Normally, if a user program wishes to fork its code stream into two parts, N=2 and two control table words are used. When rejoining, N=1 and only one control table word is required. The RECONFIGURE request is postponed and control is not given to the user program until all pending I/O requests are completed. The RECONFIGURE request causes the B6500 to be interrupted in order to modify future communication control with the job partner(s), but no loading of program segments is performed. If the user program does not contain the proper code segments for the new configuration, they must be obtained via LOAD requests before initiating the RECONFIGURE request. OS4 is structured in such a manner that reconfigurations of any type may be performed without requiring reloading of OS4. Joining separate code streams is performed after receiving the proper RECONFIGURE request from each of the streams so that proper synchronization of the larger union can be effected.

TRACE is a utility request for monitoring the flow of a program. Each time the instruction counter (ICR) is altered by an Exchange, Load, Skip, or Jump instruction, a branch trace interrupt causes OS4 to record the contents of ICR before and after the branch. The TRACE request contains three parameters in AC3:

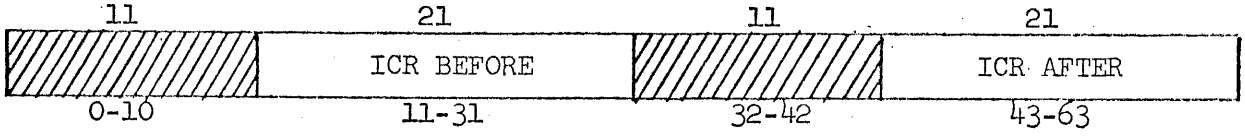


FN Logical file number, nine bits. A buffer of 100 words is maintained in PE memory. Each time the buffer is filled by 100 branch trace interrupts, the buffer is emptied by writing it out to the specified file on the ILLIAC IV disk.

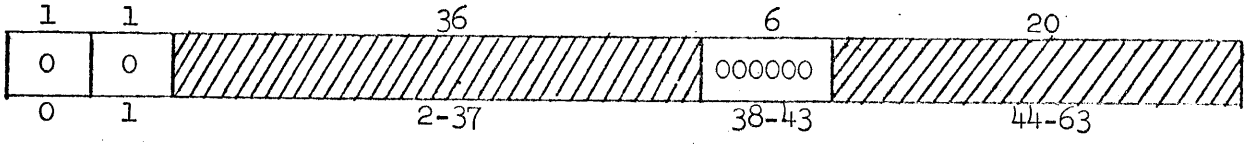
LA Lower address, twenty bits. Branches with ICR values less than LA are not recorded.

UA Upper address, twenty bits. Branches with ICR values greater than UA are not recorded.

The TRACE request can be nullified at any time by another TRACE request with LA=UA=0, at which time the partially filled 100-word buffer is written to the disk file. The buffer is also written at normal job termination if a TRACE request had been in effect. Each word of the buffer contains the before and after ICR values for one interrupt:

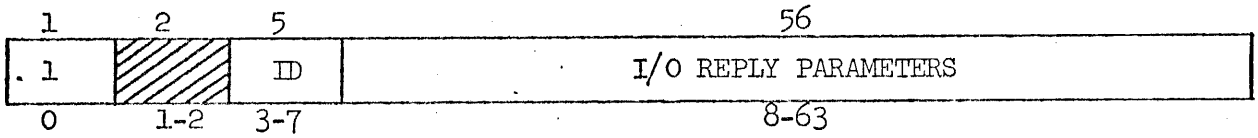


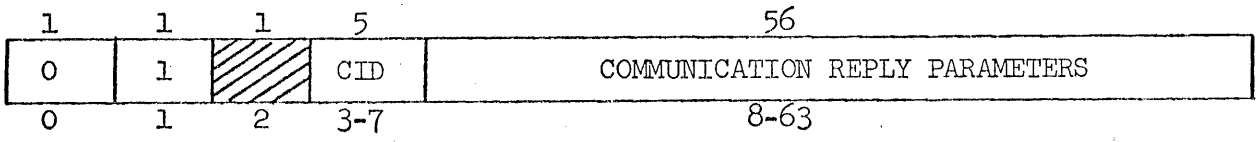
EXIT is a utility request used for terminating a job. It causes input files to be purged, and output files to be closed and transmitted to the B6500. The request has no parameters in AC3:



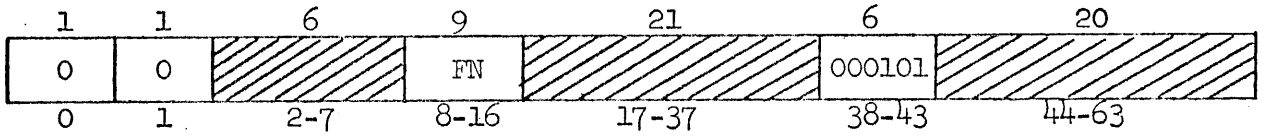
Responses to communications from the B6500 fall under three categories: replies to I/O and communication requests, commands from the B6500 job partner, and B6500 executive control.

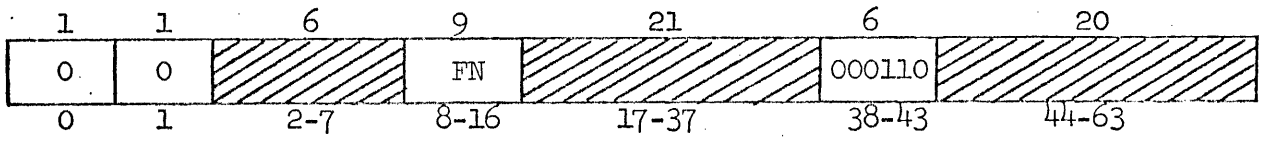
Replies to I/O and communication requests interrupt the user's program on ILLIAC IV via the TRI-loaded interrupt. The left-most two bits of TRI indicate the type of reply, and bits 3-7 contain the index to the proper 32-entry table:

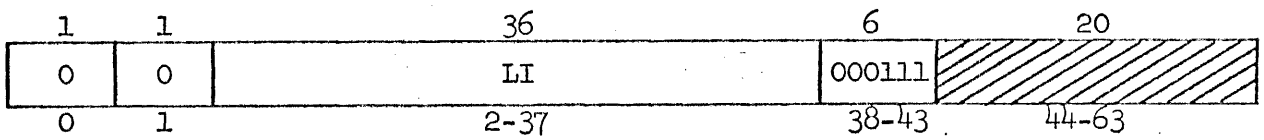




Job partner commands also interrupt the user's program on ILLIAC IV via the TRI-loaded interrupt. Included in the first implementation of the operating system are four job partner commands: BREAK, RESTART, LOAD, and TRAP. Bits 0-1 of TRI contain zeroes. The BREAK, RESTART, and LOAD commands perform the same functions as described above under utility requests. The formats of TRI are:







Upon completion of the BREAK, RESTART, or LOAD commands, OS4 places the command in TRI into TRO to signal the job partner that the command was performed. The RESTART command causes control to be given to the restarted program, but the BREAK and LOAD commands cause control to be returned to the ILLIAC IV program at the point of interruption.

TRAP is a job partner command that causes interruption of the ILLIAC IV program and execution of an interrupt routine on ILLIAC IV. The interrupt routine, either supplied by the user, the compiler, or the operating system, is executed in the normal state (not in the interrupt mode), as are all interrupt routines. The TRAP command format in TRI is:

1	1	36	6	20
0	0	TRAP COMMAND PARAMETERS	111111	IA
0	1	2-37	38-43	44-63

OS4 transfers control to the interrupt routine at the interrupt address (IA) after loading AC3 with the TRAP command contents of TRI. Return from the interrupt routine must be made with AC3 containing:

1	1	36	6	20
0	0	TRAP REPLY PARAMETERS	111111	TRAP REPLY PARAMETERS
0	1	2-37	38-43	44-63

Upon return from the interrupt routine, OS4 sends the contents of AC3, containing the TRAP reply parameters, back to the B6500 job partner via TRO and returns control to the user program at the point of interruption.

B6500 executive control is used for starting a job on ILLIAC IV and recovering from errors. It is also used for diagnostic control. It does not employ the use of the TRI-loaded interrupt, but rather forces control of ILLIAC IV by use of TMU commands. Job initialization consists of halting the control unit(s), setting the configuration control registers, clearing local memory, initializing the interrupt and advast control registers, bringing OS4 and the loader into PE memory from the ILLIAC IV disk, and issuing a LOAD command. Upon receipt of the LOAD complete signal, a TRAP command starts the ILLIAC IV job execution.

IX. THE LOADER

General

The Loader is an ILLIAC IV program located in the protected area of ILLIAC IV memory. It performs the tasks necessary to transform relocatable program segments into executable code and to overlay program segments during program execution.

The load file, prepared by the Collector, is read from the ILLIAC IV disk. The Loader ascertains the mode of memory allocation (see below). A Memory Data Table and Subroutine Return Stack are constructed for the job, all necessary relocation and instruction parity formation is performed, and the job is initiated by passing the address of the first location to be executed to OS4. During job execution, the Loader stacks all subroutine call return addresses and performs all explicitly and implicitly requested overlay. At the end of the job, the Loader supplies information necessary for diagnostic memory dumps, if requested.

Memory Allocation Modes

The Loader provides for the use of ILLIAC IV memory in one of three modes: static, fixed overlay, and dynamic overlay.

In the static mode, all program segments remain in memory throughout the execution of the job, in fixed locations. Thus this mode can be used only for memory-contained jobs.

In the fixed overlay mode, program segments may be overlayed by others, but the locations occupied by segments when in memory are fixed throughout the execution of the job. Segments may be explicitly called in by the user program at any time, and any overlay implicitly initiated by subroutine calls will also be performed. This mode may be used when a program is not memory-contained, and it is possible and practical to map the overlay scheme of the program.

In the dynamic overlay mode, program segments are not assigned to specific memory locations before job execution begins. Instead, each program segment is brought into memory and its relocatable addresses made absolute only when implicitly called for by user subroutine calls. This mode must be used when the program is not memory contained, and it is impractical to map the overlay scheme of the program.

Subroutine Linkage: The Memory Data Table

A Memory Data Table is constructed for each job by the Loader. It resides in the write-protected area in ILLIAC IV memory, and contains information used in subroutine calls and requests for overlay of program segments. A table allocation is made for each location declared as an ENTRY in the program. Each table location consists of two ILLIAC IV words. The first word is divided into inner (bytes 1-4) and outer (bytes 0 and 5-7) parts. The inner part contains the memory address of the ENTRY, as well as information relating to its presence in memory and the type of segment which contains it. The outer part contains the address of the appropriate Loader routine to process the call on the ENTRY. The second word of the table location contains a disk descriptor skeleton describing the segment on disk.

Subroutine Linkage: The Subroutine Return Stack

A Subroutine Return Stack is maintained by the Loader in order to provide the facility for recursive subroutine calls. Each time a subroutine is called, the Loader places the return address in the top of the stack. Each return from a subroutine is made to the address in the top of the stack, and the top of stack is deleted.

Subroutine Linkage: Standard Calls and Returns

The Standard Subroutine Call has the form:

```
SLIT(3) = SUB;
LØAD(3) $C3;
EXCHL(3) $ICR;
```

At execution time, the Loader will have placed in the address field of the SLIT instruction the address of the Memory Data Table location corresponding to SUB. Execution of these three instructions will cause control to

be passed to the appropriate Loader routine to process the call. When the called subroutine is entered, a Return Word is in ACAR 3. In order to return to the calling program, the called routine must place this Return Word in ACAR 3 and execute a

```
STL(3) $ICR.
```

This instruction passes control to the Loader return routine.

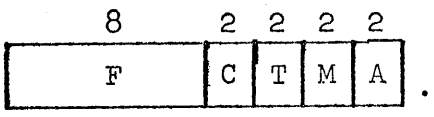
In the case of a call with parameters to be passed to the called subroutine, the form:

```
FILL;
SLIT(3) = SUB;
LØAD(3) $C3;
EXCHL(3) $ICR;
SKIP, 2 × (number of parameters);
```

followed by the parameters, each 64 bits long, must be used. On entry to the subroutine, the inner part (bytes 1-4) of the Return Word will contain the address-1 of the first parameter word in the calling program. Return is accomplished as in the parameterless case.

Relocatable Code Segments: Program Words

Program segments as produced by Assemblers and Compilers and passed to the Loader via the Collector are termed Relocatable Code Segments (RCS). The bulk of each Relocatable Code Segment consists of Program Words, composed of a 32-bit Program Syllable (PS) associated with a 16-bit Loader Information Field (LIF). The LIF informs the Loader of the action to be taken in processing the PS. The format of the LIF is:



The meaning of the codes is as follows:

C = 0 No loader action required (constant).

C = 1 The PS is an ILLIAC IV instruction syllable.

T = 0 SLIT, ALIT, JUMP

T = 1 Other CU instruction

T = 2 PE instruction.

M = 0 Absolute address

M = 1 Relocatable address

M = 2 Externally defined address. F points to the position in the External table containing the name of the external symbol. The address of the instruction contains a displacement which is added to the address of the external symbol at load time.

A = 0 Row type address arithmetic

A = 1 Word type address arithmetic

A = 2 Syllable type address arithmetic

C = 2 The PS represents bit string data. The bit string data facility provides the means for constructing data constants formed by concatenating bit strings derived from relocatably or externally defined quantities. F = the length of the string (in bits). A and M are as defined in the ILLIAC IV instruction case (C = 1). If M = 0 (absolute address) then up to 255 bits may follow, 32 per PS. If a PS contains less than 32 significant bits, the significant bits are left justified in the PS. If M = 1, the relocatable address is right justified in the PS. If M = 2, the high-order 8 bits of the PS contain the External table position and the low order bits contain the displacement.

C = 3 The PS contains a loader pseudo-instruction. The high-order 8 bits of the PS are the loader pseudo-operation code. The low order bits are defined according to the instruction. The loader-pseudo-instructions include:

ALJ absolute jump

BRLJ base-relative jump

SRLJ self-relative jump

ALS absolute skip

BRLS base-relative skip

SRLS	self-relative skip
PZRØ	place zeroes
PNØP	place no-ops
FILL	place no-ops conditional
SEGEND	End of program segment

Relocatable Code Segments: Entry Table

Included in each RCS is an Entry table, consisting of an eight character identifier and its associated relative location for each identifier declared as an ENTRY in the program segment.

Relocatable Code Segments: External Table

Included in each RCS is an External table, consisting of an eight character identifier for each identifier declared as EXTERNAL to the program segment.

Relocatable Code Segments: Miscellaneous

Included in each RCS is information relating to the Entry and External tables, confirmation of the segment as an ILLIAC IV program segment, length of the program segment, and the first executable location in the

X. THE HARDWARE SUPERVISOR

Several procedures, called intrinsics, are added to the standard B6500 Master Control Program (MCP) to handle ILLIAC IV I/O and Interrupts. These additional intrinsics are collectively referred to as the hardware supervisor.

I/O intrinsics are used by the operating system to transmit I/O commands to the IOC, and to verify disk I/O requests from the job partners and to generate the absolute disk addresses for these requests.

Interrupts are handled by an intrinsic that scans in IOC result descriptors and then uses the contents of the I/O command and ID fields to look up the job partner which should handle the result descriptor. The intrinsic then places the result descriptor into the job partner's RECEIVE queue so that the job partner can process the interrupt.

The hardware supervisor performs the control state functions that the other system programs cannot perform since they run in normal state. Therefore, it passes interrupts on to the appropriate program that must take action or issues an I/O descriptor that was built by a system module. The hardware supervisor performs a more complex task when it verifies all job partner requests for validity and maps virtual disk addresses into absolute disk addresses for the job partners.

The hardware supervisor receives all of its job partner-related information from the execution monitor. This information is used to check job partner requests for validity. In order to route interrupts to the proper partners and prevent interference in another quadrant, the hardware supervisor must know which job partners are responsible for which quadrants. The execution monitor provides the hardware supervisor with a table that identifies the appropriate partner for each quadrant and also marks the stack of the job partner program with this identification. In addition, the file map table, which was built by the disk file allocator to specify the mapping of user files on the disk, is presented to the hardware supervisor. The hardware supervisor puts this table into core in the form of the double dimensioned array described in section IV. This table is used to verify all I/O that uses the disk. Finally, a BIOM map is provided--and

updated as necessary--that describes the blocks of BIOM that the job partner may use.

In order to illustrate some of the hardware supervisor's activities, a description of the steps followed in a typical disk to array I/O request follows. It is important to realize that it is the job partner which eventually receives and acts on all CU and I/O interrupts. The hardware supervisor passes interrupts to the job partner, puts I/O descriptors for the job partners into their final form after checking for validity, and issues the verified descriptors in control state. The steps are numbered to correspond with the diagram in figure 10-1.

1. The user program running on ILLIAC IV decides to perform an
2. I/O request, which interrupts the program and transfers control to
3. OS4, the ILLIAC IV resident portion of the operating system. OS4 reacts by taking four actions: It
4. Constructs a one-word I/O request message from the user program's I/O request parameters and places it in the control unit's test maintenance unit output register (TRO). It also enters the request in a
5. Status table in the write-protected area of PE memory. The status table is referenced in subsequent requests for I/O status and upon completion of the I/O transaction. OS4 then performs a
6. CACRB instruction which sets a bit in the ADVAST control register and causes an interrupt of the I/O subsystem's controller descriptor control (CDC). OS4's last action is to
7. Return control to the user's program.
8. The CDC, having been interrupted by OS4, interrupts the
9. MPX (B6500 multiplexor), which in turn interrupts the
10. CPU (B6500 processor). The CPU is interrupted immediately if it is in normal state, processing a normal B6500 user program. The ALGOL compiler, data reformatting utility routines, and most of the ILLIAC IV operating system are normal-state programs. If the CPU is already processing another interrupt, it is in control state, and cannot respond immediately to this interrupt. Eventually, the CPU is interrupted, and control is given to the

11. MCP (B6500 Master Control Program). The MCP runs in the control state in response to various classes of interrupts, and controls I/O with all peripheral devices, and handles all the B6500 time-sharing activities such as paging of program and data segments in B6500 core memory. The hardware supervisor portion of the ILLIAC IV operating system, which is embedded in the MCP, also runs in control state. The MCP, upon gaining control because of this interrupt, causes the CPU to read the multiplexor's

12. Interrupt register. The contents of this register indicate which peripheral device is seeking attention. One bit is reserved for ILLIAC IV interrupts. The MCP, recognizing the interrupt as having come from ILLIAC IV rather than a standard B6500 peripheral, transfers control to the Hardware supervisor.

13. The hardware supervisor causes the CPU to "scan-in" a

14. Result descriptor from the CDC. This descriptor was generated by the CDC as a result of the interrupt from OS4, and indicates that a particular ILLIAC IV control unit is desiring attention. Recognizing this, the hardware supervisor causes the CPU to initiate an I/O signal to the CDC. The CDC then fetches an I/O descriptor from

15. B6500 memory. The I/O descriptor is part of the hardware supervisor's tables, and specifies a "read CU' operation. The CDC places the I/O descriptor into its .

16. CU descriptor control, which causes the TRO to be read into B6500 memory. When the read operation is finished,

17. The CDC is again interrupted, which in turn interrupts the MCP via the MPX and CPU, and the

18. Hardware supervisor again gains control. The hardware supervisor scans-in the result descriptor which indicates the "read CU" to be finished, and transfers control to the appropriate

19. Job partner. The job partner decodes the request read from the TRO, and asks the hardware supervisor to perform the necessary disk I/O. The hardware supervisor consults the

20. Execution monitor's tables to check the request for validity, and causes the CPU to initiate another I/O signal to the CDC. This time, the CDC fetches the I/O descriptor from B6500 memory which describes the "disk to array" read operation. The CDC places this I/O descriptor into its

21. Disk queuer. The I/O descriptor contains an absolute disk address. As the proper disk storage unit turns, eventually this address agrees with the

22. Disk timing track. If the storage unit and its disk file controller are not busy, the

23. Transfer of data from the disk to the PE memory begins. When the data transfer is finished, or an error is detected, the

24. CDC list control interrupts the CDC. The CDC in turn interrupts the MCP via the MPX and CPU, and provides the hardware supervisor with the disk-to-array result descriptor.

25. The job partner again gains control and checks the result descriptor. If an error is indicated, the job partner may decide to try again. If it does not, or if the result descriptor did not indicate any errors, the job partner requests the hardware supervisor to send an I/O-finished message back to OS4. The hardware supervisor checks the execution monitor's tables and causes the CDC to fetch the proper I/O descriptor from B6500 memory to cause the "write CU" operation. This operation writes the job partner's reply message from B6500 memory into the control unit's

26. TMU command register (TCR). When the operation is finished, the CDC is again interrupted, causing the repetition of interrupts and resultant actions through the MPX, CPU, and MCP. The hardware supervisor, upon receipt of the I/O finished result descriptor, terminates the B6500 actions for this request. Meanwhile, the TCR register has received the job partner's message in the form of a command to place the reply information into the control unit's

27. TMU input register (TRI). This action causes an interrupt of the

28. User's running ILLIAC IV program, which passes control to OS4.

29. OS4 responds by taking the message from the TRI register and placing it into the

30. Status table. If the original I/O request from the user program had contained an "interrupt address", OS4 leaves the interrupt mode and transfers control to an

31. Auxiliary interrupt routine supplied by the user. Entrance is made to the auxiliary routine at the specified interrupt address. When finished processing the reply message, the auxiliary routine returns control to OS4 via a program request interrupt, and OS4 returns to the

32. Main user program at the point of interruption. At any time during or after the I/O transaction, the user may perform a

33. Status request, which interrupts the program and transfers control to OS4. OS4 consults its protected

34. Status table, and returns to the user with the current status of the designated I/O request.

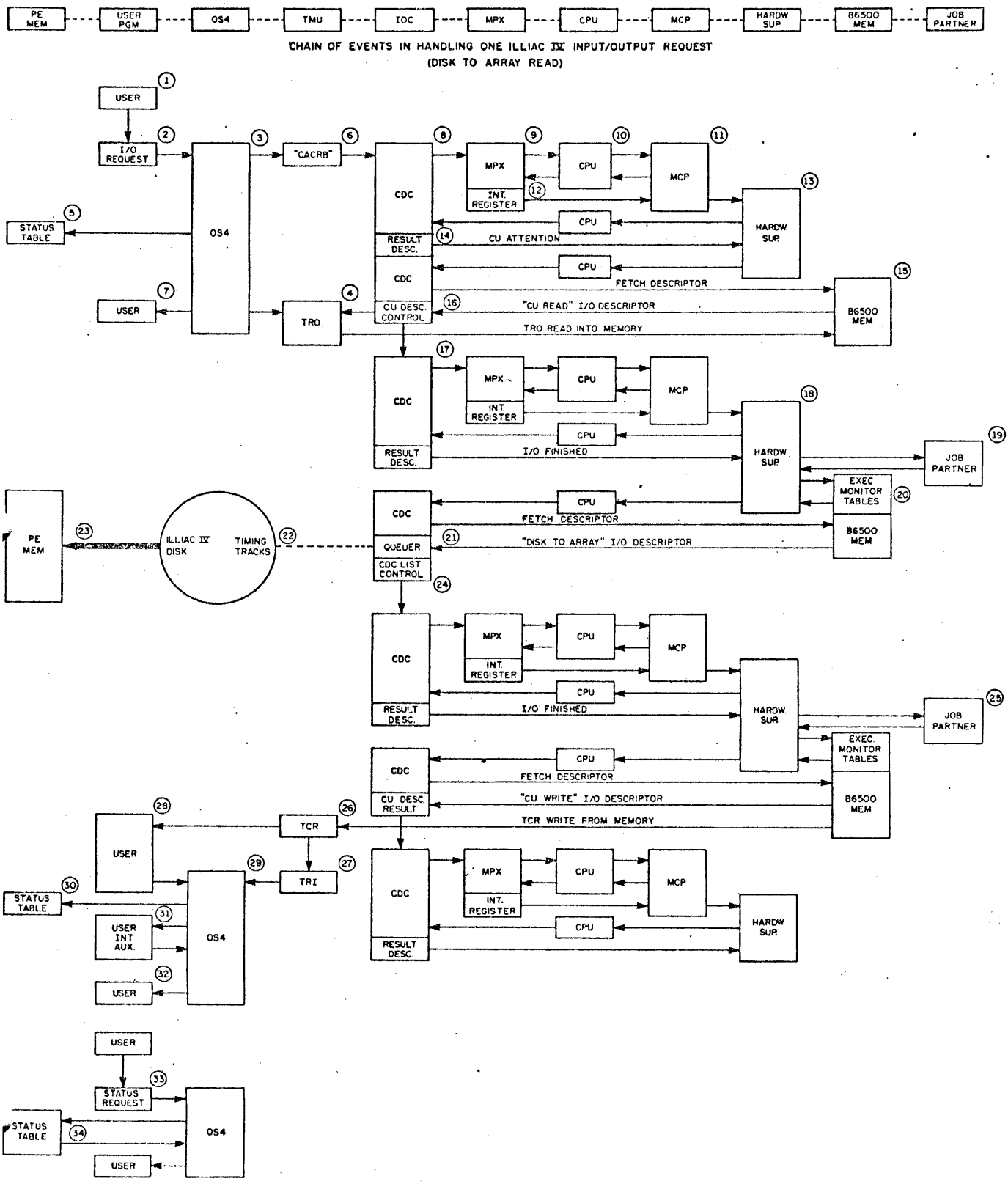
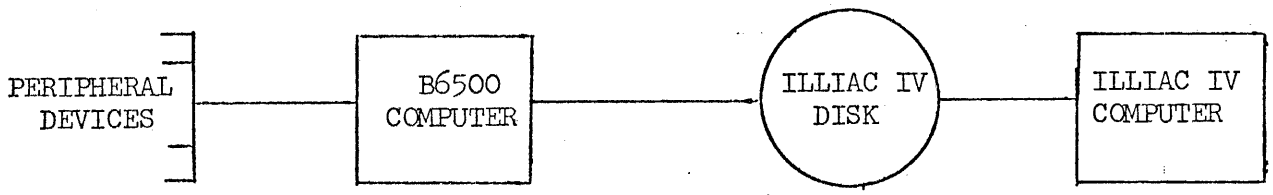

PE MEM
USER PGM
OS4
TMU
IOC
MPX
CPU
MCP
HARDW SUP
B6500 MEM
JOB PARTNER
CHAIN OF EVENTS IN HANDLING ONE ILLIAC IV INPUT/OUTPUT REQUEST
(DISK TO ARRAY READ)
USER
I/O REQUEST
"CACRB"
CDC
MPX
INT. REGISTER
CPU
MCP
HARDW. SUP.
STATUS TABLE
OS4
RESULT DESC.
CU ATTENTION
FETCH DESCRIPTOR
CU DESC. CONTROL
"CU READ" I/O DESCRIPTOR
TRO READ INTO MEMORY
TRO
I/O FINISHED
EXEC. MONITOR TABLES
QUEUER
CDC LIST CONTROL
"DISK TO ARRAY" I/O DESCRIPTOR
ILLIAC IV DISK
TIMING TRACKS
"CU WRITE" I/O DESCRIPTOR
TCR WRITE FROM MEMORY
TCR
TRI
USER INT AUX.
STATUS REQUEST

Figure 10-1

Appendix A. ILLIAC IV HARDWARE DESCRIPTION

The ILLIAC IV system contains four basic components: the ILLIAC IV computer, the ILLIAC IV disk, the B6500 computer, and peripheral devices. The ILLIAC IV computer is a parallel array of four identical computers, each configured into a control unit and a matrix of 64 identical processing units. Each processing unit contains its own thin-film memory of 2048 64-bit words. The ILLIAC IV computer, in itself, contains no input-output peripheral devices other than displays for hardware maintenance purposes. Its only major data path to the outside world is to the ILLIAC IV disk.



The ILLIAC IV disk subsystem is a set of sixteen disk storage units, each with a capacity for 921,600 64-bit words, with an average rotational latency of 20 milliseconds. The disk units are head-per-track files. The disk subsystem is capable of simultaneous reads and writes at a transfer rate with the ILLIAC IV array of 10^9 bits per second. This transfer rate, being matched with the ILLIAC IV memory, is faster than the B6500 memory. For this reason, a buffer input-output memory (BIOM) is used between the disk and the B6500 computer. The BIOM (not shown in the above diagram) is really identical to two of the ILLIAC IV processing unit memories. The B6500, for purposes of data transfers to the ILLIAC IV disk, views the BIOM as an extension of its own memory. The BIOM has a capacity of 4096 64-bit words, or 5456 48-bit B6500 words.

The B6500 computer is a large-scale time-sharing computer. It has a five-fold advantage in speed over its predecessor, the B5500. It contains a 600 nanosecond thin-film memory, expandable in 16,384 word modules to 524,288 48-bit words. The B6500, unlike the ILLIAC IV, has a port to the outside world, namely a multiplexor. The multiplexor has a direct connection with the control units of ILLIAC IV for control information, but not a wide enough path for large amounts of data. The path data normally takes to the ILLIAC IV is from peripheral devices through the multiplexor, through the BIOM, to the ILLIAC IV disk.

The B6500 peripheral devices are connected to the multiplexor. They include the standard set of input-output devices, namely card readers, card punches, supervisory printers, line printers, magnetic tape drives, teletypes, telephone data sets, mass storage, and B6500 disk. Also included is an interface message processor (IMP) connected to the ARPA net of remote equipments.

Appendix B. FILE SECURITY

1. A user is required to log in with a project name and a user name. He may also be required to use an authentication code.

2. Once a user is logged in, provided he is the administrator of the project under which he logged in, he may add projects and other users to the system. He uses the control statement:

EXECUTE FILE/SECURITY

3. The file security program takes note of the project name and user name under which he has logged in.

4. He may add a project to the security file via the statement:

ADD PROJECT <project name> UNDER <user name>

The FILE/SECURITY program uses a file called SECURITY/FILE. It checks this file for the name of the logged-in project name and ascertains whether the logged-in user is in fact the administrator of that project. If he is not, it replies with the statement:

YOU ARE NOT PROJECT ADMINISTRATOR.

If he is the administrator, the file is checked for a duplicate entry of the <project name>. If found, it replies with the statement:

DUPLICATE PROJECT NAME. USE ANOTHER.

If it does not find a duplicate, it records the project name and user name and responds with the statement:

PROJECT <project name> UNDER <user name> ADDED TO SECURITY/FILE.

The named project becomes a sub-project under the logged-in project name. The named user gains admittance to the system, may access all files accessible to the named project, and becomes administrator of the new project. The named user may be new to the system, may be a previously admitted user, or may in fact be the logged-in user himself.

5. The logged-in user may add other users to the system via the statement:

ADMIT USER <user name>

The FILE/SECURITY program searches the SECURITY/FILE for the logged-in project name. If the logged-in user is not the administrator of that project, it responds with:

YOU ARE NOT PROJECT ADMINISTRATOR.

If he is the administrator, the program compares the user name with recorded user names under that project name. If a duplicate entry is found, it replies with:

DUPLICATE USER NAME. TRY ANOTHER.

If a duplicate name is not found, the named user is admitted to the system as a user of that project, and the program responds with the statement:

USER <user name> ADMITTED TO PROJECT <logged-in project name>.

6. A project name, together with the names of all its sub-projects, may be disassociated from its parent project and moved to become a sub-project under another project via the statement:

MOVE PROJECT <project name> TO <name of new parent project>

Only a project administrator who has jurisdiction over the old and new parent projects may use this statement. If the logged-in user is found to be the administrator of the logged-in project, and the <project name> is a sub-project of the logged-in project, and the <name of new parent project> is also a sub-project name under the logged-in project name, the <project name> and all attached sub-project names and user names associated with the <project name> and sub-project names are moved from the old parent to the new, and the reply is made:

PROJECT <project name> MOVED FROM <old parent name> TO <new parent name>.

If all the above conditions are not met, one of the following replies is made:

YOU ARE NOT PROJECT ADMINISTRATOR.

PROJECT NOT UNDER YOUR JURISDICTION.

7. A project name, with all its sub-project names, may be removed from the SECURITY/FILE via the statement:

DELETE PROJECT <project name>

The program checks to see that the logged-in user is the administrator of the logged-in project. It then compares the <project name> with the logged-in project name and all other project names which are sub-projects of the logged-in project. One of three replies is made:

YOU ARE NOT PROJECT ADMINISTRATOR.

PROJECT NOT UNDER YOUR JURISDICTION.

PROJECT <project name> DELETED FROM SECURITY/FILE.

If the project is deleted, all users associated with that project and its sub-projects are deleted. Since a user may be associated with many projects, occurrences of his name are deleted only with reference to the deleted projects.

8. A user may be removed from the system via the statement:

DELETE USER = <user name> FROM PROJECT <project name>

If the logged-in user is found to be the administrator of the logged-in project, and the <project name> is the same as, or a sub-project of the logged-in project name, and the <user name> is found to be a user of that project but not the administrator of that project, the <user name> entry is disassociated from the <project name>. Entries with the same <user name>, but associated with other projects, are not disturbed by this action. One of four replies is made:

YOU ARE NOT PROJECT ADMINISTRATOR.

PROJECT NOT UNDER YOUR JURISDICTION.

USER IS ADMINISTRATOR. CANNOT BE DELETED.

USER <user name> DELETED FROM PROJECT <project name>.

9. In order to delete a project without at the same time deleting its sub-project, or in order to insert a project between a parent and sub-project, a combination of ADD, MOVE, and DELETE statements must be used. In order to delete every entry of a user's name from the system, a separate DELETE statement must be used to remove the user from the separate projects. Files created by the named user under the named project are not purged. They are

attached to the administrator of the named project. In order to delete an administrator of a project, his user name must be changed. Since an administrator cannot be deleted from a project, he can be replaced only by changing his user name:

REPLACE USER <user name> WITH USER <user name> UNDER PROJECT <project name>.

10. When a project is deleted, all files attached to that project are purged. In order to save such files, a separate statement should be used before deleting the project:

MOVE FILE <file name> FROM PROJECT <old project name> TO PROJECT <new project name>

Only a project administrator who has jurisdiction over the old and new projects may use this statement.

11. Once a user is admitted to the system, he may obtain protection of his admittance via the statement:

PROTECT WITH <authentication code>

Subsequent entries to the system are made by logging in with the authentication code in addition to the project name and user name. This authentication code is not revealed to any other user in the system. It is the user's private code. When lists of projects and users are provided, such as for billing, only the project names and user names appear. If at any time a user feels his authentication code has been compromised, he may change it by using the statement:

PROTECT WITH <new authentication code>

12. Project names, user names, and authentication codes are each restricted in length to eight alphanumeric characters. The format for logging in to the system is:

?LI:<project name>,<user name>

or

?LI:<project name>,<user name>,<authentication code>

In response to the log-in, the system replies with one of five statements:

PROJECT NAME NOT IN SYSTEM.

USER NOT PROJECT MEMBER. SEE ADMINISTRATOR = <project administrator name>.

AUTHENTICATION CODE INCORRECT.

PLEASE AUTHENTICATE.

STATION <station number> LOGGED IN AT <time>.

13. Any user may obtain a complete list of projects, administrators, and users in the SECURITY/FILE via the statement:

LIST

Authentication codes, of course, are not shown. The listing has the following format:

1: Project name - Administrator name
User name
User name
⋮

2: Project name - Administrator name
User name
User name
⋮

2: Project name - Administrator name
User name
User name
⋮

3: Project name - Administrator name
User name
User name
⋮

4: Project name - Administrator name
User name
User name
⋮

2: Project name - Administrator name
User name
User name
⋮

3: Project name - Administrator name
User name
User name
⋮

3: Project name - Administrator name
User name
User name
⋮

14. When a user creates a file, he, and only he, has full access to that file. He may let others have access to it via the statement:

OPEN <file names> TO <accessors> VIA <access type>

A file name can have any number of parts, each separated by slashes, and each restricted to eight alphanumeric characters. Several files, each to have the same access, can be named separated by commas. The accessors can have one of the following five forms: (1) A project name permits access by any user of the project; (2) A project name, followed by one or more user names restricts access to only those named users of the project; (3) The construct <project name> + PARENTS permits access by all users of the project and all users of projects in a direct ancestral line to the named project; (4) the construct <project name> + SUBS permits access by all users of the named project and all users of every project that is a sub-project of the project and their sub-projects; (5) the word ALL permits access to all users in the system.

Any number of representatives of the five accessor forms may be strung into a list, separated by semicolons. Within type 2, the users are separated from the project name and each other by commas. The word VIA ends the accessor list. The access type can have one of three forms: READ, WRITE, or ADD, or any combination of the three, separated by commas. A list can

be formed by use of semicolons to correspond one-for-one left to right with a list of accessors. A distinction is made between the access types WRITE and ADD. Any part of a file may be written into, including overwriting information via the access type WRITE. Access type ADD permits writing additional records into the file, but not overwriting previously recorded information.

Even though a creator of a file is the only one who has access to it when the file is first created, the file is created under a project. The file creator, the administrator of the project, and the administrators of parent projects are all permitted to use the OPEN statement for that file. Providing access to a file does not provide the facility for using the OPEN statement.

15. Any user can construct private accessor-type declarations for use in OPEN statements. They are entered into the SECURITY/FILE for future use of the particular logged-in project-user. The declarations are identified by means of the statement:

DECLARE <access identifier> = <accessors> VIA <access types>

Subsequent uses of OPEN statements can take the form:

OPEN <file names> TO <access identifier>

16. Users can be denied access to files via the statement:

CLOSE <file names>.TO <accessors> VIA <access types>

The file names, accessors, and access types are structured as in the OPEN statement. A file may be closed to all access types via a shortened version of the statement:

CLOSE <file names> TO <accessors>

The file creator and parent administrators are permitted to use the CLOSE statement.

17. Files may be purged from the system by its creator or any parent administrator via the statement:

PURGE <file names>

18. Any user who has access to a file may increase the length of time the file is to be kept in the system via the statement:

SAVE <file names> UNTIL <date>

Several files may be named separated by commas. Access by the user is checked independently for each file. Rejection of the request for a non-accessible file does not affect acceptance for accessible files. Files already to be saved to a later date than <date> are unaffected. For each file, one of two replies is made:

YOU HAVE NO ACCESS TO <file name>

FILE <file name> SAVED UNTIL <date>

19. The administrator of the parent project of all projects in the system has the capability of obtaining billing information via the statement:

BILLING FROM <starting date> TO <ending date>

A separate page of the output is started for each project in the system. Included with that project are all of its sub-projects. Thus, each project administrator can receive a copy of the charges for his project and the sub-projects under his jurisdiction.

20. The administrator of the parent project of all projects in the system has the capability of erasing billing history via the statement:

ERASE BILLING TO <ending date>

21. Execution of the program FILE/SECURITY is terminated via the statement:

END

22. The SECURITY/FILE consists of four files: a directory of project names, called SECURITY/FILE/PROJECTS; a directory of user names, called SECURITY/FILE/USERS; a directory of all files in the system, called SECURITY/FILE/FILES; and a directory of all file access identifiers, called SECURITY/FILE/ACCIDS. The formats of these files are as follows:

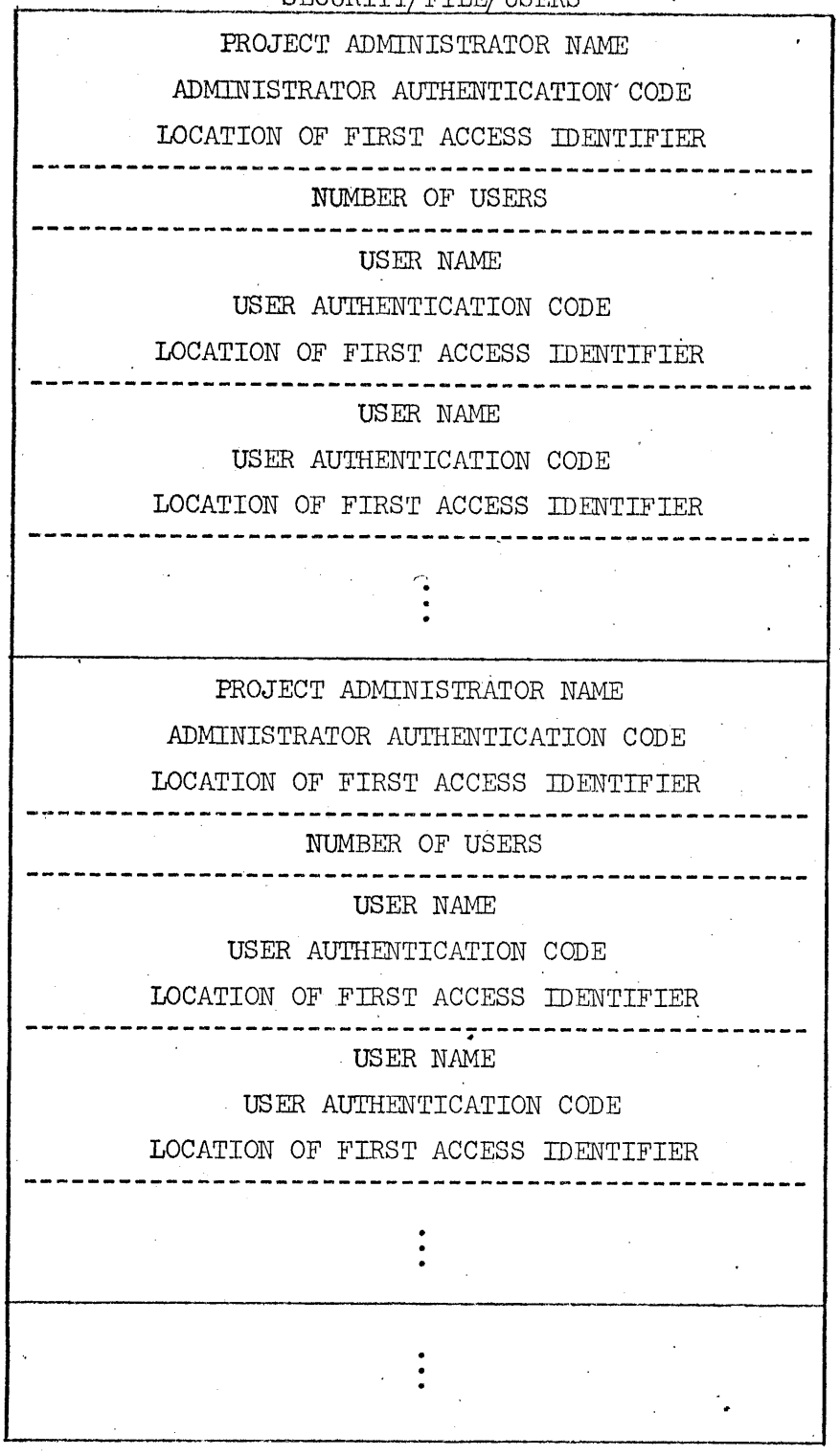
SECURITY/FILE/PROJECTS

PROJECT NAME
LOCATION OF PROJECT ADMINISTRATOR NAME
LOCATION OF PARENT PROJECT NAME
NUMBER OF SUB-PROJECTS
LOCATION OF SUB-PROJECT NAME
LOCATION OF SUB-PROJECT NAME
⋮

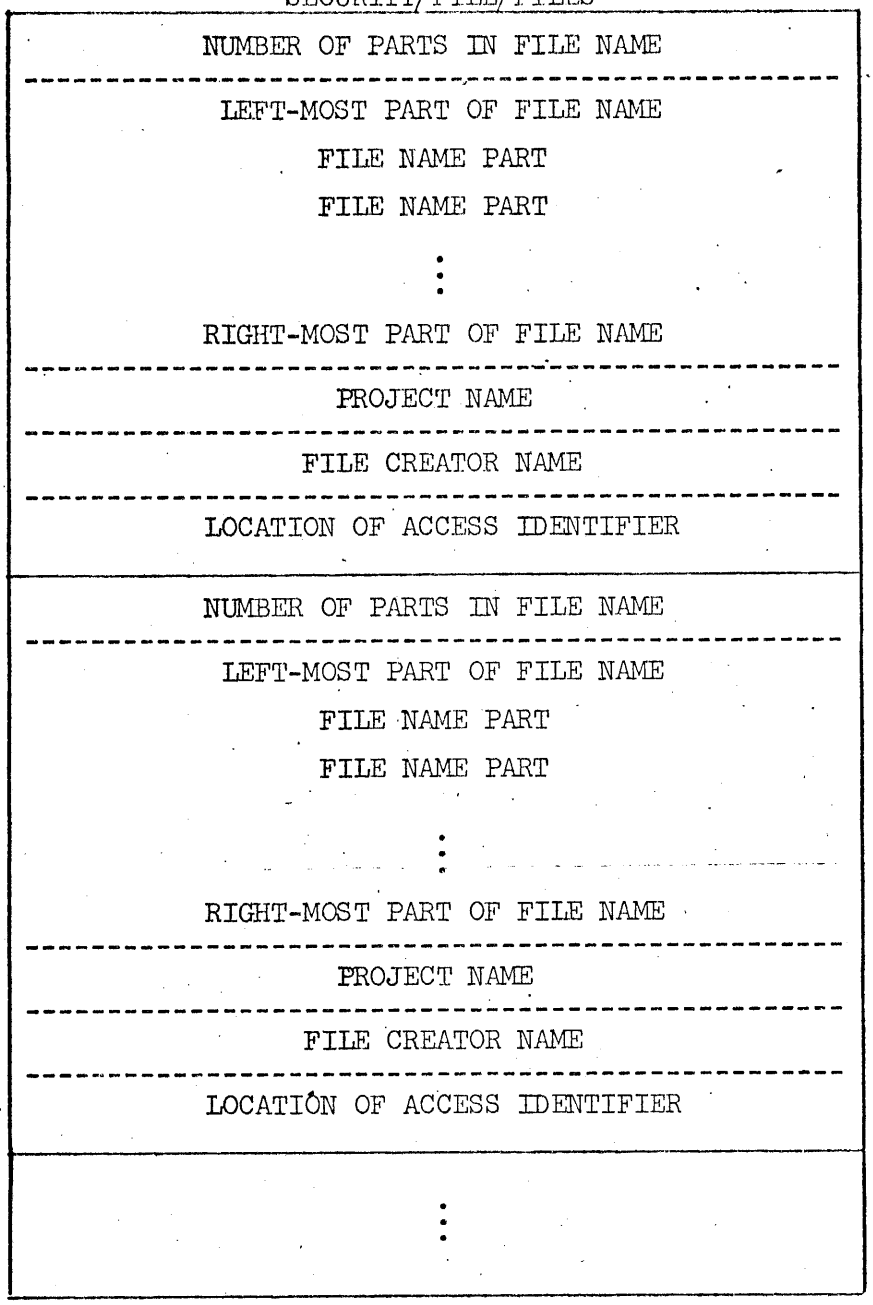
PROJECT NAME
LOCATION OF PROJECT ADMINISTRATOR NAME
LOCATION OF PARENT PROJECT NAME
NUMBER OF SUB-PROJECTS
LOCATION OF SUB-PROJECT NAME
LOCATION OF SUB-PROJECT NAME
⋮

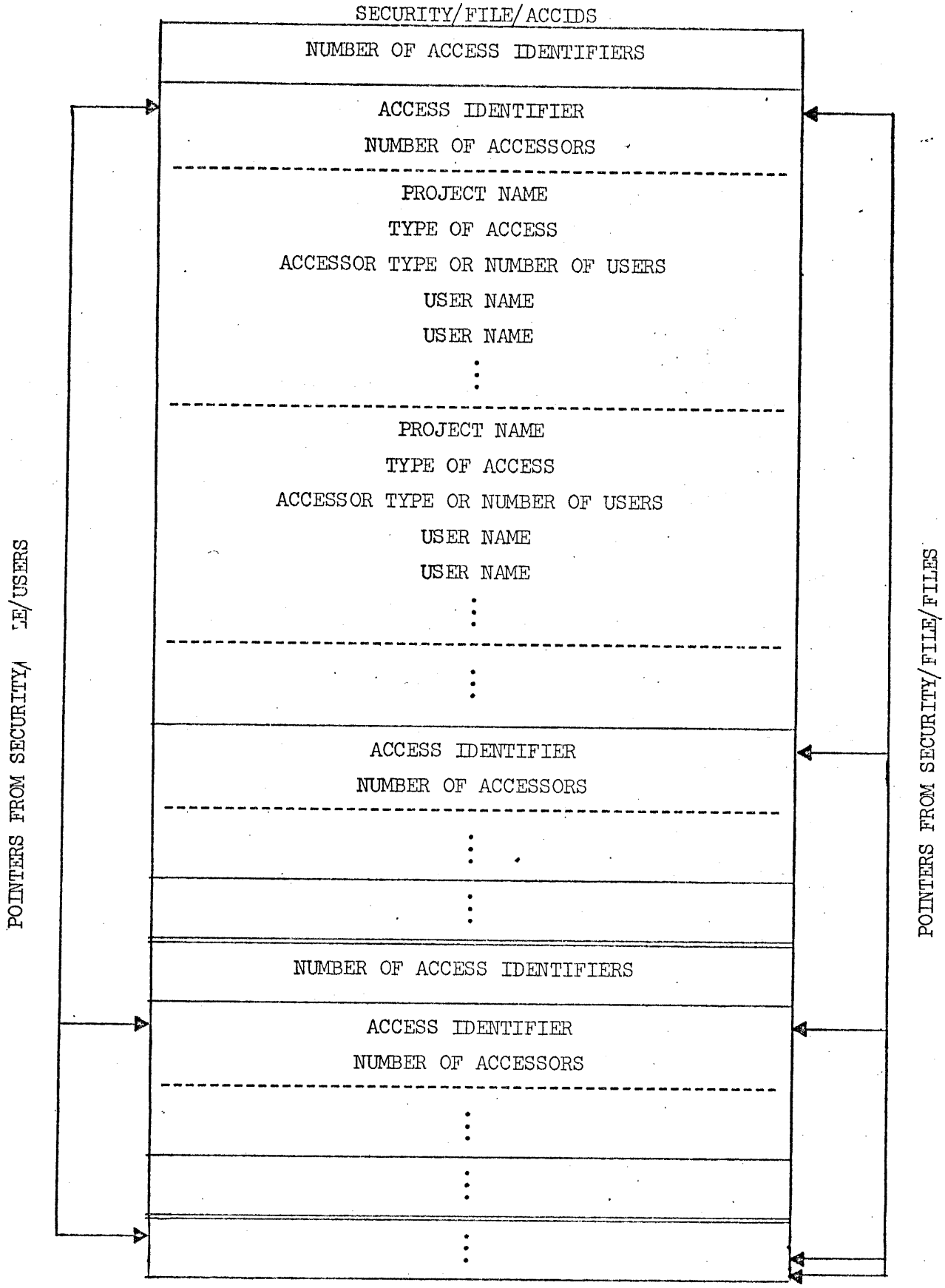
⋮

SECURITY/FILE/USERS

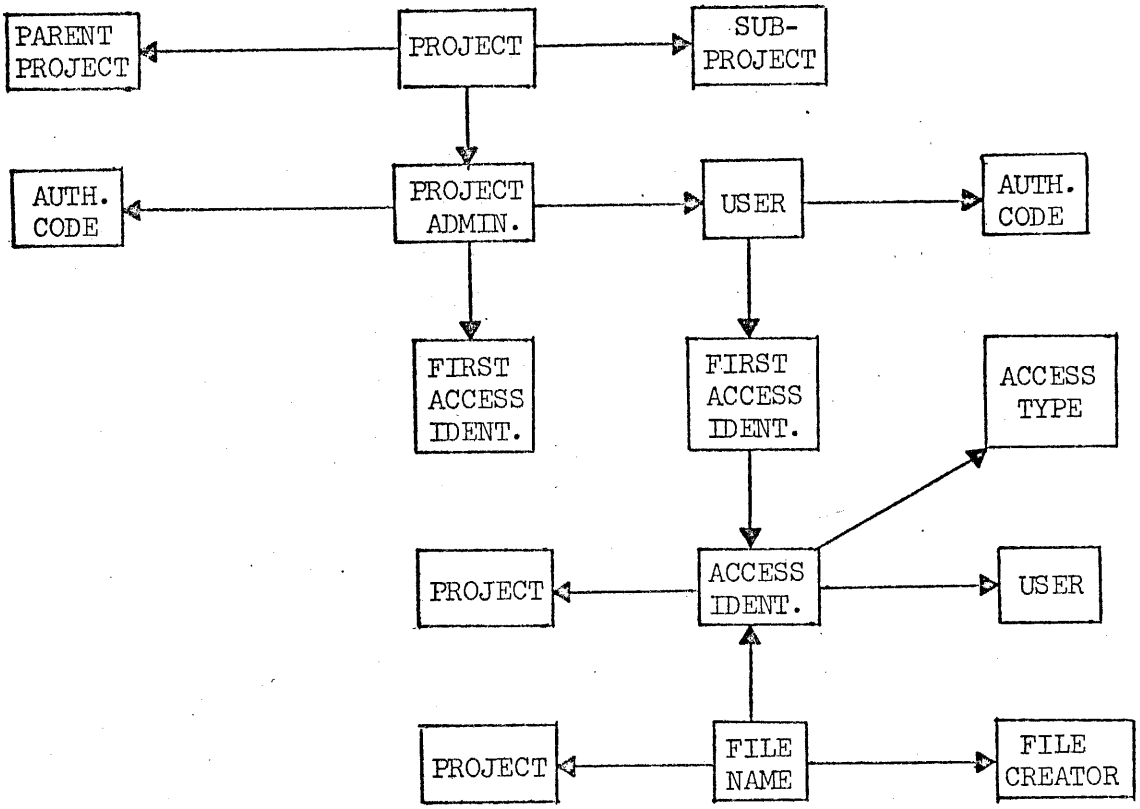


SECURITY/FILE/FILES




SECURITY/FILE/ACCIDS
NUMBER OF ACCESS IDENTIFIERS
ACCESS IDENTIFIER
NUMBER OF ACCESSORS
PROJECT NAME
TYPE OF ACCESS
ACCESSOR TYPE OR NUMBER OF USERS
USER NAME
USER NAME
PROJECT NAME
TYPE OF ACCESS
ACCESSOR TYPE OR NUMBER OF USERS
USER NAME
USER NAME
ACCESS IDENTIFIER
NUMBER OF ACCESSORS
NUMBER OF ACCESS IDENTIFIERS
ACCESS IDENTIFIER
NUMBER OF ACCESSORS
POINTERS FROM SECURITY/ LE/USERS
POINTERS FROM SECURITY/FILE/FILES

23. The FILE/SECURITY set of files forms a tree. A subset of the tree, with its file pointers, can be diagramed as follows:



Appendix C. INTERACTIVE USES OF ILLIAC IV

It can be argued that there should be no interactive use of ILLIAC IV. The machine is too powerful to be allowed to sit idle while awaiting human response. On the other hand, it is clear that (1) interactive use of computers has become very popular among programmers and (2) there will be occasions when the interactive use of ILLIAC IV would be a useful extension.

Obstacles in Problem Solving

When computers were scarce, the lack of computer time and the failure of numerical algorithms were the primary hurdles which faced the physical scientist in attempting to solve his problems. While both these problems persist as machines become faster, a new obstacle also arises. The effort required to debug a written program can limit the number of new problems that can be solved, regardless of how fast the machine is or how applicable the numerical algorithms are. Great strides are being made in developing appropriate numerical algorithms. The advent of fourth generation machines will make available several hundred times the computing power that was available with third generation machines. Thus the obstacles imposed by limited resources in numerical mathematics and computer time are shrinking. It seems only appropriate to also attack the new obstacle, which lies not in mathematics or in the computer but in the man-machine interface.

Compilers and Programming Effort

No one disputes the usefulness of higher level languages. Their sole purpose is to make it easier to compose code and thus increase the productivity of a programmer. The inefficiency in the object code produced by the compiler is generally accepted because considerably less programmer effort is required to compose a code. Clearly, in using a higher level language, the resource of computer time is compromised to enhance the resource of programmer effort.

Considerable effort has been expanded on the Tranquil compiler so that the amount of programmer time required to compose a program is not exorbitant. But composing the program does not complete the programming phase of the job. The program must be debugged before it can be run productively.

On conventional computers, the length of time required to compose a large code and the length of time required to debug a written code require roughly the same length of time. Why not devote some effort to shortening the time required by a programmer to debug the code he has so conveniently written in Tranquil?

Debugging Aids

The ILLIAC IV is a radical new design. It will be used principally by people who are familiar with conventional machines. These people will attempt to implement a new generation of problems in a language which is totally foreign to them. (They use FORTRAN.) The software will have errors. Many of the people using the machine will reside elsewhere and will wish to minimize their stay in Illinois. All of these factors will complicate the debugging process. Users must be given adequate tools to offset these complications or the result will be a long delay between the completion of coding and the first production run.

In order to reduce the time and effort required to debug a larger code, the following features will be implemented in the system. CRT-keyboard devices will be available to allow the user to "interact" with his ILLIAC IV program. A user will be able to:

(1) Breakpoint through the instruction stream and display the current value of any memory location or any operating register. Breakpoints may be specified relative to machine instructions or Tranquil statements.

(2) Change the value of any memory location or operating resistor and resume execution with the new value. The programmer will be able to select one of several formats (floating point, integer, BCD, etc.). The memory locations to be displayed may be referred to by location number or symbolic name.

Since a CRT can display many values (perhaps 30-40) simultaneously, there is no reason to restrict the information displayed at one time. This "interactive" mode may be entered (that is, the first breakpoint may be imposed) in one of the following ways: (1) at the command of the operator, (2) upon entering a specific subprogram during the execution of an ILLIAC IV program, (3) after executing for a certain length of time.

No Degradation of Performance

These features can be implemented without significantly degrading ILLIAC IV's performance, and at the same time it will appear to the user that he is truly interacting with ILLIAC IV. When execution of a program that is being run in "interactive" mode is suspended (execution of a breakpoint), a file will be written on ILLIAC IV disk which contains a core dump and a dump of all the PE registers. When the user has identified those values he wishes to have displayed, the B6500 will load a buffer in its memory from the ILLIAC IV disk and paint the CRT from this buffer, performing suitable encoding and formatting. If the user asks for the value of a variable by its symbolic name, the location of that variable in ILLIAC IV memory will be obtained from the loader. If any values are to be modified, they will be entered from the keyboard, decoded, and overwritten on ILLIAC IV disk. The program will then be restarted by loading ILLIAC IV from the disk file. (It is implicit in this implementation that the user will have sufficient memory maps provided by the compiler or loader.) Since the human response time will be in seconds while the B6500-ILLIAC IV interaction will be in milliseconds, the process will appear to the user as an interactive one, but to ILLIAC IV it will be offline.

Interacting During Production

Some form of monitoring during a production run is necessary. If the CRT-keyboards equipment has the capability to draw vectors, it should be almost straightforward to develop software packages to search ILLIAC IV data files and display "snapshot" graphical output, which can be monitored continuously during the execution of a production run. This is a powerful extension of the interactive process since any breakdown in a production code can be detected immediately, and the job can be aborted.

Advantages of Interactive Debugging

The conventional debugging procedure is to insert print commands at strategic points in the code, execute until a disaster occurs, then attempt to isolate the erroneous logic. Generally, this requires many repetitions, with the print commands being expanded and rearranged. The programmer must make repeated runs, often encountering scheduling delays, often frustrated because he did not print all that was needed. If an exhaustive print is

attempted or if repeated dumps are taken, the printers are occupied for extended times and pouring through the massive output is very time-consuming. Subtle errors which are encountered when the compiler or operating system "goofs" are almost impossible to detect using these procedures, and often costly delays of several weeks result.

Using interactive debugging, the program is not required to submit several runs in order to find the values he needs. Any values can be displayed at his command, so the troubleshooting process is uninterrupted. The printers are not tied up, and the programmer does not have to thumb back and forth through reams of printout, diligently searching for a vital value. Compiler errors can be detected almost as easily as logic errors and can be corrected on the spot, rather than waiting for a "fix" to the compiler.

There are certain problems which require "interaction" as the only practical method of solution. These problems include very complex non-linear equations in many unknowns and optimization problems with non-linear constraints. Solution of these equations by ordinary techniques is extremely difficult and requires many separate runs. Many decision-making processes must be programmed. With an interactive CRT, a graph is displayed, and parameters are adjusted (with the human mind performing the problem-solving decision process) until a solution having the desired properties is generated.