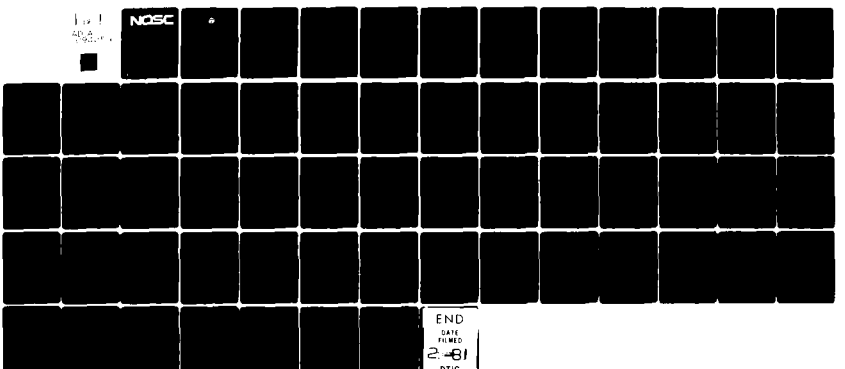
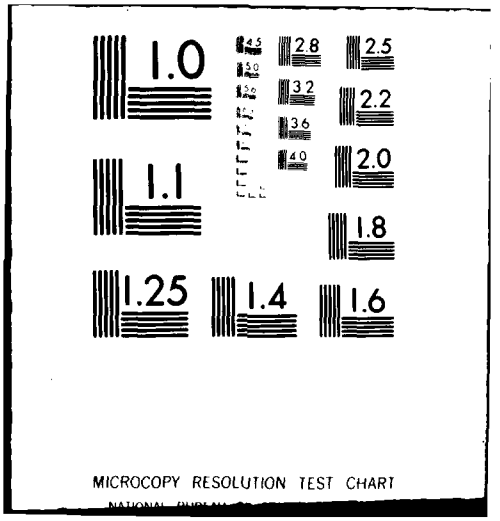
AD-A094 259 NAVAL OCEAN SYSTEMS CENTER SAN DIEGO CA F/G 5/1
DESIGN ACHIEVEMENTS OF THE COMMAND CENTER INFORMATION SUBSYSTEM--ETC(U)
AUG 80 D L SMALL, D O CHRISTY
UNCLASSIFIED NOSC/TR-581 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART

NOSC

NOSC TR 581

NOSC TR 581

Technical Report 581

DESIGN ACHIEVEMENTS OF THE COMMAND CENTER INFORMATION SUBSYSTEM (CCIS)

A dynamic and extensible computing system for natural language understanding of Navy command center information

DL Small and DO Christy

30 August 1980

Prepared for
Naval Electronic Systems Command

Approved for public release; distribution unlimited

NAVAL OCEAN SYSTEMS CENTER
SAN DIEGO, CALIFORNIA 92152

NAVAL OCEAN SYSTEMS CENTER, SAN DIEGO, CA 92152

AN ACTIVITY OF THE NAVAL MATERIAL COMMAND

SL GUILLE, CAPT, USN
Commander

HL BLOOD
Technical Director

ADMINISTRATIVE INFORMATION

Work was done under program element 62721N, subproject XF21241100, by the Communications Support System Branch (Code 8121).

Released by
HF Wong, Head
Ship and Shore Communications Systems Division

Under authority of
HD Smith, Head
Communications Systems and Technology Department

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(9) **REPORT DOCUMENTATION PAGE**

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NOSC Technical Report 581 (TR 581)

2. GOVT ACCESSION NO.: AD-A094259

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): (6) DESIGN ACHIEVEMENTS OF THE COMMAND CENTER INFORMATION SUBSYSTEM (CCIS), A dynamic and extensible computing system for natural language understanding of Navy command center information.

5. TYPE OF REPORT & PERIOD COVERED: (14) NOSC/TR-581

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): (10) DL/Small [redacted] DC/Christy

8. CONTRACT OR GRANT NUMBER(s): (17)

(12) 61

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Naval Ocean Systems Center
San Diego, CA 92152

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
62721N XF21241100

11. CONTROLLING OFFICE NAME AND ADDRESS: (11)
Naval Electronic Systems Command
Washington, DC

12. REPORT DATE: 30 Aug[redacted]80

13. NUMBER OF PAGES: 58

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): (16) F21241

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Command Center Information Subsystem (CCIS)
Information systems
Management information systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

CCIS is a revolutionary new hardware and software system for Navy data management, giving a command center the capability for providing high quality information to its commander as an aid in his decision making. It allows processing of natural language queries to proceed at the same time as dynamic data update. It provides extensibility so that the computer can "understand" the commander's individualized description of data received in the command center from standard static and dynamic sources. It also allows those dynamic sources to be customized uniformly to the particular command center.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

393159

DESIGN ACHIEVEMENTS OF THE COMMAND CENTER INFORMATION SUBSYSTEM (CCIS)

A Dynamic and Extensible Computing System for Natural Language Understanding of Navy Command Center Information.

I. INTRODUCTION

The Command Center Information Subsystem (CCIS) is a Navy exploratory development project, begun in FY76 and completed in June 1980. It is a revolutionary new hardware and software system for Navy data management, giving a command center the capability for providing high quality information to its commander as an aid in his decision making. The system allows processing of natural language queries to proceed at the same time as dynamic data update. It provides extensibility so that the computer can "understand" the commander's individualized description of data received in the command center from standard static and dynamic sources. It also allows those dynamic sources to be customized uniformly to the particular command center. Such alterations can facilitate using a data base for, among other things, analyzing the state of enemy forces, assessing the readiness condition of own forces, determining the status of communications possibilities for a task force and planning of possible engagement with the enemy. CCIS' approach to dynamic data update provides current "real time" information to the data base to support those functions.

The stated objective of CCIS has been and is today to improve the quality of information available for use in the command center. Responsible command center commanders are presently inundated with massive quantities of data and often cannot readily access the data they need to aid in making the decisions they face.

The questions hardest to deal with and understand are

1. which of that data IS relevant in aiding the commander's decision making,
2. how does he describe that data,
3. how can the computer understand his description, and
4. how can timeliness of retrieval and integration of relevant data be assured.

It is NOT the intent of CCIS to help the commander decide which data IS relevant. Rather it provides EXTENSIBILITY so that the commander, for ease of understanding, can individualize the description of data received in the command center from standard sources and so that those sources can be customized uniformly to the particular command center (sources may vary between ships depending on the degree of automation available or the function of the command center; i.e., carrier task force command center, amphibious command center). For any given CCIS installation, the standard database description; i.e., all static database elements and the vocabulary which describes both static and dynamic data base structure, is stored permanently on CCIS's bulk storage unit. Those descriptions can be altered only in the commander's working copy of the standard database description.

Such alterations can facilitate using the database for planning and threat analysis. For instance, the commander could pose the hypothetical question of what would be the shortest time required for the most ready of the Task Force's units to intercept a disabled vessel which had been picked up on Task Force radar, where positions are dynamically updated in real time for all platforms involved and the most ready units are defined to be those with the most fuel. Or the commander could ask what would be the Task Force's radius of detection, where radius of detection was the greatest range of the Task Force carrier's aircraft. Or, the commander could ask what would be the weapon range of hostiles, where hostiles

are Russian boats whose distance is less than 100 miles from the track of the Task Force carrier. Such alterations do not modify the standard database definition, allowing a new commander to use the system without having to learn the specialized vocabulary of the previous commander and without being constrained by it.

These alterations are all explicit extensions to the CCIS system by the operational commander and his staff to facilitate its use in command center problem solving. The computer's understanding of his intent thus is limited to the processing of those functions and vocabulary he defines plus processing of a basic set of database management utilities for data update, retrieval and analysis. There is no attempt by the computer to "guess" his intent through construction of semantic models of his reasoning or through inferring meaning from informal English text which he or others might draft. Rather attention is paid to high performance understanding by the computer of those problems which the commander himself can explicitly, yet naturally, describe and understand. Such performance is accomplished through a functional separation of translation, performed by the Query Processor (QP) of natural language queries into data analysis and update commands from the execution of those commands in a back end Data Processor (DP). A third processor, the Interface Processor (IP), is used for interface to non-CCIS automated data sources (see Figure 1).

The objective of this report is to summarize the CCIS system capabilities and outline design issues uncovered in the process of its development. The following describes in detail the CCIS static and dynamic data models, extensibility capabilities provided, types of queries possible and the system architecture developed to support this processing.

II. THE DATA MODEL

The data model in which the standard database is described and which the user can manipulate is a primitive relational one. The primitive unary relations are termed "CLASS", and are used to designate groups of data with one characteristic in common; i.e., "SHIP" is a CLASS which contains elements such as the CLASS(es) CVN and CGN or the NAME(s) EISENHOWER and NIMITZ. The primitive relation term "RELATION" is used to link two alphanumeric data items, such as two "NAMES", two "CLASSES", or a "NAME" and a "CLASS". The primitive relation term NUMBER RELATION is used to link a number and a "NAME", or a number and a "CLASS". In the examples which follow, relations will be represented as tables with each row representing one n-tuple (relational statement), where n represents the tuple-size of the relation; i.e., n is the number of columns in the table. For all primitive relations (be it CLASS, RELATION or NUMBER RELATION) the system maintains a begin and end time and a quantity field. For data entered by the user through the Query Processor, begin time is 0 and end time is day 377000000 (+ infinity) in Julian time and the quantity of an item is 1. Other values for these fields can be entered through the Interface Processor. However, there is no syntax to support this second method of data entry in the Query Processor. Thus, for a CLASS, from the Query Processor view of data entry n = 1 and for the other structures described n = 2. Internally, the tuple-size for CLASS is 5 and for the other structures it is 7. In special cases of query, the time fields can appear; i.e., begin time is greater than 0 and end time is less than 377000000; and the quantity field can appear; i.e., it is greater than 1. A quantity of 0 will cause the tuple not to be printed.

As can be seen from the example primitive relations described in the next section, there is no way for the system to distinguish between different words, other than as the primitive relation designation "NAME", "CLASS", RELATION", or "NUMBER RELATION". Words have no other intrinsic meaning until grouped in classes or related by

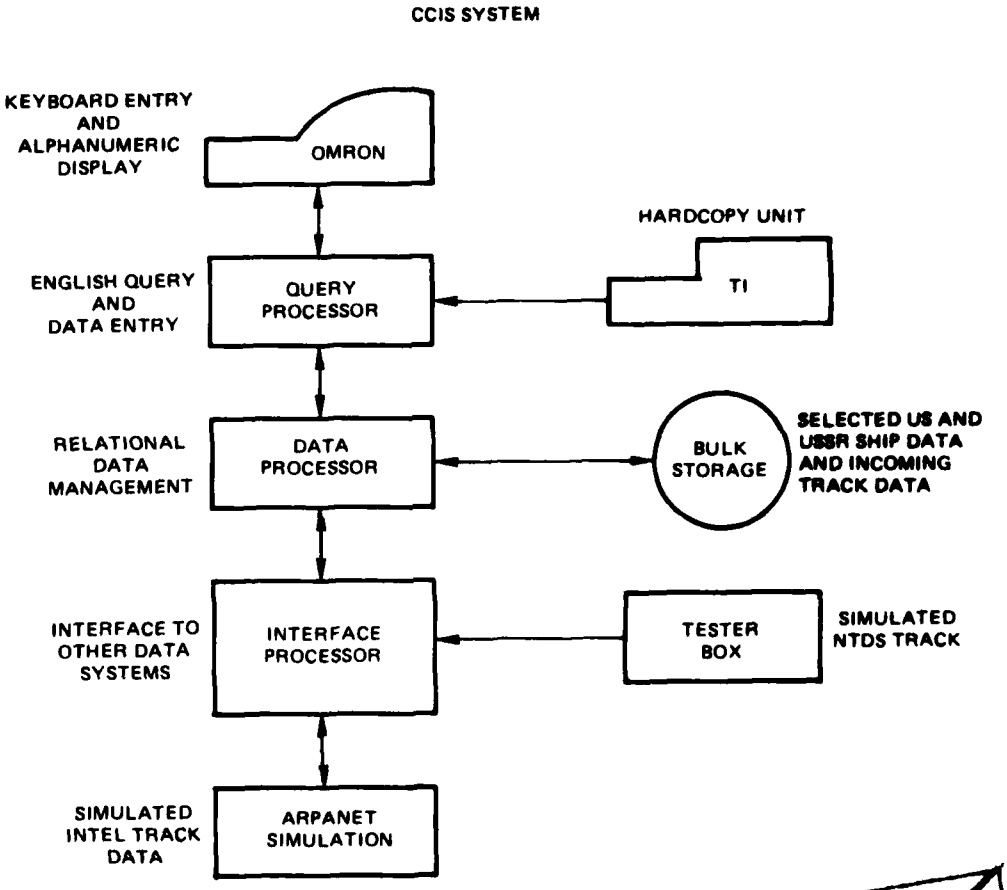


Figure 1. CCIS SYSTEM.

relations to other words or numbers. These meanings are supplied to the CCIS system by declarative sentences via the Query Processor or by special Data Processor semantic command structures sent via the Interface Processor.

Conceptually, the simplest operations to be performed by the user to build and extend his data model are the creation of names, classes and binary relations. Creation of a NAME, i.e., "NIMITZ" or "US", can be done by typing "NIMITZ: = NAME" or "US: = NAME". Creation of a CLASS; i.e., the class of "CVN", can be done by typing "CVN: = CLASS". Creation of a binary RELATION is done in a similar manner; i.e., by typing "WEAPON: = RELATION" or "TOP SPEED: = NUMBER RELATION".

After creating the RELATION or CLASS, the user may add information to it; i.e., add a new tuple, by typing the new data in a sentence format; i.e., "US IS NATIONALITY OF NIMITZ.", "35 IS TOP SPEED OF THE NIMITZ.", or "NIMITZ IS CVN." To do such data entry, the words "NIMITZ" and "US" must already have been identified to the system by defining them as NAMES. Relations and classes can be added to, but they cannot be updated; i.e., a tuple changed; nor can tuples be deleted.

Data can also be entered in an automatic manner, with the same syntax as above, by using the Interface Processor as a means of sending it to the Query Processor, or by sending it to the Data Processor using Data Processor semantic program commands directly.

III. EXTENSIBILITY

There are several types of extensions, or definitional facilities, which will be available for the commander to easily individualize his data description. The first of these, creation of NAMES, CLASSES and RELATIONS and insertion of data values into CLASSES and RELATIONS has been discussed. The creation of primitive relations allows data to be grouped into meaningful relationships. For instance, by creating a CLASS consisting of the ship classes of a specific type; i.e.,

CVN: = CLASS

CGN: = CLASS

VIRGINIA: = NAME

NIMITZ: = NAME

VIRGINIA IS CGN.

NIMITZ IS CVN.

the user can record the concept of "ships of a type" permanently. In a command center, this power to abstract concepts in the data can be particularly useful for dynamic formulation of flights by missions, categorization of potential threat, etc.

The second type of extensibility provided is the ability to uniformly add new data sources for dynamically updating the CCIS data base. The CCIS Interface Processor (IP) provides specialized hardware interfaces and matching software for translation of data from one or more non-CCIS sources into a CCIS compatible format. It then passes data it has translated to the Data Processor, which is responsible for all data maintenance, including addition of dynamically changing data to its database.

The executive of the IP accepts real time data as the processor is interrupted and when not busy with real time processing requests "batch data" for reading. The format of that dynamic input data is sent to the IP from external sources before input records are sent (in the CCIS hardware model constructed the sources are either the ARPANET or the CCIS Tester Box which send data over standard RS232 serial ASCII lines). The Format record defines whether input fields are alphanumeric or numeric and the relations to which the data is to be added. The resultant external input records then are reformatted by the IP for consumption by the DP. The IP also sends the current time as declared in intervals specified by the Time Report record.

All names used in format descriptions and in data entry to the IP must be predefined in the static data base structure of the CCIS Data Processor database and in the Query Processor lexicon using the CREATE commands of the data model. Dead Reckon software is provided in the Data Processor to project current position to current time using the most recent data known about any dynamic track. Use of this technique combined with the IP policy of only sending "significant changes" to the database for simulated real time NTDS tracks minimizes the amount of storage required to maintain current dynamic data ("significant changes" for the simulation are changes in COURSE, SPEED or track TYPE).

The third type of extensibility envisioned, but not yet implemented, is that of query extension. These extensions include the user being able to define synonyms and functions (with variables) which are then added to the language by the Query Processor. Addition of synonyms is straightforward. For instance, it might be more convenient in many cases to use the identifier EISENHOWER than it is the entire name, DWIGHT D. EISENHOWER. The system will support defining EISENHOWER to mean the same as DWIGHT D. EISENHOWER. The user simply types:

DEF: EISENHOWER: DWIGHT D. EISENHOWER

where DWIGHT D. EISENHOWER has already been created as a NAME.

Function definition (with variables) is very similar to the capability provided in the Rapidly Extensible Language System (REL) [7]. A function has variables, as does a mathematical function. A concept can be defined using input variables (enclosed in " " marks) which can take the value of any noun phrase(s) known to the system. Consider the following definition:

DEF: RUSSIAN "SHIP": "SHIP" WHICH HAS USSR NATIONALITY

Then one may ask, "WHAT ARE RUSSIAN BOATS?" where BOATS becomes the value of "SHIP". In order to use such a definition, all vocabulary items must have been previously defined; i.e., here SHIP was already defined as a CLASS, USSR defined as a NAME, and NATIONALITY defined as a RELATION. The system then links the given noun, BOAT, with the definition RUSSIAN, even though BOAT has not actually been used in the definition; i.e., BOAT has only been used as follows: SSBN IS BOAT. DELTA III IS BOAT. USSR IS NATIONALITY OF DELTA III.

IV. QUERY TYPES

The primary method of querying the data base is through a limited form of English. The queries must pertain to data of which the system has knowledge. Only first order predicate sentences are permitted with the exception of "What are classes?" and "What are relations?" These last two questions are trapped with a string recognition. The system uses a limited set of quantifiers and relative subordinate clauses. It does not include queries involving

"where", "when" (except in the limited context of "current position" and the printing of current time or report time when meaningful), "how many", conjunctives (except in the limited use for distance and intercept time functions) and pronouns. Also, it cannot answer questions where a hidden relation is implied but not explicitly stated; such as, "WHAT IS LIERMAN'S SHIP?" where Lierman is the commander of Kennedy which is a ship. In order to ask this question the system would have to have the relation commander explicit in the query; i.e., "WHAT IS THE SHIP WHOSE COMMANDER IS LIERMAN?".

USE OF ARTICLES, PLURALS, AND THE VERBS "HAS" AND "IS":

1. THE – Can be used at any time, where appropriate.
2. Plurals are made by adding "S". Adding "ES" is not allowed.
3. IS and ARE are interchangeable, depending on the grammar:

   WHAT ARE WEAPONS?
   WHAT IS THE KNOX?
4. HAS and HAVE are interchangeable, depending on the grammar:

   WHAT ARE WEAPONS OF SHIPS WHICH HAVE USSR NATIONALITY?
   WHAT ARE WEAPONS OF EACH SHIP WHICH HAS USSR NATIONALITY?
5. HAVE without ARE and WHICH:

   WHAT SHIPS HAVE USSR NATIONALITY?

PREDEFINED QUERIES:

1. WHAT ARE CLASSES?
2. WHAT ARE RELATIONS?

These two queries must be typed as shown.

SIMPLE QUERIES:

1. WHAT ARE "CLASS name"?; i.e., WHAT ARE AIRCRAFT?, will give all entries in the CLASS "AIRCRAFT".
2. WHAT ARE "RELATION name"?; i.e., WHAT ARE SONAR?, will give both columns of all entries in the relation "SONAR".
3. WHAT IS "NAME"?; i.e., WHAT IS KNOX?, will give the CLASSES and RELATIONS in which "KNOX" is an actual entry.

BASIC QUERY TYPES – ARE . . . OF

ARE . . . OF confines the answer to one column of the relation, as the second column is given by the "OF" phrase, "WHAT ARE THE WEAPONS OF THE KNOX?" yields only the weapon NAMES; i.e. SH-2 SEASPRITE and MK25 BPDMS for the KNOX.

1. WHAT ARE "RELATION" OF "NAME"? (WHAT ARE WEAPONS OF KNOX?).
2. WHAT ARE "RELATION" OF "CLASS"? (WHAT ARE WEAPONS OF FFS?) This type of query utilizes the system's ability to join information from different relations and classes in order to satisfy a query. This can also be seen in the examples which follow.

USE OF "EACH":

"EACH" can be used as a quantifier preceding the name of a CLASS or RELATION to cause the information for that CLASS or RELATION to be printed in tabular format.

1. WHAT ARE "RELATION" OF EACH "CLASS"? (WHAT ARE WEAPONS OF EACH FF?)
   This type of query will expand the CLASS (FF) down to the shipclass NAMES and will print both the weapon NAME and the shipclass NAME; i.e., "SH-2 SEASPRITE" and "GARCIA-C"; for each of the shipclass NAMEs. EACH can also be used as a multiple quantifier to force extra columns to be printed.
2. WHAT IS THE "RELATION" OF EACH "RELATION" OF EACH "CLASS"? (WHAT IS THE RANGE OF EACH WEAPON OF EACH FF?)
   The answer to this query will print out the range, weapon NAME, and shipclass NAME. The user can continue to add RELATIONS as long as the relations have elements in common. For instance, WHAT IS RANGE OF EACH WEAPON OF EACH RADAR . . . makes no sense, but WHAT IS RANGE OF EACH WEAPON OF EACH WEAPON . . . does, as SPARROW is a WEAPON of F-14 TOMCAT which is a WEAPON of the NIMITZ-C.

CLASS AND NAME USED AS LIMITERS TO RESTRICT A RELATION:

1. WHAT ARE "CLASS" "RELATION"? (WHAT ARE AIRCRAFT WEAPONS?)
   In this example, AIRCRAFT limits the RELATION "WEAPON" to those WEAPONS that belong to the CLASS "AIRCRAFT". Both the aircraft NAME and the shipclass NAME it belongs to are printed.
2. WHAT ARE "NAME" "RELATION"? (WHAT ARE USSR NATIONALITY?)
   This query would give all items in the given RELATION which have USSR as an entry. Both columns of the RELATION are printed.

USE OF "WHOSE", "WHICH", "THAT":

The subordinate relative clause constructions "ARE. . . WHOSE", "ARE . . . WHICH HAVE", and "ARE . . . THAT HAVE", act to restrict the relation as a whole to the conditions in the "WHOSE", "WHICH", and "THAT" parts. The result is that both columns of the target relation are printed.

1. "WHOSE" links the condition phrase to the CLASS or RELATION immediately preceding it. There is no other way for the system to interpret the query. For example, "WHAT ARE WEAPONS OF SHIPS WHOSE RANGES EXCEED 60?" asks for "SHIPS with RANGE greater than 60; WEAPONS OF".
2. "WHICH HAVE" and "THAT HAVE" are interpreted in the same way as "WHOSE", UNLESS A COMMA IS INCLUDED. Inclusion of a comma causes the system to process and give answers to ALL POSSIBLE interpretations. For example:
   - A WHAT ARE WEAPONS OF SHIPS WHICH HAVE RANGE GREATER THAN 60? is interpreted as in 1) above. (Note that "IS" is omitted from the comparative when used with "WHICH HAVE").
   - B WHAT ARE WEAPONS OF SHIPS, WHICH HAVE RANGE GREATER THAN 60? will have answers for both interpretations, below:
     1 SHIPS which have RANGE greater than 60; WEAPONS of.
     2 WEAPONS which have RANGE greater than 60, which are carried on SHIPS.

USE OF COMPARATIVES (USING "WHOSE"):

The comparatives are:

1. (IS) GREATER THAN
2. (IS) LESS THAN
3. (IS) EQUAL TO
4. (IS) GREATER THAN OR EQUAL TO
5. (IS) LESS THAN OR EQUAL TO
6. (IS) AS GREAT AS (BIG, LARGE, etc.)
7. (IS) AS LITTLE AS (SMALL, etc.)
8. IS (To mean EQUAL TO)
9. EXCEED (Note: "EXCEEDS" is not allowed)
10. (IS) >
11. (IS) <

The word "IS" must be used when using "WHOSE", unless using the comparative EXCEED.

1. WHAT ARE "RELATION" WHOSE "NUMBER RELATION" "COMPARATIVE" "NUMBER"? (WHAT ARE WEAPONS WHOSE RANGE IS GREATER THAN 6Ø?)

   The user can also use "CLASS" or "CLASS" "RELATION" in place of "RELATION".

2. WHAT ARE ("RELATION", "CLASS", or "CLASS" "RELATION") WHOSE "NUMBER RELATION 1" "COMPARATIVE" "NUMBER RELATION 1" OF "NAME"? (WHAT ARE WEAPONS WHOSE RANGES EXCEED THE RANGE OF THE PHOENIX?)

PUTTING IT ALL TOGETHER:

WHAT IS THE RANGE OF EACH WEAPON OF EACH AIRCRAFT WEAPON WHOSE TOP SPEED IS GREATER THAN THE TOP SPEED OF THE FORGER?

NON-STANDARD QUERY TYPES:

There are four non-standard query types. Three of them involve the functions CURRENT POSITION, DISTANCE, and INTERCEPT TIME. The functions use only dynamic data sent from the IP (NTDS and INTEL Track data) as explained in the section on Extensibility. The other one is a constraint on the use of relations where the quantity field is filled. WEAPON STRENGTH is an example of a special-purpose RELATION which has the quantity field filled for the number of weapons on each platform in the data base. There is no syntax or semantics for manipulating this quantity field other than asking "WHAT IS WEAPON STRENGTH OF 'CLASS' or 'NAME'?" All data for filling the relation is entered via the IP using DP semantic language.

The Current Position function is used to find the geographical location of a given Track at the current time (current time is determined by the system's clock located in the IP). The function takes the latitude, longitude, course, speed and the time of the last report for a given Track, and extrapolates using dead reckoning to find the estimated current position of the track. The system will accept only the following syntax:

WHAT IS (THE) CURRENT POSITION OF NTDSTN 1?

where "NTDSTN 1" is the name of the desired track. Words in parentheses are optional.

or

WHAT IS (THE) CURRENT POSITION OF (EACH) NTDS/TRACK?

where NTDS/TRACK is a CLASS containing the NAMES of dynamic data Tracks (NTDSTN 1, etc.).

The Distance function accepts the names of two tracks as its arguments in a limited conjunctive form. It first computes the present position of each given track, and then finds the distance (using great circle distance computations), in nautical miles, between the two positions. The query syntax for this function is:

WHAT IS (THE) DISTANCE BETWEEN INTELTN 1 AND NTDSTN 4?

where "INTELTN 1" is one desired track and "NTDSTN 4" is the other desired track.

or

WHAT IS (THE) DISTANCE BETWEEN (EACH) INTEL/TRACK AND NTDSTN 4?

The INTERCEPT function accepts two track NAMES, as does DISTANCE. The first Track is the one doing the intercepting, and the second Track is the target.

The system replies with the time (in minutes to the tenth) needed for the pursuing Track (first Track) to reach the intercept point, and the coordinates for the interception point in degrees, minutes, and seconds (to the tenth second). Longitude is reported first, then latitude.

WHAT IS (THE) INTERCEPT TIME BETWEEN NTDSTN 1 AND NTDSTN 5?

or

WHAT IS (THE) INTERCEPT TIME BETWEEN (EACH) NTDS/TRACK AND NTDSTN 1?

In all of these special functions current time will be reported out as the "begin time" of the resultant tuple.

V. USER AIDS

CCIS is designed to be a friendly system. There are a number of editing features available for correcting or editing queries which correspond closely to the ARPANET XED text editing functions. Certain commands can be utilized during the normal (input) mode. These include minor editing commands and parsing and print related commands. To invoke more extensive editing commands, the user must enter the edit mode by typing a control E (^E). The system then positions the cursor at the beginning of the current query. The available subcommands are as follows:

User types	Result
	EDIT MODE COMMANDS
?	If typed following ^E, prints out edit instructions.
n SPACE	Spaces n characters forward copying line as is.
B	Break Line – Inserts a carriage return into new line. Edit subcommands may be continued with rest of the old line. Note: the rest of the old line is not shown on the screen.
n D	Deletes n characters FORWARD from the old copy.
E	Moves cursor to end of line copying line as is.
I	Insert mode – all input is inserted until next CR or LF.
n K x	Deletes FORWARD from present cursor position until the nth occurrence of "x". If "x" is a CR, it means the end of the line.
P	Prints the rest of the old line and the current new line.
n R	Deletes next n characters forward from cursor and enters insert mode.
n S x	Spaces forward to before the nth occurrence of "x". If "x" is a CR, it means the end of the line.
CR	Move to end of query, terminate Edit and enter Input mode.
LF	Deletes remainder of line after the cursor and enters Input mode.
U	Same as CR but cannot be used to terminate insertion commands such as I or nR.
ESCAPE	Copies rest of old line to new, updates current line to new, and re-enters edit mode on the updated current line. Abort all changes so far, and start edit over.
	INPUT MODE COMMANDS
^O	Temporarily suspends output. Restart by typing any character.
^D	Resets the system to start over on all queries of the database the user is in.
^U	Delays parsing (processing) until after editing. Keeps system from parsing an erroneous expression before editing.
^B	Forces the beginning of parsing.
^N	Does not allow system to begin parsing until ^B is typed.
^P	Causes the output to also be printed on the T1700 terminal.
^S	Turns off the effect of ^P (stops printout).
	EITHER MODE
^A	Back spaces over one character (acts as a Delete to the LEFT)
^H	Same as ^A.
^W	Backs over (deletes) the last word that the user typed, or the word to the LEFT of the cursor's current position.
^X	Deletes to the beginning of the current line. In INPUT mode, the system then re-enters INPUT. In EDIT mode, it waits for the user to type another EDIT mode command.

^R Causes the print of the new expression (reprints the current query, including all changes).

^Q Reset system to start over on input, killing any old output.

LEXICAL ASSISTANCE

Lexical assistance is provided in the INPUT MODE using the ESC (escape) command as a means of causing the lexicon to be searched for the longest match between the lexicon and the last characters of the query. This assistance, together with the aforementioned editing features, is provided as an efficient (from a computational point of view) alternative to automated spelling correction.

VI. RESPONSE FORM

When ambiguous situations arise, all alternatives are considered. If one survives to form a non-trivial output it is output as the result. When there is more than one such output, each is displayed. The user can obtain more information about these answers by asking simpler queries.

All outputs will be alphanumeric with the capability of receiving the output on a CRT screen or as a printout.

The normal procedure followed for printing out information is for the system to output the information in columnar form. If one of the spellings of a word in a row exceeds the size for the column that column is readjusted to have a larger size. The output form is principally that of row column matrix. If the row exceeds the width of the screen it wraps around. An extra carriage return is added to keep rows of data distinguished from wrap arounds. The end of the answer is signified by printing a series of dots below the last line of the answer; i.e., ". . . . . ". However, there are other responses which the system gives when confronted with non-standard situations:

1. In the case where there is no information of the type requested, the system will simply output the end of response indicator, ". . . . . ".
2. In the case of ambiguous queries such as those using the ". . . , WHICH HAVE" phrasing, there will be MORE THAN ONE REPLY. If all replies have answers, the information for each answer will be terminated by the ". . . . . " message. If any (or all) of the answers are null (no information in the database), EACH null answer will be signified by the ". . . . . " message.
3. If the Query Processor encounters words in the query which have not been defined in the QP lexicon, it will terminate processing of the query, and output the message "WHAT?" to the user. The user may then retype the query, using correctly spelled words which are known to the lexicon, or type in another query.
4. Occasionally, all the words contained in a query will be contained in the QP lexicon, but the phrasing of the query is not interpretable by the QP. In this case, the QP simply ignores the query, as it considers it to be invalid, and will wait for the user to type a valid query. However, there is no message to inform the user that this is the case. Therefore, if a considerable delay in response time is encountered, the user may assume that his query was ignored, and retype the question, using a different phraseology. If, in fact, the delay was due to extended processing time, the user should obtain the system's response before he finishes retyping the query. If the system is still parsing his previous query, any new input typed by the user will not be echoed on the CRT screen.

VII. SYSTEM ARCHITECTURE

The CCIS system consists of a Query Processor (QP), a Data Processor (DP), a buffer memory, bulk storage and an Interface Processor (IP). All data flow between processors is low speed (1200 baud in the CCIS demonstration system), ASCII coded, serial interface. Data-flow control between the Query Processor, Interface Processor and buffer memory is exercised by the Data Processor, the principal function of which is data manipulation. The buffer memory provides buffering between bulk storage (currently a disk memory) and the Data Processor.

The processing organization is developed as follows: principal processing of the input user statements, which are in a form of extensible English, is performed by the Query Processor which provides the dialog with the user and parses the English statements. The Query Processor translates the parsed statements into an action sequence which, in turn, drives the Data Processor in its manipulation of relations and updates the dictionary of vocabulary words and relations stored in the Query Processor's and Data Processor's lexicons.

The Data Processor has control over the disk data management, stores and retrieves data as needed by the Query Processor, controls the loading of data from the Interface Processor and loads the programs and data structures for both the Query and Interface Processors. Functions are available for relational data management such as ad hoc creation of relations, addition of data tuples to them, joins, class intersections and restricts. Special functions which provide the computational features for printing current position and calculating distance and intercept time are provided for dynamic data as are those routines necessary for adding to those relations.

In operation, the principal function of the Query Processor is to interface with the user of the data base. This is accomplished by providing a language to the user that is nearly English and by providing editing features. The Query Processor takes a statement given it by the user, converts it to Data Processor control language, and sends it to the Data Processor. A sequence of action statements is transmitted to the Data Processor where their execution provides a final result, such as one or more formatted relations. The resulting record is transmitted to the Query Processor for display. In this manner, query translation and command execution are carried out by separate processors in a manner conceptually similar to the Bell Telephone Laboratories' front-end back-end processor data management system [1]. The Interface Processor operates like a data transformation unit. Only a simple syntax is processed by this processor and its principal functions are to recognize data structures of the retrieval system and to relate the incoming data with the appropriate records in the retrieval system. Finally it sends the data to the Data Processor for final disposition.

The functional or logical structure of the system is shown in Table 1.

TABLE 1. CCIS Functions.

Common Modules

1. Linkage Table
2. Executive Module
3. Activity Tree
4. Buffer Control Module
5. Buffer Control Blocks
6. Communication and I/O Module
7. Processor Instructions

Query Processor (QP) Modules

1. Protocol Module
2. Lexicon Module
3. The Lexicon
4. Utility Module
5. English Parsing Module
6. The Parsing Dictionary
7. CND Functions Module
8. Semantic Processing Module
9. Miscellaneous Pieces

Data Processor (DP) Modules

1. Compiler Handler
2. Relational System Module
3. Output Module
4. Logical File Management System
5. Bulk Memory Page Management
6. Buffer and Local Memory Management
7. Auxiliary Memory System

Interface Processor (IP) Modules

1. NTDS and INTEL Data Formatting
2. IP Executive
3. Current Time Generation

Processor-Processor Communications

VII.1 Common Modules

Each of the various modules functionally works together to provide the overall functions of the Query Processor and Data Processor. The linkage table provides a way for one module to call functions of another module. The executive module provides the passage of program control from one module to another. It uses the activity tree to keep track of how the modules are to be executed. When data must be passed from one module to another the buffer control module provides a means of symbolically sending data from one module to another. The buffer control blocks characterize the channels between the modules as well as keep track of where the data is located. Some modules use the same mechanisms internally also. The communication module provides mechanisms for sending and receiving data to and from other processors, terminals, and communication lines. These modules described so far are general system modules that are not specifically unique to one processor of the CCIS system.

The software of the Query and Data Processors is composed of a number of modules which interact at several levels. In compiling, some modules tell other modules how to convert expressions to code. Others tell how to link the programs of one module into another. Each module has a set of parameters which keep track of the progress of the module as well as tell where data is from other modules. Each module can have a series of programs called activities which interact with the executive module. The programsare free of interaction otherwise except by the three methods: compiler macros, linkage table, and parameter table of each module. Some of the modules are data modules which are not programs but data structures used within the system. Normally these will not be called modules in the table of contents or parts headings. The interaction with these is still through the parameter tables. The exception to this is the buffer control blocks wherein locations of programs from the modules defining and using them are required.

VII.1.1 Linkage Table

The linkage table is designed primarily to allow one module to call programs in another module without prior knowledge of the location of the called programs. There are two aspects of the linkage table. First, in the last stage of assembling a module a file is read that tells where in the QP or DP each link reservation of each program is located. Then, when a new module is loaded into the QP or DP the addresses of programs called by other modules have to be entered into the copy of the linkage table that is to be loaded into the QP or DP.

QP Linkage Table Functions (including common modules)

FUNCTION NAME	HEX	FILE
EXIT	0040	EXEC
QUIT	0041	EXEC
THEN	0042	EXEC
ELSE	0043	EXEC
BOTH	0044	EXEC
THEN/QUIT	0045	EXEC
ELSE/QUIT	0046	EXEC
BOTH/QUIT	0047	EXEC
ACTIVATE	0048	EXEC
SET/QUIT/FLG	0049	EXEC
SET/TO/DEPTH/SCAN	004A	EXEC
LAST/ACT	004B	EXEC
CURRENT/ACT	004C	EXEC
NEXT/ACT	004D	EXEC
CONNECT/BRANCH	004E	EXEC
GET/NUMBER	004F	Not Used
DISTANCE	0050	Not Used
COMPARELINK	0051	Not Used

ACTIVITYVALUES	0052	EXEC
ACT/CONS	0053	EXEC
GET /FAST/TAIL	0054	EXEC
GET/SLOW/TAIL	0055	EXEC
GET/SLOW/OF/CONSTRUCTION	0056	EXEC
GET/TAIL/OF/CURRENT	0057	EXEC
LOCKSTACK	0058	EXEC
RETURN/FROM/SUSPEND	0059	EXEC
SETACTIVITYVALUES	005A	EXEC
SET/CONSTRUCTION	005B	EXEC
SUSPENDF	005C	EXEC
SET/TO/WIDTH/SCAN	005D	EXEC
SETSCANTOEND	005E	BUFF
SCANBACKWARD	005F	BUFF
ADVTOSCAN	0060	BUFF
BACKUPSTORE	0061	BUFF
FULLP	0062	BUFF
EMPTYP	0063	BUFF
BACKUPREAD	0064	BUFF
RESTORESCANPTR	0065	BUFF
GETSCANPTR	0066	BUFF
SCANTESTOF	0067	BUFF
SCANNEXT	0068	BUFF
SCAN	0069	BUFF
SETSCANOF	006A	BUFF
READSTORED	006B	BUFF
READNEXT	006C	BUFF
READLAST	006D	BUFF
READFROM	006E	BUFF
WRITETO	006F	BUFF
CLEARBUFFER	0070	BUFF
STUFFBYTECODE	0071	BUFF
GETBYTECODE	0072	BUFF
CLEARGRAPHICS	0073	
SCANTESTEND	0074	BUFF
SCANTESTBNG	0075	BUFF

TELL/BUFFERCODE	0076	BUFF
CONS	0077	UTIL
DIFFLIST	0078	UTIL
GETASCNU	0079	UTIL
GETNEWCELL	007A	UTIL
GIVEUPCELL	007B	UTIL
GIVEUPTREE	007C	UTIL
INTERSECTION2	007D	UTIL
LDIFF	007E	UTIL
LENGTH	007F	UTIL
MAKENUMBER	0080	UTIL
NCONC2	0081	UTIL
TCONC	0082	UTIL
TCONCER	0083	UTIL
UNION2	0084	UTIL
SENDLEXWORD	0085	LEX
GETINTRPBYTE	0086	BUFF
GETFEATBITS	0087	RULE
ENTER/A/WORD	0088	LEX
SEARCHPOS	0089	SEM
RESETLEXICON	008A	LEX
LINKGARBAGE	008B	UTIL
SAVECHKBITS	008C	RULE
PARSEIT	008D	RULE
SAVETREE	008E	SEM
COPYTR	008F	SEM
INSTALLTREE	0090	SEM

Abbreviations (All file names abbreviated herein are accessible under the account name SMALL on ISIC on the ARPANET.)

EXEC	QPEXEC.;57
BUFF	BUFFERHANDLER.;60
COMM	COMMUNICATIONMODULE.;55
PROT	USERPROTOCOL.;202
LEX	USERLEXICON.;149
UTIL	QPSMUTIL.;26
RULE	SMRULEHANDLER.;42
CND	QPSMCNDFUNCTIONS.;48
SEM	QPSMSEMANTICS.;153

DP Linkage Table Functions (including common modules)

EXIT	0040	EXEC
QUIT	0041	EXEC
THEN	0042	EXEC
ELSE	0043	EXEC
BOTH	0044	EXEC
THEN/QUIT	0045	EXEC
ELSE/QUIT	0046	EXEC
BOTH/QUIT	0047	EXEC
ABORT	0048	EXEC
ACTIVATE	0049	EXEC
SET/QUIT/FLG	004A	EXEC
SET/TO/DEPTH/SCAN	004B	EXEC
LAST/ACT	004C	EXEC
CURRENT/ACT	004D	EXEC
NEXT/ACT	004E	EXEC
CONNECT/BRANCH	004F	EXEC
CREATE/NEW/PAGE/DIRECTORY/ ENTRY/FOR/LOGICAL/RECORD	0050	PAGM
CREATE/FIRST/PAGE/DIRECTORY/ PAGE	0051	PAGM
FETCH/EMPTY/DISK/LOCATION	0052	Not Used
LOCK/A/PAGE	0053	PMGT
UNLOCK/A/PAGE	0054	PMGT
GET/PAGE	0055	PMGT
CREATE/NEW/NAME	0056	CATM
CREATE/NEW/LOGICAL/RECORD	0057	CATM
FIND/CATALOG/ENTRY	0058	CATM
INSERT/PAGE/POINTER	0059	Not Used
UNPACK/BUFFER	005A	Not Used
CREATE/IND/CLASS	005B	Not Used
RESET/BUFFER	005C	Not Used
OUTPUT/PARAM	005D	Not Used
LEXICON/SEARCH	005E	Not Used
FIND/DIRECTORY/LOC/OF/ ELEMENT/POINTER	005F	DIRF
FIND/NEXT/PAGE/POINTER	0060	DIRF

COPY/HDR	0061	DIRF
FIND/AREA/TYPE	0062	DIRF
FIND/ROOT/PAGE/ID	0063	DIRF
UPDATE/A/RECORD	0064	PAGM
GET/NEW/PAGE	0065	PMGT
LEXICON	0066	LEXI
COPY/INPUT/BUFFER	0067	Not Used
CREATE	0068	Not Used
UPDTE/REC	0069	Not Used
GET/NEW/DISK/PAGE	006A	PMGT
TEST/PAGE IN/CORE	006B	PMGT
CHANGE/PAGE	006C	PMGT
GET/NEW CORE/PAGE	006D	PMGT
NAME/CORE/PAGE	006E	PMGT
DELETE PAGE	006F	PMGT
COPY/PAGE	0070	PMGT
NO/OF/AVAIL/CORE/PAGES	0071	PMGT
CLEAR INPUT/BUFFER	0072	Not Used
JOIN	0073	Not Used
DISCONNECT	0074	EXEC
PROJECT	0075	Not Used
RESTRICT	0076	Not Used
GET/NUMBER	0077	Not Used
DISTANCE	0078	COMH
COMPARELINK	0079	Not Used
ACTIVITYVALUES	007A	EXEC
ACT/CONS	007B	EXEC
GET/FAST/TAIL	007C	EXEC
GET/SLOW/TAIL	007D	EXEC
GET/TAIL/OF/CONSTRUCTION	007E	EXEC
GET/TAIL/OF/CURRENT	007F	EXEC
LOCKSTACK	0080	EXEC
RETURN/FROM/SUSPEND	0081	EXEC
SETACTIVITYVALUES	0082	EXEC
SET/CONSTRUCTION	0083	EXEC

SUSPENDF	0084	EXEC
SET/TO/WIDTH/SCAN	0085	EXEC
TEST/PAGE/IN/CHANGE/MODE	0086	PMGT
NO/OF/AVAIL/DISK/PAGES	0087	PMGT
MAX/NO/OF/AVAIL/CONT/CORE/PAGES	0088	PMGT
GET/N/NEW/CONT/CORE/PAGES	0089	PMGT
REL/N/CONT/CORE/PAGES	008A	PMGT
UNPACKLEXICON1	008B	LEXI
MAKENUMBER	008C	Not Used
UNPACKNUMBER1	008D	COMH
DELETE/LOGICAL/RECORD	008E	CATM
DELETE/CAT/PAGE	008F	DIRF
DWRITETO	0090	Not Used
DREADFROM	0091	Not Used
CREATE/CHANNEL	0092	Not Used
FINIS	0093	Not Used
SCANTESTBNG	0094	BUFF
SCANTESTEND	0095	BUFF
SETSCANTOEND	0096	BUFF
SCANBACKWARD	0097	BUFF
ADVTOSCAN	0098	BUFF
BACKUPSTORE	0099	BUFF
FULLP	009A	BUFF
EMPTYP	009B	BUFF
BACKUPREAD	009C	BUFF
RESTORESCANPTR	009D	BUFF
GETSCANPTR	009E	BUFF
SCANTESTOF	009F	BUFF
SCANNEXT	00A0	BUFF
SCAN	00A1	BUFF
SETSCANOF	00A2	BUFF
READSTORED	00A3	BUFF
READNEXT	00A4	BUFF
READLAST	00A5	BUFF
READFROM	00A6	BUFF
WRITETO	00A7	BUFF

CLEARBUFFER	ØØA8	BUFF
GETBYTECODE	ØØA9	BUFF
STUFFBYTECODE	ØØAA	BUFF
DEMPTYP	ØØAB	Not Used
READ	ØØAC	Not Used
WRITE	ØØAD	Not Used
NOT/USED	ØØAE	Not Used
TELL/BUFFERCODE	ØØAF	BUFF
CONTROLCHAR	ØØBØ	Not Used
CAT/GARBAGE/COLLECT	ØØB1	CATM
FIND/LAST/PAGE/OF/RECORD	ØØB2	DIRF
FIND/LENGTH/PAGES	ØØB3	DIRF
CREATE/RELATION	ØØB4	RELM
ADD/PAGE/TO/RELATION	ØØB5	RELM
FIND/SPACE/UTILIZED/ON/PAGE	ØØB6	RELM
FIND/NEXT/PAGE/FROM/HDR	ØØB7	RELM
FIND/PSEUDO/RECORD/NAME	ØØB8	RELM
FIND/TUPLE/SIZE	ØØB9	RELM
ADD/TO/SPACE/UTILIZED	ØØBA	RELM
FIND/PREVIOUS/PAGE/FROM/HDR	ØØBB	RELM
FIND/START/OF/USER/PAGE/STRING	ØØBC	RELM
FIND/INITIAL/LOCATION/OF/LRN	ØØBD	RELM
GET/CURRENT/TIME	ØØBE	Not Used
OPEN/LRN	ØØBF	LGFL
CLOSE/LRN	ØØCØ	LGFL
READ/TUPLE	ØØC1	LGFL
WRITE/TUPLE	ØØC2	LGFL
MULTIPLY	ØØC3	ART1
DIVIDE	ØØC4	ART1
CIRCLEDISTANCE	ØØC5	ART2
DEADRECKON	ØØC6	ART1
NEGCONV	ØØC7	ART1
CONVERT1	ØØC8	ART1
SINECOMP	ØØC9	ART1
TCONVERT	ØØCA	ART1
DELTAT	ØØCB	ART1
INTERCEPT	ØØCC	ART3
CLEAR/PAGE	ØØCD	CLPG
RENDEZVOUS	ØØCE	Not Used

Table of Abbreviations (All file names whose abbreviations are expanded below are found under the account SMALL on ISIC on the ARPANET)

CLPG	DANACLEARPAGE.;2
CATM	CATMANAGE7.;7
PAGM	PAGEMAN1.; 121
DIRF	BRDNFUN.; 113
LEXI	DPLEXICONNOPAGED.; 4
RELM	CREATEN-COL.; 47
EXEC	COPYEXEC.; 67
COMM	DPCOMMUNICATIONMODULE.; 22
BUFF	BUFFERHANDLER.; 60
COMH	COMPILERHANDLER.; 167
ART1	BETH.NEWPOSITION; 1
ART2	BETH.DISTANCE; 6
ART3	BETH.NEWINTERCEPT; 11
LGFL	GOK.LOGICALFILE; PAPERNOTES
PMGT	GOK/PAGEMAN.CR; 1

VII.1.2 Executive module

The executive module is a collection of programs which allow for the control of machine code execution to proceed in a manner that avoids excessive entanglement of the logically formulated modules of the system. Thus by calling the function EXIT one leaves the current modules and the executive determines the next module to be called. The current module does not need to know what that next module is. Further when a module calls SUSPEND the action of the current module is interrupted until the condition specified by the suspend is met at which time the action of the module is continued where it left off. Thus a single module may have several programs, or activities, that can interact with the executive. Each activity has a block on the activity tree with an activating location.

Location of Executive Functions in Linkage Table

QP LOCATIONS

EXIT	0040	EXEC
QUIT	0041	EXEC
THEN	0042	EXEC
ELSE	0043	EXEC
BOTH	0044	EXEC
THEN/QUIT	0045	EXEC

ELSE/QUIT	0046	EXEC
BOTH/QUIT	0047	EXEC
ACTIVATE	0048	EXEC
SET/QUIT/FLG	0049	EXEC
SET/TO/DEPTH/SCAN	004A	EXEC
LAST/ACT	004B	EXEC
CURRENT/ACT	004C	EXEC
NEXT/ACT	004D	EXEC
CONNECT/BRANCH	004E	EXEC
ACTIVITYVALUES	0052	EXEC
ACT/CONS	0053	EXEC
GET/FAST/TAIL	0054	EXEC
GET/SLOW/TAIL	0055	EXEC
GET/TAIL/OF/CONSTRUCTION	0056	EXEC
GET/TAIL/OF/CURRENT	0057	EXEC
LOCKSTACK	0058	EXEC
RETURN/FROM/SUSPEND	0059	EXEC
SETACTIVITYVALUES	005A	EXEC
SET/CONSTRUCTION	005B	EXEC
SUSPENDF	005C	EXEC
SET/TO/WIDTH/SCAN	005D	EXEC
DP LOCATIONS		
EXIT	0040	EXEC
QUIT	0041	EXEC
THEN	0042	EXEC
ELSE	0043	EXEC
BOTH	0044	EXEC
THEN/QUIT	0045	EXEC
ELSE/QUIT	0046	EXEC
BOTH/QUIT	0047	EXEC
ABORT	0048	EXEC
ACTIVATE	0049	EXEC
SET/QUIT/FLG	004A	EXEC
SET/TO/DEPTH/SCAN	004B	EXEC
LAST/ACT	004C	EXEC
CURRENT/ACT	004D	EXEC

NEXT/ACT	004E	EXEC
CONNECT/BRANCH	004F	EXEC
DISCONNECT	0074	EXEC
ACTIVITYVALUES	007A	EXEC
ACT/CONS	007B	EXEC
GET/FAST/TAIL	007C	EXEC
GET/SLOW/TAIL	007D	EXEC
GET/TAIL/OF/CONSTRUCTION	007E	EXEC
GET/TAIL/OF/CURRENT	007F	EXEC
LOCKSTACK	0080	EXEC
RETURN/FROM/SUSPEND	0081	EXEC
SETACTIVITYVALUES	0082	EXEC
SET/CONSTRUCTION	0083	EXEC
SUSPENDF	0084	EXEC
SET/TO/WIDTH/SCAN	0085	EXEC

Activity Functions

FUNCTIONS	FILE NAME
EXIT	EXEC

VII.1.3 Activity Tree

The activity tree is a data structure that contains a block of eight words for each activity. In this block are pointers to other blocks on the tree. The tree has essentially three roots: the fast tree, the slow tree, and the construction tree. Each block has two subsidiary blocks which may be executed depending on the conditions of how the current block returns to the executive. In addition it points to one additional block where the normal continuation of the tree is found. Between the execution of each accessible block of the slow activity tree the executive scans through every accessible block of the fast activity tree. Each block also points to a location containing the address of the code to be executed by this activity. This address table points to the addresses of the executive programs of each module. Lastly, each block contains four words that can be used by the activity program to pass data from one activation to the next of the same activity.

Activity Functions

FUNCTIONS		
MOVEBUFFERIF	COMM	test semantic interlock
PRINTLEXGROUPS	LEX	
SEMANTICDRIVER	SEM	

LEXITDRIVER	PROT	
QUERYDRIVER	PROT	
READDATA	COMM	
PARSEDRIVER	PROT	in protocol
MOVEBUFFER	COMM	
PARSEDRIVER	RULE	in parsing
WRITEDATA	COMM	
READINBUFFERS	COMM	
EXIT	EXEC	
WRITEOUTPUTBUF	COMM	

MOVEBUFFER and MOVEBUFFERIF are essentially the same program. The difference is not found in the program but in the activity block associated with each. The activity block contains the address of the buffer control block associated with the source and one associated with the sink of the move. It moves data from the source, until empty, or to the sink until full, which ever occurs first. MOVEBUFFERIF has a preamble which tests an interlock parameter of the system and if it is set does not do the move.

VII.1.4 Buffer Control Module

The buffer control module provides a service that allows for sending data to and receiving data from a buffer. One module can send data to a buffer and another receive data from a buffer. The collection of programs in the buffer control module provides different types of control over how the data is sent or received from the buffer. It also contains a number of utility programs that are commonly used to describe the different buffers. The data about each buffer is contained in a table called the buffer control block for the buffer.

Location of Buffer Control Functions in Linkage Table

QP LOCATIONS

SETSCANTOEND	005E	BUFF
SCANBACKWARD	005F	BUFF
ADVTOSCAN	0060	BUFF
BACKUPSTORE	0061	BUFF
FULLP	0062	BUFF
EMPTYP	0063	BUFF
BACKUPREAD	0064	BUFF
RESTORESCANPTR	0065	BUFF
GETSCANPTR	0066	BUFF
SCANTESTOF	0067	BUFF
SCANNEXT	0068	BUFF
SCAN	0069	BUFF
SETSCANOF	006A	BUFF
READSTORED	006B	BUFF

READNEXT	006C	BUFF
READLAST	006D	BUFF
READFROM	006E	BUFF
WRITETO	006F	BUFF
CLEARBUFFER	0070	BUFF
STUFFBYTECODE	0071	BUFF
GETBYTECODE	0072	BUFF
SCANTESTEND	0074	BUFF
SCANTESTBNG	0075	BUFF
TELL/BUFFERCODE	0076	BUFF
GETINTRPBYTE	0086	BUFF

DP LOCATIONS

SCANTESTBNG	0094	BUFF
SCANTESTEND	0095	BUFF
SETSCANTOEND	0096	BUFF
SCANBACKWARD	0097	BUFF
ADVTOSCAN	0098	BUFF
BACKUPSTORE	0099	BUFF
FULLP	009A	BUFF
EMPTYP	009B	BUFF
BACKUPREAD	009C	BUFF
RESTORESCANPTR	009D	BUFF
GETSCANPTR	009E	BUFF
SCANTESTOF	009F	BUFF
SCANNEXT	00A0	BUFF
SCAN	00A1	BUFF
SETSCANOF	00A2	BUFF
READSTORED	00A3	BUFF
READNEXT	00A4	BUFF
READLAST	00A5	BUFF
READFROM	00A6	BUFF
WRITETO	00A7	BUFF
CLEARBUFFER	00A8	BUFF
GETBYTECODE	00A9	BUFF
STUFFBYTECODE	00AA	BUFF
TELL/BUFFERCODE	00AF	BUFF

VII.1.5 Buffer Control Blocks

The buffer control block for each buffer contains information on how to write or read from the buffer, how to advance in the buffer and how to backup. It contains links to programs (which may be suspend programs) that are to be executed if one reads from an empty buffer or writes into a full one. In addition to allowing one to read and write to a buffer, there is a mechanism to allow scan of the buffer. Also contained in the block are parameters that give the limits of the buffer, where data is read from, where data is written to, and where data is scanned from. Flags are available to tell whether the buffer is empty or full and whether the scan is at the end or beginning of the buffer.

VII.1.6 Communication and I/O module

The communication and I/O module has the programs for handling the interrupts of the system and multiplexing data on several channels between the QP and DP. This includes up to 16 channels between QP and DP of which only two are used. In addition it includes 16 transmit and receive token channels between QP and DP. For the QP, input and output to an Omron terminal, input and output to a Texas Instruments (TI) terminal, input and output to a graphics terminal (not used) and input and output to the ARPANET are handled by this module. For the DP, these special input and output functions include the input and output for IP and DP, for the Omron terminal and for the ARPANET.

Activity Functions

FUNCTIONS		
MOVEBUFFERIF	COMM	test semantic interlock
READATA	COMM	
MOVEBUFFER	COMM	
WRITEDATA	COMM	
READINBUFFERS	COMM	
WRITEOUTPUTBUF	COMM	

VII.1.7 Processor Instructions

All of the CCIS processors are specially built using micro electronic technology based on the AMD 2901 microprocessor. The instruction repertoire was designed to have certain properties that would facilitate the generation and maintenance of code. The limitation of the technology dictated limitations on the code. This section will discuss some of the tradeoffs encountered in generating this machine.

The principal architecture of the processor is that of the so called Van Neuman machine. It has a processor that loads its instructions from a memory in the same way it loads data and can only do one instruction at a time. The principal constraining limitation of this architecture is that the memory has to be serially accessed to read or write data. No effort was made to make the instruction repertoire more flexible than the serial memory limitation would dictate. However, certain serial loops do not require instruction fetch when using the multiple registers available to the AMD 2901 micro architecture. Effort was made to utilize this feature in some instances.

Another design criterion was that the bulk memory was expected to be that of a disk. The access to disk for programs not in processor main memory or in buffer memory would run 30 to 100 milliseconds. Instructions that could compress strings of instructions to a single instruction were examined and a few were implemented. However, as the criterion was not an overriding consideration, it played only a minor part in the design.

A far more important consideration was that the programs had to be modular with minimization of side effects of one program on another. The complexity of the system also dictated that non procedural techniques would be used. To facilitate this a stacking architecture was desired and implemented. Some of the instructions would be stack oriented, especially the subroutine calling sequence. With the stack, the storage and retrieval of data local to a module could be placed on the stack in a well controlled way. Also the desire to be able to pass functions as arguments dictated that calls from the contents of a register were required.

Because of the desired modularity of the system, the instruction repertoire contained data loading and storing functions that allowed for easy implementation of absolutely relocatable code. About 99.9 percent of the programs are absolutely relocatable in blocks. Jumps, branches, calls, data loading, were primarily relative. We tried to be pure in this endeavor but it was found that on occasion absolute reference was needed; i.e., to reference global system parameters. A few such instructions were added at a later time.

Of course, the standard arithmetic and logical functions were needed and implemented. These followed closely the architecture of the AMD 2901 ALU capability.

Some instructions were provided for I/O and display control.

The system was planned to have interrupt capability. Therefore the usual interrupt synchronization was required. Some of the instructions did not allow interrupts immediately following them. Release interrupt lockout would execute a return before the system was interrupted again and hence keep the stack clean.

A number of special instructional capabilities were designed for the CCIS processor(s) to improve performance for large complex data structures. As described below, appropriate testing of these features never came about because the need for overall functioning of the system overshadowed constructing large enough data bases to test thoroughly high performance features. Descriptions of these capabilities follow.

The AMD 2901 is not optimal for shifting. A special card that could do a number of complicated data rearrangement operations normally was required for extensive packing and unpacking of data. A card was designed to accomplish this, but was late in being implemented and proved to be more complicated in access than desired. By the time it was ready the programs had been written, paying the time delay price, and instructions for the byte and bit rotations had been implemented.

The half word data access and storage were added at a later time. They were placed on the desired list at the very first but other pressing needs delayed their implementation. They were not necessary.

A set of special instructions were designed to allow a nesting of local variables in the subroutines. These were required because the query processing design model had such a structure, used in particular for the design of parsing rules.

The access instruction was designed to trap access to data not in memory. Thus a technique of loading data before restarting the process (such as might be useful in checking data access authorization) could be programmed. It was never used because the system had not evolved to use this sophistication.

The repeat instructions were part of the original design. They were designed with the interrupt in mind. In some architectures; i.e., that of the Univac CP-642B, repeat instructions had been designed and implemented but they could not be used in an interrupt environment. In this system care was taken to insure that the repeat instructions were interruptible and that return from the interrupt would allow the continued repeat to proceed.

The move instructions proved to be valuable in many ways; i.e., clearing areas, and bulk moves. The search for match also was useful in multiway branches. The other search instructions were not used much, not because of any problem with the instruction, but because the development time did not allow for the improvements in the data comparing needed for fast joins of records. The search within limits was very useful in the string garbage collector.

The repeat instructions that had self termination because of internal conditions were originally designed to have the B register the terminating value. But the instruction as implemented cleared the B register. This should be corrected eventually.

The binary search was a good idea, but the situation has to be correct to use it That situation did not occur. Some improvements are needed to handle more cases to make it more useful. The same can be said for the bubble sort.

The special feature of generation of a break point is desired and usable, but development time did not allow for its proper utilization. The principal value of its use is in the delimiting of a growing table. Only after experimentation in deployment would its value be seen. Such testing never came about.

In summary, the instructions were designed to provide the general service of any normal processor plus special instructions and hardware to accommodate the critical processing situations of a data base processor. Some of these latter critical processing situations were never tested, the principal reason being that the overall functioning of the system overshadowed the detailed polish form needed to use the features. When the overall function of the system was working, no time was left for development of those techniques to make the system fast, robust, and more flexible.

A detailed description of the instructions available is in the ARPANET file (located on ISIC) accessible by the name <SMALL>CCIS. INSTRUCTIONS; 17.

VII.2 Query Processor Modules

The following modules are specific to the Query Processor. The protocol module provides editing and special interactions specifically with OMRON and Texas Instruments (TI) terminals. The lexicon module stores information about words that are used in the editing feature of the protocol module and in the English parsing module. The lexicon itself is the data structure that remembers the spelling and parameters associated with each word. The utility module provides programs and macros to handle the manipulation of data in the knotted memory used in the parsing process. The English parsing module, patterned after the Rapidly Extensible Language (REL) use of the Martin Kay parsing algorithm extended with feature checking on grammar rules [2] [3], takes data from the protocol module and parses to form a semantic tree that is passed to the semantic processing module. The parsing dictionary is a data structure that describes the total control of the parser. The parser itself is general purpose; the parsing dictionary is that part that makes the parser work as an English parser. In the process of parsing, the parsing dictionary tells the parser to make certain tests. These tests are the programs collected in the condition (CND) functions module. The

dictionary also tells the parser what semantic functions to mention in the semantic tree. There is a correspondence table that relates the information in the parsing dictionary and the functions of the CND module and another table that relates the information of the parsing dictionary with the semantic functions in the semantic module. The semantic module also contains functions that interpret the semantic tree to generate the language to be passed to the communication module for sending control to the Data Processor. Miscellaneous pieces contain some debugging tools, programs to transfer programs to the Data Processor, and some initialization programs.

Query Processor Buffer Blocks

Name of Buffer	# of Bytes	Location of control block	Location of buffer	Byte address
Newspell	112	347Ø	DCØØ	37ØØØ
Parsebuffer	32Ø	328Ø	DC7Ø	371CØ
Parzebuffer	32Ø	329Ø	DCCØ	373ØØ
Neweditbuffer	32Ø	32AØ	DD1Ø	3744Ø
Oldeditbuffer	32Ø	32BØ	DD6Ø	3758Ø
Userbuffer	32Ø	32CØ	DDBØ	376CØ
Outputbuffer	32Ø	32DØ	DEØØ	378ØØ
Datain	256	32EØ	DE5Ø	3794Ø
Data out	256	32FØ	DE9Ø	37A4Ø
Destination	64	33ØØ	DEDØ	37B4Ø
Source	64	331Ø	DEEØ	37B8Ø
From grafics	64	332Ø	DEFØ	37BCØ
To grafics	64	333Ø	DFØØ	37CØØ
Omron inbuf	64	334Ø	DF1Ø	37C4Ø
Omron outbuf	64	335Ø	DF2Ø	37C8Ø
ARPA receive	64	336Ø	DF3Ø	37CCØ
ARPA transmit	64	337Ø	DF4Ø	37DØØ
TI inbuf	64	338Ø	DF5Ø	37D4Ø
TI outbuf	64	339Ø	DF6Ø	37D8Ø
Graf inbuf	64	33AØ	DF7Ø	37DCØ
Graf outbuf	64	33BØ	DF8Ø	37EØØ
Input intØ	64	33CØ	DF9Ø	37E4Ø
Output intØ	32	33DØ	DFAØ	37E8Ø
Input int1	32	33EØ	DFBØ	37ECØ
Output int1	32	33FØ	DFB8	37EEØ

VII.2.1 Protocol module

The protocol module provides editing features to the user as well as screen handling for the Omron and TI terminals. A mechanism is provided by this module to recognize when a statement is completed and is ready to be processed by the parser. The module does some preprocessing to remove leading spaces, convert carriage returns to spaces, reduce multiple spaces to a single space and fill out words found in the lexicon (using the lexicon module services) when requested by the use of the terminal ESC key.

Activity Functions

FUNCTIONS		
LEXITDRIVER	PROT	
QUERYDRIVER	PROT	
PARSEDRIVER	PROT	in protocol

VII.2.2 *Lexicon Module*

The lexicon module is a collection of programs that allows the creation of new words in the lexicon, the reading of the contents of the lexicon, and the matching of words with the words of the lexicon. It provides a matching service to both the editor in the protocol module and the parser in the English parser module. It provides a listing service to the semantic module. It also provides a spelling service to the semantic module. At the present time it does not coordinate with the DP lexicon. There are a few other services available but not being used: such as, delete word, delete all words after a given word, and find closest spelling in the matching process.

Functions in Linkage Table

QP LOCATIONS		
SENDLEXWORD	0085	LEX
ENTER/A/WORD	0088	LEX
RESETLEXICON	008A	LEX

Activity Functions

FUNCTIONS	
PRINTLEXGROUPS	LEX

VII.2.3 The Lexicon

The lexicon is a data structure consisting of four parts: A spelling table, an index table, a fast access table, and a collection of four control blocks. The information parameters (48 bits) associated with each word are found in the index table with 8 additional bits in the spelling table.

VII.2.4 Utility Module

The utility module is a collection of macros and programs to manipulate trees and charts. These are used principally in the English parsing module, CND functions module, and the semantic processing module.

Functions in Linkage Table

QP LOCATIONS		
CONS	0077	UTIL
DIFFLIST	0078	UTIL
GETASCNU	0079	UTIL
GETNEWCELL	007A	UTIL
GIVEUPCELL	007B	UTIL
GIVEUPTREE	007C	UTIL
INTERSECTION2	007D	UTIL
LDIFF	007E	UTIL
LENGTH	007F	UTIL
MAKENUMBER	0080	UTIL
NCONC2	0081	UTIL
TCONC	0082	UTIL
TCONCER	0083	UTIL
UNION2	0084	UTIL
LINKGARBAGE	008B	UTIL

VII.2.5 English Parsing Module

The English parsing module is an interconnected system of programs that performs the number parsing and the lexicon parsing, as well as the English parsing. The end product of the parsing is a chart which is passed to the semantic module.

Functions in Linkage Table

QP LOCATIONS		
GETFEATBITS	0087	RULE
SAVECHKBITS	008C	RULE
PARSEIT	008D	RULE

Activity Functions

FUNCTIONS		
PARSEDRIVER	RULE	in parsing

VII.2.6 The Parsing Dictionary

The parsing dictionary is a data structure which describes the English language, including grammar rules used for CCIS and features (e.g., should one of the noun phrases in the rule be a number) which apply. It is highly compressed to save space. For each rule in English, it not only describes the structure to be recognized but also the CND function (if any) to be called and the semantic function to be placed in the semantic tree, a part of the parsing chart. The grammar transformation (TNF) required for a given grammar rule is also encoded in a compressed program which is interpreted upon activation of the rule.

VII.2.7 CND Functions Module

The parsing of an English sentence occasionally encounters a rule that is so complex that the feature mechanism cannot determine whether the rule should be applied. In these cases a CND function is called to determine whether the rule applies. In other cases, although the rule does apply, the manipulation of the chart is more complex than can be described by the TNF functions and the CND can be used to do these manipulations. The parsing dictionary tells which CND function to call when the rule is being tested for selection.

VII.2.8 Semantic Processing Module

The semantic module is a collection of programs which will cause the DP to do various processes; e.g., create, add to a relation, join relations. It also has a program which interprets the parsing chart to find and accomplish the programming structure represented by the semantic trees.

Functions in Linkage Table

QP LOCATIONS		
SEARCHPOS	0089	SEM
SAVETREE	008E	SEM
COPYTR	008F	SEM
INSTALLTREE	0090	SEM

Activity Functions

FUNCTIONS	
SEMANTICDRIVER	SEM

VII.2.9 Miscellaneous pieces

Special Memory Monitor
QPXmt
MOVEBUFFERIF
Call Unpack Lexicon
Clear Special Memory
and Create Dictionary Tree.

There are a few pieces which have not been incorporated in any of the above modules. These are a collection of programs to interpret the knotted memory, a program to transmit programs from the QP to the DP, a calling sequence that calls programs from the lexicon to convert a compact lexicon to the standard lexicon, a calling sequence that calls programs from the English parsing module and a calling sequence which initializes the knotted memory area. The last two are short programs of not more than 20 instructions each.

The cleargraphics routine is a dummy routine consisting of a simple return, since the graphics module was not built into the system.

There is a small hand coded test on the semantic interlock before going to MOVEBUFFER. This is indicated by the function MOVEBUFFER IF.

Functions in Linkage Table

FUNCTION NAME		HEX	
CLEARGRAPHICS		0073	Dummy Routine

Activity Functions

FUNCTIONS		
MOVEBUFFERIF	COMM	test semantic interlock

VII.3 Data Processor (DP) Modules

The Data Processor provides semantic command compilation (by the compiler) and interpretation (execution of commands as received from the compiler handler), fetching, updating and analysis of data (i.e., the relational system) and storage-management processes. The latter consists of logical file and record maintenance, disk-page management and local- and buffer-memory management. The common modules described above (in Section VII.1) are, of course, a part of the Data Processor.

Data Processor Buffer Blocks

Name of Buffer	# of Bytes	Location of control block	Location of Buffer	Byte address
Omron outbuf	64	320C	3280	CA00
Omron inbuf	16	31FC	32C0	CB00
Composite	128	317C	32D0	CB40
IP HI rec	64	31DC	3350	CD40
From QP	64	319C	3390	CE40
To QP	64	31AC	33D0	CF40
Graf inbuf	16	31BC	3410	D040
Graf outbuf	16	31CC	3420	D080
IP inbuf	16	321C	3430	D0C0
IP outbuf	16	322C	3440	D100
Input int0	16	323C	3450	D140
Output int0	8	324C	3460	D180
Input int1	16	325C	3470	D1C0
Output int1	4	326C	3480	D200

VII.3.1 Compiler Handler

The compiler handler is a collection of functions which controls the allocation, compilation, execution, and cleanup of the parts of the DP semantic command sequence used to generate relational and output module effects desired. It also contains a library of programs that are called by the generated code. Some of the simpler processes are contained in the library directly, but the more substantial programs are set up and then called elsewhere.

The tuple compiler itself is a reverse polish driven code generator. This architecture was chosen because of simplicity of design, the speed of compiling, the flexibility of generation and control of flow, rapidity of implementation, and compression of notation. It is totally table driven with no searches. It compiles long expressions in less than 50 milliseconds. This is a shorter time than loading a comparable program from disk. The compiled code generally takes about 4 to 5 times as much memory as that of the string which was compiled. The table used for generating structured calls is found in the compiler handler.

The compiler handler module consists of about a half dozen compiler loading, allocation, and execution routines and uses the sequence compiler to generate code to effect the command sequence presented to it. About 100 routines are used to match the compiler handler with other routines in the system and simple utilities. It also contains two tables, one of which is used by the tuple compiler and the other of which is linked to the compiler by one of its registers so that the user does not have to call them by absolute address. It contains programs that read the data and generate strings for the input. It garbage collects the input string area when required.

The operations of the tuple compiler are given in Figure 2. The instruction repertoire contains operations for data loads and moves, reading and writing files, branching,

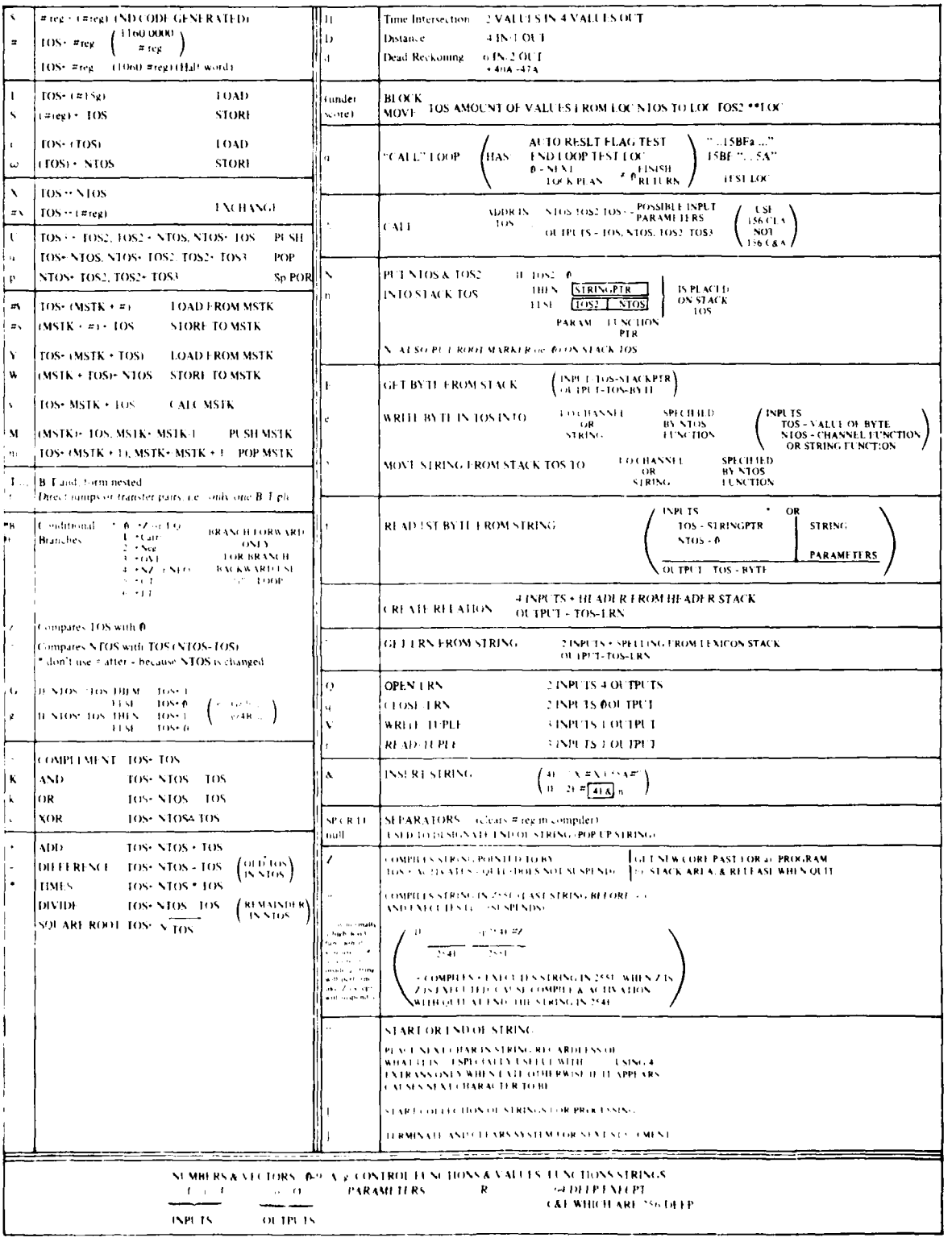


Figure 2. Symbol definition.

loops, stack manipulation, constrained jumps, compares, logical and arithmetical functions, subroutine calls, macro expansion, block moves, I/O, compile with suspend and compile without suspend, and calls to system functions. Beside the file handling, create relation, and use of the lexicon are provided. A double branch notation is provided to facilitate construction of Boolean lattices. All branching and jumps are forward. Only loops have control paths going backwards. Loops are self contained with no branches in or out. Also, each loop has an abort test which will cause the loop to terminate in addition to the normal loop termination test.

In addition to instructions there is a collection of single letter names used to identify arrays of memory locations to store temporary values. These are called vectors. They are labeled with the letters: A C F i I j J o O P R. Each of these has associated with it a normal usage, which is usually arbitrary and assigned by the environment in the system. The exceptions to this are the F vector, which has meaning to the string garbage collector, and the C vector, which contains a preset table pointing to locations of where compiler handler routines are located. Also each vector has a size, which is the number of cells set aside for that array. These are summarized in the following table (Table 2):

Table 2

Name	Size	Normal use
A	64	Dynamic variables which are assigned to different functions or modules.
C	256	The more constant quantities, tables of functions and pointers to permanent strings.
F	256	The table of string functions and macros. The garbage collector uses this table to determine where changing strings are located. Changing strings are located in just this one area of memory. Other string pointers are left alone.
i	64	General purpose vector, normally used to select and specify records used in the data processing.
I	64	General purpose vector, normally used to select and specify records used in the data processing.
j	64	General purpose vector normally used to load tuples for processing.
J	64	General purpose vector normally used to load tuples for processing.
o	64	General purpose vector normally used to compose tuples in the processing.
O	64	General purpose vector normally used to fetch tuples for outputting.
P	64	General purpose vector normally used to save old results that will be used in future processing.
R	64	General purpose vector normally used to control the outputting of data from the system.

In the compiling process numbers found in the string are composed in a register for that purpose. Normally non digits clear that register. As each new digit is read the old result in the register is multiplied by 10 and the new digit is added. However, when a vector letter is read a table position is found which holds a number which is then added to the contents of the number register. This becomes the address of the vector. Code can be generated to do various things to the cell or address so indicated. For example, 12345:32IS is compiled to generate code to load the number 12345 and then to store that value into the position 32 of vector I. In another example 21JL42IS will cause code to be generated that will take the contents of the 21 position of the vector J and store it into the 42 position of the vector I. Another example: A:60iS will generate code to take the address of the 0 position of vector A and store that address into the 60 position of i. If there is a space in front of A, the number register is cleared, thus making it the 0 position of A. The : after the A causes code to be generated to load the value of the number register at the time it was compiled. In this case the number register would contain the address of the vector A. In the case above it would have contained the number 12345. It is also possible to load and store any cell in memory.

VII.3.2 Relational System Module

The logical file management system modules (VII.3.4) are used to support the implementation of the relational data-management operations. These operations include creation of new data structures, adding data to those structures by the human user of the system and also dynamically adding data to them using the Interface Processor, performing the classical relational operations of join, restrict and project, stringing those primitive operations together, and performing arithmetical operations such as finding the distance between dynamic data elements, finding the current position of one or more such data elements, or finding the intercept time between two dynamic data elements.

All primitive relational system operations are accessed by using various pointers to strings found in the "F" array (see VII.3.1 on the compiler handler). Each of these pointers is a reference to the memory location of the first byte of a string programmed in the language described in VII.3.1 and Figure 2 of that section. Following is a description of the function of each of the strings which corresponds to a primitive relational operation. The programs for these strings can be found in the ARPANET file on ISIC named <SMALL>DPQUANSEM:73.

64F Create a Name
65F Create a Class
66F Create a Number Relation
67F Create a Relation
68F Add to a Class
69F Add to a Relation
70F Add to a Number Relation
78F Add to a Dynamic Relation

The strings 77F, 79F and 99F are utility strings supporting 78F.

80F Set the current time cell (34A)

82F The primitive string which is used in various combinations to cause joins, restricts, class intersections and projects. These combinations are arrived at by setting where the key (in the vector location 4I or 4i) is for any relation used and by setting what information is to be carried from one operation to another (using the vector location 2I or 2i): i.e., data preceding the key, the key itself, or data following the key. The

variable string 5F is used in this program to determine the type of compare to be used in its processing; i.e., key equality or inequality, numerical restriction or special function computations such as distance or current position.

The strings 75F, 76F, 6F, 7F, 8F, 9F, 10F, 11F, 12F, 13F, 14F, 15F, 16F, 17F, 18F, 84F 85F, 86F, 88F, 89F, 87F and 83F are utility strings supporting 82F. 81F is a string used to facilitate debugging.

92F – Used to invoke the computation of current position

93F – Used to invoke the computation of distance

98F – Used to invoke the computation of time to intercept

The strings 90F, 91F, 94F, 95F, 96F, 97F, 100F and 101F are utilities supporting 92F, 93F and 98F.

Linkage Table Functions

DP LOCATIONS		
MULTIPLY	00C3	ART1
DIVIDE	00C4	ART1
CIRCLEDISTANCE	00C5	ART2
DEADRECKON	00C6	ART1
NEGCONV	00C7	ART1
CONVERT1	00C8	ART1
SINECOMP	00C9	ART1
TCONVERT	00CA	ART1
DELTAT	00CB	ART1
INTERCEPT	00CC	ART3
LEXICON	0066	LEXI
UNPACKLEXICON1	008B	LEXI
DISTANCE	0078	COMH
RENDEZVOUS	00CE	NOT USED

VII.3.3 Output Module

The output module takes the resulting record from the relational system module and steps through it one small item at a time forming the appropriate character sequence and sending it to channel 1 for communication from the Data Processor to the Query Processor. The module is suspended if the channel is full. The data passes through channel 1 to the output buffer of the Query Processor. Data is taken from there and passed to the OMRON screen and/or the TI terminal. The programs for these strings can be found in the ARPANET file on ISIC named <SMALL>DPPRINTQUANTITYSEMSTRINGS;3.

Linkage Table Functions

DP LOCATIONS		
UNPACKNUMBER1	008D	COMH

VII.3.4 Logical File Management System

The data structure for the relational records is essentially that of storing and reading data from pages as if the data were organized in a table. The table is read one row at a time. The new table is constructed one row at a time. In the processing of data each table is scanned one row at a time and the field manipulated and processed to obtain a row of a new table. The rows of the table are called tuples. In many of the processes the same record could be accessed by different parts of the process or different processes altogether. The steps of scanning down the rows of the tables is a common process. There is no arbitrary access to the middle of a table. The principal access was to start at the beginning of the table and scan forward until the end, or when putting new data into a table to place the data at the end of the table and assign new pages if necessary. A set of programs was generated to facilitate these operations. The open record would cause the creation of a logical file name which would then be used to keep track of the parameters to control the scan. With this one could also get values of headers and parameters to point to pseudo records if needed. Also the tuple size could be obtained. When the processing was finished the close record could be used to clean up the control parameters as well as make sure the last data was written if necessary. The read tuple allowed for reading a tuple from the beginning or end and advancing and backing up as need be. The write tuple allowed for the placement of data into the record. One could write at the end or cause the data to be moved and write it at the beginning.

Linkage Table Functions

DP LOCATIONS

CREATE/RELATION	ØØB4	RELM
ADD/PAGE/TO/RELATION	ØØB5	RELM
FIND/SPACE/UTILIZED/ON/PAGE	ØØB6	RELM
FIND/NEXT/PAGE/FROM/HDR	ØØB7	RELM
FIND/PSEUDO/RECORD/NAME	ØØB8	RELM
FIND/TUPLE/SIZE	ØØB9	RELM
ADD/TO/SPACE/UTILIZED	ØØBA	RELM
FIND/PREVIOUS/PAGE/FROM/HDR	ØØBB	RELM
FIND/START/OF/USER/PAGE/STRING	ØØBC	RELM
FIND/INITIAL/LOCATION/OF/LRN	ØØBD	RELM
OPEN/LRN	ØØBF	LGFL
CLOSE/LRN	ØØCØ	LGFL
READ/TUPLE	ØØC1	LGFL
WRITE/TUPLE	ØØC2	LGFL

VII.3.5 Bulk Memory Page Management

The bulk memory (currently disk) is organized into pages of fixed length (Figure 3 shows a typical data page structure) with pages grouped into logical records. A pointer to the head of each logical record is maintained as an index to bulk storage by use of the logical record catalog (Figure 4 shows a typical catalog entry).

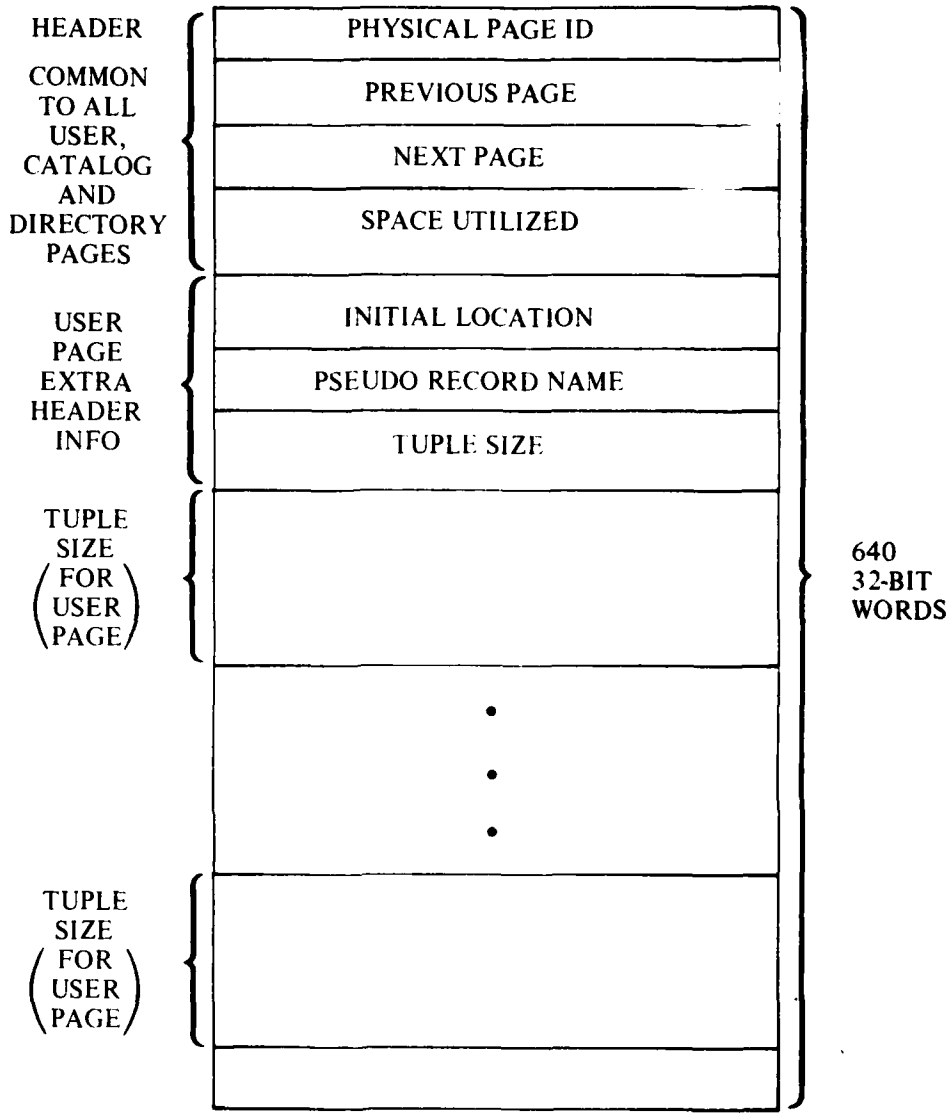


Figure 3. Page structure.

LOGICAL RECORD NAME (LRN)
AREA TYPE (RELATION NUMBER RELATION, CLASS, NAME)
ROOT PAGE ID (FIRST PAGE OF RECORD)
PAGE DIRECTORY POINTER
LENGTH IN NUMBER OF PAGES

Figure 4. Typical catalog entry.

The next layer of indices into bulk storage is maintained in the page directory, where the connectivity of each page to other than itself is maintained. Pages in use are kept in this directory (Figure 5 shows a typical directory entry).

PHYSICAL ID OF PAGE ENTRY
LOGICAL RECORD TO WHICH IT BELONGS
LAST PAGE (ID) FOR RECORD
FIRST ELEMENT VALUE FOR THIS PAGE IN RECORD
PAGE DIRECTORY POINTER OF NEXT PAGE OF RECORD
ADDRESS OF FIRST WORD OF DIRECTORY ENTRY

Figure 5. Typical page directory entry.

The handlers and utilities are designed to provide redundancy in the pointer structure and to provide additional means for finding those pages belonging to a relation or logical record without loading all pages of the record. Additionally, all records are named (other than just through their physical address) using a double hashing scheme. This promotes independence of the record from its location on disk or in memory. The next-page-pointer redundancy for each page of the record also is maintained in order to avoid loss of record structure if the page directory and/or catalog becomes lost.

Linkage Table Functions

DP LOCATIONS		
CREATE/NEW/NAME	0056	CATM
CREATE/NEW/LOGICAL/RECORD	0057	CATM
FIND/CATALOG/ENTRY	0058	CATM
DELETE/LOGICAL/RECORD	008E	CATM
CAT/GARBAGE/COLLECT	00B1	CATM
CLEAR/PAGE	00CD	CLPG
CREATE/NEW/PAGE/DIRECTORY/ENTRY FOR/LOGICAL/RECORD	0050	PAGM
CREATE/FIRST/PAGE/DIRECTORY/PAGE	0051	PAGM
UPDATE/A/RECORD	0064	PAGM
FIND/DIRECTORY/LOC/OF/ELEMENT/ POINTER	005F	DIRF
FIND/NEXT/PAGE/POINTER	0060	DIRF
COPY/HDR	0061	DIRF
FIND/AREA/TYPE	0062	DIRF
FIND/ROOT/PAGE/ID	0063	DIRF
DELETE/CAT/PAGE	008F	DIRF
FIND/LAST/PAGE/OF/RECORD	00B2	DIRF
FIND/LENGTH/PAGES	00B3	DIRF

VII.3.6 Buffer and Local-Memory Management

These functions are designed for maintaining memory status with respect to space available for pages. Bits are kept for each page location in buffer and in local memory to show if the page is locked into memory and cannot be written to disk, if the page is protected against writing, has been written on, and so forth. One of the functions copies pages from disk to buffer and/or local memory as space is available.

Linkage Table Functions

LOCK/A/PAGE	0053	PMGT
UNLOCK/A/PAGE	0054	PMGT
GET/PAGE	0055	PMGT
GET/NEW/PAGE	0065	PMGT
GET/NEW/DISK/PAGE	006A	PMGT
TEST/PAGE/IN/CORE	006B	PMGT
CHANGE/PAGE	006C	PMGT
GET/NEW/CORE/PAGE	006D	PMGT
NAME/CORE/PAGE	006E	PMGT
DELETE/PAGE	006F	PMGT
COPY/PAGE	0070	PMGT
NO/OF/AVAIL/CORE/PAGES	0071	PMGT
TEST/PAGE/IN/CHANGE/MODE	0086	PMGT
NO/OF/AVAIL/DISK/PAGES	0087	PMGT
MAX/NO/OF/AVAIL/CONT/CORE/PAGES	0088	PMGT
GET/N/NEW/CONT/CORE/PAGES	0089	PMGT
REL/N/CONT/CORE/PAGES	008A	PMGT

VII.3.7 Auxiliary Memory System

Most requirements for Navy information storage and retrieval demand random-access auxiliary storage. An initial choice for such a storage device was the Model 44 Diablo disk drive with a nominal disk capacity of 6250000 sixteen-bit words and an access time of about 50 milliseconds. This drive is reasonably typical of the lower-cost disk drives which are available.

The functions of the disk controller and the I/O channel interface were performed by a combination controller and buffer-memory (12000 thirty-two bit words) designed and constructed at the Naval Electronics Laboratory Center (NELC) (now NOSC).

There are a number of advantages in having a buffer-memory interface between a computer and a disk drive. These include a reduction in the total disk-drive access time and the provision of a temporary storage area between disk and main memory and are discussed in the following two paragraphs. Development time did not allow for their full exploitation.

When using a disk drive, head positioning and latency of the disk drive (waiting for the disk drive to reach the proper track and the sector within the track) present a delay problem. Without the buffer-memory (a direct interface between the computer and the disk drive), the computer would have to wait the length of this disk-access time which is about 50 milliseconds for the Diablo disk drive. With the buffer-memory interface, the computer instructs the main controller to input a certain page into the buffer. The 50-millisecond access time of the Diablo disk drive is very significant and will slow the processing time of the computer if the disk is interfaced directly.

The buffer can be used to store intermediate data of an incomplete operation (temporary scratchpad memory). The buffer will allow small changes to be made in a page from the disk without having to bring a whole page of memory into main memory. The page is first brought from disk to the buffer and then the changes are sent from the computer to the desired location in the buffer. As few computer words as one may be rewritten from the buffer back to the disk. This saves computer I/O time and main-memory space. Page thrashing can be reduced with the use of the buffer as a temporary storage when generating and recognizing item names (especially on a small-memory computer).

VII.4 Interface Processor (IP) Modules

The CCIS Interface Processor (IP) provides specialized hardware interfaces and matching software for translation of data from one or more non-CCIS sources into a CCIS compatible format. It then passes data it has translated to the Data Processor, which is responsible for all data maintenance, including addition of dynamically changing data to its database.

VII.4.1 NTDS and INTEL Data Formatting

The format of dynamic input data (for demonstration purposes simulated Intel and NTDS data is used) must be sent to the IP from either the ARPANET or the CCIS Tester Box over a standard RS232 serial ASCII line before input records can be understood by the IP's executive. The format record is described below by use of Example 1 (See ** note).

EXAMPLE 1.

(FN/ */ A "TYPE"/ N "LAT"/ N "LONG"/ N "COURSE"/ N "SPEED"/NØ)

WHERE:

(	=	Begin Record Boundary
FN	=	Format Definition of an NTDS Report (an NTDS report format triggers the mechanism which sends only data whose course, speed or type fields has changed)
or		
FI	=	Format Definition of an INTEL Report
/	=	Delimiter Between Fields
*	=	Track ID Field (i.e.,; predefined CLASS name of track)
A	=	Alpha-numeric field is being defined
" ----"	=	Predefined RELATION name for Alpha-numeric field
N	=	Field is numeric.
" ----"	=	Predefined NUMBER RELATION name for numeric field
Ø	=	No Definition Following this Field (i.e., field has a fixed numeric field definition)
)	=	End Record Boundary

Any change in the input record input or output field definition format must be preceded by a format definition. (** All predefined names used in the format record must be defined in the static data base prior to dynamic data loading. There is a current limitation of 5 data fields per format definition due to CCIS Data Processor semantic software limits.)

A sample data input for an NTDS track, NTDSTN 1, follows:

EXAMPLE 2.

IP INTERFACE RECORD FOR NTDSTN 1

(N/NTDSTN 1/SURFACE/ØØØØ5396/Ø7955Ø66/Ø4ØØ/ØØØ15Ø/Ø87134ØØØ)

WHERE:

(	=	Begin Record Boundary
N	=	Data Type Indicator (i.e., N – NTDS, I – INTEL)
/	=	Delimiter Between Fields
NTDSTN 1	=	Name previously defined in the static database designating NTDS Track Number 1.
SURFACE	=	Name previously defined in the static database designating the type of track (i.e., SURFACE for Surface Track, SUBSURFACE for subsurface track, AIR for Air Track).
ØØØØ5396	=	Latitude of the track. Given in degrees, minutes, seconds and tenths of seconds, expressed as an integer number, signed or unsigned
Ø7955Ø66	=	Longitude of the track. Data is in the same format as latitude, except all inputs are positive.
Ø4ØØ	=	Course of track. Given in degrees and tenths of a degree, expressed as an integer.
ØØØ15Ø	=	Speed of track. Given in knots and tenth knots, expressed as an integer (i.e. 15.Ø knots).
Ø87134ØØØ	=	Time of sighting. Given in Julian format of Julian day, hours, minutes, seconds (i.e., 87th day, 13th hour, 4Øth minute, Øth second).
)	=	End Record Boundary.

VII.4.2 IP Executive

The IP executive monitors the input buffer flag (checking for the presence of simulated NTDS data input) and the clock flag. If neither flag is set, it will request a simulated Intel (batch input) record from the ARPA net. Data from the ARPA net is loaded into the output working buffer and then reformatted for sending to the Data Processor.

Simulated NTDS data interrupts the IP executive and is stored in the input buffer, a ring buffer which can hold up to 16K characters. When a complete record has been received, the input buffer flag is incremented. The IPEXEC will then read the entire record from the input buffer into the working buffer. Only NTDS records which have changed significantly;

i.e., type, course, or speed for that track have changed, are sent, reformatted, to the Data Processor from the working buffer. If the working buffer overflows, the most recent data is sent to the DP in preference to older data. The program for the IP Executive can be found on the ARPANET file on ISIC named <SMALL>IPEXEC.CR;8.

VII.4.3 Current Time Generation

The interval timer interrupts the IP executive every 10 ms to update the internal clock. The clock may be initialized by sending a time report from the ARPA net or the Tester Box, as shown in the following example:

EXAMPLE 3.

(T/127134500/120)

WHERE:

(	=	Begin Record Boundary
T	=	Identifies this as a Time Report Record to the IP
/	=	Delimiter between Fields
127	=	Example for Julian Day Start Time
13	=	Example for Hour of Day Start Time
45	=	Example for Minute of Hour Start Time
00	=	Example for Second of Minute Start Time
120	=	Interval in seconds between current time reports to be sent to the Data Processor. (120 = 2 minutes between time reports sent)
)	=	End Record Boundary

The clock flag is set whenever the seconds count is equal to the time-report interval, which is set by a time report. The IP executive will output the current time to the Data Processor when it sees the clock flag set.

As mentioned above, all names used in format descriptions and in data entry to the IP must be predefined in the CCIS Data Processor database and in the Query Processor lexicon using the CREATE commands of the data model. Dead Reckon software is provided in the Data Processor to project current position from the most recent data known about any dynamic track data and current time. Use of this technique combined with the IP policy of only sending "significant changes" to the database for NTDS tracks minimizes the amount of storage required to maintain current dynamic data.

VII.5 Processor-Processor Communications

In the description of the communication modules the techniques of handling the data were considered in a general way. Here a more logical description of the processor-processor communications will be discussed.

To review the techniques, consider that in each processor except the IP the buffer handler provides tools to form character buffers. This set of programs provides methods of putting character streams into and removing character streams from the buffers. There are

three basic interfaces to the buffers. First there are the interrupt routines. During the interrupt processing data is placed into and removed from the buffers, with special care to ensure the data is not lost or extra characters added. Second there are activities whose responsibility is the movement of data from one buffer to another. Third, the major processing modules either take data from a buffer and in the process do the testing for empty; are structured to suspend if there is no data; or to put data into the buffer with three options; i.e., data can be placed in the buffer until the buffer is full and the remainder dropped out of the data stream; the buffer can be tested and appropriate action taken if full; or data can be placed in the buffer,but the activity suspended if it won't go into the buffer, with the structure that reentry from the activity is appropriately controlled.

As can be understood by the tools data can flow from one processing module to another in a controlled fashion without loss of characters even though the processing modules are on different machines.

The data flow between the Query Processor and the Data Processor is of special interest. In this particular communication module a service is provided to allow for the movement of data back and forth in such a way that up to twenty independent streams of data can be handled. At the present time only two are used. Handshaking is provided in such a manner that when data is not removed at the receive end no new data is placed in the transmit end of a channel. Thus data flow and generation can be delayed until the receiving module can properly use it. In addition to the data flow, sixteen special flags allow one processor to set the flag of the other. Special data hold flags which will stop the flow of data through a channel are also provided.

In the design of the system the IP was treated separately and does not have a uniform design of common modules as is found between the Query Processor and the Data Processor. The communication between the Data Processor and the IP is constrained in the following way. The principal design of the interface with the IP is that data flows mostly from the IP. In early considerations of the development of CCIS it was thought that the Data Processor would be interfaced to the IP in such a manner that it would trap all data from the IP without flow control from the IP. Upon further consideration it was decided that allowing IP data flow without control was dangerous. The present design uses a flow control to the IP. It works as follows: The IP waits until it receives the null character, at which time if it has data it will send not more than twenty characters to the Data Processor and will wait to receive another null character before it will be allowed to send the next twenty characters. The IP does not remember nor count the null characters. It is not critical when the IP discovers that a null character has been sent after the sending of data to the Data Processor. For instance, the I/O channel allows two characters to be placed into the out channel and the processor will believe the characters have been sent. If the IP then receives a null character it will not cause any problems. No special consideration that two characters are in the channel is needed.

In the Data Processor the data from the IP is double buffered. The only time the null is sent is when the interrupt buffer is empty. Data is moved from the interrupt buffer to the IP data buffer when there is room in the IP data buffer.

The above discussion reviews the tools for communication. In the following paragraphs the multiplexing and processing of data will be discussed.

In the initial setup of the Data Processor, the data from the IP is sent to the user buffer of the Query Processor via channel 2 from the Data Processor to the Query Processor. The user buffer is the buffer where data typed on the Query Processor keyboard is generally sent. From there it is parsed and processed. However, when dynamic data is entering the system the IP is switched to flow into the Data Processor multiplexor.

When a person queries the system the characters typed pass from the OMRON interrupt buffer to the user buffer. Data is taken from the user buffer and control protocol is extracted. The edited data is passed to the parsebuffer. From here it is reformatted slightly and passed to the parsebuffer. The parser takes the characters out of the parsebuffer and generates a chart which is stored in the special extended Query Processor memory. The semantic module extracts the data from the special memory and generates command sequences which it sends to channel 1 in the Data Processor.

The data arriving from the Query Processor via channel 1 and the data arriving from the IP in dynamic data mode via the IP data channel are of the same format.

The format to the Data Processor multiplexer is composed of a left bracket, a sequence of command messages, and finally an unescaped right bracket. Percent sign is the escaping mechanism.

The multiplexer continually scans the IP data channel and channel 1 from the Query Processor. It drops all characters until it sees the first left bracket, at which time it locks onto that channel. It continually reads that selected channel, copying the data into the command buffer until an unescaped right bracket is encountered. It copies the brackets also. It then goes back into the scanning process, alway starting with the other channel. It continually fills the command buffer using this process.

It is important that the IP or Query Processor not open a command sequence prematurely. If it does so it locks the data processor so that transactions from the other processor cannot come in.

The relational system module and the output module are data and program structures controlled by the compiler handler of the data processor. It takes the command message sequence and calls the sequence compiler to generate code to do the commands indicated. The sequence will call appropriate subroutines to do the steps.

The relational system module executes the program to accomplish the indicated task. If the task requests a report to be generated then the data record indicated is passed to the output module and the data processing module resumes processing the data from the command buffer without waiting for the report to be generated.

There are three groupings of command sequences in the system: those that report the results, those that process the data, and those that control the flow of processing. Besides the data passed from the Query Processor to the Data Processor are control programs which represent the different actions that the Data Processor can take on the data. In the actual communication path only a selection token is sent. That token is expanded into the sequence that is compiled. Then it is executed and the results or partial results are saved for the next step. Each query may have anywhere from two to twenty compiles, depending on the number of steps needed to get the result.

To give an example, consider the following:

The selection token 38C& would be expanded to 53C& 4iLX2:4iS 7:2IS1:2iS 56C&
82F:%@ :iS1iS

In this case 53C& will be expanded to get a previous result and place it in small i registers. The previous result does not mean the result that was calculated immediately before the last calculation, but means the one stored on the control stack to be the previous result. The next phrase, 4iLX2:+4iS, will generate code to select the range part of the relation instead of the domain part for the small i registers. This is followed by 7:2IS1:2iS, which generates code to copy the entirety of each tuple when taken from the last result and only the extra field of the next previous result. The 56C& is expanded to exchange the contents of regs i and I. The main join is compiled and executed when the code for 82F:%@ is executed. This essentially generates the new result. The code: iS1iS cleans up after the operation. The small i regs are marked as scratch. In effect this entire expression forms a restriction of the last result by having elements equal to the range of the previous result.

VIII. ISSUES

VIII.1 System Response Time

When entering a query the estimated time for the parsing and lexicon is 11 to 40 seconds. The communication time is about 2 seconds. The data processing time is about 1 to 10 seconds per join. This time is determined by the size of the records and the number of expands required. A relation of about 100 entries without expands takes about 3 seconds to process. Each expand takes 300 to 500 milliseconds to process. An expand rich relation can take much longer. Each query averages about 3 to 4 joins. The response from the Data Processor can be between 1 and 30 seconds. The total time to respond can be from 15 to 130 seconds. The time to write and edit a query depends on the person asking the query, but estimates of performance noted place the average time at about 5 to 40 seconds. The time to write and edit a query is not remembered by the user because he is doing something. However, when he gives the final carriage return, he becomes aware of how long it takes the system to respond. The time it takes for the results to be printed out is about 1 to 180 seconds. This time is also not noted by the user because he is actively analyzing the response. However, if the response is quite long he gets bored and does note the time.

The total time of a query is about 15 to 180 seconds of which 10 to 80 seconds is idle time for the user.

When loading a data base through the Query Processor, pipelining occurs and the entry time of a new statement is about 7 to 13 seconds.

The time that it takes to enter a new statement directly into the Data Processor from the Interface Processor is about 2 to 3 seconds.

VIII.2 Data Storage Capability

VIII.2.1 Number of user generated words

The number of words that can be added to the new system is about 500 . This information is described in more detail in the CCIS Operations Manual [4]. To make the system accept more words would require moving the lexicon in the Query Processor to a different area with more space and rewriting the lexicon module of the Data Processor to be paged.

VIII.2.2 Size of storable data

The demonstration system uses 250 pages. It started with 2346 pages immediately accessible. After loading the demonstration data 2096 pages are still free to use. The selection of pages in the demonstration system uses only pages of the removable disk of disk unit 1. 2448 pages are available on the fixed disk of disk unit 1. This area at the moment is used to store permanent data so that the demonstration can be refreshed if need be without re-loading the data base. Without doing reorganization of the disk unit 0, an additional 2106 pages could be added. At the present time this area is used for program development. The addressing structure is designed to include two more disk units. If these units are added, an additional 9792 pages can be added.

The amount of data page space is not a limiting factor in the present system demonstration.

VIII.3 Utility to Navy

VIII.3.1 Utility of Disciplined English Query Language

One of the principal values of the CCIS function is the improvement in efficiency of personnel. The latter is accomplished by providing a forgiving English interface to the staff personnel so that special training is not required and in organizing the information in a manner that facilitates rapid development of complex questions without an overburden of extraneous records and files to be examined.

VIII.3.2 Utility of Dynamic Update Techniques in CCIS

To improve the quality of information in the Command Center, data must be readily available, timely and accurate. To improve the timeliness of that data for analysis and planning, the CCIS architecture provides the Interface Processor for the function of receiving formatted data which can be provided from automated real time or batch sources and reformatting it for dynamic update of CCIS, primitive relation data base in the Data Processor.

The Data Processor is in the position of always being ready to accept formatted data from the Interface Processor when it is not busy processing a user query from the Query Processor. To minimize the amount of data which the Data Processor must hold in its data base, the Interface Processor provides a screening function so that extraneous and/or duplicative information is not sent to the DP. In the demonstration CCIS system, this function is demonstrated for NTDS position data. Here, only changes to course or speed are sent to the DP along with an initial position on a track. The DP, on request, then dead reckons position from last reported position. So the screening function performed results in sending only "significant" changes to the data base. Thus, a technique is provided which minimizes storage requirements for dynamic data and yet provides currency, or timeliness, of that data.

VIII.3.3 Utility of primitive knowledge

The structure of data bases has been the subject of considerable investigation for some time. Three basic types are discussed in the literature; the tree structure, the chain structure, and the relational (table) structure. Another structure not commonly discussed in the literature as a data base structure, but nonetheless used in many systems, is the arc link representation. Many of the ARPA English systems use this technique to represent their semantic structure.

One of the techniques of speeding up access to data in a data base has been to form inversion tables. This method can work on the tree structured data bases effectively. However, the inversion table creates a redundancy in the data. Also, the analysis of relational data bases results in primitive relations representing inversions on the big relations which are built. The arc link systems have the data arc floating around in virtual memory with little structure, hence not being well suited for structured storage of data. Recently, some have begun to note a basic similarity between each of these types of data bases. In the tree structured data bases the total inversion creates a complete redundancy in the data stored in the data base, progressing from no inversion at all with little or no redundancy, to a total inversion with complete redundancy. If one removes the original tree from the data base the inversion codifies the data. But the total inversion results in a number of primitive relations. When the chain structured data bases are analyzed and the address structure is symbolized out of hardware considerations the result is also in terms of primitive relations. When the arc linked data base structure, such as found in LISP [5] based systems, is restructured to collect arcs of the same type, this also results in the primitive relational structure.

Each one of the data bases discussed is a structure of sufficient complexity to represent collections of triplets. The primitive relational form is simply to group the triplets such that each group has one of its three fields identical. The common field is used to label the group.

The inefficiency of most data base systems can be traced to the data being stored in such a way that when retrieving data about any reference, much more data has to be retrieved than will be used. In the arc link systems an arc represents knowledge. When an aspect of an arc is accessed, the virtual memory brings in the arc and the page on which the arc is found. Because of the accident of generation the other arcs on the same page have little or no relationship to the arc of interest. Consequently a lot of overhead is required. In the tree structured data bases a similar process happens. The retrieval is accomplished bringing in a tree or subtree of which only one sub branch is of interest. The overhead in this retrieval is again high. In the classical non-primitive relational data base system each relation consists of many fields. To retrieve data required, the loading of entire rows of each relation is required, again resulting in much more data than necessary to answer the query. Lastly, the chained data bases sometimes required more data to be loaded than necessary. But this depended largely on the system designed and how carefully the designer removed redundancy: i.e., how much he made it a primitive relational data base unbeknownst to the user of the data base.

The primitive relational data base requires the least overhead in the retrieval of data in an unstructured query environment. This organization, together with special treatment of "belongs to", is very effective for retrieval of data and is closely matched to the structure of the English language. However, the primitive relational data base with all fields grouped for fast retrieval is triply redundant. Also, some relations are so pervasive that it is expedient to build them into the design of the system. The most common of this type is "belongs to". We do this by designating some names to be "classes" whereby the mechanism of belonging to can be implemented. The data retrieval is sensitized to also know about "belongs to" and ranges are expanded to include the domain when encountered in another class or relation. The designation of names of relations and individuals tells the English how the knowledge is structured about that name. The designation of a number relation indicates that only storage about the relation and one name is needed. In the English system the entities called number are not considered important enough to store data around the number for direct number recognition. This restricts the system so that the question "What is known about 7?" does not have a meaningful answer.

VIII.3.4 Utility of Microelectronic Technology

One of the questions to be answered by the development of CCIS was whether an English query system could be designed and implemented which had adequate response time, yet was not excessive in hardware costs. Microelectronic technology has developed in the last 7 years to provide design options that allow for the inexpensive high performance implementation of hardware support functions to which a CCIS like system could be matched. That system was implemented and demonstrated with the microelectronic circuit, the AMD 2901 bit slice processor, and used 4k ram memory chips. Its cost was that of a small system and response times were very tolerable, although no attempt was made to optimize performance. The demonstration clearly showed that an English query system could be implemented in inexpensive microelectronic hardware.

VIII.3.5 Utility of multiprocessor design

In the classical design of computer systems a trend was to design for a single processor with the functions swapped in and out. This centralized approach was questioned at the beginning of the project on the basis of whether one processor (constructed from microelectronic circuitry) could handle the computation load. As a consequence of the possibility that a single processor would not be fast enough, a triple processor approach was taken. It then became possible to evaluate the performance of a three processor system versus the sharing of a single processor for three functions. The three processors were descriptively called the Query Processor, whose function was to interface with the human user, the Data Processor, whose function was to interface with mass storage units, and the Interface Processor, whose function was to interface with the rest of the automated world.

One of the aspects noted in the exercise of the CCIS processors was that the Query Processor and the Data Processor were not simultaneously busy. This would suggest that the total function of both the Query Processor and the Data Processor could be squeezed into one processor. The problems noted with this were that there was not enough memory to accommodate both the programs and needed data area for the query processing and the data processing, and that there was not enough time to swap programs. Possibly large auxiliary read only program memory could alleviate the first of these problems, but this would require a very new architecture, for the address space would have to be increased and as a consequence the programs would have to increase in size also. The idea of using virtual memory would work only if all the programs were locked in. Swapping even when hidden by virtual memory would be intolerable, as it would cause the time to respond to a query to increase 5 or 6 times; i.e., the 40 seconds needed to answer a query would be increased to 4 or 5 minutes, an impossibly long time.

Another consideration in trying to combine the Query Processor functions and the Data Processor functions is the effect on the executive. At the present time the executive is mostly idle on each of the processors. When a query is asked, first the Query Processor executive becomes very busy and a little later the Data Processor becomes quite busy. If the two processors were combined, and the executive had to handle both, the time necessary to get back to an activity would be more than doubled, this in turn slowing the data retrieval from the disk by a factor of two. It will also make the editing of queries very sluggish. The total throughput would likely drop by a factor of two, where the throughput is measured as the number of questions answered in an hour. To avoid this drop in the throughput would require a radical redesign of the executive.

To combine the Interface Processor with the Query Processor or the Data Processor would present a modularity design problem. The programs in the Interface Processor change radically from application to application, from implementation to implementation, and from time to time. To integrate the Interface Processor functions with those of either the Query Processor or the Data Processor would cause a redesign of the unit whenever there was the slightest change.

Because of the multiprocessor design and concerns with modularity, a considerable effort was spent in the CCIS development to ensure independence of the query processing function (with the exception of commonality of lexicon English vocabulary) from the data base management function. A secondary concern was high performance for the data base search, the latter being of more concern as experiments were conducted with very large data bases. These concerns led us to consider different strategies for expressing formally the description of the communications between the Query Processor and the Data Processor. The ideal situation would have been to utilize a non-procedural notation which did not allow implicit decisions to be made by the Query Processor as to when to retrieve relations from bulk storage. An experiment was conducted in that regard using a formalized machine readable notation for Query-by-Example [6] for the communications between the QP, DP and IP. The notation forced use of another parsing scheme in the DP to extract relational data management commands to be executed and of heuristics to determine the order in which they were to be executed (if different than the order received by the DP). The software complexity involved would have required developing a strategy for paging large programs in the Data Processor as well as a strategy for breaking up the programs into pieces of length of one page or less (640 words or less). Because of these complexities, the decision was made to utilize the formal procedural notation described in VII.3.1 to ensure independence of the data base management function and to utilize the buffer memory, page directory and special processor instructions to attain high performance in data base search (a situation not tested in the course of the CCIS development).

VIII.4 Possible Capabilities

VIII.4.1 How to Design an English Query System

The key to designing an effective English query system is to have a well developed English grammar with a concept of the structure of a data base that matches the structure of English. Fred Thompson's Rapidly Extensible Language (REL) system [7] is by far the most developed of the natural language systems available. The REL system [7] is modularly designed with well defined data structure for the grammar, the parser, and the semantics. It has a very fast parser, in its original setting parsing in 100 milliseconds compared to a minute for the LISP [5] based systems. Wood's transition network system at BBN [8] is also well developed but its architecture is embedded in the LISP [5] language and system. Since its data base was not structured to be stored as separate from LISP [5], the data base size is limited to that of the address space of LISP [5]. The LADDER system at SRI [9] was considered but is also a LISP [5] embedded system. However, it does have the ability to access external data structures, for example, the Data Computer data base system [10]. It uses a parse by exception process, which is slow. It also has to go through several layers to retrieve the data.

It is unreasonable to design an English data base system in hardware only. In hardware only, it would run a thousand times faster than in a programmed machine. But such speed was shown not to be necessary. Programmable machines are much less expensive to design

and implement. Even multiple microprogrammable machines are better than hardware only. If a commercial machine is to be used, it should have a virtual machine architecture, writable control memory, and 576K bytes of lockable in core memory. If two machines are to be used each should have about 320K bytes of lockable in core memory. In both instances, the machine(s) should have instructions or writable control store. One also should have accessible a compiler/assembler facility and an easy way to load the machine(s) from the output of the cross compiler/assembler.

VIII.4.2 What Features are Important

Experience with CCIS demonstrations has shown an English interface for users of computerized data bases to be a very important concept. People who are not experts in machine languages or special languages for data bases have had no trouble learning how to use an English language query. Also, it was observed that in order to avoid excessive overhead in finding a solution or in finding critical data, the use of chained limiters and of clauses: i.e., use of the multiple depth English query: was of significant importance.

One of the most important subsidiary features has turned out to be the editing feature provided for the user when typing in a query. Without this, countless hours would have been wasted just in testing the system, let alone the hours wasted by the user of such a system. It is highly recommended that powerful editing features be part of any English query system.

VIII.4.3 What Features are not Important

The spelling correction process has turned out to be nearly unusable. It is not clear that in this limited arena it is useful. When typing a query or statement it cannot be evoked automatically and the only reason one would want it is to extend words. It is unclear what need a word extension capability would have for a speller. In the practical observations a word extender has worked out well without spelling correction. In the parsing, automatic spelling correction could be useful if the parse is not completed properly without spelling correction. However, there are two stikes against it. First, the complexity of the system would have to be increased to know when and when not to use the spelling corrector: and secondly, the parse explodes when all variations are permitted in the spelling. The time to do the parse would increase by an order of magnitude. The size of the parse chart would increase the same way. The size of the area set aside for containing the chart was designed to accommodate lengthy and complicated parses. Spelling is generally needed more on the lengthy and complicated statement -- just when the parse is likely to take the most space. It is not clear whether the benefits gained would outweigh the cost in response time, the need for more chart space and the added complexity to determine when and when not to use spelling correction. Normally when one asks a complex query he carefully reviews the statement anyway to ensure he asked the right question. In this situation most people find spelling errors. The line editing features allow for quickly correcting noted spelling errors without having the machine do it. Also a spelling corrector must be interactive because many corrections are not intended. This is another layer of complexity. Until other considerations and data are available spelling correction is an unnecessary function.

It does not appear that the yes and no questions are of much value. In all the demonstrations and scenarios only once did someone attempt to ask a yes or no question. A conjectured reason for this is that one has to have extreme confidence in the analytical capability

of the machine to ask a yes or no question. The information content of the answer is so low that one has to have other means of checking the accuracy of the answer. When asking questions that have data output the data itself generally provides a check on the correctness of the answer. Although the yes and no questions should be completed for completeness of the system, they should not be considered important.

VIII.4.4 What Organizational Changes Should Be Made

In the present system only one active data base is permitted in the system. To accommodate multiply developed data bases would require changes to the page directory system and the create function to allow for data that is labeled by the virtue of user access. But this in itself will not get very far in adding multiply developed data bases. The module that sends data to the user would have to be more sophisticated to screen out data that was added by other users. The lexicons of both the Data Processor and the Query Processor would have to be able to label the words so that it is known what users can use each word. The same spelled word of one user may have a different interpretation. One user uses a word as a class, another as a relation. In the Data Processor lexicon one user has the word associated with one logical record whereas another user has the same word associated with another logical record. It is clear that a new architecture is needed to support this feature.

A similar situation exists for developing the system to simultaneously accommodate several users of the same data base. Having different users of the same data base might require saving in the Query Processor the definitions of each separate user. Also it is possible that the lexicon structure may have to be developed to be layered, the base layers being common and the top layer added when a user desired to use the data base.

VIII.4.5 What Changes Should Be Made to the Modules

Changes to the page loading module can be made by improving loading of data from disk to load only what is needed and by pipelining page loads using buffering provided by the buffer memory. In addition, the page directory could be used to decide whether a page should be loaded according to a sort key or its context.

Clearly the Query Processor lexicon must be redesigned to avoid unnecessary code. It was noted that the principal search cycle involved calling routines that are general purpose and would take much more time to execute than direct machine code.

Also the spelling correcter in the Query Processor lexicon is shown not to be useful and should be removed to speed up the lexical process. At the present time the Query Processor critical processing load is in the lexicon module, not in the parser. A similar module in the Data Processor runs 100 times faster than the one in the Query Processor.

The lexicon in the Data Processor should be redesigned to be paged. At the moment it is totally contained in memory, which for large lexicons is not tolerable.

An intervening relation module should be added to the semantic processing section. At the present time the English is limping along without the powerful intervening relation process.

Pronoun processing should be added. This is not a large task. It consists of activating the functions that parsing uses to pull up prestored nouns and to systematically save each noun in a structured replacement list.

A module that creates a correspondence between the modules of the Query Processor and the Data Processor should be created. Many of the tools for this are already in place. What is still needed is a redesign of the lexicon for the Query Processor and the Data Processor.

Scratch pages should be used for temporary processing; i.e., when output results are obtained. Care should be taken to retain enough memory of temporary results, however, to handle pronoun processing and ellipses.

VIII.4.6 What Instructions Should Be Added to the Repertoire

An indirect call from register should be added ignoring the Y field. Thus if the D register is named it would have the effect of an indirect call to the Y field.

Correction to repeat instructions should be made not to clear the B register and to decrement B in a normal manner when terminating on a match.

A single instruction to fetch a byte from a byte address should be added as should a single instruction to store a byte to a byte address leaving other bytes alone. A special hardware card that could do a number of complicated data rearrangement operations required for extensive packing and unpacking of data could be used in support of these instructions and other such data rearrangements (see Section VII.1.7).

IX. Conclusions

IX.1 Utility of English

For the last fifteen years there has been concern as to whether the English language could be used to control and query automated Navy Systems. Research in utilizing English language had indicated a complexity that made the question very germane. Demonstration systems, such as the LADDER system at Stanford Research Institute (SRI) [9], have been developed that work slowly and require some of the fastest machines available. Could the Navy afford the machines and slowness that was indicated by the techniques developed? Meanwhile relatively unpublicized work the Rapidly Extensible Language System (REL) [7], was being done at California Institute of Technology by Dr. Fred Thompson and his students in developing an English language data base system that was fast and highly structured. In contrast, other systems developed were based on a process of exception and consequently were not structured and slow in response to queries. However, the system developed by Fred Thompson could only be executed on large expensive commercial machines. The question still remained as to whether the REL system could be of utility to the Navy.

What was demonstrated in this project was not only that the Thompson English system was real and not a figment of a canned demonstration, but also that it could be developed into a system that could be useful for shipboard placement in a task force and in some selected single ship systems.

IX.2 General Utility

In general the utility of a system to the Navy is difficult to measure. The principal mission of each platform generates the potential utility of a system for that platform. The CCIS system is no different than any other system in this respect. For instance, the CCIS system is too slow to be of much use in tactical systems. Improvements to the presently demonstrated speed could place the CCIS close to the threshold of utility in tactical systems

but the psychological interface was not designed with this in mind. On the other hand, the CCIS system could be used for inventory and purser functions, but it has much more power than is generally required for these functions. In between these two extremes, a number of command functions are adequately matched in both time and complexity to justify an English query system such as the CCIS system.

CCIS is more of an analytical tool than a planning and construction tool. Its principal forte is in analysis of the state of enemy forces (as in intelligence gathering), of the state of one's own forces, of the readiness condition of one's own ship or force, of the status of communication possibilities of the task force, and of possible engagements with an enemy. Hence, CCIS would have more potential in a task force command situation than in a shipboard command situation, because the essential information retained is of broader scope than that required by one ship. However, in such subsystems as communication on a ship a CCIS like system could be used effectively.

It was shown in a variety of demonstrations of CCIS over a period of time that the principal value of a CCIS function was in the improvement of timeliness of information, accuracy of the relationships of the data, and efficiency of the personnel. This is accomplished by collecting information directly into the system via the Interface Processor without it having to be loaded by manual and interfering techniques, by providing a forgiving English interface using the Query Processor to the staff personnel so that special training is not required and in organizing the information in the system in a manner that facilitates rapid development of complex questions without an overburden of extraneous records and files to be examined.

The general utility of CCIS can be assessed thoroughly only by a series of meaningful experiments conducted in both a simulated environment such as the Advanced Architectural Testbed at NOSC [11] can provide and in live operational use of the system. The next phase of CCIS development should see this type of assessment.

X. Acknowledgements

The authors wish to thank Mr. Robert Ebert, Mr. Henry Gok, Ms. Lois Sczepaniak and Ms. Elizabeth Trottier, all of the Naval Ocean Systems Center, for their many technical contributions towards ensuring the completion of the CCIS experimental system. Additionally, the authors express their appreciation to Drs. Bozena and Fred Thompson of California Institute of Technology for sharing their insights into the REL development with members of the CCIS project.

XI. References

[1] Canady, R.H. et al, "A Back End Computer for Data Base Management," Communications of the Association for Computing Machinery (CACM), October, 1974

[2] Dostert, B.H. and Thompson, F.B., "How Features Resolve Syntactic Ambiguity," Symposium on Information Storage and Retrieval, University of Maryland, April, 1971

[3] Kay, M., "Experiments with a Powerful Parser," Deuxieme Conference Internationale sur le Traitement Automatique des Languages, Grenoble, France, 1967

[4] Christy, D.O., Small, D.L., and Trottier, E., "Command Center Information Subsystem (CCIS) Operations Manual," NOSC TN 899, September, 1980

[5] Teitelmen, W., "INTERLISP Reference Manual," December, 1975

[6] Zloof, M.M., "Query by Example: A Data Base Language," IBM Systems Journal, December, 1977

[7] Dostert, B.H., REL - An Information System for a Dynamic Environment," California Institute of Technology REL Report 3, December, 1971

[8] Woods, W.A., "Transition Network Grammars for Natural Language Analysis," Communications of the ACM, Vol. 13, pp 591-606, 1971

[9] Sacerdoti, E.D., "Language Access to Distributed Data with Error Recovery (LADDER)," Stanford Research Institute (SRI) Artificial Intelligence Center Technical Note TN 140, Menlo Park, California, February, 1977

[10] Rothnie, J.B. and Goodman, N., "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases," Computer Corporation of America (CAA) Technical Report CAA-77-04, March, 1977

[11] "Advanced Command and Control Architectural Testbed Concept of Operations Plan," NOSC Code 8321 Internal Informal Report, July, 1977