N91-22796

Space Communications Artificial Intelligence For Link Evaluation Terminal (SCAILET)

Anoosh Shahidi
Sverdrup Technology, Inc.
Lewis Research Center Group
2001 Aerospace Parkway
Brook Park, Ohio 44142

1. INTRODUCTION

A software application to assist end-users of the Link Evaluation Terminal (LET) for satellite communications is being developed. This software application incorporates artificial intelligence (AI) techniques and will be deployed as an interface to LET. The high burst rate (HBR) LET provides 30 GHz transmitting/20 GHz receiving, 220/110 Mbps capability for wideband communications technology experiments with the Advanced Communications Technology Satellite (ACTS). The HBR LET and ACTS are being developed at the NASA Lewis Research Center in Cleveland, Ohio. The HBR LET can monitor and evaluate the integrity of the HBR communications uplink and downlink to the ACTS satellite. The uplink HBR transmission is performed by bursting the bit-pattern as a modulated signal to the satellite. By comparing the transmitted bit pattern with the received bit pattern, HBR LET can determine the bit error rate (BER) under various atmospheric conditions. An algorithm for power augmentation will be applied to enhance the system's BER performance at reduced signal strength caused by adverse conditions.

The HBR LET terminal consists of seven major subsystems (Fig. 1):

- Antenna subsystem
- Radio frequency (RF) transmitter subsystem
- RF receiver subsystem
- Control and performance monitor (C&PM) computer subsystem
- Local loopback subsystem at RF
- Modulation and BER measurements subsystem
- Calibration subsystem

The C&PM computer controls and monitors all the other subsystems through an IEEE488 interface. HBR LET experiments with the ACTS satellite will be initiated by users through the C&PM experiment control and monitor (ECM) software. The ECM software was developed on a Concurrent 3205 minicomputer in FORTRAN, which provides the end-user with the following capabilities:

- Individual instrument control
- Interactive interface used to communicate with the digital ground terminal
- Ability to conduct BER measurements

• User-controlled data acquisition

Programming scripts, defined by the design engineer, set up the HBR LET terminal by programming subsystem devices through IEEE488 interfaces. However, the scripts are difficult to use, require a steep learning curve, are cryptic and are hard to maintain. The combination of the learning curve and the complexities involved with editing the script files may discourage end-users from utilizing the full capabilities of the HBR LET system. In the following sections, I describe an intelligent assistant component of SCAILET that addresses critical end-user needs in the programming of the HBR LET system as anticipated by its developers. We will then take a close look at the various steps involved in writing ECM software for a C&PM computer and at how the intelligent assistant improves the HBR LET system and enhances the end-user's ability to perform the experiments. (Although a hypertext documentation module plays an important role in familiarizing end-users with all the LET HBR subsystems, the description of this module is beyond the scope of this paper.)

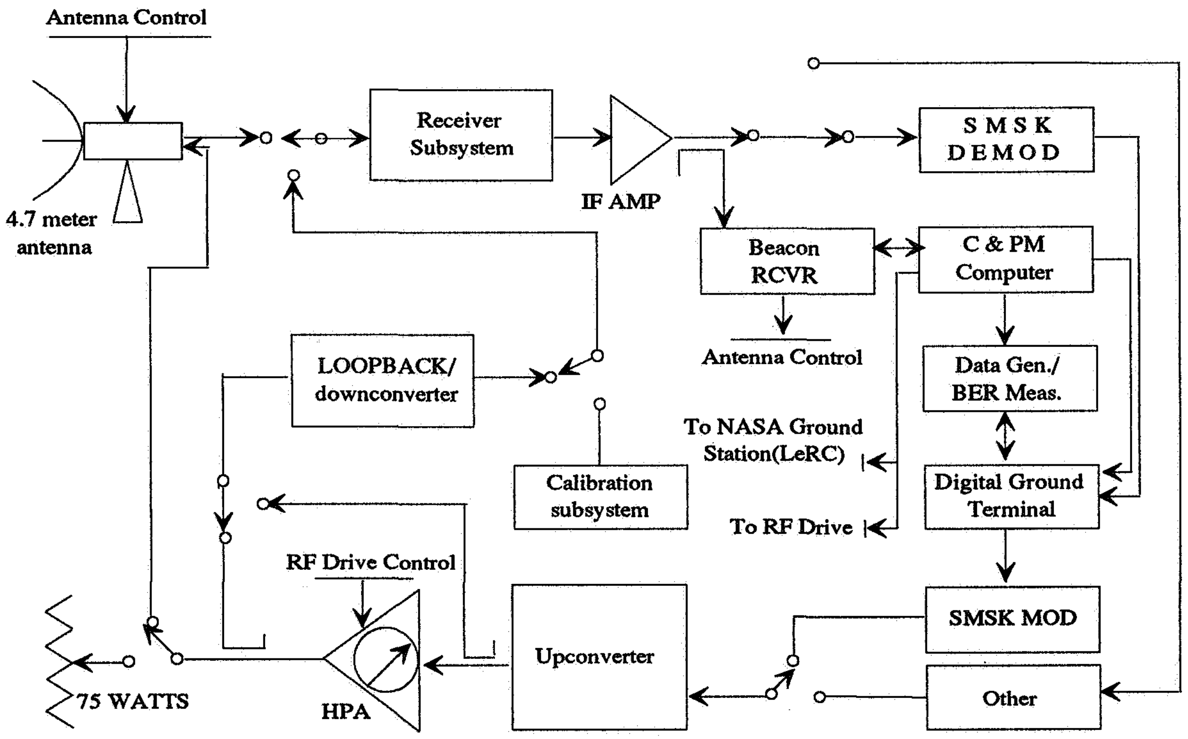

Figure 1. HBR LET Block Diagram

2. DILEMMA OF SOPHISTICATED INTERFACES

The fundamental dilemma in designing practical software is how to provide more power to the user without sacrificing ease of use. By designing intelligent interfaces the gap between the novice user who is a domain expert and the software can be bridged. In my research, I view the user as a "planner." Planning is a problem-solving technique, the process of finding a sequence of steps to accomplish some goal. In the system described here the computer user manipulates knowledge structures called *plans*. Plans are bundles of knowledge about the standard subtasks in a domain and are designed and organized from a typical user's point of view.

There are many different approaches that address planning in artificial intelligence. Two major approaches are hierarchical planning (e.g., Sacerdoti, 1974) and script-based planning (Shank, and Abelson, 1977). Most traditional planners try to generate a plan of action for a specific task in a domain. In contrast, my planning system is designed to provide a framework in which executable forms of domain tasks can be specified by using a planning hierarchy. A planning hierarchy provides a view of a procedural specification (a sequence of actions) that achieves a domain task which includes an explicit notion of levels of detail. Therefore, the novice user can be supplied with a portfolio of functionality - predefined high-level plans that omit many details - for using the software in each task domain.

Level 1:	John Doe's BER.Back to Back terminal Only
Level 2:	Generic BER, Generic Loopback, Generic Calibration
Level 3:	ECM Software
Level 4:	FORTRAN

Figure 2. Implementation Hierarchy for SCAILET Software. Each level represents a specialization built out of the primitives provided at the next lower level.

Figure 2 uses the example of SCAILET software to illustrate the application of a planning hierarchy to intelligent interfaces. The SCAILET development environment is a Gateway 486 personal computer and uses the Choreographer graphical user interface tool. SCAILET communicates with the Concurrent computer through a TCP/IP link. At the top level there is experiment-specific software for a particular project. The second layer is general categories of software (e.g., BER, calibration,and loopback). The third layer is the ECM software developed on the Concurrent computer. Finally, the fourth layer is FORTRAN. Each layer has its own "primitives," which encapsulate a sequence made up from primitives at the next level down.

I divide my computer users into three categories: domain expert, computer novice (DECN); domain novice, computer expert (DNCE); and domain expert, computer expert (DECE). For instance a DECN would be an experienced satellite operator who wants to use HBR LET to test video transmission on the ACTS satellite but has no previous experience on the Concurrent computer. A DECE could be an experienced communications engineer who uses our software applications to get data. A DNCE could be a computer scientist who knows how to use software applications but knows nothing about communications.

Currently, a researcher (DECN) begins with the ECM software layer (level 3); the two upper levels only exist in his or her mind. Occasionally, the only DECE has enough time to actually create a version of level 2 or level 1 for use by a DECN. More likely, the DECN must learn the ECM software's commands - which are the primitives for level 2 - that are necessary to perform actions at the higher levels. The highest level, which is customized to the needs of a specific user, can only be built by a DECE user who is proficient in levels 2, 3, and 4 and can make proper use of the primitives at each level. The point is to provide an environment that allows these higher levels - levels 1 and 2 - to be created by a DECE for use by a DECN.

2.1 Barrier Between DECN and DECE

This general model of software use represents domain tasks by nodes at the top level of a tree. These tasks are gradually "unpacked" by each subsequent level below, so that at some level the nodes describe a sequence of how the task domain is accomplished by using the ECM software. The specific goal of this research is to build a system that allows a DECN to start at level 1 and then facilitate his or her movements through the first 3 levels.

In using a general-purpose application a DECN user must first learn the commands and the macro language of the software. This step lacks context; the DECN is forced to deal with an abstract formalism divorced from his or her expertise. In the context of this example, typically force researcher John Doe (DECN) to use the ECM package. After mastering the ECM package's commands, John Doe has to mentally create intermediate plans that are primitive relative to his domain of expertise. Using these intermediate plans, John Doe must then create overall plans that are specific only to his situation.

The operators at level 1 are overall goals - they resemble goals in the task domain; while the operators at level 3 are data specialized goals - they are the commands of a general purpose applications package. The goal of this research is a framework and an environment in which the general purpose software can be specialized by a DECE, allowing the DECN to draw analogies between the task domain and his or her own knowledge at the overall goal level and initially avoiding abstract formalisms of the data specialized level.

2.2 STANDARD APPROACH TO BARRIER

Most intelligent interfaces that have tried to break down this barrier attempt to monitor user actions and try to infer an overall plan (e.g., Johnson, 1986). These systems are usually partial matching schemes, based on a plan catalog. The inferences made by partial matches allow the system to correct mistakes, to complete actions, and to infer "higher" plans. Since these systems mainly monitor low-level user actions and infer higher level plans, I refer to these intelligent interfaces as plan-matcher systems. One drawback of these systems is that matching always requires a detailed understanding of the user's goals - something that is not typically available. Also, since a novice's actions are erratic, plan-matchers that try to infer the overall plan from a novice's actions are often brittle. In the example a plan-matching system would have to monitor the ECM package primitives used by the user, refer to a standard set of novice goals, and infer plans that the novice is using to accomplish those goals. In most domains this a very difficult approach.

Besides the implementation difficulties, the overall approach, of necessity, focuses on helping a novice select a set of primitive steps in using the software. In contrast, the plan language of SCAILET will focus on the organizational and intermediate steps that allow the user to achieve an overall plan in the software domain. The actual primitive steps provided by the software package typically are never used by a novice. For instance, an engineer knows that he wants to transmit and receive video signals to and from the ACTS satellite but does not know the first thing about how to use the ECM application and how to structure his domain knowledge to achieve this high-level goal. Given the standard knowledge that has to be in the system explicitly, the traditional bottom-up approach makes the engineer work much harder than necessary. The traditional approach makes the engineer learn the low-level ECM commands and then tries to infer that all the engineer wants to do is to transmit and receive a signal. Since explicit knowledge for this task already exists, SCAILET plan language gives the engineer a transmitting and receiving plan directly and lets him specialize it. The SCAILET approach is explained in detail in the following section.

3. THE SCAILET PLAN LANGUAGE

3.1 Related Work

Programmer's Apprentice (PA) Project (Waters et. al., 1985) was one of many automatic programming research projects that had the goal of improving programmer productivity by developing tools based on AI techniques. This project also studied human-problem solving behavior by using the programming domain. The long-term goal of the PA Project has been to develop a theory of programming (i.e., how expert programmers understand, design, implement, verify, modify, and document programs). Although the Emacs knowledge-based editor (KBEmacs) in the PA project falls well short of the long-term goal, it offers interesting insight into the task of program construction. This knowledge-based editor is tightly integrated with a standard Emacs-style editor. This integration allows the programmer to freely intermix knowledge-based program editing with text-based and syntax-based program editing.

The most dramatic development in programming has been the development of high-level languages. Now, automatic programming is attempting to perform middle-level programming decisions automatically, and hence is, bringing about a second improvement. AI techniques make it possible to represent knowledge about programming in general and use this knowledge to understand particular programs. There are three main ideas that were implemented in KBEmacs:

(1) The assistant approach: Since fully automated programming is too hard to implement with what is now known, Waters et al., (1985) decided on an assistant approach, a division of labor between the programmer and his or her assistant, the KBEmacs. The assistant will take

care of the low-level operations, in order to make a human programmer more productive. This approach can also serve as a research ground for fully automated programming.

(2) Cliche: A cliche is a standard method of dealing with a task, a lemma or partial solution. Cliches are aggregates of code that achieve a stereotypical operation (e.g. searching a one-dimensional structure). When a cliche is used, it is instantiated by filling in the roles (i.e., input or output variables) with appropriate computational tasks. This creates a cliche that is specialized to the task.

An important aspect of cliches is reuse. Once something has been thought out and given a name, it can be used as a component in future thinking. Cliches provide an appropriate vocabulary for relevant intermediate and high-level concepts. Both man and machine are limited in the complexity of the lines of reasoning they can develop and understand. In order to deal with more complex lines of reasoning, intermediate-level vocabulary can be used that summarizes parts of the line of reasoning.

```
cliche EQUALITY_WITHIN_EPSILON
      Primary roles X, Y, EPSILON;
      comment "determines whether {the x} and {the y}
            differ by less than {epsilon}";
      constraints
            DEFAULT ({the epsilon}, 0.00001);
      end constraints;
begin
      return abs({the input x} - {the input y}) < {the epsilon};
end EQUALITY_WITHIN_EPSILON;
```

Figure 3. The Cliche EQUALITY_WITHIN_EPSILON From the PA Project.

A corollary of the cliche idea is that a library of cliches can be viewed as a machine-understandable definition of the vocabulary programmers use when talking about programs. In KBEmacs a large portion of the knowledge that is shared between the programmer and the computer is in the form of a library of algorithm cliches.

Figure 3 is a simple example of a cliche in KBEmacs. The `EQUALITY_WITHIN_EPSILON` cliche compares two numbers and returns a boolean value that specifies whether or not the numbers differ by less than a given epsilon. The roles that must be filled are `X` and `Y`, which are numbers to be compared with one another. A constraint is used to specify a default value for epsilon, but the user can specify his or her own. When cliches are used, they can be specified as an indefinite noun phrase (e.g. "`an EQUALITY_WITHIN_EPSILON of A and B`"). The primary roles definition lists the roles that must be specified and the order in which they must be specified. When this cliche is instantiated in a program the roles in braces(i.e., {.....}) are replaced with actual numbers. For example, if two values, A and B, were passed to this cliche, the comment would read " `determines whether A and B differ by less than 0.00001.`"

(3) Plans: Many AI systems make use of plans as a way of dealing with complex operations. As a representation, plans can deliberately ignore some aspects of a problem in order to make it easier to reason about other aspects of a problem. Plans are designed to represent two kinds of information in KBEmacs: the structure of the particular programs, and the knowledge about cliches. The two basic operations performed by KBEmacs are simple reasoning about programs (i.e., the source of data flow) and combining cliches together to create programs. The plan formalism is particularly designed to handle these operations (e.g., explicit arcs to show data flow make it easy to determine the source of the data). The user can construct programs by specifying the cliches that will be used within that program and specializing the roles of those cliches.

The plan formalism abstracts away from the syntactic features of a programming language and allows the programmer to focus directly on the semantic features of a program. This also has the added advantage of making the internal operations of KBEmacs language independent.

The major advantages of KBEmacs claimed by Waters et al. (1985)are:

• Programs can be constructed more quickly.

• Since programmers are limited in the amount of code they can produce per day, it is more productive to specify which cliches and what roles are used in a program than to write the code for each program. Since cliches are intended to be reused, the time invested making cliche libraries is worth while.

• A program built out of cliches is more reliable.

The major disadvantage of programming in cliches claimed by Waters et al. (1985) is that to get the full benefit of cliches, the programmer has to think in terms of them as much as possible.

KBEmacs was only a research prototype fraught with bugs. It was a 40,000 line Lisp program that had only a dozen cliches. Designing exact cliches to be used was a lengthy task.

The idea of plans as a method of program construction was studied by Soloway and Ehrlich (1985). Soloway's empirical studies suggest that expert programmers use two types of programming knowledge: (1) programming plans, which are generic program fragments that represent stereotypical action sequences in programming; and 2) rules of programming discourse, which capture the conventions in programming and govern the composition of the plans into programs. Experts seem to have a portfolio of these plans that can be used in problem solving.

As a representation, plans can deliberately ignore some aspects of a problem (i.e., syntactic structures) in order to make it easier to reason about other aspects of a problem (e.g., program design). One of the most powerful ideas in AI is the idea of a representation shift (shifting from text representation to plan representation). Initially, the novice would have to learn which language structures (i.e., to use syntactic choices) and then code those structures correctly (i.e., syntactic structures). Kurland et al. (1986) see the stage of learning the syntactic structures of a programming language as a barrier that poses significant difficulty for the novice. By removing or reducing the syntactic structures, it possible to reduce the cognitive load on the novice and allow him or her to focus on design issues. It is important for the novice to build a model of the computational process, so that he or she can predict actions and debug programs. When the cognitive load resulting from the syntactic structures of the language is reduced, the novice can focus on the flow of control and build his or her computation model.

3.2 The SCAILET Approach to the Barrier between DECN and DECE

The SCAILET approach provides a plan language with which to specify the layers of plans that make up a hierarchy for a range of tasks and goals. The plan language provides a knowledge-structuring scheme that will house a DECE's understanding and structuring of domain knowledge in a form usable by a DECN. Although this is much like an object hierarchy, but it includes enough information for the plan language to help a DECN use the hierarchy at multiple levels.

The DECE users will be provided with an environment where they can build domain knowledge into layers, so that the coded knowledge will not just be visible to another programmer, but will also be usable and visible to a DECN who wishes to achieve domain tasks. The top level of the SCAILET application will be goal driven: A DECN will be able to draw analogies between this level and domain tasks and will then use the top-level plans to achieve domain tasks. The application will be provided to the DECN as a set of plans that represent the tasks and subtasks which are of interest to him or her.

3.3 Current ECM Program Structure

A typical ECM program contains the following modules:

(1) Instrument definition software
- (a) Specifies instrumentation to be included in the experiment or test
- (b) Provides initial configuration and control parameters for each instrument

(2) Sequence definition software
- (a) Encodes the experiment as a sequence of commands
- (b) Distinguishes between the main sequence and subsequences

(3) Sequence execution software

BER sequence commands are given in the appendix. As you can see, it is very cryptic and requires a steep learning curve.

3.4 Formalizing SCAILET Programming Plans

The plan language system provides HBR LET system designers with an easy and structured way to construct plans as bundles of programming code with data and control links to other plans. Using this mode, a designer will be able to build his or her portfolio of programming plans that can be reused in various problems.

The plan language has the following design goals:

(1) Support the system designers in developing a portfolio of plans.

(2) Support the use of a plan-like composition of programs.

(3) Allow a distinct mechanism for data flow between plans.

(4) Allow a distinct mechanism for the flow of control within a program.

The plan formalism is based on object-oriented programming. For each plan there is a class that specifies the local data and operations of that plan. Plan class can then be specialized into instances, each with its own copy of local data. The DECE can sequence these instances into a particular execution order. Each plan object consists of four parts:

(1) PARENTCLASS - This is a hierarchical link that is part of the inheritance of the plan language.

(2) SLOTS - Each plan can have zero or more slots that specify data or plan links. Data slots are used to store data that are used during the execution of a plan. A plan slot is viewed as a component within the owner plan.

(3) INITIALIZATION - This part contains executable code that is performed once when control first flows through the plan.

(4) EXECUTION - This part contains executable code that is performed whenever control flows through the plan.

3.5 An example: BER testing

SCAILET plan language envelops procedural commands of ECM software in a declarative subplan that can be referenced by the end-user. Using ECM software, the user can conduct BER testing, loopback, and calibration. Figure 4 is a specialized BER plan language interface with three subplans for create files, instrument definition, and develop sequence. The structure of the program is intuitively apparent. ECM commands are bundled in subplans at the lowest level. The user does not have to know specific ECM commands. He or she only has to choose among various subplans. I intend to deploy SCAILET with a large number of typical plans. The user can then specialize or modify any of the SCAILET plans for his or her specific needs. Subplans know their parent plans, and hence tell the user when they are being misused. The system can also detect if there are any initial values missing for any subplans. Plans and subplans are designed by using graphical screens.

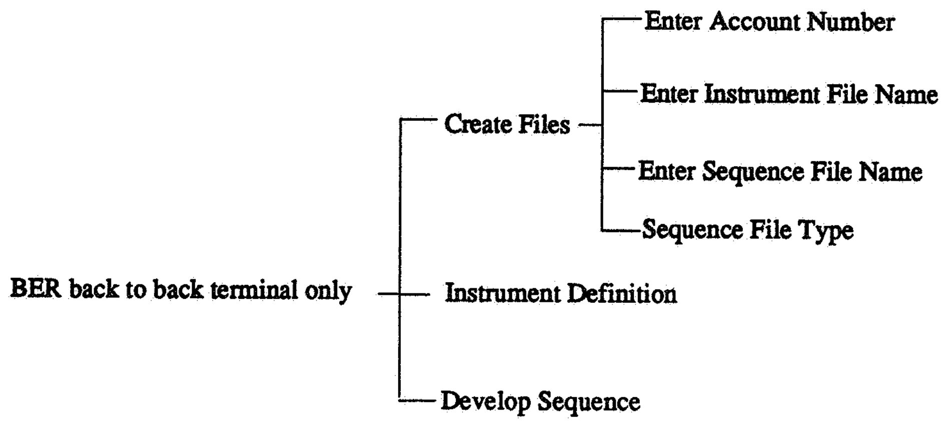

Figure 4. Specialized BER Plan Language Interface With Three Subplans (The user can click on a plan and "open up" its subplans).

4. FUTURE DIRECTION

By 1992, a complete portfolio of typical plans will be developed by using the system developers. Since the system will also have a completed hypertext documentation system, links between the plan language subplans, and their corresponding documentation will be established.

ACKNOWLEDGMENTS

This project is being developed at and funded by the Space Electronics Division of the NASA Lewis Research Center. I would like to thank Mr. Edward Petrik of NASA Lewis for his support and direction on this project. I would also like to thank Mr. Rich Rienhart of Analax Corp., the programmer of the ECM software, and Mr. Rich Schlegelmilch of the University of Akron for his system support, and his contributions to this project and to the hypertext documentation module.

REFERENCES

Johnson, W.L. (1986). "Intention-Based Diagnosis of Novice Programming Errors". Morgan Kaufmann Publishers, Los Altos, CA

Kurland, D.M., Pea, R.D., Clement, C., and Mawbey, R. (1986). "A Study of Development of Programming Ability and Thinking Skills in High School Students". *Journal of Educational Computing and Research*, 2(4), pp. 429-458. Baywood Publishing.

Sacerdoti, E. D. (1974). "Planning in a Hierarchy of Abstract Spaces". *Artificial Intelligence* 5:115-135.

Shank, R. and Abelson, R. (1977). "Scripts Plans Goals and Understanding." Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ

Soloway, E., Ehrlich, K., (1985) "Empirical Studies of Programming Knowledge". *Readings in Artificial Intelligence and Software Engineering*. Rich, C., Waters, R.C. (Eds) Morgan Kaufmann.

Waters, R.C. (1985) "The Programmer's Apprentice: A Session with KBEmacs". *Readings in Artificial Intelligence and Software Engineering*. Rich, C., Waters, R.C. (Eds) Morgan Kaufmann.

```
M::SDALET.DOC/111
SEQUENCE ARRAY  SDA(24,500)

INT - 1  SEQUENCE COMMAND NUMBER +LABEL #       ACTION NUMBERS
          - 100000  SINGLE EXECUTION             21  SET A PARAMETER
          - 200000  START LOOP SEQUENCE          60  ZERO POWER METERS
          - 299999  END LOOP SEQUENCE            61  STEP A PARAMETER
          - 300000  CALL SUB-SEQUENCE           101  WAIT
          - 399998  END SUB-SEQUENCE            102  GOTO STATEMENT
          - 900000  BEGIN STOP SEQUENCE         110  CHECK A PARAMETER
          - 999997  END SEQUENCE COMMAND        301  START DATA GENERATOR
INT - 2 NUMBER OF TIMES TO EXECUTE LOOP         302  STOP DATA GENERATOR
INT - 3 NUMBER OF STEPS IN LOOP SEQUENCE        303  STOP DATA CHECKER
INT - 4 ACTION TO BE TAKEN                      311  SET NUMBER OF ERRORS FOR GENERATOR
                                                313  PERFORM BER MEASUREMENT
INT - 5 CHECK PARAMETER - ARRAY SLOT OF INST    315  DGT COMMAND
        SET PARAMETER - ARRAY SLOT OF INST
        STEP PARAMETER - ARRAY SLOT OF INST
        START DG - ARRAY SLOT OF INST
        STOP DG - ARRAY SLOT OF INST
        STOP DC - ARRAY SLOT OF INST
        SET ERRORS -  ARRAY SLOT OF INST

INT - 6 SET PARAMETER - WAVETEK -> MARKER EDGE 1-RISING 2-FALLING
        WAIT - TOTAL TIME IN SEC
        SET ERRORS - NUMBER OF ERRORS TO SEND TO DG

INT - 7 SET PARAMETER - WAVETEK -> OPTION 0-15
                        BEACON RECEIVER -> OPTION 1-6
        STEP PARAMETER - INCREMENT / DECREMENT FLAG  1 UP  2 DOWM

REAL -8 SET PARAMETER - VALUE / PS VOLTAGE
        STEP PARAMETER - INC VALUE
        CHECK PARAMETER - LOW LIMIT


REAL -9 CHECK PARAMETER - HIGH LIMIT

INT -10 FUNCTION TO BE PERFORMED (CHAR)

INT -11 CHECK PARAMETER - LABEL TO GOTO IF UNDER RANGE
        GOTO - LABEL NUMBER TO GOTO
        END - TOTAL NUMBER OF SEQUENCE COMMANDS

INT -12 CHECK PARAMETER - LABEL TO GOTO IF OVER RANGE

CHAR-13 SUB-SEQUENCE - FILENAME
    -17 DGT COMMAND  - 20 CHARACTER COMMAND
        END SEQUENCE - SDA FILE NAME
```

```
INT -18 SET PARAMETER - TOGGLE SWITCH -> INPUT
                        WAVETEK MARK   -> CHANNEL MODE 0-12 OF WTPM TO SET
                        FREQUENCY -> CHANNEL MODE 0-12 OF WTPM TO SET
        CHECK PARAMETER - MODE OF WTPM TO CHECK 0-12
        START DG - DATA TYPE
        PERFORM BER - # OF MEASUREMENTS PER EB/NO

INT -19 SET PARAMETER - TOGGLE SWITCH -> OUTPUT
        START DG - DATA RATE
        PERFORM BER - DG NUMBER

INT -20 PERFORM BER MEASUREMENT - TIME TO WAIT BETWEEN READINGS
        START DG - DESTINATION



INT -21 START DG - MODEM RATE
        SET PARAMETER - EBNO -> MODEM RATE

INT -22 START DG - BURSTED

INT -23 START DG - CONTINUOUS

INT -24 SET PARAMETER - SWITCH SETTINGS = BIT POSITIONS 1,10
```

WAVETEK PARAMETER OPTIONS/MODE

0	- FREQUENCY
1	- REFERENCE DELAY
2	- CURSOR DELLAY
3	- START DELAY
4	- WINDOW DELAY
5	- REFERENCE POWER
6	- PULSE RISE TIME START
7	- PULSE RISE TIME END
8	- PULSE FALL TIME START
9	- PULSE FALL TIME END
10	- PULSE WIDTH TIME START
11	- PULSE WIDTH TIME END
12	- MARKER 1
13	- MARKER 2
14	- MARKER 3
15	- MARKER 4

WAVETEK MODES

0	-
1	- CW A&B
2	- CW A
3	- PEAK A&B
4	- CW B
5	- GRAPH A
6	- PEAK A
7	- GRAPH B
8	- PEAK B
9	- MARKER A
10	- MARKER B
11	- PULSE A
12	- PULSE B

BEACON RECEIVER OPTIONS

0	- FREQUENCY
1	- 2ND LO VFO
2	- VIDEO ATTENUATION
3	- SIG STR OFFSET
4	- CARRIER INDICATOR
5	- SIG STR SLOPE