START with P = 1000

Honeywell

COMPUTER AND COMMUNICATIONS GROUP

PROGRAM SPECIFICATION

Series 16 Basic Compiler/Interpreter

1. SCOPE:

This document describes an interactive interpretive compiler for the BASIC language.

2. APPLICABLE DOCUMENTS:

The following documents (latest revision) form a part of this specification to the extent specified herein:

70 110 011-397, H-316 Product Specification (includes ASR control Unit)
70 110 010-527, DDP-516 Product Specification
70 110 010-523, DDP-516 Control Unit Product Specification
70 130 072-442, DAP-16 and DAP-16 Vol 2 Assembler Manual

3. DEFINITIONS:

3.1 Basic

Acronym for Beginners All-purpose Symbolic Instruction Code. BASIC is an engineering and science oriented programming language.

3.2 Interpretive Compiler

A compiler which compiles and executes source without producing object code.

4. DESCRIPTION:

The Basic interpretive compiler provides an interactive environment in which users may compose, edit and execute programs written in the BASIC language. The compiler assembles an entire program and calls for execution of the enter language statements to be executed immediately. Statements are checked for syntactic and logical errors during both entry and execution. All errors are reported to the user when detected.

5. REQUIREMENTS:

5.1 Environment.

5.1.1 Required Hardware.

1 - H316 or DDP-516 Central Processor Model 316/516-01

REV AUTH	WR	REL ENG	SIZE A	CODE IDENT 07577	DOC NO
	SPEC	DES ENG	SCALE	REV	SHEET OF

1 - 4K of Core Memory
1 - ASR-33 or ASR-35 Console Typewriter Model 316/516-53/55

5.1.2 Supported Hardware.

Up to 16K of Core Memory.

5.1.3 Required Software.

None.

5.1.4 Supported Software.

Will read in and execute any program written in BASIC. A method is included for linking to programs written in FORTRAN or DAP-16.

5.1.6 Special Relationships.

This system can be configured to execute under the RTX-16 Executive or the OP-16 Operating System.

5.2 Performance.

5.2.1 Design Goals.

This compiler will have as many of its features as possible operative in a 4K version. The following features will definitely be included in the 4K version: READ, PRINT, DATA, LET, GO TO, IF-THEN, FOR-TO-STEP, NEXT, GOSUB, RETURN, END, STOP, DEF, DIM, interactive operation, and source program editing.

This compiler is designed to be upward-compatible to a multi-user configuration.

A subset of this language will also be a subset of the 1648 Time Sharing BASIC.

Input/Output will be handled by an IOS package to allow expansion to new devices.

5.2.2 Timing Considerations.

This compiler is designed to be compatible with the multi-programming environment of OP-16.

5.3 Restrictions.

All programs must be written in BASIC as described herein. A subset of this language operates properly in a computer with 4K of memory.

6. PROCEDURES:

6.1 Use.

6.1.1 Stand-Alone Version.

The stand-alone version is loaded from self-loading paper tape and executed starting at a certain location. All further communication is via the ASR console.

6.1.2 OP-16 Version.

BASIC is called from the console using the RTX Keyboard program. All further communication is directly with BASIC via the console. A "QUIT" command is provided for terminating BASIC.

6.2 Data Formats.

6.2.1 Input Data Formats.

6.2.1.1 Statement Inputs

Detailed statement formats are described under Syntax (6.2.5). Statements must be terminated by a carriage return. A line feed is optional. The sequence X-OFF, RUBOUT may optionally follow the carriage return or carriage return, line feed. A null statement has a special meaning--see Section 6.2.2.1.

6.2.1.2 Data Inputs

The data must be in the format described for numbers under Syntax (6.2.5). Data must be terminated by a carriage return. A line feed is optional. The sequence X-OFF, RUBOUT may optionally follow the carriage return or carriage return, line feed.

6.2.1.3 Input Error Correction

A leftward arrow or succession of leftward arrows may be used to delete one or a succession of preceeding characters. A commercial at (@) may be used to delete the entire line.

6.2.2 Output Data Formats.

6.2.2.1 Request for Statement Input

BASIC outputs a question mark followed by an X-ON each time it is ready to receive a statement input. If the previous statement was a null statement, the question mark is output but the X-ON is not. This feature allows orderly termination of program input from paper tape.

6.2.2.2 Request for Data Input

BASIC outputs an exclamation point (!) followed by an X-ON each time it is ready to receive data input.

6.2.2.3 Listing Output

Statements are output by order of line numbers with all spaces except those in comments and messages deleted. Numbers are output in a standard format described in Section 6.2.6. Each statement is terminated with the sequence carriage return, line feed, X-OFF, RUBOUT. The program is terminated with a null statement.

6.2.2.4 Data Output

Data are output in the standard format described in sections 6.2.6 and 6.2.7.3.

6.2.3 Error Message Formats.

6.2.3.1 Stop/End Message

Whenever a STOP or END is encountered, the line number and the word EXIT is printed. If the program terminates by executing the highest numbered statement without encountering either a STOP or an END, line number 0000 and EXIT are printed.

6.2.3.2 Error Messages

The following message is printed for each error encountered during either statement input or execution:

ERROR AA LINE BBBBB

where AA stands for an error code and BBBBB for a line number.

6.2.4 Internal Data Formats.

Programs are stored internally as compressed statements. A line number table gives the starting byte location in core of each statement.

6.2.4.1 Variable Storage

All variables are stored as standard floating-point numbers (two words per variable). This format is described in the DAP-16 Mod 2 assembler manual.

6.2.4.2 Constant Storage

Constants are store as either one-word fixed-point numbers or two-word floating-point numbers. The preceding byte indicates whether the constant occupies one word or two.

6.2.4.3 Control Bytes

Bytes with special values between 0 and 127 are used as compressions for the standard statement types, indications of constants and variables, etc.

6.2.4.4 Alphanumeric Bytes

Bytes with USASCII significance (values between 128 and 255) are used to hold variable names, arithmetic operators, etc.

6.2.5 Statement Syntax.

The following syntax is described in Backus Normal Form. Quantities enclosed within diamond brackets (< >) are metalinguistic variables representing a class of syntactic variables. A colon followed by an equal sign (:=) means "is defined as." A vertical line (|) connecting two elements means logical OR. An element or group of elements enclosed in square brackets followed by a subscript and superscript $([]_a^b)$ may be repeated any number of times within the inclusive range of the subscript and superscript. All letters and symbols not enclosed in diamond brackets are actual characters of the syntax. Blanks are ignored anywhere within any BASIC statement except a comment statement or a message. Thus GOTO, GO TO, and G OT O are all equivalent.

<alphabetic character> := A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<digit> := 0|1|2|3|4|5|6|7|8|9

<special character> := +|-|*|/|↑|=|(|)|<|> | .|,|;|Δ

<integer> := $[\text{<digit>}]_1^9$

<decimal number> := $[\text{<digit>}]_1^{N<9}$. $[\text{<digit>}]_0^{9-N}$

<sign> := $[+|-]_0^1$

<exponent> := E <sign> $[\text{<digit>}]_1^2$

<number> := $[\text{<integer>}|\text{<fraction>}|\text{<decimal number>}]_1^1$ $[\text{<exponent>}]_0^1$

<signed number> := <sign> <number>

<simple variable> := <alphabetic character> $[\text{<digit>}]_0^1$

<fraction> := . <integer>

<subscripted variable> := <alphabetic character>(<expression> [,<expression>] $_{0}^{\infty}$)

<variable> := <simple variable>|<subscripted variable>

<function name> := SIN|COS|TAN|ATN|EXP|ABS|LOG|SQR|INT|RND|SGN| FN<alphabetic character>

<function term> := <function name> (<expression>)

<term> := <number>|<variable>|<function term>|(<expression>)

<involution factor> := <term>|<involution factor>↑<term>

<multiply factor> := <involution factor>|<multiply factor> [* | /] $_{1}^{1}$ <involution factor>

<expression> := <multiply factor>|<sign><expression>|<expression> [+|-] $_{1}^{1}$ <involution factor>

<assignment statement> := LET <variable> = <expression>|<variable> = <expression>

<READ statement> := READ <read list>

<INPUT statement> := INPUT <read list>

<read list> := <variable> [, <variable>] $_{0}^{\infty}$

<DATA statement> := DATA <number list>

<number list> := <expression> [, <expression>] $_{0}^{\infty}$

<RESTORE statement> := RESTORE

<PRINT statement> := PRINT[<print list>] $_{0}^{1}$

<print list> := <print item> [[,|;] $_{1}^{1}$ <print item>] $_{0}^{\infty}$ [,|;] $_{0}^{1}$

<print item> := <expression>|<message>|<message><expression>| TAB (<expression>)

<message> := "[<alphabetic character>|<digit>|<special character>] $_{1}^{\infty}$"

<comment> := REM [<alphabetic character>|<digit>|<special character>] $_{0}^{\infty}$

<GOTO statement> := GOTO <line number>

<GOSUB statement> := GOSUB <line number>

<RETURN statement> := RETURN

<ON statement> := ON<expression> GOTO <line number> [,<line number>] $_{0}^{\infty}$

<line number> := [<digit>] $_{1}^{5}$

<logical IF statement> := IF <expression> <relational operator> <expression> [THEN <statement body> [: <statement body>] $_{0}^{\infty}$ | THEN <line number>|GOTO <line number>] $_{1}^{1}$