Electronic Devices Division
3310 Miraloma Avenue
P.O. Box 3669
Anaheim, CA 92803

Toll Free (800) 854-8099
in California (800) 422-4230

Rockwell International

Rockwell

instant Pascal user's manual

AIM 65

July 1981

instant Pascal user's manual

AIM 65
advanced interactive microcomputer

Rockwell International

...where science gets down to business

instant Pascal
user's manual

AIM 65
advanced interactive microcomputer

Rockwell International

...where science gets down to business

Document No.
29650N85

TABLE OF CONTENTS

TABLE OF CONTENTS (con't)

LIST OF FIGURES

LIST OF TABLES

SECTION 1

INTRODUCTION

Pascal is a powerful high level computer programming language originally designed for educational purposes. Developed by Niklaus Wirth of the ETH Technical Institute of Zurich in 1970 - 1971, Pascal has gained acceptance world-wide as a standard language to teach computer programming. The rich variety of Pascal language features allows a wide range of data structures to be specified and complex algorithms to be implemented. Programming in Pascal using structured programming techniques produces programs that are easy to write, understand and maintain. The wide-spread teaching of the language, coupled with the increased productivity of the programmer and the improved reliability of the generated program, is causing Pascal to be increasingly adopted in industrial and scientific applications as well as in the classroom.

The Pascal language defined by Niklaus Wirth is recognized as Standard Pascal and is referred to as such in this manual. Numerous world-wide installations of Pascal on main-frames, minicomputers, and microcomputers have resulted in many variations of the language in the form of extensions (added language features beyond Standard Pascal) and restrictions (Standard Pascal features that have not been implemented). These installations typically require a large amount of computer resources, e.g. up to 64K of RAM, a floppy disk, and a CRT-based terminal.

AIM 65 Instant Pascal* is a unique implementation of an extensive subset of Standard Pascal which combines the immediacy of ROM based system software, interactive debug facilities at the source code level, on-board AIM 65 printer and display peripherals along with low-cost expansion memory , to provide a complete Pascal educational, development and application system.

*Instant Pascal is a trademark of Melvin E. Conway

1.1 AIM 65 INSTANT PASCAL OVERVIEW

Instant Pascal is an implementation of the Pascal programming language for the Rockwell AIM 65 Microcomputer. The firmware is embodied in a five-ROM set which is installed by means of the AIM 65 expansion connector. Instant Pascal incorporates facilities to write and debug programs entirely at the source language level. These include source-level editor, statement and assignment trace and immediate source-statement execution for examination and modification of data.

Instant Pascal is a highly interactive programming tool which substantially extends the power of the AIM 65. A major subset of the Standard Pascal language, it incorporates all of the simple and structured statements, and the most widely used simple and structured data types. Extensions to the language permit direct control of memory-mapped I/O of the microcomputer systems and allow interfacing to machine-language programs developed with the AIM 65 assembler.

Features of Instant Pascal include:

- All editing occurs at the source language level
- Translation at program input and output obviates any need to know internal program formats
- Lister contains a built-in prettyprinter which displays program and data structure
- Source programs may be conveniently modified during the program testing process
- Source-level trace lists statements as they are executed

- Independent assignment trace lists values as they are changed
- Immediate statement execution permits operator examination and modification of data
- String extensions to language facilitate development of interactive programs

AIM 65 Instant Pascal implements a substantial subset of the Standard Pascal defined in the Jensen and Wirth book "Pascal User Manual and Report" (see Appendix F). Extensions to their standard included in AIM 65 Instant Pascal and language features in the standard but not included in AIM 65 Instant Pascal are described in detail in Section 7.

AIM 65 Instant Pascal is both a compiler and an interpreter. It compiles source statements written in Pascal directly from the keyboard, from the AIM 65 Text Editor or from external media (audio tape or user defined) into an internal format. The internal format contains both the object code to be interpreted upon execution and source code identifiers for formatted printout to support debugging and editing at the source code level.

1.2 AIM 65 INSTANT PASCAL USER'S MANUAL CONTENTS

Section 1, Introduction, introduces the AIM 65 Instant Pascal product with language features, extensions and restrictions summarized. Common Pascal language and programming books are also referenced.

Section 2, Installation and Initialization, tells how to install the AIM 65 Instant Pascal ROMs and how to enter, re-enter and exit Instant Pascal.

Section 3, Instant Pascal Operation, explains how to operate AIM 65 Instant Pascal in conjunction with the AIM 65 Debug Monitor and Text Editor. This includes program input, output, editing, checking and execution.

Section 4, Program Debugging, describes how to use Instant Pascal commands to debug a program written in Pascal. This includes statement and assignment tracing, as well as immediate statement execution.

Section 5, Instant Pascal Text Units, explains how the Pascal Source code is treated for the purpose of editing, checking and execution.

Section 6, Diagnostic Messages, describes the meanings of diagnostic messages that may occur during translation, execution and listing as well as the effect on Instant Pascal operation.

Section 7, Instant Pascal Language Definition, defines the language extensions and restrictions with respect to Standard Pascal.

Appendix A, Summary of commands, summarizes all commands used for source program entry, editing and saving, as well as debugging and execution.

Appendix B, Language Summary Tables, includes several summary tables of language words, symbols and capabilities.

Appendix C, Page 0 Memory Map, defines variables that are alterable by the user.

Appendix D, ASCII Character Set, lists the hexadecimal and decimal codes for letter, number and special characters as well as control commands.

Appendix E, Execution-Time Diagnostics, defines the meanings of error codes displayed during Pascal program execution.

Appendix F, Selected Bibliography, lists several reference books on Pascal and programming.

1.3 REFERENCE DOCUMENTS

Users of this manual should have the manuals supplied with their AIM 65, particularly the AIM 65 User's Guide (29650 N36), the R6500 Programming Manual (29650 N30) and the R6500 Hardware Manual (29650 N31).

Readers who are unfamiliar with Pascal are advised to read one of the many tutorials on this language (See Appendix F). These three books are especially recommended:

- Cherry, George W., Pascal Programming Structures. Reston Publishing Company, Reston, VA, 1980.

- Grogono, Peter, Programming in Pascal. Addison-Wesley Publishing Company, Reading, MA, 1980.

- Fox, D. and Waite, M., Pascal Primer. Howard W. Sams & Co. Inc., Indianapolis, IN, 1981.

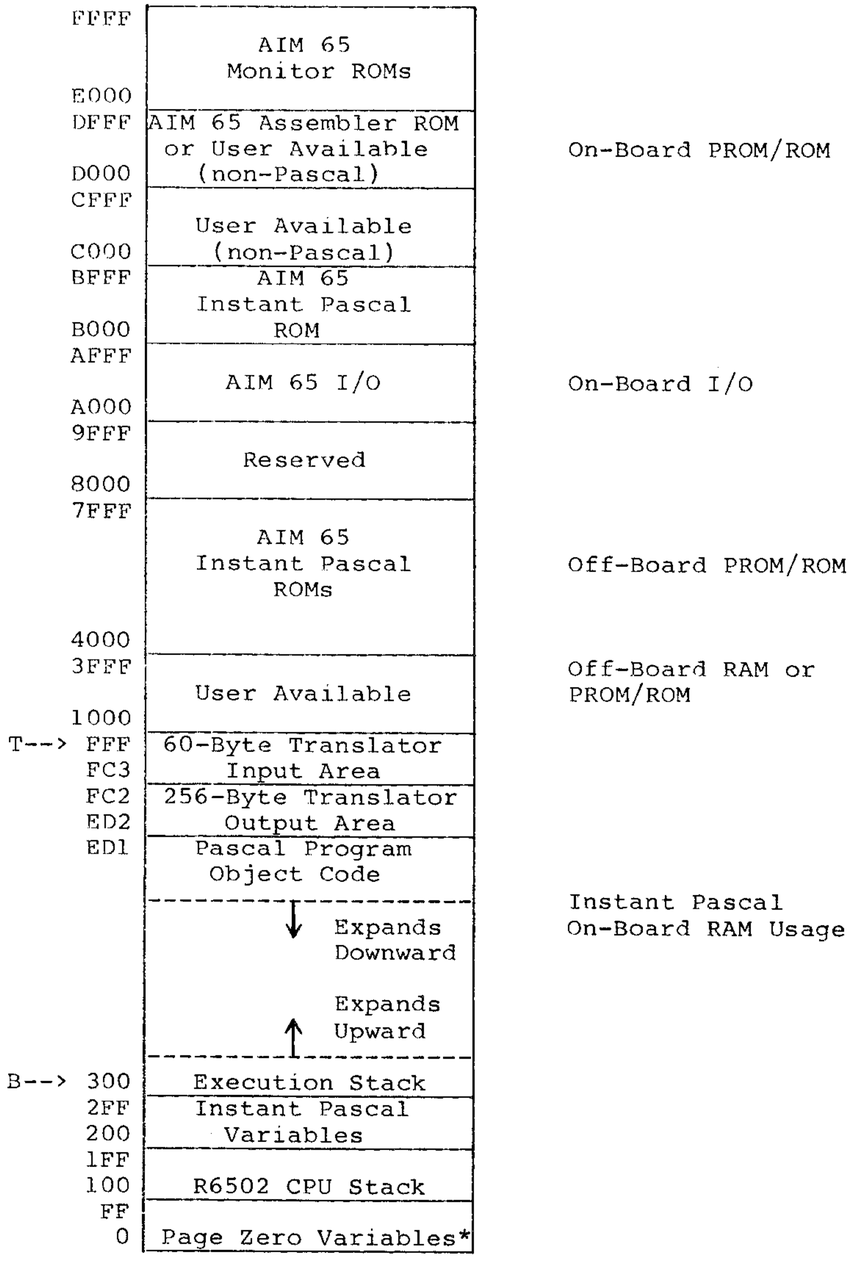
SECTION 2

INSTALLATION AND INITIALIZATION

The AIM 65 Instant Pascal is provided in five Rockwell R2332 4K-byte ROMs. Four of the ROMs must be installed in a PROM/ROM module external to the AIM 65 while the other one may be installed on-board the AIM 65 Master Module. Figure 2-1 shows the memory map of AIM 65 Instant Pascal. Note that Instant Pascal operates in conjunction with the AIM 65 Monitor, therefore the Monitor ROMs must be installed. Since Instant Pascal can link to machine code, the optional AIM 65 Assembler may reside on-board concurrently with Instant Pascal, but is not required. After installing the Instant Pascal ROMs and connecting the PROM/ROM module to AIM 65, Instant Pascal is ready for operation.

2.1 INSTALLING OF INSTANT PASCAL ROMS

Before removing the ROMs from the shipping package, be sure to observe the handling precautions listed in Section 1.4 of the AIM 65 User's Guide. Since MOS devices may be damaged by the inadvertent application of a high voltage, be sure to discharge any static electrical charge accumulated on your body by touching a ground connection (e.g. a grounded equipment chassis) before touching the ROMs or the AIM 65. This precaution is especially important if you are working in a carpeted area or in an environment with low relative humidity. Ensure that power is turned off to the AIM 65. Carefully remove any PROM or ROM device that is installed in socket Z26 on the AIM 65 Master Module. Remove the ROMs from the shipping package and verify that the pins are straight and free from foreign



T = Top of the user program including Translator input and output areas (default value = $0FFF)
B = Bottom of the user-program (default value = $0300)

* See Appendix D

Figure 2-1. AIM 65 Instant Pascal Memory Map

material. Install the ROMs on the AIM 65 Master module and on the PROM/ROM Module in accordance with ROM address ranges listed in Table 2-1. You can install R32P6 on-board the AIM 65 in socket Z26, however the other ROMs must be installed off-board. After the ROMs are installed in the PROM/ROM module, connect the module to the AIM 65 Expansion connector in accordance with the PROM/ROM or adapter module user instructions.

The 4K bytes of AIM 65 on-board RAM provide 2764 bytes available for the application program after subtracting memory dedicated to page 0, page 1 and AIM 65 Instant Pascal overhead (see 3.3.1). This allows a Pascal program of about 1800 source characters in length to be entered when memory is dedicated to the Pascal program object code, i.e. compiler input is from the keyboard or external media, or about 1100 source characters when the compiler input is from memory, i.e. the text buffer in the AIM 65 Text Editor. For larger programs, add off-board expansion RAM from $1000 - $3FFF.

NOTE

It is recommended that a Rockwell RM 65 16K PROM/ROM Module be used installed in an RM 65 card cage and connected to the AIM 65 through an RM 65 Adapter/Buffer Module. Additional RAM can easily be added using RM 65 8K Static RAM modules or an RM 65 32K Dynamic RAM module. The RM 65 modules that may be used and their part numbers are:

RM65-3216	16K PROM/ROM Module*
RM65-3132	32K Dynamic RAM Module
RM65-7004	4-Slot Piggyback Module Stack
RM65-7008	8-Slot Card Cage
RM65-7016	16-Slot Card Cage
RM65-7101	Single Card Adapter*
RM65-7104	Adapter/Buffer
RM65-7116	Cable Driver Adapter/Buffer

*Minimum Expansion Set for AIM 65 Instant Pascal operation.

Table 2-1. AIM 65 Instant Pascal ROM Addresses

ROM No.	Address	Module (1)	Socket
R32P2	4000-4FFF	RM 65 PROM/ROM	Z12 (2)
R32P3	5000-5FFF	RM 65 PROM/ROM	Z13 (2)
R32P4	6000-6FFF	RM 65 PROM/ROM	Z14 (2)
R32P5	7000-7FFF	RM 65 PROM/ROM	Z15 (2)
R32P6	B000-BFFF	AIM 65 Master Module	Z26

NOTES

(1) ROMs may be installed in any PROM/ROM module that operates with R2332 ROMs in the corresponding address range.

(2) Recommended sockets - ROMs may be installed in any socket that can be configured for the corresponding address ranges.

2.2 ENTERING, EXITING, AND RE-ENTERING INSTANT PASCAL

2.2.1 Entering Instant Pascal (5 Key)

Press 5 to enter Instant Pascal from the AIM 65 Monitor when the Monitor prompt "<" is displayed. When power has just been turned on, or when Instant Pascal has not been previously entered, the AIM 65 will respond with the first of two initialization questions:

```
<5> MEMORY SIZE?
```

Press RETURN to enter the default value of 4096. If more contiguous RAM than 4K bytes is available and you wish to take advantage of it, enter the number of bytes in decimal, up to a maximum of 16384. The second question,

```
WIDTH?
```

requests the page width for program listings and data output. Press RETURN to enter the default value of 20 for the AIM 65 printer. If the KB/TTY switch is, or is about to be, set to TTY (see AIM 65 User's Guide, 1.9.2), first enter the number of printer columns in decimal. (This value may be changed later by using the +<W> command in Instant Pascal; see 3.3.4).

2.2.2 Exiting Instant Pascal (ESC Key)

Any time Instant Pascal is in an input wait state, pressing ESC will return control to the AIM 65 Monitor. This can occur when the AIM 65 is waiting for an Instant Pascal command after the "+<" prompt or when an Instant Pascal program is waiting for input data. Also, a listing produced by the +<L> command or a program trace may be aborted and control returned to the AIM 65 Monitor by holding down the ESC key until the printing stops.

2.2.3 Re-entering Instant Pascal (5 Key)

Whenever Instant Pascal is exited to the AIM 65 Monitor, you can return to Instant Pascal without loss of the program in memory, provided that the RAM occupied by the program is not disturbed. Just press 5 when the Monitor prompt is displayed. Instant Pascal checks locations $60 and $61 for a value which is set up during the initialization sequence. If this value is present, the initialization sequence is bypassed and the Instant Pascal prompt "+<" is displayed. When this happens, Instant Pascal has assumed that the integrity of RAM storage has not been disturbed. If RAM occupied by the Instant Pascal program was altered while the AIM 65 Monitor was in control, the results are unpredictable. This can be avoided (at the cost of the Pascal program in memory) by reinitializing memory after entering by means of the +<N> command (see 3.3.2).

2.2.4 Start Application Program in PROM (6 Key)

When the AIM 65 Monitor is waiting for a command, pressing 5 causes the Monitor to execute a JMP to $B000, which enters Instant Pascal. Pressing 6 executes a JMP to $B003. The Instant Pascal ROM contains a JMP $C000 in $B003, so that Key 6 may be used to enter a function at $C000-$D000 from the Monitor. A Pascal program is limited to $300-$3FFF, however (see Figure 2-1).

2.2.5 Transitions Between Wait States

Table 2-2 expands upon Section 2.2.2 by describing certain transitions which may be forced by other keys as well as ESC.

Table 2-2. State Transitions in Instant Pascal

Cmd	State	Key	Effect
I	Waiting for source input	ESC RETURN	Returns to AIM 65 Monitor Completes input, command prompt
R	Waiting for source input	ESC RETURN (2)	Returns to AIM 65 Monitor Completes input, command prompt
G	Executing	DEL(held)	Breaks to command prompt
	Waiting for data input	ESC	Returns to AIM 65 Monitor
	Tracing	ESC(held) SPACE(held)	Returns to AIM 65 monitor Pause; any key resumes
L	Listing	ESC(held) SPACE(held)	Returns to AIM 65 Monitor Pause; any key resumes

SECTION 3

INSTANT PASCAL OPERATION

All resources of the Instant Pascal system are made available to you by means of a command interpreter which executes one-letter commands. The Instant Pascal command prompt is "+<". This chapter describes the six major processes of the Instant Pascal system which you may invoke:

a. Source input from keyboard or other devices available to the AIM 65 Monitor.

b. Source output (listing) to the AIM 65 printer or other output devices available to the AIM 65 Monitor.

c. Initialization to an empty program condition.

d. Editing of the Pascal program.

e. Binding (identifier references) and syntax checking of the Pascal program.

f. Execution of the Pascal program.

3.1 READ SOURCE INPUT (R COMMAND)

Turn on AIM 65 power, then press 5 after the Monitor prompt. Press RETURN in response to both of the initialization questions. This will ready Instant Pascal to accept the first command:

```
ROCKWELL AIM 65
<5> MEMORY SIZE? 4096
 WIDTH? 20
+<
```

The "R" (Read) command causes Instant Pascal to accept Pascal source input until it receives an empty line; i.e., two RETURNs in a row. Press R, then RETURN in response to the IN= prompt; this denotes keyboard input. (Other input devices may be designated; see the AIM 65 User's Guide, 3.8.1.)

Now enter the following from the keyboard as shown:

```
PROGRAM SQUARES;
VAR I:INTEGER;
BEGIN
FOR I:=1 TO 10 DO
WRITELN(I,SQR(I))
END
```

Finish with an extra RETURN to terminate input and return to the command prompt.

For input and editing purposes, the language of Instant Pascal is grouped into indivisible units called "text units". Each line of input must contain a sequence of complete, syntactically correct text units. An input line may be up to 60 characters long, including the final RETURN.

If, after keying in a line, the line is immediately printed back out with two "#"s, one at the beginning of the line, then an input syntax error was detected. The second "#" is immediately to the left of the character which led the input translator into an impasse. No part of the line went into the program memory, so key it in again.

3.2 SOURCE OUTPUT COMMANDS

The Pascal program in memory is a sequence of text units. The Instant Pascal editor maintains a pointer, called the text unit pointer, which always points to one text unit in the program.

When the program is initialized to an empty state, it contains one text unit, which is the "." at the end of every Pascal program. This text unit is always present and is never entered.

All editing is done relative to the point in the program denoted by the text unit pointer. In particular, text units entered by the R and I commands are inserted into the program immediately before (above) the text unit designated by the text unit pointer, and the text unit pointer remains pointing at that text unit. Therefore, after the program of 3.1 is entered, the text unit pointer still points to the ".".

3.2.1 List Current Text Unit (SPACE Command)

The SPACE command +< > lists the designated text unit (i.e., the text unit designated by the pointer). Press the SPACE bar after entering the program in 3.1.

```
+< >
 .
+<
```

3.2.2 Move Text Pointer to the Top (T Command)

The pointer may be moved to the top of the program by the T command. Type T:

```
+<T>
PROGRAM SQUARES;
+<
```

3.2.3 List Text Units (L Command)

The L command lists a specified number of text units (not lines), beginning with the designated text unit. The listing is generated by a process called "the lister". The text unit pointer is not moved by the L command. After you key L, the system will prompt for a numeric input with "/". The standard AIM 65 conventions apply:

```
RETURN = 1
. = forever (actually 9999)
sequence of digits terminated by RETURN = that number
```

Then the OUT= prompt will ask for the output device. RETURN specifies the AIM 65 printer. (Other output devices may be specified; see the AIM 65 User's Guide, 3.8.2.)

Now execute the standard sequence for listing a program:

```
    +<T>
PROGRAM SQUARES;
 +<L>/.
 OUT=RETURN
 PROGRAM SQUARES;
 VAR
  I:INTEGER;
 BEGIN
1 FOR I:=1 TO 10 DO
2  WRITELN(I,SQR(I))
 END.
```

Details:

a. The VAR and I:INTEGER; are on separate lines even though they were keyed in on the same line. They are separate text units; the formatting algorithm works entirely from the internal form of the program, which disregards the number of text units appearing on an input line.

b. The numbers in the left-hand margin of the statement part of the program count the indentation level. (The blank preceding BEGIN and END is really a zero, which is blanked for the sake of appearance.)

c. The listing format is such that corresponding structured statement delimiters (BEGIN...END, REPEAT...UNTIL, IF...ELSE, CASE...END) are at the same indentation level, and can thus be matched up over a span of program lines by use of the marginal numbers.

d. If corresponding statement delimiters are not shown by the lister to be at the same indentation level, the program has a sequence (syntax) error which will be caught by the Binder (see 3.5).

If a tape recorder is connected to your AIM 65, you can avoid rekeying this program later by writing it to tape now. First adjust the interblock gap parameter in $A409 to a value appropriate to your recorder. ($20 is a reasonable place to start.)

```
+<ESC                    Return to Monitor
<M>=A409 08 xx xx xx     Display $A409...$A40C
</> A409 20              Change $A409 to $20
<5>                      Return to Instant Pascal (pointer at
                           top)
<5>+<L>/.                List all; position tape
OUT=T F=SQRS RETURN      Start tape motion in record mode
T=1 RETURN               Recording begins
+<                       Recording is complete; turn off
                           recorder
```

You may wish to examine the contents of this tape by reading it into the AIM 65 Text Editor and listing it. The text is essentially as listed by Instant Pascal except for indentation and marginal numbers. You therefore have the option of using the AIM 65 Text Editor for preparing and correcting Pascal source program tapes.

3.3 INITIALIZATION

Before destroying this program you can execute it by keying the Go command:

```
+<G>OK
 1 1
 2 4
 3 9
 4 16
 5 25
 6 36
 7 49
 8 64
 9 81
 10 100
```

3.3.1 Report Available Memory (M Command)

The number of free bytes in memory may be interrogated with the M (memory) command.

```
+<M>=2653
```

Details:

a. Excluded from the definition of free space is space allocated to all fixed tables (most of page 2), pages 0 and 1, and the translator input and output buffers (317 bytes at the very end of declared RAM). Also, 250 is subtracted so that even when free space reads zero there will be 250 bytes available for the system's stacks to permit execution to begin.

b. You will find that memory occupied by programs is independent of the use of spaces as separators in the input text.

c. Typically, the space occupied by a program is 150% of the space occupied by the ASCII text of the Pascal source. This number is obtained by subtracting the program's M value from the M value of an initialized system. The source code identifiers are included in the program to allow the lister to reconstruct the source program for output.

3.3.2 Initialize Memory (N Command)

The program may be cleared by the N (New) command. Because N is irreversible, you get a chance to change your mind; you must answer "Y" to R U SURE? for N to work.

```
+<N> R U SURE? Y
```

From this point on, a new program can be entered as described in 3.1.

3.3.3 Initialize Instant Pascal (Z Command)

Instant Pascal may be completely restarted by typing Z. In addition to initializing memory like the N command, the memory size and page width prompt messages are displayed the same as upon initial entry from the AIM 65 Monitor. The R U SURE? prompt is also displayed to minimize accidental clearing of the memory. (You even get one more chance to save your program if you push ESC in response to the MEMORY SIZE? prompt.)

```
+<Z> R U SURE?  Y
MEMORY SIZE?  4096
WIDTH?  20
```

3.3.4 Redefine Page Width (W Command)

The WIDTH? part of the initialization sequence can be executed on its own with the W command. This is useful if you wish to change output devices.

```
+<W>WIDTH? 80
```

3.4 PROGRAM EDITING

After initializing to an empty program recreate the program SQUARES in memory, either by keying it in (see 3.1) or by reading it from tape, as follows. First position the tape.

```
+<R>IN=T F=SQRS T=1
```

Now start tape reading. (See the AIM 65 User's Guide, 9.1.5 and 9.1.6, for a more detailed discussion of the use of tape.) The command prompt +< indicates a successful completion of the read operation. (If the tape is not read successfully, you may have to RESET the AIM 65 and then re-enter Instant Pascal. Listing the program will then show you how much has been entered.)

In this section, the following commands will be used to modify the program SQUARES.

U/n	Move the pointer up n text units (toward the top)
D/n	Move the pointer down n text units (toward ".")
B	Move the pointer to the bottom, or ".", text unit
F	Find the line containing the indicated string
K/n	Delete n text units beginning with the designated one
I	Insert one input line of source
V	View a neighborhood of the designated text unit

First, list the program (T L . RETURN). Now verify that the pointer is at the top by executing the +< > (SPACE) command.

3.4.1 Move the Text Pointer to the Bottom (B Command)

Move to the bottom by executing the +<B> command. The "." that indicates the end of source code is displayed.

3.4.2 Move the Text Pointer Up (U Command)

Now move up to the I:INTEGER; by executing +<U>/5 RETURN. Note that after U the new designated text unit is listed.

```
+<U>5
I:INTEGER;
```

3.4.3 View Five Text Units (V Command)

Sometimes the designated text unit will be a semicolon or some other indistinguishable feature such as BEGIN or END. You can view a neighborhood of the designated text unit with the V command. Try it.

```
+<V>
PROGRAM SQUARES;
VAR
 I:INTEGER;
BEGIN
1 FOR I:=1 TO 10 DO
2
```

Details:

a. V is equivalent to U/2 L/5 except that the text unit pointer is not moved.
b. Normally V displays five text units with the designated text unit at the center. Exceptions occur near both ends of the program.

3.4.4 Move the Text Pointer Down (D Command)

After satisfying yourself that you are at the I:INTEGER, move down three text units to the output statement by executing D/3:

```
+<D>/3 RETURN
WRITELN(I,SQR(I)
+<
```

3.4.5 Delete Text Units (K Command)

Now delete the text unit WRITELN(I,SQR(I)) by executing +<K>/RETURN. (Remember that RETURN means 1 when a numeric input, prompted by "/", is being requested.)

```
+<K>/RETURN
WRITELN(I,SQR(I))
***
END.
```

Instant Pascal lists exactly what it is deleting, followed by "***", followed by the new designated text unit.

Detail:

> K/n is implemented as L/n followed by an internal block move which overlays the portion which was just listed. If you have second thoughts during the listing part of a K command, quickly hit RESET. Upon returning to Instant Pascal you will see that nothing has been deleted (assuming you were fast enough).

3.4.6 Insert a Text Line (I Command)

Insert one line of text by pressing I, typing in the next text and terminating the entry with RETURN. The text will be entered immediately before the current text unit (use the SPACE command if needed).

Detail:

> The command sequence K/RETURN I is the standard method of altering one text unit. The typical Pascal line A:=1; consists of two text units; if the A:=1 is the designated one, this command sequence will not touch the semicolon.

Before inserting something in place of the WRITELN, list the program.

```
+<T>
PROGRAM SQUARES;
+<L>/.
OUT=RETURN
PROGRAM SQUARES;
 VAR
  I:INTEGER;
 BEGIN
1 FOR I:=1 TO 10 DO
2
 END.
```

Note that this is a syntactically correct program with an empty statement in the scope of the FOR. You can execute it by pressing G. If you don't believe that anything happened, you can re-execute it with the assignment trace on (see 4.1). This traces assigned variables. Remember to toggle the assignment trace off after the program has executed.

```
+<A>ON
+<G>
FOR I:=1 TO 10 DO
> 1
> 2
> 3
> 4
> 5
> 6
> 7
> 8
> 9
> 10+<A>OFF
```

To insert a new statement in the scope of the FOR, you must find the END, because Pascal source is inserted before the designated text unit. This can be done by commanding a +<B> 1<U>/RETURN.

3.4.7 Find a Text String (F Command)

You can also find the END using the F (Find) command. Position to the top (+<T>) then key the +<F> command. At this point the

AIM 65 is waiting with a text input prompt in the display. (If the KB/TTY switch is set to TTY, the text prompt is "*" at the terminal.) Key in some string characteristic of the FOR statement, such as FOR or :=, then return. The editor will find the text unit. Now move down one text unit to the END.

```
+<T>
PROGRAM SQUARES;
+<F>
:= RETURN
FOR I:=1 TO 10 DO
+<D>/RETURN
END.
+<
```

Details:

a. Find is implemented by internally performing a L/. and diverting the lister's output to an internal buffer, in which a string match is performed at the end of each line (not text unit).

b. Some lines contain two text units, notably lines ending in ";" and ".". If a string match is found, the designated text unit is the last one listed, namely the ";" or ".". This can be disconcerting if the search for "X" IN "X:=1;" ends up on ";", so the following special case has been internally programmed: if a string match is found and the designated text unit is ";", +<U>/1 is expected. This special case does not test for ".", so a search for END in this example will end up pointing to ".".

c. Because the spaces that come out follow the rules of the lister and bear no necessary connection to the use of spaces (as separators) during source input, it is inadvisable to use space separators in a search argument until one understands how the lister puts them out, namely as seldom as possible.

3.4.8 Program Change Example

Now that the END has been found, change the program to generate a table of factorials by inserting a new statement before the END, using the I (Insert) command.

```
+<I>
WRITELN(I,FACT(I))
+<
```

This program is incomplete, because FACT is an undefined function; this can be verified by pressing G:

```
+<G>
*UNDEF* FACT
WRITELN(I,FACT(I))
*SEQUENCE*
WRITELN(I,FACT(I)
END.
```

Details:

a. The diagnostics come from the Binder, which is called by the C (Check) command (see 3.5) or whenever execution is called for (+<G>) but the program has been changed or has not been bound. The Binder looks up all identifier and label references and performs a total syntax check on the program. The lookup function of the Binder led to the *UNDEF* message; the syntax check function led to the *SEQUENCE* message.

b. Normally an undefined identifier would not lead to a *SEQUENCE* error, but an undefined identifier in an expression is assumed to be a variable, and the "(" following FACT is erroneous under that assumption.

c. The "OK" following +<G> or +<C> is produced by the Binder when it finds no errors.

Now key in a definition of FACT:

```
+<T>
PROGRAM SQUARES;
+<F>                                  Find
BEGIN                                the string argument
BEGIN                                Found it.
+<R>IN=RETURN                        Begin input.
FUNCTION FACT(N:INTEGER):REAL;
VAR I:INTEGER;
J:REAL;
BEGIN
J:=1;
FOR I:=2 TO N DO
J:=J*I;
FACT:=J
END;
RETURN
```

Go to the top and list the program:

```
+<T>
PROGRAM SQUARES;
+<L>/.
 OUT=
PROGRAM SQUARES;
 VAR;
  I:INTEGER;
 FUNCTION FACT(N:INTEGER):REAL;
 VAR
  I:INTEGER;
  J:REAL;
 BEGIN
1 J:=1;
1 FOR I:=2 TO N DO
2  J:=J*I;
1 FACT:=J
 END;
 BEGIN
1 FOR I:=1 TO 10 DO
2  WRITELN(I,FACT(I))
 END.
```

Now, pressing G will give you the first ten factorials.

```
+<G>OK
1  1
2  2
3  6
4  24
5  120
6  720
7  5040
8  40320
9  362880
10  3628800
```


3.5 PROGRAM BINDING (C COMMAND)

Binding is normally an automatic process, called by +<G> before execution if a program is unbound, that is, if it has not yet been bound since it was entered or if it has been changed since it was last bound.

Since the Binder produces intelligible diagnostic messages, it can be used as a syntax and undefined identifier checker; the C (Check) command exists for this purpose. It calls the Binder only, and produces the message "OK" or one or more Binder diagnostics (see Section 6.2).

Details:

a. The Binder does not check for type mismatches in expressions, assignments, or actual parameters. These are checked at execution time.

b. If the binding is without error, the text unit pointer is left at the top of the program. If the binding terminates with a *SEQUENCE* (syntax) error, the text unit pointer is at the first text unit printed after the word *SEQUENCE*. If the binding goes to completion (i.e., does not terminate on a *SEQUENCE* error) but causes diagnostics such as *UNDEF* (undefined identifier), *DUP* (duplicate identifier or label), *UNDECL* (undeclared label), or *TYPE* (type inconsistency in declaration), the text unit pointer will be left at the bottom of the program.

3.6 PROGRAM EXECUTION

3.6.1 Execute Program (G Command)

The G (Go) command calls the Binder if necessary and then, if the program has been bound without error, gives control to the Pascal Interpreter to execute the program.

Normally execution begins at the beginning of the program. There is one exception to this rule. Break-in-progress is a command-time condition which is true if the program was executing and execution was discontinued either by execution of the BREAK predefined procedure or by depression of the DEL key during execution. If Break-in-progress is true, G will cause execution to resume at the point of the break.

Details:

a. Break-in-progress is preserved even if ESC causes an exit to the AIM 65 Monitor. It is turned off by the New (+<N>) command.

b. Break-in-progress is turned off by invocation of the Binder. This has two consequences:

   (1) If the program is changed after a break, G will cause execution to start at the beginning, because the Binder will have been invoked.

   (2) The C command may be used explicitly to force G to start at the beginning after a break.

3.6.2 Execute and Trace Program (, Command)

The +<,> (Step) command is basically the same as +<G>, with the following additional properties.

a. Execution is traced. That is, text units are printed out before they are executed and the values of variables are printed out after they are changed by assignment.

b. Execution continues only until the start of the next statement, at which time the command prompt occurs with Break-in-progress true.

The +<,> is, in effect, a source-level single-step command. Any time that +<,> is in use, execution may be resumed with the G command, since break-in-progress is set.

Continuous execution may be interrupted and control returned to the command interpreter by holding down the DEL key. This key is sampled at the start of execution of every text unit. If it is down, KEY BREAK is printed and control is returned to the command interpreter with Break-in-progress set. Then the +<,> may be used to single-step for a while.

Details:

a. A text unit is printed just before the command prompt when execution is discontinued, both when the DEL key and the +<,> command are used. This is the text unit about to be executed.

b. The text unit pointer (which is not the same thing as the Interpreter execution pointer) is set to the text unit printed out before the command prompt. If it is not clear where the stop occurred, you can use the V command to enlarge the view.

SECTION 4

PROGRAM DEBUGGING

Chapter 3 described the basic facilities of Instant Pascal for creating, changing, and executing a Pascal program. In addition, the Step (+<,>) command and DEL key have been shown to be useful for analyzing the dynamic behavior of a program. This chapter describes several additional facilities which are available for program debugging.

4.1 TRACING

When the Step command is used, tracing is automatic. In general there are two kinds of tracing, which can be independently controlled in Instant Pascal during continuous execution (+<G>).

a. Statement tracing

b. Assignment tracing

These two kinds of trace may be independently controlled by two toggles, called S and A, respectively. These toggles are switched by the S and A commands.

4.1.1 Statement Tracing (S Command)

Statement tracing prints each text unit just before it is executed.

Detail:

The state of the toggle after entering the S command is indicated by the word ON or OFF printed immediately after the command.

4.1.2 Assignment Tracing (A Command)

Assignment tracing prints values changed by assignment and FOR statements.

Details:

a. The state of the toggle entering the A command is indicated by the word ON or OFF printed immediately after the command.

b. If S is off and A is on, assignment statements are nevertheless printed in order to associate them with the printed values. FOR statements are also printed, but only the first time through the loop.

4.2 IMMEDIATE STATEMENT EXECUTION (X COMMAND)

It is possible, under well defined conditions, to key in a single statement and have it executed immediately, independent of the execution of the program in memory. This is useful during a break or after an abnormal program termination for examining and changing the values of variables.

The +<X> (eXecute) command immediately sets up a text input prompt. Key in one statement; it will be executed upon depression of RETURN. Note the restrictions to the use of X which are listed below.

Details:

a. There must be a program in memory which has been successfully bound and whose execution has begun. That is, X should be executed only during a break or after completion of a program, either a normal completion or one accompanied by an ERROR message. Other use of X will lead to unpredictable results.

b. If the program is not bound when X is executed, it will first be bound before the statement is executed. This program binding is indicated by the printout ".C.". If it leads to a diagnostic, the statement will not be executed.

c. The statement must itself be bound before it can be executed. This can lead to diagnostics, in which case the statement will not be executed.

d. There is the question of visibility of identifiers. That is, the statement WRITELN(A[I]) may have different interpretations of A and I depending on the block in which it is presumed to be executed. This block is identified by the position of the text unit pointer at the time +<X> is executed. The text unit pointer must be in the statement part of an active block or at the bottom of the program. The movement commands used in text editing (see 3.2, 3.4) may be used to position the pointer before using X.

e. Note that at a break, the text unit pointer is at the text unit listed and about to be executed.

f. If the program terminates with an ERROR message, the text unit pointer is at the text unit listed, i.e., the one whose execution caused the error.

g. If the program is bound in response to X (i.e., if ".C." is printed), and there are no Binder diagnostics, the interpretation of the statement will be with respect to the statement part of the main program, since, in this case, the Binder leaves the text unit pointer at the bottom of the program.

h. The single statement executed can be a structured statement. Its complexity, therefore, is limited only by the 60-character length of the AIM 65 input line.

SECTION 5

INSTANT PASCAL TEXT UNITS

The internal form of a Pascal program in the Instant Pascal system has been designed to satisfy three objectives not previously considered compatible in systems supporting structured languages:

- The internal form of the program must be backwards translatable to equivalent source code. This objective permits building a system which can support the illusion of directly executing the source code.

- Execution time should be insensitive to complexity of data or statement structures. For example, the time required by an IF statement to skip over an unexecuted compound statement should be about the same whether the compound statement is three lines or one hundred lines long.

- The consequences for the structure of the internal form of the program of any source-program change must be confined to the portions of the program which are changed. As a counterexample, changing one letter, VAR A,B:SEAL; to VAR A,B:SEAT; (where SEAL and SEAT are defined types) in a compiler-based system can completely alter the object program and therefore requires total recompilation.

The approach which has been taken to reconcile these objectives in Instant Pascal divides the process of transforming source text into answers into three phases.

a. Translation (elicited by the R, I, and X commands) processes Pascal source code and translates it into the internal program format which is in executable form except for certain addresses and integer values, such as pointers to variable declarations and lengths of structured types. The proper values cannot be computed for these values because the program cannot be assumed to be whole. Space is left, however, for these values to be later added. (These entities are called "spanners" in this section).

b. Binding (elicited by the C and X, and possibly by the G and step commands) scans the entire program (or in the case of X, the statement), verifying its structural integrity and assigning values to all spanners.

c. Execution (elicited by the G, step, and X commands) sequentially interprets program statements and alters data values in accordance with the rules of Pascal.

5.1 TEXT UNITS IN EDITING

In order to ensure that the user's freedom to edit programs is not seriously constrained by the design approach taken above, this design approach also incorporates the following principle.

> Every program is conceived as being built of a single-level, linear sequence of syntactic building blocks called "text units". From the editor's point of view, the text unit is the throwaway unit. Program building and altering reduces to inserting text units into, and deleting text units from, this linear sequence.

As a consequence of this principle, character-level editing within a given text unit is accomplished by deleting the text unit and inserting a replacement. In some cases, text units are small enough that the editing process, viewed as deletion followed by insertion, is essentially the same as conventional character-oriented editing. For example, the following are text units:

BEGIN, END, ; (as a statement separator), REPEAT, ELSE

On the other hand, some text units are larger. For example, all simple statements are text units; in fact, expressions are always contained within larger text units.

The person familiar with Pascal will readily incorporate the text-unit orientation into his or her editing style. Here is a brief and informal definition of most of the Pascal text units.

a. Program, procedure, and function headings, including formal parameter lists.

Example:

```
FUNCTION ISPRIME(N:2..MAX):BOOLEAN;
```

b. Certain part header words:

```
CONST
TYPE
VAR
```

c. Constant definitions, including the final semicolon.

Example:

```
GREETING='HELLO,THERE!';
```

d. Type definitions, including the final semicolon, except in the case of record type definitions, in which case the word RECORD stands in place of the type and final semicolon.

Examples:

```
TABLE=ARRAY[RANGE]OF COST;
HASHTABLE=RECORD
```

e. Variable and field declarations (they are the same).

Example:

```
GAP=$A409,INPUTCHAR:CHAR;
```

(Note that the final semicolon after a field declaration is always required in Instant Pascal unless the field type is RECORD.)

f. END and ; at the end of a RECORD definition.

g. Simple statements.

Examples:

```
GOTO 5
SUBR(CRLOW)
A:=B[N+1]*C
```


h. The connective fragments which are used to build structured statements.

Examples:

```
REPEAT
UNTIL A=N
IF ODD(I) THEN
WITH R[P] DO
```

i. Labels and case constant lists preceding statements.

j. Comments.

Example:

```
(*THIS IS A COMMENT*)
```

In general, the lister begins each text unit on a new line. Exceptions occur when labels and case constants precede statements and, most commonly, when semicolons follow statements. This fact is important in editing because the formatted listing is frequently used as a reference document in the editing process, and the editing commands which require numeric parameters (+<L>/n, +<K>/n, +<U>/n, +<D>/n) always count text units, not lines. The most common case to remember when counting text units on a listing is to count as two text units any line containing a simple statement or a structured statement connective fragment followed by ";".

Examples:

```
A:=1;
END;
UNTIL NOT ODD(N);
```

There is a special case in the input translator which is necessitated by a syntactic ambiguity in Pascal. Consider the plight of the input translator on encountering the following text in the course of program creation.

```
A,B,C,D:E;
```

Is it a variable declaration, or is it a case constant list followed by a procedure statement? These two objects must be translated differently. This is the only case in which the translation process is context-sensitive. The translator decides how to treat this input by examining a context flag of type (DECLARATIVE,STATEMENT). The flag's value is maintained according to the following rules.

a. It is initialized to DECLARATIVE.

b. Otherwise, its value depends on the identity of the text unit immediately preceding the text unit pointer, and the value is subject to change every time the identity of the preceding text unit is changed, according to the following rules:

(1) If there is no preceding text unit, the value is unchanged.

(2) If the preceding text unit is END or the separate ";", the value is unchanged.

(3) Otherwise, if the preceding text unit is one which occurs in declaration or definition parts, the value is changed to DECLARATIVE; or if the preceding text unit is one which occurs in the statement part, the value is changed to STATEMENT.

The reason for excepting END and the separate semicolon is that they can occur in both statements and at the end of RECORD types.

5.2 DEFINITIONS OF TEXT UNITS

This section presents a list of the text units of Instant Pascal. It is not a formal definition of the syntax of the source language. Such a definition is implied in Chapter 7. Rather, this section uses the reader's assumed knowledge of existing syntax definitions of Pascal to present the source language from the linear, single-level text unit point of view.

There are two metalinguistic elements used in these lists.

a. Square brackets [ ] enclose an optional element.

b. Elipses ... mean that the previous element is optionally repeated any number of times.

The detail notes accompanying the lists clarify specific points and cite deviations from standard Pascal practice.

5.2.1 Program Heading

```
PROGRAM identifier;
```

5.2.2 Text Units Appearing in Both Declarative and Statement Parts

1. (* comment text *)
2. END
3. ;

Details:

a. Comment text may not contain the character sequence "*)".

b. The character ";" appears at the end of certain heading, declaration, and definition text units. This use of ";" is not a separate text unit but is an inseparable part of the heading, declaration, or definition text unit.

5.2.3 Text Units Appearing in Declarative Parts

The text units are:

1. LABEL integer [,integer]...;
2. CONST
3. identifier=constant;
4. TYPE
5. identifier=type;
6. identifier=RECORD
7. VAR
8. identifier[=$hhhh][,identifier[=$hhhh]]...:type;
9. identifier[=$hhhh][,identifier[=$hhhh]]...:RECORD
10. PROCEDURE identifier[(formal parmeter list)];
11. FUNCTION identifier[(formal parameter list)]:type;

Details:

a. The constant definition identifier=identifier; is not in the language.

b. The option [=$hhhh] denotes a global variable assigned to the absolute hexadecimal address hhhh. If the type is longer than one byte, hhhh is the lowest-numbered address in the variable.

c. The identifier[=$hhhh] cases above are used for both variable and field declarations. The option [=$hhhh] is not permitted in field declarations.

d. The semicolons at the end of cases 1,3,5,8,10,11 are not optional. Therefore, the optional semicolon permitted by Pascal after the last field declaration in a record definition is not optional in Instant Pascal; it is required.

5.2.4 Simple Statements

The statements are:

1. variable:=expression
2. function designator:=expression
3. GOTO integer
4. identifier([actual parameter list)]

Details:

a. Statement type 2 expresses assignment to the value of a function.

b. The integer in the GOTO statement must appear in the LABEL declaration of the block containing this statement part. GOTO may not jump outside its own block.

c. The identifier in the statement type 4 is a declared or predefined procedure name.

5.2.5 Statement Connective Fragments

The connective fragments are:

1. BEGIN
2. END (the same as in 5.2.2)
3. ; (the same as in 5.2.2)
4. IF Boolean expression THEN
5. ELSE

6. WHILE Boolean expression DO
7. REPEAT
8. UNTIL Boolean expression
9. FOR identifier:=expression TO expression DO
10. FOR identifier:=expression DOWNTO expression DO
11. CASE expression OF
12. case constant[,case constant]...:
13. OTHERWISE:
14. WITH record variable[,record variable]...DO
15. label:

Details:

a. Case constants are not strongly type checked against the expression in the CASE fragment. A case constant will be selected at run-time if its ORDinal value matches the ORDinal value of the expression.

b. OTHERWISE: is the default case "constant".

SECTION 6

DIAGNOSTIC MESSAGES

This chapter describes diagnostic messages which can occur during translation, binding, execution, and listing.

6.1 SOURCE INPUT DIAGNOSTICS

Each correct source input line is a sequence of one or more syntactically correct text units. There is syntax checking during translation, but only within each text unit. With the exception noted in 5.1, there is no concern at translation time about the order in which text units are entered or about the overall structure of the program.

During source input (I, R, or X command), an entire line is input before any of the line is translated. This permits free use of the DEL key to correct keying errors. Once translation begins (immediately after keying RETURN) each text unit is translated without regard for any text units either preceding or following it on the same line. If an error is discovered during the translation of any text unit in an input line, no text unit in that line is incorporated into the program. If no error is discovered, all text units in the line are incorporated into the program.

Three diagnostic messages can occur during source input.

- The syntax error message

  #...#

  See 3.1 for a discussion of this message.

6. WHILE Boolean expression DO
7. REPEAT
8. UNTIL Boolean expression
9. FOR identifier:=expression TO expression DO
10. FOR identifier:=expression DOWNTO expression DO
11. CASE expression OF
12. case constant[,case constant]...:
13. OTHERWISE:
14. WITH record variable[,record variable]...DO
15. label:

Details:

a. Case constants are not strongly type checked against the expression in the CASE fragment. A case constant will be selected at run-time if its ORDinal value matches the ORDinal value of the expression.

b. OTHERWISE: is the default case "constant".

SECTION 6

DIAGNOSTIC MESSAGES

This chapter describes diagnostic messages which can occur during translation, binding, execution, and listing.

6.1 SOURCE INPUT DIAGNOSTICS

Each correct source input line is a sequence of one or more syntactically correct text units. There is syntax checking during translation, but only within each text unit. With the exception noted in 5.1, there is no concern at translation time about the order in which text units are entered or about the overall structure of the program.

During source input (I, R, or X command), an entire line is input before any of the line is translated. This permits free use of the DEL key to correct keying errors. Once translation begins (immediately after keying RETURN) each text unit is translated without regard for any text units either preceding or following it on the same line. If an error is discovered during the translation of any text unit in an input line, no text unit in that line is incorporated into the program. If no error is discovered, all text units in the line are incorporated into the program.

Three diagnostic messages can occur during source input.

- The syntax error message

  #...#

  See 3.1 for a discussion of this message.

o The message

"NO SPACE LEFT"

This indicates that the current number of free memory bytes (see +<M>, 3.3.1) minus the number of bytes into which this line has been translated is negative.

o The message

"TOO MUCH CODE"

This message, which is expected never to occur, says that the 256-byte buffer which receives the output of the translator is not large enough to hold the internal form of the entire input line. If the message occurs and if the input line contains multiple text units, break the input into more than one line.

Details:

a. In the case of the syntax error message, the right-most "#" usually gives a strong clue about the error, since the character immediately to its right is the one which led the syntax analyzer into the impasse. If the right-hand "#" is the last character of the diagnostic message, something is missing at the end of the input, for example, a final semicolon in a declaration or definition.

b. The usual response to the "NO SPACE LEFT" message is to save the program on tape and then to figure out how to do what needs to be done in less space. If a larger memory size could have been declared because RAM is available which wasn't used, it is still necessary to write the program out and then read it back in.

6.2 BINDER DIAGNOSTICS

Diagnostic messages which occur during binding have the following distinctive form: an indicator line describing the error, followed by one or more program text lines describing where the error occurred. The following indicator lines can occur:

SEQUENCE
UNDEF identifier
DUP identifier
UNDECL label
DUP label
TYPE identifier

6.2.1 *SEQUENCE* Diagnostic

The *SEQUENCE* diagnostic indicates a syntax error. This is usually an inter-text-unit syntax error, for example, a missing semicolon, but intra-expression syntax errors which escape the checking in the translator are possible, as the example in 3.4.8 illustrates.

The indicator line is followed by four text units to display the context of the error. The actual error was discovered, that is, the impasse occurred, at or in the first text unit listed.

A *SEQUENCE* diagnostic immediately aborts binding with the text unit pointer at the first of the four listed text units.

Details:

a. If the only thing following the *SEQUENCE* line is ".", this indicates a missing END in the program. Listing the whole program will show this in the form of a nonzero indentation on the last source line.

b. If examination of the program shows that the syntax error is due to something missing immediately before the first listed text unit (a missing semicolon is a common example), it can be immediately entered with the +<I> command without moving the text unit pointer.

6.2.2 Identifier Lookup Diagnostics

The indicator lines *DUP* identifier and *UNDEF* identifier are each followed by one text unit which contains the cited identifier occurrence.

The *DUP* diagnostic is generated at a defining occurrence of an identifier if the identifier has a prior defining occurrence at the same block level.

The *UNDEF* diagnostic is generated at a referring occurrence of an identifier if the identifier has no prior defining occurrence at the same or at any higher block level which would be visible to this referring occurrence.

Neither of these diagnostics stops the Binder.

6.2.3 Label Lookup Diagnostics

The indicator lines *DUP* label and *UNDECL* label are each followed by one text unit which contains the cited label occurrence.

The *DUP* diagnostic is generated at a label: preceding a statement if there was a prior occurrence of the same label: in the statement part of the same block.

The *UNDECL* diagnostic is generated at a GOTO label if there is no label declaration in this block which names the label.

The case in which a label is declared and occurs in a GOTO but does not occur preceding a statement is caught at run-time when (and if) the GOTO is executed.

Neither of these diagnostics stops the Binder. Note that labels are stored and searched as character strings, except that insignificant zeros are suppressed. Instant Pascal does not impose the conventional limit of four digits to the length of a label.

6.2.4 Type Consistency Diagnostics

The indicator line *TYPE* identifier can indicate an inconsistency between the types of the upper and lower bounds of a subrange type; it can also be generated by a STRING type with a noninteger length. The identifier in the indicator line is a type identifier appearing in either the subrange or string type; this identifer refers to a type which is inconsistent with the definition in which it occurs. This definition occurs in the text unit following the indicator line.

This diagnostic does not stop the Binder.

Details:

a. If no Binder diagnostic occurs, the Binder will output the message "OK" and the text unit pointer will be at the top of the program.

b. If a Binder diagnostic occurs but no *SEQUENCE* diagnostic occurs, the text unit pointer will be at the bottom of the program.

c. It is not possible to execute a program without an error-free binding.

6.3 EXECUTION-TIME DIAGNOSTICS

The are 53 possible execution-time diagnostics enumerated in Appendix E. This section deals with what is common to all of them.

All execution-time diagnostics are fatal to execution. The error reported by the diagnostic message is discovered in the process of interpreting a text unit. This text unit is printed as part of the diagnostic message; then execution terminates (with Break-in-progress false) and control returns to the command interpreter with the text unit pointer at the offending text unit.

The form of the diagnostic message is.

```
ERROR #nn
text unit
```

The definitions of error codes nn are listed in Appendix E.

After control returns to the command interpreter, a series of WRITELNs may be executed with the +<X> command to examine the state of the data. The position of the text unit pointer insures that visibility of identifiers is the same as that which prevailed at the time of the error. The data structures built during execution are not initialized until the next +<G> or +<,> is executed, so data are available for examination. (This is true after an error-free termination also, but first do a +<B> before the +<X> command to position the text unit pointer to the bottom of the program).

6.4 SOURCE OUTPUT DIAGNOSTICS

Under normal conditions, the lister will never encounter something which it cannot translate. If memory becomes corrupted,

however, the lister may encounter a value which does not translate meaningfully; this will lead to the output of a single "?" as part of the output text.

Certain corrupted memory configurations can cause the lister to loop continously; it may be necessary to RESET the AIM 65 under these conditions.

Detail:

> Under certain conditions the lister will only print a partial text unit as part of a diagnostic message. This can occur where lists are involved and an error was caught in the middle of a list: in LABEL and VAR declarations, parameter lists, and case constant lists. In one case, namely parameter lists, the backward translation algorithm may become confused if the translation starts in the middle of the list, and an extra semicolon might come out. Also, the text unit pointer may actually be positioned in the middle of a list. This will not lead to difficulty, however, and the text unit pointer can be repositioned as usual. Care should be taken not to delete anything until the text unit pointer is back to the beginning of a text unit.

SECTION 7

INSTANT PASCAL LANGUAGE DEFINITION

The language definition in this chapter presupposes a knowledge of Pascal concepts, particularly as they are presented in the "Report" section of "Pascal User Manual and Report", Second Edition, by Kathleen Jensen and Niklaus Wirth (see Appendix F). Following is a list of deviations from Jensen and Wirth Pascal; it may be presumed that any feature of Pascal which is not explicitly discussed here corresponds to Standard Pascal.

7.1 SUMMARY OF EXTENSIONS AND RESTRICTIONS

7.1.1 Extensions

Following is a list of the features of the AIM 65 Instant Pascal language which are not found in Standard Pascal.

a. Variables, both simple and structured, may be given absolute memory addresses. The syntax "=$hhhh" (where h is a hexadecimal digit) after any variable identifier in a variable declaration fixes the variable at address $hhhh and makes it global. (That is, its value survives the block in which it is declared.)

b. The underline character "_" may appear in user-defined identifiers wherever a digit may appear. This character is not on the AIM 65 keyboard, but it can be entered from a terminal.

c. The OTHERWISE: default clause may precede the last statement in a CASE statement in place of a case constant list.

d. Identifiers and labels may have any length; the entire string (except for insignificant zeros in labels) is significant.

e. The STRING data type may have values whose length varies dynamically up to the declared maximum.

f. The predeclared procedure BREAK causes interruption of program execution, if it is enabled.

g. The predeclared procedure SUBR(ENTRANCE) calls the machine-language subroutine whose entrance address is the declared address of the absolute CHAR variable ENTRANCE. SUBR(ENTRANCE,DATA) does the same, but before entering the subroutine it places the (lowest) address of DATA in $FE,$FF and places the first (lowest-addressed) byte of DATA in A.

h. The predeclared function FUNC(ENTRANCE) has a value of type CHAR. It calls a subroutine the same way that SUBR(ENTRANCE) does; upon return, it uses the contents of A for its value. FUNC(ENTRANCE,DATA) does the same as SUBR(ENTRANCE,DATA); upon returning, it uses the content of A for its CHAR value.

7.1.2 Restrictions

The following features of Standard Pascal are not found in AIM 65 Instant Pascal.

a. FILE types are not implemented, nor are the predeclared procedures GET, PUT, RESET, REWRITE, PAGE or the predeclared functions EOLN, EOF. The predeclared procedures READ, READLN, WRITE, WRITELN, are implemented to work with all character-serial devices available to the AIM 65

Monitor. These devices act like TEXT files and can read into and write from the following types: CHAR, INTEGER, REAL, BOOLEAN (Y or N on input), variable-length STRING.

b. Set expressions ([expression..expression] and operators +, -, *) are not implemented. One-byte sets (up to eight elements in the base type) and the relational operator IN are implemented; together with absolute-addressed variables these permit Pascal-level testing of bits in memory, for example, I/O status.

c. Records are implemented, but record variants are not.

d. Dynamic storage allocation (procedures NEW, DISPOSE) and the pointer type are not implemented.

e. The directive FORWARD is not implemented.

f. The constant definition identifier=identifier; is not implemented.

g. Ambiguity between field and variable names is not supported. Other Pascal visibility rules are fully supported.

h. The procedures PACK and UNPACK are not implemented. Packing of data is not done below the byte level. The word-symbol PACKED is accepted but nonfunctional.

i. Procedural and functional parameters to procedures and functions are not implemented.

j. GOTO may not jump outside its own block.

7.2 WORD-SYMBOLS AND PREDEFINED IDENTIFIERS

This section lists all word-symbols and predefined identifiers in Instant Pascal. If no comment accompanies a particular list item, it may be presumed to follow Pascal in its definition.

7.2.1 Word-Symbols

The list of word-symbols in Instant Pascal is the same as in Standard Pascal, with one addition: OTHERWISE. Two Pascal word- symbols, FILE and NIL, are not implemented and should be avoided as identifiers. (In fact, they cannot be used as identifiers.)

The Instant Pascal word-symbols are:

AND, ARRAY, BEGIN, CASE, CONST, DIV, DOWNTO, DO, ELSE, END, FILE, FOR, FUNCTION, GOTO, IF, IN, LABEL, MOD, NIL, NOT, OF, OR, OTHERWISE, PACKED, PROCEDURE, PROGRAM, RECORD, REPEAT, SET, THEN, TO, TYPE, UNTIL, VAR, WHILE, WITH.

7.2.2 Predefined Type Identifiers

The predefined type identifiers are the same as in Pascal:

BOOLEAN, CHAR, INTEGER, REAL, STRING, TEXT.

TEXT is recognized but not implemented and should not be used as an identifier.

STRING is implemented differently from Pascal, as described below.

As a group, the predefined type identifiers deviate from Pascal in the fact that they cannot be redefined. In theory, Pascal

allows the identifier REAL to be used as a defined type name; Instant Pascal does not allow that.

The type STRING[length] where length is a positive integer constant (either a literal or identifier) whose value is less than 256, is defined as ARRAY[0..length] OF CHAR. The dynamic value of a variable S:STRING[N] is the sequence of characters S[0],S[1],...,S[ORD(S[0])], where 0<=ORD(S[0])<=N. That is, the first byte S[0] stores the dynamic length of the string.

Character and string literals are internally realized as values of type STRING. That is, the constant 'A' is a two-byte object whose bytes have the hexadecimal values $01 $41. For this reason, type compatibility constraints have been relaxed to the following extent: any value of type STRING, if its dynamic length is 1 (i.e., if the value of the first byte is $01), may stand in place of a value of type CHAR in assignment and comparison. For example,

```
PROGRAM STRINGDEMO;
CONST
 CHARCONST='A';
VAR
 S:STRING[10];
 C:CHAR;
BEGIN
 READ(S);
  IF S<>CHARCONST THEN
   C:=S
END
```

will execute without a diagnostic if, and only if, one character followed by RETURN, is entered in response to the READ.

Pascal allows a statement like X:='ABC' where X is of type PACKED ARRAY[3] OF CHAR; Instant Pascal does not. The compensation for this is that, in Pascal, X:='ABC' would not work if X were of type PACKED ARRAY[4] OF CHAR, but in Instant Pascal, any STRING type of declared length 3 or greater can be

assigned the value 'ABC'. In general a STRING[N] variable can be assigned a value of type STRING whose dynamic length is N or less, even zero, and even if the declared maximum length of the source is greater than N. Reference to the individual array components of a STRING type value is allowed. For example, given S:STRING[N], the CHAR-type variables S[0],...,S[N] are available to the program.

7.2.3 Predefined Boolean Constants

The BOOLEAN type is implemented as a one-byte enumerated type with the two values (FALSE,TRUE). The predefined identifiers FALSE and TRUE have the same meaning as in Pascal, as the constants of type BOOLEAN. For example, REPEAT ... UNTIL FALSE is a way of bracketing a nonterminating loop. The identifiers FALSE and TRUE may not be defined; they may be used only as the Boolean constants.

7.2.4 Predeclared Procedures

As in Pascal, the predeclared procedure identifiers are considered to be declared at a level above the program block; they can be redefined in the program.

The following procedure identifiers are predeclared in Instant Pascal; their definitions are given below.

```
BREAK
READ
READLN
SUBR
WRITE
WRITELN
```


The following Pascal procedure identifiers are not predeclared in Instant Pascal and are available as user-defined identifers.

```
DISPOSE
GET
NEW
PACK
PAGE
PUT
RESET
REWRITE
UNPACK
```

The definitions of the predeclared procedures follow.

a. BREAK. See 4.2.

b. READ, READLN. READ and READLN are the same except that after reading into the last parameter, READLN outputs a new line (CR or CR/LF) to the printer or attached terminal. READ(P1,...,Pn) sequentially executes READ(P1),...,READ(Pn); it is therefore necessary to describe only the single-parameter procedure READ(P).

The behavior of READ(P) is consistent with the Pascal READ(P) from the implied TEXT file INPUT, where the INPUT file is the AIM 65 Monitor's input device. Specifically it depends on the type of P, as follows:

BOOLEAN: Only two one-character values are accepted: "Y" and "N". "Y" sets P to TRUE; "N" sets P to FALSE. Other inputs cause "?" to be output and the READ waits for a correct input.

CHAR: One keystroke is accepted; its ASCII value is assigned to P.

INTEGER: A signed or unsigned integer in the range -32768..32767 is accepted and assigned to P. Inputs are read into an input buffer and only scanned after RETURN is keyed; therefore the DEL key may be used to correct keying errors. Scanning terminates at the first nondigit (after the possible initial sign). If the input buffer is empty, "?" is output and the READ again waits for input.

REAL: A signed or unsigned real number, with or without an exponent part, is accepted and assigned to P. The syntax accepted is exactly as in the source language, which corresponds to Pascal in this respect. Inputs with integer syntax are also acceptable. Remarks under INTEGER regarding the input buffer apply to REAL.

STRING: If the declared length of P is N, up to N characters followed by a RETURN are accepted and assigned to P. More than N characters of input will produce a run-time error. The DEL key is available for correcting keying errors. Empty input strings are acceptable.

d. SUBR. See 7.1.

e. WRITE, WRITELN. WRITE and WRITELN are the same except that after writing the value of the last parameter, WRITELN outputs a new line (CR or CR/LF) to the printer or attached terminal. WRITELN with no parameter is equivalent to SUBR(CRLF) where CRLF=$E9F0:CHAR;. WRITE(P1,...,Pn) sequentially executes WRITE(P1),...,WRITE(Pn); it is

therefore necessary to describe only the single parameter procedure WRITE(P).

The behavior of WRITE(P) is consistent with the Pascal WRITE(P) to the implied TEXT file OUTPUT, where the OUTPUT file is the same as the AIM 65 Monitor's output device. The ":" parameter separators of Pascal, and the formatting function they control, are not supported by Instant Pascal. The behavior of WRITE(P) depends on the type of P; what follows also describes output during tracing.

BOOLEAN: The letter "F" or "T" is output, if the value is true or false, respectively.

CHAR: The character is output untranslated. If the character is an ASCII nongraphic, behavior depends on the output device.

INTEGER: Values are output in the range -32768..32767. Nonnegative values are output with a leading space.

REAL: Values are output in nonexponential or exponential format, depending on their magnitude. Nonnegative values are output with a leading space.

STRING: The dynamic value of the string is output untranslated. The length byte is not output but defines the number of characters written.

ENUMERATED: The ordinal value is output as a two-digit hexadecimal number preceded by "$".

7.2.5 Predeclared Functions

As in Standard Pascal, the predeclared function identifiers are considered to be declared at a level above the program block; they can be redefined in the program.

Instant Pascal implements, in accordance with Pascal, all the predeclared functions of Pascal except the Boolean functions EOF, EOLN. In addition, Instant Pascal includes the predeclared function FUNC.

Following is the list of Instant Pascal predeclared functions:

ABS, ARCTAN, CHR, COS, EXP, FUNC, LN, ODD, ORD, PRED, ROUND, SIN, SQR, SQRT, SUCC, TRUNC.

All are defined as in Pascal except FUNC, which is defined in 7.1.

7.3 COMPARISON TO STANDARD PASCAL

This section is organized to facilitate a direct comparison between Instant Pascal and Standard Pascal, as reported in the "Report" section of "Pascal User Manual and Report", Second Edition. The number in parenthesis parallels the numbering of the Report. The content of each paragraph is a statement of any deviation which exists from Standard Pascal as defined in that paragraph of the Report.

7.3.1 Notation, Terminology, and Vocabulary (3)

a. Lower-case letters are not supported.
b. The character "[" is F1 on the AIM 65 keyboard; "]" is F2.
c. "^" is not supported.

d. "{" is replaced by "(*"; and "}" is replaced by "*)".
e. NIL is not supported.
f. FILE is not supported.
g. OTHERWISE is added to the list of special symbols.
h. Comments may only occur in places where text units are accepted.

7.3.2 Identifiers, Numbers, and Strings (4)

a. The underline "_" may appear in an identifier wherever a digit is allowed.

b. Strings of any length N in quotes are constants of the type ARRAY[0..N] OF CHAR, where the 0th element contains the value N.

7.3.3 Constant Definitions (5)

<constant identifier> is excluded from the <constant> definition.

7.3.4 Data Type Definitions (6)

<pointer type> is not supported.

a. Structured Types (6.2)

<file type> is not supported.

b. Record types (6.2.2)

(1) Variants are not supported.
(2) There must be a semicolon following the <field list> and preceding the END.

c. Set types (6.2.3)

(1) The operators +, -, and * on sets are not supported.
(2) The base type of a set may have a cardinality of at most 8.

d. File type (6.2.4)

(1) FILE type is not supported.
(2) TEXT type is not supported.

e. Pointer types (6.3)

Pointer type is not supported.

7.3.5 Declarations and Denotations of Variables (7)

Redefine <variable declaration> as follows:

```
<variable declaration> ::=
     <variable id> { , <variable id> } : <type>
<variable id> ::= <identifier>|<identifier>=$ <hex digit>
     { <hex digit> }
<hex digit> ::= <digit> | A | B | C | D | E | F
```

(1) Component variables (7.2)

<file buffer> is not supported.

(2) File buffers (7.2.3)

This form of variable is not supported.

(3) Referenced variables (7.3)

This form of variable is not supported.

7.3.6 Expressions (8)

a. NIL is not supported.

b. <set> is not supported.

(1) Multiplying operators (8.1.2)

Operations on sets are not supported.

(2) Adding operators (8.1.3)

Operations on sets are not supported.

7.3.7 Statements

a. Procedure statements (9.1.2)

Procedure and function parameters are not supported.

b. Goto statements (9.1.3)

It is not possible to jump out of a procedure or function.

c. Case statements (9.2.2.2)

(1) Redefine <case statement>:

```
<case statement> ::= CASE <expression> OF
<case list element> { ;<case list element>}
     <otherwise element> END
<otherwise element> ::= OTHERWISE:<statement> |
     <empty>
```

(2) The END may be preceded by a semicolon.

d. With statements (9.2.4)

(1) On page 49 of the Jensen and Wirth Pascal User Manual and Report, there is an example showing ambiguity between a variable and field identifier. Instant Pascal does not support such ambiguity.

7.3.8 Procedure Declarations (10)

The word symbols PROCEDURE and FUNCTION may not be used in a formal parameter section.

a. File handling procedures (10.1.1)

These procedures are not supported.

b. Dynamic allocation procedures (10.1.2)

These procedures are not supported.

c. Data transfer procedures (10.1.3)

These procedures are not supported.

7.3.9 Predicates (11.1.2)

EOF, EOLN are not supported.

7.3.10 Input and Output (12)

a. The predeclared files INPUT and OUTPUT are not named. (However, see 7.2.4 of this document.)

(1) The procedure read (12.1)

(a) The only file supported is INPUT, which is never named.

(2) The procedure readln (12.2)

(a) The only file supported is INPUT, which is never named.

(b) READLN is interpreted as in 7.2.4 of the present document.

(3) The procedure write (12.3)

(a) The only file supported is OUTPUT, which is never named.

(b) The write-parameter separator ":" is not supported, nor are the formatting functions which they control.

(4) The procedure writeln (12.4)

The only file supported is OUTPUT, which is never named.

(5) Additional procedures (12.5)

PAGE is not supported.

7.3.11 Programs (13)

The <program parameters> part is empty.

APPENDIX A
SUMMARY OF COMMANDS

TEXT UNIT POINTER MOVEMENT

↑<T>	Position to top text unit.
↑<B>	Position to bottom text unit. (Always ".")
↑<U>/n	Position up n text units. ("."=forever; RETURN=1)
↑<D>/n	Position down n text units.
↑<F> argument string	Find the line containing the argument string.

DEVICE SOURCE I/O

↑<R>IN=device	Read lines of Pascal source text until an empty line. Device=RETURN is the keyboard.
↑<L>/n OUT=device	List n text units to the specified device. Device = CR is the on-board printer or attached terminal.

SOURCE TEXT EDITING

↑<K>/n	Delete n text units.
↑<I>	Insert one line of source text.
↑< >	(space) List current text unit.
↑<V>	View five text units centered on the current text unit.

TOGGLES

+<S>	Toggle statement trace.
+<A>	Toggle assignment trace.

PROGRAM CONTROL

+<C>	Check program syntax; invoke the Binder.
+<G>	Execute program. Starts at beginning of program unless Break-in-progress is true.
+<X> one statement	Immediately execute the statement.
+<,>	Execute and trace program, returning to command interpreter with Break-in-progress true at start of next statement. Starts at beginning of program unless Break-in-progress is true.

MISCELLANEOUS

+<W>	Redefine page width.
+<N>	Initialize to empty program.
+<M>	Report number of free memory bytes.
+<Z>	Initialize Instant Pascal (cold start).

APPENDIX B

LANGUAGE SUMMARY TABLES

Reserved Words

AND	END	NIL	SET
ARRAY	FILE	NOT	THEN
BEGIN	FOR	OF	TO
CASE	FUNCTION	OR	TYPE
CONST	GOTO	OTHERWISE	UNTIL
DIV	IF	PROCEDURE	VAR
DO	IN	PROGRAM	WHILE
DOWNTO	LABEL	RECORD	WITH
ELSE	MOD	REPEAT	

Reserved Symbols

.	,	'	;		
+	-	*	/	:=	:
=	<=	>=	<>	<	>
(	)	(*	*)	[	]

Standard Identifiers

Constants:	FALSE	TRUE		
Types:	BOOLEAN	CHAR	INTEGER	
	REAL	STRING	TEXT	
Functions:	ABS	FUNC	ROUND	SUCC
	ARCTAN	LN	SIN	TRUNC
	CHR	ODD	SQR	
	COS	ORD	SQRT	
	EXP	PRED	SUBR	
Procedures:	READ	READLN	WRITE	WRITELN
	BREAK			
Procedures:	READ	READLN	WRITE	WRITELN
	BREAK			

Real and Integer Functions

Word	Function	Operand (s)	Result
ABS	Absolute Value	Integer, Real	Same as Operand
ARCTAN	Arc Tangent	Integer, Real	Real
COS	Cosine	Integer, Real	Real
EXP	Exponent	Integer, Real	Real
LN	Natural Logarithm	Integer, Real	Real
ODD		Integer	Boolean
ROUND	Round	Real	Integer
SIN	Sine	Integer, Real	Real
SQR	Square	Integer, Real	Same as Operand
SQRT	Square Root	Integer, Real	Real
TRUNC	Truncate	Real	Integer

Character Functions:

Word	Function	Operand (s)	Result
CHR	Character	Integer	Character
ORD	Ordinal	Scalar except Real	Integer
PRED	Predecessor	Scalar except Real	Same as Operand
SUCC	Successor	Scalar except Real	Same as Operand

Machine Language Functions

Word	Function	Operand (s)	Result
SUBR	Subroutine Call	Address	-
FUNC	Function Call	Address, Data	Char

Debug Functions

Word	Function	Operand (s)	Result
BREAK	Interrupt Execution	-	-

Arithmetic Operators

Symbol/Word	Operation
+	Addition (plus sign)
-	Subtraction (minus sign)
*	Multiplication
/	Division
DIV	Division (yields a truncated integer result)
MOD	Modules (yields the remainder of division)

Relational Operators

Symbol/Word	Operation
>	Greater than
<=	Greater than or equal to
<	Less than
<=	Less than or equal to
=	Equal to
<>	Not equal to

APPENDIX C

PAGE 0 MEMORY MAP

The Instant Pascal software uses page 0 locations 06-B4 and FC-FF. (All addresses in this Appendix are hexadecimal.) The following locations, in particular, may be of direct interest to the user.

Hex Address	No. Bytes	Cold Start Value	Warm Start Value	Parameter Description
06-07	2	0200	-	The first address used by the software.
08-09	2	0FFF	-	The last address used by the software. (one less than the answer to the MEMORY SIZE? question).
12-13	2	0EC2	-	The address of the first byte of the Instant Pascal program. This address decreases as the program grows.
14-15	2	0EC3	-	The address of the last byte of the Instant Pascal program.
16-17	2	0FC4	-	The starting address of the translator's 60-byte input area. The input area's last byte is at the address in 08.

Hex Address	No. Bytes	Cold Start Value	Warm Start Value	Parameter Description
18-19	2	OEC4	-	The starting address of the translator's 256-byte output area. The output area immediately precedes the input area, and the object program immediately precedes the output area.
1A-1B	2	02FC	-	Location $0200-$02FF contains fixed tables. A run-time stack normally begins at $0300 and builds upward. $1A-$1B contains the initial address of the the top of the empty stack which is 4 less than the address of the first byte to be used by the stack, that is, $02FC. The answer to the + <M> question is ($12)-($1A)-250, that is, 246 less than the number of free bytes from the bottom of the stack to the start of the program.

NOTE

You may open up a hold from $0300 to a higher address by increasing the address in $1A after the initialization sequence. For example, if you wish Instant Pascal not to use $0300-$03C9, the address is $1A should be set to $03C6 or greater.

Hex Address	No. Bytes	Cold Start Value	Warm Start Value	Parameter Description
1E	1			Holds the contents of the processor's S register after an execution-time error or break. Upon return to the interpreter from a break, a check is made to ensure that the current value of S does not exceed the contents of $1E, which would indicate the loss of information.
2A-2B	2	0014	-	Contains the answer to the WIDTH? question.
FE-FF	2			Used by SUBR and FUNC to communicate the address of a Pascal data item to a machine-language subroutine (See 7.1).

APPENDIX D

ASCII CHARACTER SET

HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII
00	0	NUL	20	32	SP	40	64	@	60	96	
01	1	SOH	21	33	!	41	65	A	61	97	a
02	2	STX	22	34	"	42	66	B	62	98	b
03	3	ETX	23	35	#	43	67	C	63	99	c
04	4	EOT	24	36	$	44	68	D	64	100	d
05	5	ENQ	25	37	%	45	69	E	65	101	e
06	6	ACK	26	38	&	46	70	F	66	102	f
07	7	BEL	27	39	'	47	71	G	67	103	g
08	8	BS	28	40	(	48	72	H	68	104	h
09	9	HT	29	41	)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	|
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94		7E	126	
1F	31	VS	3F	63	?	5F	95		7F	127	DEL

NUL - Null
SOH - Start of Heading
STX - Start of Text
ETX - End of Text
EOT - End of Transmission
ENQ - Enquiry
ACK - Acknowledge
BEL - Bell
BS - Backspace
HT - Horizontal Tabulation
LF - Line Feed
VT - Vertical Tabulation
FF - Form Feed
CR - Carriage Return
SO - Shift Out
SI - Shift In

DLE - Data Link Escape
DC - Device Control
NAK - Negative Acknowledge
SYN - Synchronous Idle
ETB - End of Transmission Block
CAN - Cancel
EM - End of Medium
SUB - Substitute
FSC - Escape
FS - File Separator
GS - Group Separator
RS - Record Separator
US - Unit Separator
SP - Space (Blank)
DEL - Delete

APPENDIX E

EXECUTION-TIME DIAGNOSTICS

Note: Error numbers with an asterisk are internal checks. Their occurrence may indicate loss of memory integrity.

01*

02*

03 Value Stack has exceeded available space.

04*

05*

06 Too few actual parameters

07 Too many actual parameters

08 Actual parameter of a VAR formal parameter is an expression.

09*

10*

11 Nonordinal type where ordinal (scalar) type required.

12 Maximum permissible dynamic statement depth of 24 exceeded.

13 Ordinal value computation out of range (Array and set selection)

14*

15*

16 Expression result type should be Boolean and is not.

17 Function identifier on left-hand side of assignment does not refer to a declared function.

18*

19 Type compatibility error in assignment or actual value parameter.

20 Type (s) improper for operator.

21 Attempt to make array selection on a nonarray.

22 String length excessive for operation.

23 Empty string improper for operation.

24*

25 Attempt to negate a nonnumeric.

26 Operand should be Boolean.

27*

28 GOTO with destination textual depth greater than that of GOTO.

29 Actual/formal parameter type discrepancy.

30 Control variable has changed in FOR loop.

31 Argument of CHR not in 0..255.

32 Value less than lower limit of subrange.

33 Value greater than upper limit of subrange.

34 Argument not ordinal (scalar).

35 Argument not real or integer.

36 GOTO refers to an undefined label.

37 Actual/formal VAR parameter type discrepancy.

38 Array reference to STRING with noninterger subscript.

39 Right operand of IN not a set or packed set.

40 Source String to long for receiving string.

41 Improper argument of record select (.) operator.

42*

43 No match between value of expression and CASE constants.

44 Argument of WITH is not a record variable.

45 More than the maximum of 16 records in a program.

46 Field identifier not preceded by "." and not in the scope of a WITH.

47*

48 Processor stack pointer is higher than when break discontinued execution; Break-in-progress is now false and + G will start at beginning.

49 Processor stack is about to overflow.

50 Attempted division by zero.

51 Floating-point overflow.

52 Fixed-point overflow in TRUNC or ROUND.

53 MOD with negative divisor.

APPENDIX F

SELECTED BIBLIOGRAPHY

Conway, R., D. Gries and E.C. Zimmerman, "A Primer on PASCAL", Winthrop Publisher (1976).

Dahl, O.J., E.W. Dijkstra, C.A.R. Hoare, "Structured Programming", Academic Press (1972).

Dijkstra, W.E., "A Discipline of Programming", Prentice Hall (1976).

Fox, D. and M. Waite, "Pascal Primer", Howard W. Sams Co., Inc. (1981).

Findlay, W., and D.A. Watt, "PASCAL-An Introduction to Methodical Programming", Computer Science Press (1978).

Grogono, P., "Programming in PASCAL", Addison Wesley (1978-Revised Edition 1980).

Jensen, K., And N. Wirth, "Pascal User Manual and Report", Springer-Verlag (1978)

Kiebuntz, R.B., "Structured Programming and Problem-Solving with PASCAL", Prentice-Hall (1978)

Schneider, M., S. Weingart and S. Perlman "An Introduction to Programming and Problem Solving with PASCAL", John Wiley & Sons (1978).

Tiberghien, "The PASCAL Handbook", Sybex (1980).

Welsh, J., and J. Elder, "Introduction to PASCAL", Prentice Hall (1979).

Wilson, I.R., and A.M. Addyman, "A Practical Introduction to PASCAL", Springer-Verlag (1978).

Wirth, N., "Systematic Programming-An Introduction", Prentice Hall (1973).

Yourdon, E., "Techniques of Program Structure and Design", Prentice Hall (1975).

Zaks, R., "Introduction of PASCAL Including UCSD PASCAL", Sybex (1980).