Pascal-S

Pascal Subset Compiler/interpreter

Scott A. Franco

[1 Overview of Pascal-S 4](#_Toc507581844)

[1.1 Introduction 4](#_Toc507581845)

[1.2 Language of Pascal-S 4](#_Toc507581846)

[1.3 My use of Pascal-S 5](#_Toc507581847)

[1.4 The strange story of the instruction "36" 5](#_Toc507581848)

[1.5 Onward and upward? 7](#_Toc507581849)

[2 Using Pascal-S 7](#_Toc507581850)

[2.1 Configuring Pascal-S 7](#_Toc507581851)

[2.2 Compiling and running Pascal programs with Pascal-S 9](#_Toc507581852)

[2.3 Other operations 9](#_Toc507581853)

[2.4 Reliance on Unix commands in the Pascal-S toolset 10](#_Toc507581854)

[2.5 The “flip” command and line endings 10](#_Toc507581855)

[3 Building the Pascal-S system 11](#_Toc507581856)

[3.1 Compiling and running Pascal-S with an existing ISO 7185 compiler 11](#_Toc507581857)

[3.2 Notes on using existing compilers 12](#_Toc507581858)

[3.2.1 GPC 12](#_Toc507581859)

[3.2.1.1 GPC on Cygwin 12](#_Toc507581860)

[3.2.1.2 GPC for mingw 13](#_Toc507581861)

[3.2.2 FPC 14](#_Toc507581862)

[3.2.2.1 Obtaining FPC 14](#_Toc507581863)

[3.2.2.2 Configure and build the FPC version 14](#_Toc507581864)

[4 Files in the Pascal-S package 14](#_Toc507581865)

[4.1 Directory: bin 15](#_Toc507581866)

[4.2 Directory: c\_support 16](#_Toc507581867)

[4.3 Directory: doc 16](#_Toc507581868)

[4.4 Directory: fpc 16](#_Toc507581869)

[4.5 Directory: Fpc/linux\_X86 16](#_Toc507581870)

[4.6 Directory: mac\_X86 16](#_Toc507581871)

[4.7 Directory: fpc/windows\_X86 16](#_Toc507581872)

[4.8 Directory: gpc 16](#_Toc507581873)

[4.9 Directory: gpc/linux\_X86 16](#_Toc507581874)

[4.10 Directory: mac\_X86 17](#_Toc507581875)

[4.11 Directory: gpc/windows\_X86 17](#_Toc507581876)

[4.12 Directory: ip\_pascal 17](#_Toc507581877)

[4.13 Directory: ip\_pascal/windows\_X86 17](#_Toc507581878)

[4.14 Subdirectory: sample\_programs 17](#_Toc507581879)

[4.15 Directory: source 17](#_Toc507581880)

[5 Pascal-S Machine Instruction Set Reference 17](#_Toc507581881)

[5.1 Contents 17](#_Toc507581882)

[5.2 Instruction Quick Reference 18](#_Toc507581883)

[5.3 Instruction Details 19](#_Toc507581884)

[6 Testing Pascal-S 30](#_Toc507581885)

[6.1 Running tests 30](#_Toc507581886)

[6.1.1 testprog 30](#_Toc507581887)

[7 Licensing 31](#_Toc507581888)

# Overview of Pascal-S

This section contains background material on Pascal-S. If you want to get started using Pascal-S now, skip to “Using Pascal-S”

## Introduction

Pascal-S was a companion interpreter/compiler implementation for a subset of the Pascal language created in about 1975 by Niklaus Wirth in ETH, Zurich. It was intended as a lightweight implementation for students to use on a batch or timeshare terminal based computer system in an era when computers were expensive with limited access. It was created as part of the Pascal-P family.

The prime use of the Pascal-P and Pascal-S family today is to learn how compilers work. Pascal itself is a very regular language with straightforward rules and syntax. Further, most of the various versions of Pascal-P and Pascal-S still exist, and thus you can pick a subset and level of complexity for the implementation you want to study:

|  |  |
| --- | --- |
| Implementation | Lines of code |
| Pascal-S | 1930 |
| Pascal-P2 | 4703 |
| Pascal-P4 | 5218 |
| Pascal-P5 | 9046 |

The Pascal-P family is different than Pascal-S. Pascal-S is a combined compiler/interpreter. Pascal-P generates intermediate code in text file form, and has a separate compiler and interpreter section. Although Pascal-S was created by the same group (Niklaus Wirth’s ETH students and himself) as Pascal-P, it is significantly different in form.

The original paper that discusses the Pascal-S system is here:

<http://www.standardpascal.com/Wirth-PascalS.pdf>

And also in the Pascal-S archive. That article is a very complete explaination of the implementation, and I won’t repeat it here. It includes a complete copy of the original source for Pascal-S.

## Language of Pascal-S

Pascal-S omits many features of full Pascal, including:

* Subrange types are not implemented (type a = 1..10).
* Scalar types are not implemented (type a = (one, two, three)).
* Sets are not implemented.
* Dynamic variables (pointers) are not implemented.
* Variant records are not implemented.
* Procedure/function parameters.
* gotos.
* The predefined functions succ and pred only function on type char.
* Only files of type "text" can be used, and then only the ones that are predefined by Pascal-S, which are "input", "output".
* Curly bracket comments {} are not implemented.
* Packing, the "packed" keyword, and the "pack" and "unpack" procedures, are not implemented.
* "get", "put", and file buffer variable handling are not implemented.
* Strings are unimplemented, except for literals as parameters to write/writeln, and those cannot have field lengths applied to them.
* The "forward" specifier, and forwarded procedures and functions, are not implemented.
* The predeclared identifiers maxint, text, page, new, dispose, and the functions they represent, are not present.
* The procedures reset, rewrite, pack and unpack are not implemented (they are recognized as valid predefined procedures, but give an 'unimplemented' error on use).

Pascal-S ***does*** include a few odd alternatives to standard Pascal lexical tolkens:

* “#” is an alternative to “<>”
* “&” is an alternative to “and”

## My use of Pascal-S

I encountered Pascal-S in 1982 when it was published along with a series of other papers in book form (D.W.Barron “Pascal: The language and its Implementation”). I didn’t have a computer readable form, so I used it both as a compiler example and then typed it in and used it to test my own Pascal compiler against. It’s hard to appreciate now, but it was one of the very few examples of a working compiler back then.

It was not until the 1990s before I was able to compile and run Pascal-S in its original form, since my original 1980’s compiler didn’t support variant records (at that time), and in any case, it would not have fit into the very small 64kb available in a Z80 based computer.

## The strange story of the instruction "36"

After several years of playing with Pascal-S as both a compiler test and a toy in its own right, I decided to formally verify Pascal-S. This is done using the same techniques I use to verify a full blown compiler, but cut down to the subset Pascal-s uses. Pascal-s failed this test, and my quest to find out why led me on a history tour of Pascal-S, and the CDC 6000 computer it was originally designed to run on.

In the course of running the test material through Pascal-S, I noticed that certain real values were coming out with the wrong sign, or were entirely wrong.

Examining the source for Pascal-S, I came across an interesting fact. The code for negation of integer or real appears very wrong.

Looking at the code, in simpleexpresion, a code "36" is emitted for negation:

begin (\* simpleexpression \*)  
  if sy in [plus, minus] then  
    begin op := sy; insymbol;  
      term(fsys+[plus, minus], x);  
      if x.typ > reals then error(33) else  
        if op = minus then emit(36)  
    end else

Note that x.typ is:

types = (notyp, ints, reals, bools, chars, arrays, records);

so x.typ > reals is a test for integer (ints) or real (reals), because it won't be a notyp. So the code 36 is emitted for both negate real and negate integer. Now if you look below to interpret, here is what instruction 36 does:

36: s[t].i := - s[t].i;

The .i part means the integer part of a stack record. Ie., it negates integer. But the operand, on the stack, of instruction 36 could be real !

Now my copy of Pascal-s was actually typed in from the original article by Wirth on Pascal-s, of which I have a copy via D. W. Barron's "Pascal: The Language and its implementation". I typed that in quite a few years ago. I note that some other, original copies have appeared on the net. Anyways, I figured there was probably some error. I could not believe that Wirth, a very careful (and clever) programmer would have an undiscovered bug in his code. Further, Pascal-s was used by hundreds of programmers in universities for simple Pascal work back in the 1970's.

Examining the original Wirth article carefully, there appeared to be no error in copying it. The original code had the "bug" in it as well.

Unwilling to simply declare it to be a bug and fix it, I began to think about what I knew about the CDC 6000 series computer, and looking at some CDC 6600/6000 reference material. The CDC has a very unique floating point format. The CDC 6000 uses 60 bit words for both integer and floating point formats. In fact, a floating point number is essentially the mantissa, represented by a 60 bit signed integer, with the exponent stuck into the high order bits between bit 48 and bit 59. The exponent even is allowed to appear as the complement of itself so that it remains valid if the entire number is complemented (the CDC 6600 used 1's complement numbers).

The result is that a negate instruction works equally well on both real and integer formats, on the CDC 6600 series computers.

Now the declaration of a stack operand in Pascal appears as:

s: array [1..stacksize] of    (\* blockmark:                  \*)  
    record case types of       (\*    s[b+0] = fct result      \*)  
      ints:   (i: integer);    (\*    s[b+1] = return adr      \*)  
       reals:  (r: real);       (\*    s[b+2] = static link     \*)  
       bools:  (b: boolean);    (\*    s[b+3] = dynamic link    \*)  
       chars:  (c: char);       (\*    s[b+4] = table index     \*)  
       notyp, arrays, records: ()  
     end;

IE., its a variant record, which means that all of the different types overlay each other in memory. In fact, since all of these objects appear as single machine words (60 bits in the case of the CDC 6000), these are different forms of a single machine word.

The result was that there did not need to be a different form for negation on real or integer values. Only one type of negate was needed. Hence the code was correct - but only for the CDC 6600 series computers. On any other computer the identical, and correctly compiled code, yields the incorrect result, which could be anything from the wrong sign to total data corruption, since the wrong data type is being manipulated.

Wirth probably forgot about this feature of Pascal-S. It is mentioned nowhere in the original Wirth written Pascal-S document.

Hows that for a weird code dependency ?

## Onward and upward?

One of the most instructive and fun things to do with an existing compiler/interpreter is to extend it. Of course, the other members of the Pascal-P series also represent exactly that. Pascal-P2 and Pascal-P4 for a more complete language that can possibly made to “bootstrap”, or self compile, and Pascal-P5 for a complete (non-subsetted) implementation of Pascal. However, adding back standard Pascal features to Pascal-S is also a good way to learn, and the result is a single compiler/interpreter instead of a separate compiler and interpreter as represented by the Pascal-P series.

A lot of people who work on the Pascal-P series tell me they want to add custom features to the language, or even change it to work on a different language (perhaps “Pascal like”). I think that is great, and encourage that. I would request that you don’t name the result Pascal-S (or even “Pascal” if the result is incompatible), but I admit I have lost track of the number of people who have made changes in the Pascal-P series compilers and kept the name (perhaps including me).

One of the things that Pascal-S has going for it as a starter project is that it is simpler than the Pascal-P compilers.

# Using Pascal-S

## Configuring Pascal-S

Pascal-S has a simple configuration script to set up the binary, script files and compiler in use for the system, that uses the proper defaults for your system:

[Windows]

> setpath

> configure

> make

[Linux/Mac]

$ ./setpath

$ ./configure

$ make

You can avoid “setpath” by placing the ./bin directory on your path.

The configure script attempts to automatically determine the environment you are running under, choose the correct compiler, bit width of your computer, etc. You can override this by using the options for configure:

|  |  |
| --- | --- |
| Option | Meaning |
| --gpc | Selects the GPC compiler. |
| --ip\_pascal | Selects the IP Pascal compiler. |
| --fpc | Selects the FPC Pascal compiler. |
| --help | Prints a help menu. |

The configure script will take the preconfigured versions of the Pascal-S binaries, the script files and other files and install them for the specified compiler. The Pascal-S system is configured by default for GPC Pascal running on windows, and can be left as such if desired.

Note that if you have more than one acceptable compiler resident, the configure script will choose the first one found in the order IP Pascal, GPC Pascal, and then FPC Pascal.

## Compiling and running Pascal programs with Pascal-S

To simply compile a run a program, use the ps batch file:

C:\> ps hello

When a pascal program is run this way, it gets it's input from the terminal (you), and prints its results there. The ps script accommodates the compiler that was used to build the system, and therefore you don’t need to know the exact command format of the executable.

All files in Pascal-S are anonymous (as in “no filename”), and only last the length of the program run. If you need to read from a file or write to a file use redirection:

C:\> ps test < myinputfile > myoutputfile

You will find you can get a lot of tasks done this way.

## Other operations

Within the Pascal-S toolset, you will find a series of scripts to perform common operations using Pascal-S. This includes building the compiler and interpreter using an existing ISO 7185 compatible compiler, and also testing Pascal-S.

The scripts used in Pascal-S are designed to be independent of what operating system you are running on. The Pascal-S system as been successfully run on the following systems:

* Windows
* Ubuntu linux
* Mac OS X

To enable this to work, there are two kinds of scripts available, one for DOS/Windows command shells, and another for Unix/Bash. These two script files live side by side, because the DOS/Windows scripts use a .bat extension, and Bash scripts use no extentions. Thus, when a script command is specified here, the particular type of script file is selected automatically.

The only exception to this rule is that Unix users commonly do not place the current directory in the path. This means to execute a script file in the current directory, you need to specify the current directory in front of the script. For example:

~/Pascals$ ./ps hello

## Reliance on Unix commands in the Pascal-S toolset

Most of the scripts in this package, even the DOS/Windows scripts, rely on Unix commands like cp, sed, diff, chmod and others. I needed a reasonable set of support tools that were command line callable, and these are all both standard and reasonable.

For Windows, the Cygwin toolset is available:

http://www.cygwin.com

Note that to run the cygwin tools, you will need the environment variable:

CYGWIN=nodosfilewarning

This prevents cygwin utilities from complaining about dos mode file specifications.

An alternative to Cygwin is the Mingw toolkit. Mingw uses GNU programs that are compiled as native Windows .exe files without special .dll files. It typically has better integration with Windows than Cygwin, since it does not try to emulate Unix on Windows. MinGW is available at:

http://www.mingw.org/

Where possible, I have tried to use DOS/Windows commands. The scripts are available in both DOS/Windows and bash versions. I could have just required theuse of bash, which is part of the cygwin toolkit, but my aim is not to force Windows users into a Unix environment.

## The “flip” command and line endings

Every effort was made to make the Pascal-S compile and evaluate system idenpendent of what system it is running on, from Windows command shell, to Linux with Bash shell. One common thing I have found is that several utilities don’t appreciate seeing a line ending outside of their “native” line ending, such as CRLF for Windows, and LF for linux. Examples include “diff” (find file differences) and Bash.

Therefore many of the scripts try to remove the line ending conserations, either by ignoring such line endings, or by converting all of the required files to the particular line ending in use.

The key to this is the “flip” utility. After searching for several line ending converters, “flip” was found on the most number of systems, as well as being one of the most clear and reliable utilities (it translates in both directions, it tolerates any mode of line ending as input, will not corrupt binaries, etc.).

Unfortunately, even flip was not found on some systems. The simpliest way to fix this was to include the flip.c program with the distribution, then let you compile to form a binary on your system to replace the utility.

To make the flip utility, you run:

$ make\_flip

Then flip will exist in the root directory.

# Building the Pascal-S system

## Compiling and running Pascal-S with an existing ISO 7185 compiler

You do not need to compile Pascal-S unless you are using an alternative compiler or installation. The current Pascal-S has been compiled and run with the following compilers and operating systems:

|  |  |
| --- | --- |
| Compiler | Installations |
| IP Pascal | Windows |
| GPC | Windows, Ubuntu, Mac OSx |

First, you must have a ISO 7185 Pascal compiler available. There are several such compilers, see:

http://www.standardpascal.org/compiler.html

You will probally need to compile pcom.pas and pint.pas with the ISO 7185 Pascal compatibility mode option on for your compiler. See your documentation for details.

If you are using a compiler or version of a compiler that is not tested to ISO 7185 standards, you will want to make sure that it is ISO 7185 compliant. There are tests for a full ISO 7185 compiler implementation included in the Pascal-P5 project.

To compile Pascals, the Pascal-S compiler/interpreter, use the make utility:

make Make pascals.

make pascals Make pascals.

make clean Remove all product files (usually to insure a clean remake).

To run the other programs and batch files, you should modify the following files to work with your compiler:

ps[.bat] The single program compile and run batch file.

The reason you need to change these files is because pascals uses the header file "srcfil" to read the source file. You need to find out how to connect this file in the program header to an external named file.

For example, in IP Pascal and FPC, header files that don't bear a standard system name (like "input" and "output") are simply assigned in order from the command line. Thus, ps.bat is simply:

ps %1.pas

Where %1 is the first parameter from the command line.

If your compiler does nothing with header files at all, you will probally have to change the handling of the prd and prr files to get them connected to external files. To do this, search pcom and pint for "!!!" (three exclamation marks). This will appear in comments just before the declaration, reset and rewrite of these files.

## Notes on using existing compilers

When using an existing ISO 7185 compatible compiler, the configure script will install all the files needed to work with that particular compiler. Thus these hosts will work without any modification.

### GPC

GPC (GNU Pascal Compiler) is used in the following version:

GNU Pascal version 20070904, based on gcc-4.1.3 20080704 (prerelease) (Ubuntu 2.1-4.1.2-27ubuntu2).

Copyright (C) 1987-2006 Free Software Foundation, Inc.

I have had several difficulties with other versions of GPC, which give errors on standard ISO 7185 source, or crash, or other difficulties. The GPC developers announced they were halting development on GPC in the gpc mailing list. Please see their web page:

<http://www.gnu-pascal.de>

For any further information.

The main difficulty with GPC vis-a-vie Pascal-S is that testing of the GPC compiler for ISO 7185 compatability was not regularly done on GPC releases. Thus, otherwise working GPC releases were not able to compile and run standard ISO 7185 source code.

Because of this, I can only recommend the above version of GPC be used, which compiles and runs Pascal-S error free.

In addition, please be aware that I have not run the GPC compiler, including the above version, through a current ISO 7185 compliance test such as appears here. My only concern is that GPC be able to complile and run Pascal-S, and that the resulting Pascal-S runs the compliance tests. I leave it for others to run full compliance for GPC itself.

#### GPC on Cygwin

The current Cygwin release as of 2012/03/26 does not work, since it uses GPC 2005, and is broken at that (it has the .dlls for GPC installed incorrectly).

A procedure to use GPC under the current Cygwin I have used is as follows:

1. Install the latest version of Cygwin (the one I tried is Cygwin/X, a very useful package).
2. Place c:\cygwin\bin on your path.
3. Go to the website:

<http://www.gnu-pascal.de/binary/cygwin/>

And download and install:

[gpc-20070904-with-gcc.i686-pc-cygwin.tar.gz](http://gnu-pascal.de/contrib/chief/win32/cygwin/gpc-20070904-with-gcc.i686-pc-cygwin.tar.gz)

(4.4mb, gpc-20070904, based on gcc-3.4.4, with gcc-3.4.4 support files)

1. After installing this package in an appropriate directory, say c:\gpc, modify your path to include c:\gpc\usr\bin ahead of the c:\cygwin\bin directory in the path.
2. Add cygwin=nodosfilewarning (as stated in section 2.5 “Reliance on Unix commands in the Pasca-S toolset”).

Now you will be able to follow the normal gpc instructions here to get Pascal-S running, using the standard Windows command shell. Note that this trick won’t work with the command shells Cygwin provides.

To reiterate the steps that follow:

$ configure --gpc Configure for GPC compiler.

$ make Build the Pascal-S binary.

$ regress Run the regression suites to check the Pascal-S compiler.

#### GPC for mingw

Mingw (Minimal GNU for Windows) is a different port of the GNU catalog for windows that runs directly on windows. That is, each binary is statically linked with its support library, and it is designed to work with windows directly.

As Cygwin has become more and more a full emulation of the Unix environment (a good thing), it has become less usable in interaction with other Windows programs. Thus I have found the mingw package more cooperative for every day Windows work.

Mingw does not come natively with GPC installed (or much else). I recommend you also pick up the MSYS package for mingw, which is a series of GNU programs that are compiled to run in the windows environment using Mingw.

To get the mingw distribution of GPC, follow the steps:

1. Go to the website:

<http://www.gnu-pascal.de/binary/mingw32/>

And download and install:

[gpc-20070904-with-gcc.i386-pc-mingw32.tar.gz](http://gnu-pascal.de/contrib/chief/win32/cygwin/gpc-20070904-with-gcc.i686-pc-cygwin.tar.gz)

(4.4mb, gpc-20070904, based on gcc-3.4.5, with gcc-3.4.5 support files)

1. After installing this package in an appropriate directory, say c:\gpc, modify your path to include c:\gpc\usr\bin directory in the path.

Note that this is based on a slightly different version than Cygwin.

To reiterate the steps that follow:

$ configure --gpc Configure for GPC compiler.

$ make Build the Pascal-S binaries.

$ regress Run the regression suites to check the Pascal-S compiler.

### FPC

FPC is can be used in versions 3.0.4 or later:

Free Pascal Compiler version 3.0.4 [2017/10/06] for i386

Copyright (c) 1993-2017 by Florian Klaempfl and others

#### Obtaining FPC

FPC can be found at the web site:

<https://www.freepascal.org/>

Please make sure you download and run at least version 3.0.4.

#### Configure and build the FPC version

The steps to configure and build with FPC after Pascal-S and FPC are installed are:

$ setpath

$ configure –fpc

$ make

And I recommend a check of the Pascal-S compiler with:

$ regress

Which will run a test series on the resulting compiler.

# Files in the Pascal-S package

Note: for script files, both a DOS/Windows (X.bat) and bash script (X) are provided. Their function is identical, one is for use with the DOS/Windows command shell, the other for bash shell.

configure

configure.bat Sets the current compiler to use to create Pascal-S binaries.

INSTALL Installation instructions

LICENSE License information for Pascal-S

Makefile The make file to run builds on Pascal-S. This is customized for a particular host compiler.

NEWS Contains various information about the current release.

README Brief introduction to the project, it points to this document now.

setpath

setpath.bat Adds the “bin” directory to the current path. Used to quickly run procedures in Pascal-S directory.

TODO Contain a list of "to do" items in Pascal-S.

## Directory: bin

diffnole

diffnole.bat Runs a diff, but ignoring line endings (DOS/Windows vs. Unix). Also ignores version numbers in compiler output.

doseol

doseol.bat Fixes the line endings on text files to match the DOS/Windows convention, CRLF.

fixeol

fixeol.bat Arranges the line endings on bash scripts to be Unix, and those of the DOS/Windows scripts to be DOS/Windows line endings. This is required because the editors on the respective systems insert their own line endings according to system, and this can cause problems when they are run on a different system.

make\_flip

make\_flip.bat A script to compile deoln and ueoln and create a flip script for Unix. This is used to replace the “flip” program if required.

pascals

pascals.exe The compiled binary for Windows/Unix. See comments in 2.2 for how to use this. All of the supplied batch files are customized for this version.

ps

ps.bat A batch file that compiles and runs a single Pascal program. You will need to change this to fit your particular Pascal implementation. It uses input and output from the terminal, so it is a good way to run arbitrary programs.

\*\*\* You will need to change this to fit your particular Pascal system \*\*\*

It uses input and output from the terminal, so is a good way to run arbitrary programs.

regress

regress.bat The regression test simply runs all of the possible tests through Pascal-S. It is usually run after a new compile of Pascal-S, or any changes made to Pascals-S.

repo\_ready

repo\_ready.bat Get the current Pascal-S tree ready for a push to the git repo.

testprog

testprog.bat An automated testing batch file. Runs a given program with the input file, delivering an output file, then compares to a reference file.

Testprog is used to test the program files for Pascal-S.

unixeol

unixeol.bat Fixes the line endings on text files to match the Unix convention, LF.

## Directory: c\_support

c\_support/flip.c C program to replace the local version of “flip”, the Unix line ending fixup tool. It is provided in source form here because not all Unix installations have it (for example MAC OS X didn’t have it). This allows you to compile it yourself for your target system.

## Directory: doc

iso7185rules.html A description of the ISO 7185 Pascal language.

Pascals.docx This document, in Word 2010.

The\_Programming\_Language\_Pascal\_1973.pdf

Niklaus Wirth's description of the Pascal language, the last version to come from ETH. This is the equivalent of the "Report", from "Pascal user's manual and report [Jensen and Wirth].

Wirth-PascalS.pdf The original Niklaus Wirth paper describing the Pascal-S system.

## Directory: fpc

This directory contains scripts and other files specifically modified for FPC.

Makefile The FPC specific version of the make input file for Pascal-S builds.

ps

ps.bat The FPC specific version of the ps script.

## Directory: Fpc/linux\_X86

A placeholder for Linux specific files.

## Directory: mac\_X86

A placeholder for Mac OS X specific files.

## Directory: fpc/windows\_X86

A placeholder for Windows specific files.

## Directory: gpc

This directory contains scripts specifically modified for GPC.

Makefile The GPC specific version of the make input file for Pascal-S builds.

ps

ps.bat The GPC specific version of the ps script.

## Directory: gpc/linux\_X86

A placeholder for Linux specific files.

## Directory: mac\_X86

A placeholder for Mac OS X specific files.

## Directory: gpc/windows\_X86

A placeholder for Windows specific files.

## Directory: ip\_pascal

This directory contains scripts specifically modified for IP Pascal.

Makefile The IP Pascal specific version of the make input file for Pascal-S builds.

ps

ps.bat The IP Pascal specific version of the Pascal-S script.

## Directory: ip\_pascal/windows\_X86

A placeholder for Windows specific files.

## Subdirectory: sample\_programs

Note that each test program as a .pas, a .inp (batch input, which could be empty) and output compare file .cmp. See 8.1.1 for more information.

hello.inp

hello.cmp

hello.pas The standard "hello, world" program.

prime.cmp

prime.inp

prime.pas A Pascal benchmark program.

roman.inp

roman.cmp

roman.pas Prints roman numerals. From Niklaus Wirth's "User Manual and Report".

## Directory: source

source/pascals.pas The compiler/interpreter source in Pascal.

# Pascal-S Machine Instruction Set Reference

## Contents

Contains the instruction format for Pascal-S internal instructions. Pascal-S has 64 possible instructions, labled 0 to 63. Each instruction is of the format:

f The instruction code

x 1st operand

y 2nd operand

Each instruction either gets immediate operands from its instruction x or y constants, or operates on the stack. The stack is a pushup stack of variant

records. Each location can contain an integer, real, boolean or character value. The stack contains all expression temporary values, as well as all the locals and globals for the program.

The instruction numbers are grouped according to the number of parameters:

0-3: Instruction uses both x and y parameters.

4-30: Instruction uses y parameter.

31-63: Instruction does not have parameters.

Instruction 36, normally a "no parameter" instruction, takes a single y parameter, and breaks Wirth's original rule. This was required to fix a portability problem.

This document is not meant to be a complete description of Pascal-S. See

"Pascal-s: A Subset and its Implementation" by N. Wirth, for an overall description of Pascal-S, its language, and interpretation.

## Instruction Quick Reference

0 Load address, x = Display index, y = Offset addr

1 Load value, x = Display index, y = Offset addr

2 load indirect, x = Display index, y = Offset addr

3 update display, x = Display update start addr, y = Display update end addr

8 Built in functions, x = Function code

9 Offset, y = Offset

10 Jump, y = Addr

11 Conditional jump, y = Addr

12 Switch

14 Forlup

15 For2up

16 For1down

17 For2down

18 Mark stack

19 Call

20 Index

21 Index

22 Load block

23 Copy block

24 Literal

25 Load real

26 Float

27 Read

28 Write string

29 Write with default field

30 Write with field

31 Halt

32 Exit procedure

33 Exit function

34 Fetch indirect

35 Not

36 Negate

37 Write real with field and fraction

38 Store to stack

39 Real equal

40 Real unequal

41 Real less than

42 Real less than or equal

43 Real greater than

44 Real greater or equal

45 Integer equal

46 Integer not equal

47 Integer less than

48 Integer less than or equal

49 Integer greater than

50 Integer greater or equal

51 Boolean 'or'

52 Integer add

53 Integer subtract

54 Real add

55 Real subtract

56 Boolean 'and'

57 Integer multiply

58 Integer divide

59 Integer modulo

60 Real multiply

61 Real divide

62 Read end of line

63 Write end of line

## Instruction Details

[Apologies for unfinished sections]

0 Load address

x: Display index

y: Offset address

Loads an address for a stack variable by adding the value for the display

indicated by the index y to the offset address y. The result is a stack address as a new top of stack.

1 Load value

x: Display index

y: Offset address

Loads the stack variable by adding the value for the display indicated by the index y to the offset address y, and fetching the value at that address. The result is a stack value as a new top of stack.

2 load indirect

x: Display index

y: Offset address

Loads the stack variable by adding the value for the display indicated by the index y to the offset address y, and fetching the integer value at that address, then fetching the stack value at that address. The result is a stack value as a new top of stack.

3 update display

x: Display update start address

y: Display update end address

Updates a block of display values between the start address and the end address plus one. The display is a set of block mark pointers in an array for quick reference during execution. Block pointers are collected from the stack,starting with the currently active block mark, climbing the static links in the block stack, until the indicated number of display entries are updated.

8 Built in functions

x: Unused

y: Function instruction code

Executes a built in function, by an "extended code" in y. The y codes are:

0: abs of integer

The absolute value of the integer at stack top replaces the top of stack.

1: abs of real

The absolute value of the real at stack top replaces the top of stack.

2: sqr of integer

The square of the integer at stack top replaces the top of stack.

3: sqr of real

The square of the real at stack top replaces the top of stack.

4: odd integer

The odd or even status of the integer at stack top replaces the top of stack as a boolean.

5: chr of integer

The integer at stack top is converted to its character equivalent, and that replaces the stack top as character.

6: ord of character

The character at stack top is converted to its integer equivalent, and that replaces the stack top as integer.

7: succ of character

The successor of the character at stack top is replaces the stack top as character.

8: pred of character

The predecessor of the character at stack top is replaces the stack top as character.

9: round of real

The real at stack top is rounded to its integer equivalent, and that replaces the stack top as integer.

10: trunc

The real at stack top is truncated to its integer equivalent, and that replaces the stack top as integer.

11: sin

The sine of the real at stack top is replaces the stack top as real.

12: cos

The cosine of the real at stack top is replaces the stack top as real.

13: exp

The exponential of the real at stack top is replaces the stack top as real.

14: ln

The natural logarithm of the real at stack top is replaces the stack top as real.

15: sqrt

The square root of the real at stack top is replaces the stack top as real.

16: arctan

The arctangent of the real at stack top is replaces the stack top as real.

17: eof

The eof status of the "input" file replaces the stack top as boolean.

18: eoln

The eoln status of the "input" file replaces the stack top as boolean.

9 Offset

x: Unused

y: Offset

The value of y is added to the integer address at stack top, and that replaces the stack top as integer.

10 Jump

x: Unused

y: Address

The address y replaces the current execution address.

11 Conditional jump

x: Unused

y: Address

If the boolean at stack top is false, then the address y replaces the current execution address. The top of stack is removed.

12 Switch

x: Unused

y: Address of case table

Select case entry and execute. Expects a selector value on the stack. The instruction contains the address of a case table, which consists of a series of “13” (non-executed) instructions that contain pairs of value/address, each occupying a “13” entry. The selector value is removed from the stack, then matched to each value entry in the table in turn, until a match is found, or a non-“13” entry encountered. If a match is found, the table address of the matching value is executed. If no match is found, an exeception is taken by setting ps to caschk.

13 Switch table data (not executed)

x: Unused

y: Case value or address

14 Forlup

x: Unused

y: Address of end of for loop

Start of for loop instruction for “to” loops. The end expression is at stack top, the start expression under that, and the address of the index variable under that. If the start expression is less than or equal to the end expression, then the index is assigned the start expression and execution continues. If not, the three parameters are purged from the stack and execution skips to the end of the for loop.

15 For2up

x: Unused

y: Address of start of for loop

End of for loop instruction for “to” loops. The end expression is at stack top, the start expression under that, and the address of the index variable under that. The index is incremented, and if that is less than or equal to the end expression, the index updated and execution continues at the top of the for loop, otherwise the stack is purged and execution continues (out of the loop).

16 For1down

x: Unused

y: Address of end of for loop

Start of for loop instruction for “downto” loops. The end expression is at stack top, the start expression under that, and the address of the index variable under that. If the start expression is greater than or equal to the end expression, then the index is assigned the start expression and execution continues. If not, the three parameters are purged from the stack and execution skips to the end of the for loop.

17 For2down

x: Unused

y: Address of start of for loop

End of for loop instruction for “downto” loops. The end expression is at stack top, the start expression under that, and the address of the index variable under that. The index is decremented, and if that is greater than or equal to the end expression, the index updated and execution continues at the top of the for loop, otherwise the stack is purged and execution continues (out of the loop).

18 Mark stack

x: Unused

y: Symbol table address for procedure/function

Creates a stack mark. Using the identifier table address, the current stack top and the extent of the locals allocation for the target block are added to check for overflow, and an exception results if so (ps = stkchk). Otherwise the mark is created:

Stack top: 4: Symbol table index

3: Dynamic link

2: Static link

1: Return address

0: Return value

19 Call

x: Unused

y: Offset to mark

Call a procedure. The mark created for the call created by a mark stack instruction is found using the offset. The display level is incremented and a new static link is established. The current pc is saved in the mark return address. The static link is updated from the current display level. The dynamic link is updated from the current block mark. The new block mark is set. The new stack top is set. The new pc is set from the symbol table for the called procedure/function.

20 Index

x: Unused

y: Array table entry

Array index. Expects an array index integer on stack, with the base of the array below that. The index is validated against the range of indexes in the array table entry, then the index is adjusted for low bound and added to the array base. The index is removed, and the new address replaces the array base.

This version of indexing does not scale for element size and is used for arrays of simple scalars.

21 Index scaled

x: Unused

y: Array table entry

Array index. Expects an array index integer on stack, with the base of the array below that. The index is validated against the range of indexes in the array table entry, then the index is adjusted for low bound, scaled to the array element size, and added to the array base. The index is removed, and the new address replaces the array base.

This version of indexing does scale for element size and is used for arrays of structured types.

22 Load block

x: Unused

y: Copy count

Copies a number of stack entries from the address indicated by the integer at stack top to a new stack section added at the top after the address is removed. The number of stack locations to copy is indicated in y.

This instruction is used to create value parameter copies of arrays and records.

23 Copy block

x: Unused

y: Copy count

Copies a number of stack entries from the address indicated by the integer at stack top to the address indicated by the integer second on stack. The number of stack locations is indicated in y. Both addresses are removed from the stack.

This instruction is used to assign array variables.

24 Literal

x: Unused

y: Constant

The constant y is placed as a new stack top integer. A stack overflow check is performed.

25 Load real

x: Unused

y: Address of real constant

The real constant in the real constant holding array as indexed by y is placed as a new stack top real. A stack overflow check is performed.

26 Float

x: Unused

y: Offset

The integer at the stack location offset by y is converted from integer to real.

27 Read

x: Unused

y: Type selector

If the input file is not at eof, an integer, real or character is read from the input and placed at the stack location indirectly located at the address found from the integer at stack top. The stack top is removed.

The type of read is encoded in y:

y = 1: Integer

y = 2: Real

y = 4: Character

28 Write string

x: Unused

y: Address of string in string table

Writes a string from the string table. The y address indicates where the

characters of the string begin in the string table. The field for the string

is at stack top. The stack top is removed.

29 Write with default field

x: Unused

y: Type select

Writes an integer, real, boolean or character value from the stack top to the output. The value is written using the default field for that type. The value is removed from the stack.

The type of write is encoded in y:

y = 1: Integer

y = 2: Real

y = 3: boolean

y = 4: Character

30 Write with field

x: Unused

y: Unused

Writes an integer, real, boolean or character value from the second on stack

to the output. The value is written using the field from the top of stack

integer. Both values are removed from the stack.

The type of write is encoded in y:

y = 1: Integer

y = 2: Real

y = 3: boolean

y = 4: Character

31 Halt

x: Unused

y: Unused

Halt program. Causes the program to halt normally, without an error.

32 Exit procedure

x: Unused

y: Unused

Terminates the currently running procedure. The locals and block information

for the current procedure are removed from the stack. The execution address

is restored from the return address in the block mark. Finally, the current

block is reset to the value before the procedure was activated.

33 Exit function

x: Unused

y: Unused

Terminates the currently running function. The locals and block information

for the current procedure are removed from the stack, but the return value for the function is left on the top of the stack. The execution address is restored from the return address in the block mark. Finally, the current block is reset to the value before the procedure was activated.

34 Fetch indirect

x: Unused

y: Unused

The stack location indexed by the integer at stack top replaces the stack top.

35 Not

x: Unused

y: Unused

Finds the boolean 'not' of the boolean variable at stack top, resulting in a

boolean at stack top.

36 Negate

x: Unused

y: Type selector

Negates the integer or real on top of the stack.

The type of negate is encoded in y:

y = 1: Integer

y = 2: Real

37 Write real with field and fraction

x: Unused

y: Unused

The real at third on stack is written in the "fixed point" notation. The field is the second on stack, and the fraction is at stack top. All three parameters are removed from the stack.

38 Store to stack

x: Unused

y: Unused

Stores the top of the stack to the stack location indicated by the second

integer on the stack. Both the top and second on stack are removed.

39 Real equal

x: Unused

y: Unused

Finds the real on top of stack equal to the real second on stack. Places

the result in the second stack boolean and removes the top.

40 Real unequal

x: Unused

y: Unused

Finds the real on top of stack unequal to the real second on stack. Places

the result in the second stack boolean and removes the top.

41 Real less than

x: Unused

y: Unused

Finds the real second on stack less than the real on top of stack. Places

the result in the second stack boolean and removes the top.

42 Real less than or equal

x: Unused

y: Unused

Finds the real second on stack less than or equal to the real on top of stack. Places the result in the second stack boolean and removes the top.

43 Real greater than

x: Unused

y: Unused

Finds the real second on stack greater than the real on top of stack. Places

the result in the second stack boolean and removes the top.

44 Real greater or equal

x: Unused

y: Unused

Finds the real second on stack greater than or equal to the real on top of

stack. Places the result in the second stack boolean and removes the top.

45 Integer equal

x: Unused

y: Unused

Finds the integer on top of stack unequal to the integer second on stack. Places the result in the second stack boolean and removes the top.

46 Integer not equal

x: Unused

y: Unused

Finds the integer on top of stack unequal to the integer second on stack. Places the result in the second stack boolean and removes the top.

47 Integer less than

x: Unused

y: Unused

Finds the integer second on stack less than the integer on top of stack. Places the result in the second stack boolean and removes the top.

48 Integer less than or equal

x: Unused

y: Unused

Finds the integer second on stack less than or equal to the integer on top of stack. Places the result in the second stack boolean and removes the top.

49 Integer greater than

x: Unused

y: Unused

Finds the integer second on stack greater than the integer on top of stack.

Places the result in the second stack boolean and removes the top.

50 Integer greater or equal

x: Unused

y: Unused

Finds the integer second on stack greater than or equal to the integer on top of stack. Places the result in the second stack boolean and removes the top.

51 Boolean 'or'

x: Unused

y: Unused

Finds the 'or' of the boolean on top of stack and the boolean second on stack. Places the result in the second stack boolean and removes the top.

52 Integer add

x: Unused

y: Unused

Adds the top of stack integer to the second on stack integer, and places the

integer result in the second on stack and removes the top.

53 Integer subtract

x: Unused

y: Unused

Subtracts the top of stack integer from the second on stack integer, and places the integer result in the second on stack and removes the top.

54 Real add

x: Unused

y: Unused

Adds the top of stack real to the second on stack real, and places the

integer result in the second on stack and removes the top.

55 Real subtract

x: Unused

y: Unused

Subtracts the top of stack real from the second on stack real, and places

the real result in the second on stack and removes the top.

56 Boolean 'and'

x: Unused

y: Unused

Finds the 'and' of the boolean on top of stack and the boolean second on stack. Places the result in the second stack boolean and removes the top.

57 Integer multiply

x: Unused

y: Unused

Multiplies the top of stack integer by the second on stack integer, and places the integer result in the second on stack and removes the top.

58 Integer divide

x: Unused

y: Unused

Divides the second on stack integer by the top of stack integer, and places

the integer result in the second on stack and removes the top.

59 Integer modulo

x: Unused

y: Unused

Finds the modulo of second on stack integer by the top of stack integer, and places the integer result in the second on stack and removes the top.

60 Real multiply

x: Unused

y: Unused

Multiplies the top of stack real by the second on stack real, and places

the real result in the second on stack and removes the top.

61 Real divide

x: Unused

y: Unused

Divides the second on stack real by the top of stack real, and places

the integer result in the second on stack and removes the top.

62 Read end of line

x: Unused

y: Unused

Reads the end of line from the input file. If eof is encountered on input, it is an error.

63 Write end of line

x: Unused

y: Unused

Writes an end of line to the output file. If the total number of output lines is larger than the line check limit, an error results. Pascal-S uses a limit to prevent programs that loop forever writing output.

# Testing Pascal-S

## Running tests

### testprog

The main test script for testing is:

testprog <Pascal source file>

testprog.bat <Pascal source file>

testprog is a “one stop” test resource for most programs. It expects the following files to exist under the given primary filename:

program.pas The Pascal source file.

program.inp The input file for the running program.

program.cmp The reference file for the expected output.

testprog compiles and runs the target program, and checks its output against the reference file. Several files are produced during the process:

Program.lst Contains the output from the program when run. This is all output to the standard “ output” file.

Program.dif This is the output of the diff command between program.lst and program.cmp. It should be empy if the program produced the output expected.

Not all of the files for testprog need to have contents. For example, a program that does not do input does not need to have a program.inp file. An example of this would be the “hello” program. However, testprog expects all of the files to exist.

To determine if the test ran correctly, the output of first the program.err file should be checked for zero errors, then the resulting program.dif file is checked for zero length. All of these results are announced during the test:

C:\projects\PASCAL\pascals>testprog sample\_programs\roman

Compile and run sample\_programs\roman

For a program that has a compile error.

C:\projects\PASCAL\pascals>testprog sample\_programs\roman

Compile and run sample\_programs\roman

03/18/2012 10:44 AM 76 roman.dif

For a program that does not match it’s expected output.

Regression testing

The regression test is performed as:

C:\projects\PASCAL\pascals>regress

Compile and run sample\_programs\hello

02/24/2018 11:36 PM 0 hello.dif

Compile and run sample\_programs\roman

02/24/2018 11:36 PM 0 roman.dif

Compile and run sample\_programs\prime

02/24/2018 11:36 PM 0 prime.dif

The regression test simply runs all of the sample programs and checks output.

# Licensing

Pascal-S is derived from the original sources of the Pascal-S compiler from ETH Zurich, as created by Niklaus Wirth and his students. It was and is public domain, as acknowledged by Professor Wirth, and I add my modifications to it to the public domain as well.

Public domain is a widely misunderstood concept. There is no "license" possible nor needed for public domain works. There are no restrictions on it's use, nor do it's authors have any rights to it. It can be used for any purpose, public or private, and distributed or modified for any use whatever, paid or not.

The following are typical answers to questions about public domain works in general, and this work in specific.

Q. The Berne convention states that copyright in europe, where Pascal-S originated, is automatic. Doesn't that make Pascal-S a copyrighted work?

A. The laws in all copyright countries dictate what must be done to qualify as a copyrighted work. Since there is no specific legal agreement concerning public domain work, public domain is shaped by what constitutes enforceable copyright. The most common features of public domain are, but not limited to:

1. The author has stated the work is public domain.
2. The work has been distributed freely and with knowledge of the author(s).

In the case of Pascal-S, both are true.

Q. Dosen't public domain mean that I may no longer be able to gain access to the source?

A. If every copy of the work were to be erased or burned, but that is virtually impossible. Nobody can order you to release your copy since, by definition, there are no "rights" to a public domain work.

Q. Can't someone just copyright or patent the work later?

A. Showing that a work is in the public domain is part of denying copyright or patent to a work. By definition, a legitimate public domain work cannot later be copyrighted or patented.

Q. Can't someone improve the work, then gain rights to that derived work and thus restrict it's use?

A. Anyone can improve a public domain work, but they only have rights to their improvements, not to the original work. If their improvements are trivial, then it would be trivial for others to add that functionality. If it is not trivial, then you might want to pay for it.