①

COMPUTERWARE™
Microcomputer Sales and Software

COLOR PASCAL — 32K

Box 668 • 4403 Manchester Ave. • Suite 102 • Encinitas, California 92024
Phone: (714) 436-3512

COMPUTERWARE COLOR PASCAL

TABLE OF CONTENTS

INTRODUCTION:

Computerware™ has implemented Dynasoft's PASCAL on the Color Computer so that many of our customers will have the opportunity to learn about and program in what has become one of the most popular new languages available. Previously, one had to invest substantially more money into a computer system to gain access to the PASCAL language. Although Dynasoft's PASCAL is not an 'extended' version of the language, the user will find that virtually any task can be accomplished using the commands available plus external calls to your own routines.

There are two manuals that come with the Color PASCAL: The Dynasoft PASCAL Users Manual and Computerware™'s Color PASCAL Manual (this one). Neither one make the pretense of teaching the user the PASCAL language as much as covering the syntax of PASCAL and how to use it on the Color Computer. We urge you to invest in one of the many PASCAL books available from either us or another source to get a beginning level understanding of the ASCAL language.

LICENSE:

Computerware™ has licensed PASCAL from Dynasoft for distribution on the Color Computer. Color PASCAL, in all machine readable formats, and the written documentation accompanying them are copyrighted. The purchase of CSS Color PASCAL conveys to the purchaser a license to use the Color PASCAL for his/her own use, and not for sale or free distribution to others. No other license, expressed or implied is granted.

WARRANTEE INFORMATION:

The license to use CSS Color PASCAL is sold AS IS without warrantee. This warrantee is in lieu of all other warrantees xpressed or implied. Computerware™ does not warrant the uitability of the Color PASCAL for any particular user application and will not be responsible for damages incidental or consequential to its use in a user system.

REGISTRATION:

Computerware™ wants you, the user, to be satisfied with our products. To help achieve this goal, we ask that you fill out the enclosed ORIGINAL registration form. If a problem is found in the software, we can then communicate with you concerning corrections and enhancements. If you find a problem - please document it - send to Computerware™ the documentation on the software in question, and we will make every attempt to resolve the problem/question.

Color Pascal Implementation Details

Color Pascal 1.2 as supplied assumes 32K of RAM starting at $0000. BASIC subroutines are used to interface with the terminal and the line printer. The cassette interface, for program Load and Save, is implemented within the interpreter to format ASCII files compatible with the Color Editor character file structure.

There are several locations within the interpreter that may be patched to change the system configuation. The word (2 bytes) at $0B61 contain the size of your system's RAM area in bytes (RAM is assumed to start at $0000). The tape as supplied sets this word to $7800 (30720) corresponding to a 32K system. It should be set to a different value if (a) your memory size is not 32K, or (b) you want to reserve some memory for programs or data outside the Pascal area.

The memory required by Pascal is organized as follows:

0020-0037	P-machine pseudo registers
0600-06EF	Device drivers and device control blocks
06F0-06FF	File control blocks
0700-0B4F	P-code interpreter
0B50-0CBF	SUPERVISOR initialization
0CC0-0D7F	SUPERVISOR
0D80-109F	EDITOR
10A0-26FF	PASCAL Compiler
2700-77FF	Workspace, Pascal stack, and heap.

The Workspace is used to hold source text and compiled P-Code programs, and its length is variable. The Pascal stack starts at the end of the workspace, and extends towards the end of RAM area allocated to the Pascal system. The SUPERVISOR and EDITOR use only the highest page for their data stack, but the COMPILER and user programs are allocated a stack beginning immediately after the end of the current workspace. The heap starts at the highest RAM address and grows downward toward the stack.

It should be noted that neither the EDITOR nor the COMPILER can process a workspace larger than 32768 bytes. In practice, this means that the "high RAM" pointer at $0B61 cannot be set higher than $7FFF, since the EDITOR assumes that all of the space between $2700 and one page below "high RAM" is available for the workspace.

A second location is the word at $0D78. This normally contains $A017, which is the main re-entry point into BASIC. The SUPERVISOR uses this address to jump back to BASIC when the QUIT command is executed.

The words at $0711 and $0B54 both contain $7800, which is used to initialize the system stack pointer every time the interpreter is re-entered. About 40 bytes are required for the system stack.

To patch the Pascal system, the following procedures may be followed. Load the Pascal System from BASIC using the CLOADM command. Immediately exit the Pascal System by entering 'Q' to exit the SUPERVISOR and return to BASIC. Enter any desired Patches. Mount an output tape in the tape recorder and advance the tape past the leader with the RECORD buttons engaged. Using the CSAVEM command, save your new copy from $0600 to $26FF, with a transfer address of $0700. To test your new system, load it using the CLOADM command and test it with some known programs.

Pascal's I/O Structure

The Input-Output structure is table oriented, so only the tables should have to be altered to accomodate different I/O configurations. The I/O provided by the standard system is designed to interface the system to BASIC by establishing appropriate device indicators for BASIC's device handler routines. The system uses two kinds of tables:

1) The File Control Block (FCB)

The standard procedure SETP does not actually select I/O directly, but instead selects one of 3 possible "file control blocks" located in RAM from $06F0 to $06FF. Each FCB is organized as follows:

FIELD	OFFSET	
CURBUF	0	Character buffer for this file
EOLBUF	1	EOLN flag for this file. ("1" if last input was a carriage return)
DCBPTR	2,3	Pointer to the DCB for this file

Each FCB occupies 4 bytes, and the 3 FCB's correspond to "device" 0 through 2. Only three FCB's are actually set up by the standard tape. The first, located at $06F0, corresponds to Device 0, and points to a DCB known as "TERM8DCB" at $0600. The second, located at $06F4, corresponds to Device 1, assumes to be the main control device. This FCB points to a DCB known as "TERMDCB" at $060A. The third, located at $06F8, corresponds to Device 2, assumes to be the printer control device. Example: to direct your output to the printer when you are compiling a program type '2C' instead of just 'C'.

2) The Device Control Block (DCB)

A DCB is a table of parameters located anywhere in memory which contains, among other things, the Device Flag and the address of a device driver routine. It is organized as follows:

FIELD	OFFSET	
DCBLNK	0,1	Forward link in DCB chain (not used)
DCBDID	2,3	ASCII code for device ID (2 characters)
DCBDVR	4,5	Device driver address
DCBIOA	6,7	Device physical address/flag
DCBERR	8	Error status (not used)
DCBECH	9	Echo flag (Terminal driver echos if not zero)

The standard tape contains three DCB's. The first, known as "TERM8DCB" is located at $0600. It is a dummy device control block that currently reads or writes binary (8 bit) data bytes. It's device flag is set to 0 and the driver address points to CRTDR at $061E. This DCB may be modified by the user to point to a user defined I/O handler by modifing the device address and device flag as desired. The second DCB, known as "TERMDCB" is at $060A, and is intended for the control terminal (Device 1). It's device flag is set to 0 and the driver address points to CRT7 at $0646. This device handler accesses the terminal I/O routines and strips the parity bit on input. The third DCB, known as "PRNTDCB" is at $0614, and is intended for accessing the RS-232 device which would normally be connected to a printer. It corresponds to "Device 2", and its device flag is set to -2. The device driver can only output data since the device handler within Basic for RS232 transfer is an output only driver. Care must be utilized in the use of this device to insure a printer is properly connected or the system will "hang up" and can only be recovered by the RESET button.

If the user wishes to perform I/O from a machine language subroutine, the standard calling sequence is as follows:

a) load the A register with a data byte (if this is an output or control operation)
b) load the B register with the I/O function code (see below)
c) load the X register with the DCB address
d) JSR to REQIO ($0B23)

Subroutine REQIO saves B,X and Y, loads the device physical address on the X register, and calls the device driver. Thus the driver itself is called with data in A, the I/O function code in B, and the device physical address in X.

The I/O functions codes are:

01: read a byte or character into the A register
02: write a byte or character from the A register
04: return status for the device in the A register
08: perform control function to device.

Standard DCB's

```
                        * 8 bit driver (Pascal device 0)
0600:   0000            TERM8DCB     FDB 0          CHAIN POINTER
0602:   5450                         FCC 'TP'
0604:   061E                         FDB CRTDR      DRIVER ADDRESS
0606:   0000                         FDB 0          FILE TYPE
0608:   00                           FCB 0          ERROR STATUS
0609:   01                           FCB 1          ECHO CONTROL
                        *
                        * Control Terminal DCB (Pascal device 1)
060A:   0000            TERMDCB      FDB 0          CHAIN POINTER
060C:   544D                         FCC 'TM'
060E:   0646                         FDB CRT7       DEVICE ADDRESS
0610:   0000                         FDB 0          DEVICE FLAG
0612:   00                           FCB 0          ERROR STATUS
0613:   01                           FCB 1          ECHO CONTROL
                        *
                        * Printer Control DCB (Pascal device 2)
0614:   0000            PRNTDCB      FDB 0          CHAIN POINTER
0616:   5052                         FCC 'PR'
0618:   0646                         FDB CRTDR      DRIVER ADDRESS
061A:   FFFE                         FDB -2         FILE TYPE
061C:   00                           FCB 0          ERROR STATUS
061D:   00                           FCB 0          ECHO FLAG (NO ECHO)
```

Machine Language Subroutine Linkage

Pascal programs link to EXTERN (machine language) subroutines with a JSR instruction in the P-Code interpreter. Subroutines which are not passed arguments should present no difficulties, so long as the system stack is left as it was found, and they return to the calling program with an RTS instruction. The subroutines themselves must be located outside the Pascal area. They can be placed in high memory space - memory can be set aside above the Pascal stack by setting the end-of-memory pointer below the actual end of your RAM area.

All parameters are passed 'by value' in this version of Pascal (i.e., actual argument values are passed, not their address), and all parametes are passed as 16 bit (2 byte) quantities. Arguments are pushed onto the Pascal stack in the order they are encountered in the procedure call or function reference in the calling program. A user-written subroutine MUST pull ALL ARGUMENTS from the stack before returning to the calling program, otherwise the result will be unpredictable. Arguments can be pulled from the Pascal stack with the instruction

```
        LDD ,--U
```

which pulls one argument and places it in the D register.

PROCEDURE subroutines should return with an RTS instruction. FUNCTION subroutines, however, should exit by loading the function result into D and executing the instuction

```
        STD -2,U
```

prior to the RTS. This will leave the function result on the top of the Pascal stack where it will be correctly picked up by the calling program.

Aside from the operation noted above for pulling arguments, register U should be left undisturbed by any machine language subroutine. The U register is used as the Pascal stack pointer. Note that the PULU instruction CANNOT be used to pull Pascal arguments because the stack grows UPWARD rather than downward in memory. Register Y should also be left undisturbed (or restored after use) because it is used by the interpreter as the Pseudo program counter.

APPENDIX B - SAMPLE PROGRAMS

Supervisor LOAD and SAVE procedures.

Included within the Supervisor is linkage to machine-code routines that allow for the loading and saving of data within the Pascal workspace. These routines allow the user to load the workspace with either source or pseudo object code, and the saving of the workspace contents on cassette files. The file format is compatible with the Editor/Assembler and Basic ASCII file structures.

SAVE command:

a) Install blank tape into cassette recorder.
b) Engage the RECORD buttons and position the tape beyond the tape leader.
c) While in the Supervisor (i.e., the READY prompt is displayed) enter 'S' for SAVE. The Save file routine will respond with an '=' and allow the entry of a file name of up to 8 characters. On entry of the 8th character or RETURN, the save routine will enable the cassette motor, write the Header block containing the specified file name, and write data block of 256 bytes until all specified data is transferred to tape. All data contained in the workspace is outputted.
d) On completion of the Save operation, the motor for the cassette is turned off and control is returned to the Supervisor.

LOAD Command

a) Install the tape to be loaded into the cassette recorder.
b) Rewind the cassette tape to the beginning of tape. Engage the READ buttons.
c) While in the Supervisor enter the 'L' for LOAD. The load file routine will respond with '=' and allow entry of a file name of up to 8 characters. On entry of the 8th character or RETURN, the LOAD routine enables the cassette motor and starts the search for the specified file name. If a 'null' name is entered, the first file found on the tape will be loaded. On completion of the load, control is returned to the Supervisor.
d) Errors that can occur during the load are:

   Error 2 - Memory full. File size exceeds the memory available.
   Error 4 - File not found.
   Error 9 - Checksum error on load.

APPENDIX A. ERROR CODES

Modify Appendix A subparagraph A.2 to include the following:

02: stack overflow (not enough memory to run or load program)
04: file not found on load

```
(* SUPERVISOR FOR 6809 PASCAL - APRIL 9,1981  *)
PROGRAM SUPERVISOR;

CONST COMP=53248; EDIT=3456; TERM=1;
      MON=49152; CSAVE=3056; CLOAD=3059;
TYPE  PREG=ARRAY[0..4] OF INTEGER;
VAR   CH: CHAR;
      N,N1: INTEGER;
      SCOM:^PREG;
      SOT,EOT:^CHAR;

PROCEDURE SAVE; EXTERN(CSAVE);
PROCEDURE LOAD; EXTERN(CLOAD);
PROCEDURE QUIT; EXTERN(MON);

BEGIN
  N:=TERM; SCOM:=SYSCOM;
  SOT:=SCOM^[0]; EOT:=SCOM^[1];
  WRITELN('READY'); READ(CH);
  IF (CH>='0') AND (CH<='2') THEN
    BEGIN N:=CH-'0'; READ(CH) END;
  CASE CH OF
  'C': BEGIN SETP(N); LINK(COMP,EOT) END;
  'E': BEGIN SETP(N); LINK(EDIT,0) END;
  'G': BEGIN READ(N);
       IF N<>0 THEN
         BEGIN ~~READ(N1); LINK(N,N1) END~~
         ELSE LINK(SOT,EOT)
       END;
  'M': BEGIN READ(N);
         IF N<>0 THEN MOVL(SOT,EOT,N)
       END;
  'S': SAVE;
  'L': LOAD;
  'Q': QUIT
  END (* CASE *)
END.
```

```
PROGRAM SOUND;
VAR PV: CHAR;
    I: INTEGER;
    PITCH: ^CHAR;
    DURATION: ^INTEGER;

PROCEDURE SOUND; EXTERN(43350); (* A956 SOUND *)

BEGIN
 PITCH:=140;
 DURATION:=141;
 PV:=1;
 FOR I:=1 TO 255 DO
 BEGIN
   PITCH^:=PV;
   DURATION^:=5;
   WRITELN(PITCH^,DURATION^);
   SOUND;
   PV:=PV+1
 END
END.
```

POWER .PAS   PAGE

```
(* THIS PROGRAM REQUIRES STACK RELOCATION TO EXECUTE *)
(* TO RUN:- ENTER G 4256,10000 *)

PROGRAM POWER;

CONST N=10;
TYPE DIGIT=0..9;

VAR CH:ARRAY['0'..'9'] OF CHAR;
    I,K,R:INTEGER;
    D:ARRAY[1..N] OF DIGIT;

BEGIN
  FOR K:=0 TO 9 DO CH[K]:='0'+K;

  FOR K:=1 TO N DO
  BEGIN
    WRITE('.');
    R:=0;
    FOR I:=1 TO K-1 DO
    BEGIN
      R:=10*R+D[I];
      D[I]:=R DIV 2;
      R:=R-2*D[I];
      WRITE(CH[D[I]])
    END;
    D[K]:=5;
    WRITELN('5')
  END

END.
```

```
PROGRAM PRIMES;

CONST N=1229; N1=35; (* N1 IS SQRT OF N *)

TYPE BOOLEAN=(FALSE,TRUE);

VAR  I,K,X,INC,LIM,SQUARE,L: INTEGER;
     PRIM: BOOLEAN;
     P,V: ARRAY[1..N] OF INTEGER;

BEGIN
  WRITE(2:6,3:6);L:=2;
  X:=1; INC:=4; LIM:=1; SQUARE:=9;
  FOR I:=3 TO N DO
  BEGIN  (* FIND NEXT PRIME *)
    REPEAT X:=X+INC; INC:=6-INC;
      IF SQUARE <= X THEN
        BEGIN LIM:=LIM+1;
          V[LIM]:=SQUARE; SQUARE:=(P[LIM+1])*(P[LIM+1])
        END;
      K:=2; PRIM:=TRUE;
      WHILE PRIM AND (K<LIM) DO
      BEGIN K:=K+1;
        IF V[K] < X THEN V[K]:=V[K]+2*P[K];
        PRIM:=X <> V[K]
      END
    UNTIL PRIM;
    IF I <= N1 THEN P[I]:=X;
    WRITE(X:6); L:=L+1;
    IF L=10 THEN
      BEGIN WRITELN; L:=0
      END
    END;
    WRITELN;
END.
```

```
PROGRAM DIVISORS;
  VAR  NUMBER,DIVISOR : INTEGER;
       CH : CHAR;
BEGIN
  FOR NUMBER := 100 DOWNTO 0 DO
  BEGIN
    IF NUMBER > 0 THEN
    BEGIN
      WRITELN('THE DIVISORS OF',NUMBER,' ARE:');
      FOR DIVISOR := 2 TO NUMBER DO
        IF NUMBER MOD DIVISOR = 0 THEN
        WRITELN(DIVISOR)
    END;
    READ(CH)
  END
END.
```