A DL-Pascal 1.2 and HD61700 Series
Fascicle I – ζ(3) Implementation in HD61700 using Word Arithmetic

Table of Contents

Introduction 1
Chapter 0: Reducing ζ(3) to Word Arithmetic 2
Chapter 1: The Simple Case of Addition 3
Chapter 2: Calling Bank 0 4
Chapter 3: Testing Interchangeable Implementations 5
Chapter 4: Almost HD61700 7
Chapter 5: Next Steps 9
Chapter 6: Why DL-Pascal 1.2? 11
Appendix A 12
Acknowledgments 20

Introduction

This is the first in a series of reports exploring direct programming of the HD61700 processor in the Casio PB-2000C. PB-2000C at The Computer Museum in Munich.

The work grew out of an earlier exercise: implementing ζ(3), Apéry's constant, in the programming languages available for this remarkable device. With those implementations completed—with the exception of BASIC—the natural next step was to take the calculation one level lower and explore the HD61700 itself.

DL-Pascal 1.2 provides an ideal bridge. Its INLINE statement allows HD61700 machine code to be inserted directly into Pascal routines, leaving Pascal to provide the surrounding program structure and experimental environment.

Our goals here are deliberately modest. We implement the four elementary operations on the Word data type—addition, subtraction, multiplication, and division—in HD61700 machine code, and use them to evaluate the first three terms of ζ(3), using fixed-point arithmetic scaled by 10000. DL-Pascal function types are also used to make the native Pascal and HD61700 implementations interchangeable, allowing both to be exercised by the same test and calculation code.

This work was developed jointly by Alex Barragan and ChatGPT, through a cycle of ideas, implementation, execution, observation, and revision. The collaboration proved to be a practical example of something increasingly important: human knowledge and experimentation combined with AI-assisted analysis can produce results that neither participant would readily achieve alone[1].

The destination is a complete HD61700 implementation of ζ(3). This first report establishes the machinery needed to get there.

Piotr's translation of the HD61700 manual is an invaluable reference to train humans and AIs alike.

---

1 The human co-author originally proposed the word "synergy." The AI co-author declined. "I'm afraid I can't do that, Alex."

Chapter 0: Reducing ζ(3) to Word Arithmetic

We begin with the rapidly convergent series

$$\zeta(3) = \sum_{k=0}^{\infty} C_k(205k^2 + 250k + 77),$$

where

$$C_0 = \frac{1}{64}, \qquad C_k = -C_{k-1}\frac{k^5}{32(2k+1)^5}.$$

For this first experiment we use only the first three terms:

$$\zeta(3) \approx \frac{77}{64} - \frac{532}{497664} + \frac{1397}{1555200000}.$$

To keep the calculation within the 16-bit Word type, the result is represented as a fixed-point integer scaled by 10000. The arithmetic is rearranged so that no required intermediate value exceeds the range usable by our Word primitives.

The three terms are evaluated as:

t0 = 77*156 + 77/4 = 12031

t1 = (((133*25)/24)*25)/324 = 10

t2 = (1397/240)/648 = 0

Giving 10000ζ(3) ≈ t0−t1+t2=12021.

The loss of the fractional parts is intentional. Integer division is used throughout, and once a term has been reduced to zero there is no reason to continue its remaining divisions. At this stage the objective is not numerical precision, but to obtain a small, reproducible calculation with which to exercise the HD61700 arithmetic primitives.

Chapter 1: The Simple Case of Addition

DL-Pascal's INLINE statement allows HD61700 machine code to be placed directly inside a Pascal routine. Our simplest example is addition:

```
FUNCTION HAddW(a,b : Word) : Word;
BEGIN
  INLINE ($42,$04,$00,          { LD   $4,0                    }
          $A8,$68,$04,          { LDW  $8,(IX+$4) ; a          }
          $42,$04,$02,          { LD   $4,2                    }
          $A8,$66,$04,          { LDW  $6,(IX+$4) ; b          }
          $88,$68,$06,          { ADW  $8,$6      ; a+b        }
          $42,$04,$04,          { LD   $4,4                    }
          $A0,$E8,$04);         { STW  $8,(IX-$4) ; result     }
END;
```

The interesting part is not the addition itself, but how the HD61700 code accesses the Pascal parameters and returns its result.

For a global function with the prototype FUNCTION(a,b : Word) : Word, DL-Pascal places the first Word parameter at IX+0, the second at IX+2, and reserves the function result at IX-4. These offsets therefore apply specifically to the (Word,Word):Word interface used by our arithmetic primitives. The IX register provides the connection between the Pascal calling convention and our machine code.

Register $4 is used here to hold the required offsets explicitly. The two parameters are loaded into HD61700 register pairs, ADW performs the 16-bit addition, and STW places the result where DL-Pascal expects the function value.

To the Pascal program, HAddW therefore behaves exactly like an ordinary function:

```
x := HAddW(a,b);
```

Inside that function, however, the arithmetic is performed entirely by the HD61700.

This simple mechanism—Pascal interface, IX parameter binding, HD61700 implementation—provides the basis for the remaining arithmetic primitives.

The workflow used throughout this report is deliberately simple. First, the HD61700 routine is written as a conventional assembly source file and assembled using HD61.exe. The assembler's /S option can directly produce a Pascal .PAS file containing the corresponding INLINE statement. For this report, however, we also inspect the generated .LST file and transfer the assembled bytes manually, adding the original assembly instructions as comments alongside them. This extra step is not required by the toolchain; it is done for pedagogical reasons, making the relationship between the HD61700 instructions, their machine-code representation, and the resulting Pascal INLINE statement immediately visible.

Chapter 2: Calling Bank 0

Addition and subtraction can be implemented directly with HD61700 instructions. Multiplication presents a more interesting problem. Rather than reproduce functionality already present in the PB-2000C ROM, we can use its 16-bit multiplication routine, KBM16, located in Bank 0 at &H0AD6.

KBM16 expects its operands in register pairs $5,$6 and $15,$16, and returns the product in $0,$1. The Pascal wrapper therefore only needs to load the two parameters, call the ROM routine, and store the returned value:

```
FUNCTION HMulW(a,b : Word) : Word;
BEGIN
 INLINE ($42,$04,$00,          { LD   $4,0                    }
 $A8,$65,$04,          { LDW  $5,(IX+$4) ; a          }
 $42,$04,$02,          { LD   $4,2                    }
 $A8,$6F,$04,          { LDW  $15,(IX+$4); b          }
 $FF,                  { TRP                          }
 $D6,$0A,              { address &H0AD6               }
 $42,$04,$04,          { LD   $4,4                    }
 $A0,$E0,$04);         { STW  $0,(IX-$4) ; result     }
END;
```

There is one complication: a normal CAL cannot be used directly from our Pascal code to reach a routine in Bank 0. The elegant solution came from Piotr Piątek: replace the CAL opcode with TRP. TRP itself is a one-byte instruction. The interpretation of any bytes following it is determined by the trap-handling routine elsewhere in the ROM. In this case, the following two bytes specify the address of the Bank 0 routine to be called.

The same technique is used for HDivW, which calls the ROM division routine at &H0AE8. One important detail emerged during testing: this ROM routine performs signed 16-bit division. Consequently, although DL-Pascal's Word type is unsigned, positive operands passed to HDivW must remain below 32768 if they are to retain their intended meaning.

Chapter 3: Testing Interchangeable Implementations

DL-Pascal allows functions themselves to be passed as parameters. This gives us a simple way to compare a native Pascal operation with its HD61700 equivalent using exactly the same test code.

The following reduced example demonstrates the method for addition:

```
PROGRAM Add_Test;
TYPE
  WordOp = FUNCTION(a,b : Word) : Word;

FUNCTION PAddW(a,b : Word) : Word;
BEGIN
  PAddW := a+b;
END;

FUNCTION HAddW(a,b : Word) : Word;
BEGIN
  INLINE ($42,$04,$00,          { LD   $4,0                    }
          $A8,$68,$04,          { LDW  $8,(IX+$4) ; a          }
          $42,$04,$02,          { LD   $4,2                    }
          $A8,$66,$04,          { LDW  $6,(IX+$4) ; b          }
          $88,$68,$06,          { ADW  $8,$6      ; a+b        }
          $42,$04,$04,          { LD   $4,4                    }
          $A0,$E8,$04);         { STW  $8,(IX-$4) ; result     }
END;

PROCEDURE Test(PascalOp,HD61700Op : WordOp);
VAR
  a,b,p,h : Word;
BEGIN
  a := 127;
  b := 37;

  p := PascalOp(a,b);
  h := HD61700Op(a,b);

  WriteLn('Pascal  = ',p);
  WriteLn('HD61700 = ',h);

  IF p=h THEN
    WriteLn('PASS')
  ELSE
    WriteLn('FAIL');
END;

BEGIN
  Test(PAddW,HAddW);
END.
```

Both implementations have the same (Word,Word):Word interface. Test therefore does not need to know whether an operation is written in Pascal or HD61700 machine code; it simply calls the functions it receives.

The complete test program in the Appendix extends this same idea to all four arithmetic primitives and a larger set of operand pairs. In our tests, all four implementations passed 100 comparisons:

```
ADD: 100 tests passed
SUB: 100 tests passed
MUL: 100 tests passed
DIV: 100 tests passed
```

This same mechanism will next allow the ζ(3) calculation to switch between Pascal and HD61700 arithmetic without changing the calculation itself.

Chapter 4: Almost HD61700

At this point we have two complete sets of arithmetic primitives:

Operation	Native DL-Pascal	HD61700 INLINE
Addition	PAddW	HAddW
Subtraction	PSubW	HSubW
Multiplication	PMulW	HMulW
Division	PDivW	HDivW

Only representative implementations have been discussed in the preceding chapters. The complete definitions of all eight functions are included in the program listing in Appendix A.

All eight share the same (Word,Word):Word interface. We can therefore write the ζ(3) calculation without specifying which implementation will perform the arithmetic.

```
FUNCTION ZetaStaged(AddOp,SubOp,MulOp,DivOp : WordOp) : Word;
VAR
 t0,t1,t2 : Word;
 x,y      : Word;
BEGIN
 x := MulOp(77,156);
 y := DivOp(77,4);
 t0 := AddOp(x,y);

 x := MulOp(133,25);
 x := DivOp(x,24);
 x := MulOp(x,25);
 t1 := DivOp(x,324);

 x := DivOp(1397,240);
 t2 := DivOp(x,648);

 x := SubOp(t0,t1);
 ZetaStaged := AddOp(x,t2);
END;
```

The arithmetic implementation is selected simply by the functions passed to ZetaStaged:

```
z := ZetaStaged(PAddW,PSubW,PMulW,PDivW);
WriteLn('Pascal  : ',z);

z := ZetaStaged(HAddW,HSubW,HMulW,HDivW);
WriteLn('HD61700 : ',z);
```

Both produce:

```
Pascal : 12021
HD61700 : 12021
```

In the second call, Pascal still provides the sequencing and temporary storage, but every addition, subtraction, multiplication, and division is performed by an HD61700 INLINE primitive.

Hence the title of this chapter: Almost HD61700.

Chapter 5: Next Steps

This first experiment deliberately stops short of a complete HD61700 implementation of ζ(3). Several natural steps follow:

- move the complete ζ(3) calculation into HD61700 machine code, leaving DL-Pascal only as the host;
- explore the PB-2000C's 8-byte Real arithmetic and construct equivalent HD61700 primitives;
- ultimately remove the Pascal host entirely and use the techniques described by Piotr here to calculate and display ζ(3).

The Word implementation developed here provides a small but useful foundation for those experiments.

The complete DL-Pascal 1.2 program can be found in Appendix A. It contains both implementations of the four arithmetic primitives and the test and ζ(3) routines described in the report.

Routine	Purpose
PAddW, PSubW, PMulW, PDivW	Native DL-Pascal Word arithmetic
HAddW, HSubW, HMulW, HDivW	HD61700 INLINE implementations
UnitTest	Compares native and HD61700 primitives
ZetaStaged	Three-term ζ(3) calculation using interchangeable primitives
ZetaNestedTerms	More compact, compound form of the same calculation

It produces the following output:

```
HD61700 Word arithmetic verification
ADD: 100 tests passed
SUB: 100 tests passed
MUL: 100 tests passed
DIV: 100 tests passed

Pascal arithmetic through WordOp
Zeta = t0 = 12031
t1 = 10
t2 = 0
12021

HD61700 arithmetic, staged through WordOp
Zeta = t0 = 12031
t1 = 10
t2 = 0
12021

HD61700 arithmetic, nested terms through WordOp
Zeta = t0 = 12031
t1 = 10
t2 = 0
12021

HD61700 arithmetic, direct nested terms
Zeta = t0 = 12031
t1 = 10
t2 = 0
12021
```

Chapter 6: Why DL-Pascal 1.2?

There is another reason for using DL-Pascal 1.2 in these experiments: it is an intriguing piece of software in its own right.

Only about 120 copies are known to have been released by its author, Hans Larsson. How many remain operable today is unknown. At some point DL-Pascal simply disappeared from the market, while other Pascal implementations of the period survived, circulated, and eventually found their way into software archives.

DL-Pascal followed a different path. Unlike Casio's CM-ROM languages, it is not widely available through the usual archives. Requests have been made over the years for its release, apparently without result. We will not speculate here about the reasons; the history is interesting enough without the gossip.

Perhaps more remarkable is that DL-Pascal appears never to have leaked into general circulation. This has occasionally encouraged the mythology of a small elite of ROM owners carefully guarding their treasures. Reality is probably much less dramatic.

On a French forum, the few known owners were once called “The Happy Few.” The phrase recalls Shakespeare's St. Crispin's Day speech on the eve of Agincourt: “*We few, we happy few, we band of brothers.*” Given the events of the following day, the French might have chosen another quotation.

For us, however, DL-Pascal is interesting for more than its rarity. It is an unusually capable development environment for the PB-2000C, and its INLINE facility provides the bridge between Pascal and the HD61700 that made the experiments in this report possible.

We hope that one day DL-Pascal 1.2 will become generally available, so that this remarkable piece of PB-2000C history can be studied, used, and preserved.

Appendix A

```
PROGRAM Zeta3_Fascicle1;
(*$S+,R+,T+,D+*)

CONST
  VectorSize = 10;

TYPE
  {
    A DL-Pascal function type.
    DL-Pascal requires the subroutine type to be declared first.
    The declared type can then be used for a procedure parameter.
  }
  WordOp = FUNCTION(a,b : Word) : Word;

VAR
  AV,BV    : ARRAY[1..VectorSize] OF Word;
  a,b      : Word;


{
  HD61700 Word arithmetic primitive: HAddW
  Pascal parameter layout:
    a       IX+0
    b       IX+2
    result  IX-4

  HD61700 source and assembled opcodes:
    42 04 00    LD   $4,0
    A8 68 04    LDW  $8,(IX+$4)      ; a = IX+0
    42 04 02    LD   $4,2
    A8 66 04    LDW  $6,(IX+$4)      ; b = IX+2
    88 68 06    ADW  $8,$6           ; a+b
    42 04 04    LD   $4,4
    A0 E8 04    STW  $8,(IX-$4)      ; result = IX-4
}
FUNCTION HAddW(a,b : Word) : Word;
BEGIN
  INLINE ($42,$04,$00,          { LD   $4,0                    }
          $A8,$68,$04,          { LDW  $8,(IX+$4) ; a          }
          $42,$04,$02,          { LD   $4,2                    }
          $A8,$66,$04,          { LDW  $6,(IX+$4) ; b          }
          $88,$68,$06,          { ADW  $8,$6      ; a+b        }
          $42,$04,$04,          { LD   $4,4                    }
          $A0,$E8,$04);         { STW  $8,(IX-$4) ; result     }
END;


{
  HD61700 Word arithmetic primitive: HSubW

  Pascal parameter layout:
    a       IX+0
    b       IX+2
    result  IX-4
```

```
  HD61700 source and assembled opcodes:
    42 04 00    LD   $4,0
    A8 68 04    LDW  $8,(IX+$4)      ; a = IX+0
    42 04 02    LD   $4,2
    A8 66 04    LDW  $6,(IX+$4)      ; b = IX+2
    89 68 06    SBW  $8,$6           ; a-b
    42 04 04    LD   $4,4
    A0 E8 04    STW  $8,(IX-$4)      ; result = IX-4
}
FUNCTION HSubW(a,b : Word) : Word;
BEGIN
  INLINE ($42,$04,$00,          { LD   $4,0                    }
          $A8,$68,$04,          { LDW  $8,(IX+$4) ; a          }
          $42,$04,$02,          { LD   $4,2                    }
          $A8,$66,$04,          { LDW  $6,(IX+$4) ; b          }
          $89,$68,$06,          { SBW  $8,$6      ; a-b        }
          $42,$04,$04,          { LD   $4,4                    }
          $A0,$E8,$04);         { STW  $8,(IX-$4) ; result     }
END;


{
  HD61700 Word arithmetic primitive: HMulW
  Pascal parameter layout:
    a       IX+0
    b       IX+2
    result  IX-4

  KBM16 register interface:
    $5,$6   = first operand
    $15,$16 = second operand
    $0,$1   = product on return

  KBM16 is located in bank 0 at &H0AD6. TRP is used instead
  of CAL to invoke the bank-0 routine from the Pascal program.

  HD61700 source and assembled opcodes:
    42 04 00    LD   $4,0
    A8 65 04    LDW  $5,(IX+$4)      ; a = IX+0
    42 04 02    LD   $4,2
    A8 6F 04    LDW  $15,(IX+$4)     ; b = IX+2
    FF          TRP                   ; word multiplication in bank 0
    D6 0A       DB   &HD6,&H0A        ; address &H0AD6
    42 04 04    LD   $4,4
    A0 E0 04    STW  $0,(IX-$4)      ; product -> function result
}
FUNCTION HMulW(a,b : Word) : Word;
BEGIN
  INLINE ($42,$04,$00,          { LD   $4,0                            }
          $A8,$65,$04,          { LDW  $5,(IX+$4)  ; a                 }
          $42,$04,$02,          { LD   $4,2                            }
          $A8,$6F,$04,          { LDW  $15,(IX+$4) ; b                 }
          $FF,                  { TRP ; word multiplication in bank 0  }
          $D6,$0A,              { DB   &HD6,&H0A ; address &H0AD6      }
          $42,$04,$04,          { LD   $4,4                            }
          $A0,$E0,$04);         { STW  $0,(IX-$4) ; product            }
END;
```

```
{
  HD61700 Word arithmetic primitive: HDivW
  Pascal parameter layout:
    a       IX+0
    b       IX+2
    result  IX-4

  BXWY register interface:
    $15,$16 = dividend
    $5,$6   = divisor
    $15,$16 = quotient on return

  BXWY is located in bank 0 at &H0AE8. TRP is used instead
  of CAL to invoke the bank-0 routine from the Pascal program.

  HD61700 source and assembled opcodes:
    42 04 00    LD   $4,0
    A8 6F 04    LDW  $15,(IX+$4)     ; a -> dividend $15,$16
    42 04 02    LD   $4,2
    A8 65 04    LDW  $5,(IX+$4)      ; b -> divisor $5,$6
    FF          TRP                   ; signed integer division in bank 0
    E8 0A       DB   &HE8,&H0A        ; address &H0AE8
    42 04 04    LD   $4,4
    A0 EF 04    STW  $15,(IX-$4)     ; quotient -> function result
}
FUNCTION HDivW(a,b : Word) : Word;
BEGIN
  INLINE ($42,$04,$00,          { LD   $4,0                              }
          $A8,$6F,$04,          { LDW  $15,(IX+$4) ; dividend            }
          $42,$04,$02,          { LD   $4,2                              }
          $A8,$65,$04,          { LDW  $5,(IX+$4)  ; divisor             }
          $FF,                  { TRP ; signed integer division in bank 0 }
          $E8,$0A,              { DB   &HE8,&H0A ; address &H0AE8        }
          $42,$04,$04,          { LD   $4,4                              }
          $A0,$EF,$04);         { STW  $15,(IX-$4) ; quotient            }
END;


{
  Pascal Word arithmetic primitives

  These provide a reference implementation using the native
  DL-Pascal arithmetic operators. They have the same function
  interface as the HD61700 primitives above and can therefore
  be passed interchangeably as WordOp parameters.
}
FUNCTION PAddW(a,b : Word) : Word;
BEGIN
  PAddW := a+b;
END;

FUNCTION PSubW(a,b : Word) : Word;
BEGIN
  PSubW := a-b;
END;
```

```
FUNCTION PMulW(a,b : Word) : Word;
BEGIN
  PMulW := a*b;
END;

FUNCTION PDivW(a,b : Word) : Word;
BEGIN
  PDivW := a DIV b;
END;


{
  Generic arithmetic primitive unit test

  PascalOp is the native DL-Pascal reference implementation.
  HD61700Op is the corresponding HD61700 INLINE implementation.

  AV and BV are different operand vectors. Their Cartesian product
  supplies N*N operand pairs. Only failures are reported individually.
}
PROCEDURE UnitTest(Name      : String;
                   PascalOp  : WordOp;
                   HD61700Op : WordOp;
                   N         : Word);
VAR
  i,j      : Word;
  Expected : Word;
  Actual   : Word;
  Errors   : Word;
BEGIN
  Errors := 0;

  FOR i := 1 TO N DO
    FOR j := 1 TO N DO
    BEGIN
      a := AV[i];
      b := BV[j];

      Expected := PascalOp(a,b);
      Actual   := HD61700Op(a,b);

      IF Expected <> Actual THEN
      BEGIN
        Errors := Errors+1;
        WriteLn(Name,' FAIL: ',
                a,' ',b,
                ' expected=',Expected,
                ' actual=',Actual);
      END;
    END;

  IF Errors=0 THEN
    WriteLn(Name,': ',N*N,' tests passed')
  ELSE
    WriteLn(Name,': ',Errors,' of ',N*N,' tests failed');
END;
```

```
{
  Three-term fixed-point approximation of zeta(3)

  Scale = 10000.

  t0 = 10000 * 77 / 64
     = 77 * 625 / 4
     = 77 * 156 + 77 / 4

  t1 = 10000 * 532 / 497664
     = 83125 / 7776

  t2 = 10000 * 1397 / 1555200000
     = 1397 / 155520

  The decomposition below keeps the individual constants and
  intermediate values within the usable Word range of the
  arithmetic primitives.
}



{
  Staged arithmetic implementation

  The arithmetic implementation is supplied entirely through
  WordOp parameters.

  Every primitive result is stored in a Pascal variable before
  another arithmetic primitive is invoked. This gives an explicit
  evaluation sequence and provides a reference implementation for
  the experiments below.
}
FUNCTION ZetaStaged(AddOp,SubOp,MulOp,DivOp : WordOp) : Word;
VAR
  t0,t1,t2 : Word;
  x,y      : Word;
BEGIN
  x := MulOp(77,156);
  y := DivOp(77,4);
  t0 := AddOp(x,y);

  x := MulOp(133,25);
  x := DivOp(x,24);
  x := MulOp(x,25);
  t1 := DivOp(x,324);

  x := DivOp(1397,240);
  t2 := DivOp(x,648);

  WriteLn('t0 = ',t0);
  WriteLn('t1 = ',t1);
  WriteLn('t2 = ',t2);

  x := SubOp(t0,t1);
  ZetaStaged := AddOp(x,t2);
END;
```

```
{
  Nested arithmetic with locally stored terms

  The arithmetic implementation is supplied through WordOp
  function parameters.

  Each individual zeta term is calculated by directly compounding
  the arithmetic operations, while the completed terms are stored
  in local Pascal variables before the final combination.

  This form is particularly useful for comparison with
  ZetaDirectNestedTerms below. The arithmetic expressions are
  equivalent; the important distinction is that the operations
  here are invoked indirectly through WordOp parameters.
}
FUNCTION ZetaNestedTerms(AddOp,SubOp,MulOp,DivOp : WordOp) : Word;
VAR
  t0,t1,t2 : Word;
BEGIN
  t0 :=
    AddOp(
      MulOp(77,156),
      DivOp(77,4));

  t1 :=
    DivOp(
      MulOp(
        DivOp(
          MulOp(133,25),
          24),
        25),
      324);

  t2 :=
    DivOp(
      DivOp(1397,240),
      648);

  WriteLn('t0 = ',t0);
  WriteLn('t1 = ',t1);
  WriteLn('t2 = ',t2);

  ZetaNestedTerms :=
    AddOp(
      SubOp(t0,t1),
      t2);
END;


{
  Direct HD61700 nested arithmetic with locally stored terms

  This version performs the same calculation as ZetaNestedTerms,
  but calls HAddW, HSubW, HMulW and HDivW directly rather than
  receiving them through WordOp function parameters.

  Each individual zeta term is calculated by directly compounding
```

```
  the HD61700 INLINE primitives, while the completed terms are
  stored in local Pascal variables.
}
FUNCTION ZetaDirectNestedTerms : Word;
VAR
  t0,t1,t2 : Word;
BEGIN
  t0 :=
    HAddW(
      HMulW(77,156),
      HDivW(77,4));

  t1 :=
    HDivW(
      HMulW(
        HDivW(
          HMulW(133,25),
          24),
        25),
      324);

  t2 :=
    HDivW(
      HDivW(1397,240),
      648);

  WriteLn('t0 = ',t0);
  WriteLn('t1 = ',t1);
  WriteLn('t2 = ',t2);

  ZetaDirectNestedTerms :=
    HAddW(
      HSubW(t0,t1),
      t2);
END;


BEGIN
  {
    Two deliberately different test vectors.

    BV contains no zero so that the same Cartesian product can also
    be used for division. Values are chosen so that this first test
    remains within the normal Word result range for all operations.
  }

  AV[1]  := 23;
  AV[2]  := 37;
  AV[3]  := 63;
  AV[4]  := 127;
  AV[5]  := 255;
  AV[6]  := 511;
  AV[7]  := 999;
  AV[8]  := 1024;
  AV[9]  := 2047;
  AV[10] := 3000;

  BV[1]  := 1;
```

```
  BV[2]  := 2;
  BV[3]  := 3;
  BV[4]  := 5;
  BV[5]  := 7;
  BV[6]  := 11;
  BV[7]  := 13;
  BV[8]  := 16;
  BV[9]  := 17;
  BV[10] := 19;

  WriteLn('HD61700 Word arithmetic verification');

  UnitTest('ADD',PAddW,HAddW,VectorSize);
  UnitTest('SUB',PSubW,HSubW,VectorSize);
  UnitTest('MUL',PMulW,HMulW,VectorSize);
  UnitTest('DIV',PDivW,HDivW,VectorSize);

  WriteLn;
  WriteLn('Pascal arithmetic through WordOp');
  WriteLn('Zeta = ',
          ZetaNestedTerms(PAddW,PSubW,PMulW,PDivW));

  WriteLn;
  WriteLn('HD61700 arithmetic, staged through WordOp');
  WriteLn('Zeta = ',
          ZetaStaged(HAddW,HSubW,HMulW,HDivW));

  WriteLn;
  WriteLn('HD61700 arithmetic, nested terms through WordOp');
  WriteLn('Zeta = ',
          ZetaNestedTerms(HAddW,HSubW,HMulW,HDivW));

  WriteLn;
  WriteLn('HD61700 arithmetic, direct nested terms');
  WriteLn('Zeta = ',ZetaDirectNestedTerms);
END.
```

Acknowledgments

This document was produced by Xandrian Labs. 付喪神.

Special thanks to Piotr. His dedication to preserving vintage electronics made this possible.

This is a collaboration with ChatGPT-5.6 Sol. All errors are human.

This work is freely shared with the vintage computing community. Please use it, build upon it, and cite the source.

Revision 3.