universal

Release v3

E. Theodore L. Omtzigt

Apr 13, 2021

CONTENTS:

Universal is a header-only C++17/20 template library for mixed-precision algorithm design and optimization. It contains plug-in replacements for native arithmetic types, parameterized in terms of precision, dynamic range, sampling profile, and rounding algorithms. The number systems provided in Universal enable algorithm development that optimizes performance and energy efficiency by tailoring the number systems to the requirements of the algorithm.

The motivation to find improvements to IEEE floating-point had been brewing in the HPC community since the late 90's. Most algorithms had become memory bound and computational scientists were looking for alternatives that provided more granularity in precision and dynamic range to extract more performance from the memory and networking subsystems. Even though the inefficiency of IEEE floating-point had been measured and agreed upon in the HPC community, it was the commercial demands of Deep Learning that provided the incentive to replace IEEE-754 with alternatives, such as half-floats, bfloats, TensorFloats, and DeepFloats. These alternatives are tailored to the application and yield speed-ups of two to three orders of magnitude, making it possible to scale AI deep learning algorithms to ever more capable and accurate solutions. Other computational science and engineering algorithms, such as Krylov solvers, Iso Geometric Analysis, N-Body problems, multi-grid methods, fast multipole methods, etc all stand to benefit from mixed-precision optimization, and Universal is providing the foundation for this new class of computational science and engineering performance.

The basic use pattern is as simple as:

```
#include <universal/number/posit/posit>

template<typename Real>
Real MyKernel(const Real& a, const Real& b) {
    return a * b;  // replace this with your kernel computation
}

constexpr double pi = 3.14159265358979323846;

int main() {
   using Real = sw::universal::posit<32,2>;

   Real a = sqrt(2);
   Real b = pi;
   std::cout << "Result: " << MyKernel(a, b) << std::endl;
}
```

The library contains integers, decimals, fixed-points, rationals, linear floats, tapered floats, logarithmic, interval and adaptive-precision integers and floats. There are example number system skeletons to get you started quickly if you desire to add your own, which is highly encouraged.

CHAPTER

ONE

USER GUIDE

1.1 How to build

If you do want to work with the code, the universal numbers software library is built using cmake version v3.18. Install the latest cmake. There are interactive installers for MacOS and Windows. For Linux, a portable approach downloads the shell archive and installs it at /usr/local:

```
> wget https://github.com/Kitware/CMake/releases/download/v3.18.2/cmake-3.18.2-Linux-
↪x86_64.sh
> sudo sh cmake-3.18.2-Linux-x86_64.sh --prefix=/usr/local --exclude-subdir
```

For Ubuntu, snap will install the latest cmake, and would be the preferred method:

```
> sudo snap install cmake --classic
```

The Universal library is a pure C++ template library without any further dependencies, even for the regression test suites, to enable hassle-free installation and use.

Simply clone the github repo, and you are ready to build the different components of the Universal library. The library contains tools to work with integers, decimals, fixed-points, floats, posits, valids, and logarithmic number systems. It contains educational programs that showcase simple use cases to familiarize yourself with different number systems, and application examples to highlight the use of different number systems to gain performance or numerical accuracy. Finally, each number system offers its own verification suite.

The easiest way to become familiar with all the options in the build process is to fire up the CMake GUI (or ccmake if you are on a headless server). The cmake output will summarize which options have been set. The output will looks something like this:

```
$ git clone https://github.com/stillwater-sc/universal
$ cd universal
$ mkdir build
$ cd build
$ cmake ..
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- The ASM compiler identification is GNU
-- Found assembler: /usr/bin/cc
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
```

(continues on next page)

(continued from previous page)

```
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- No default build type specified: setting CMAKE_BUILD_TYPE=Release
-- C++17 support has been enabled by default
-- Performing Test COMPILER_HAS_SSE3_FLAG
-- Performing Test COMPILER_HAS_SSE3_FLAG - Success
-- Performing Test COMPILER_HAS_AVX_FLAG
-- Performing Test COMPILER_HAS_AVX_FLAG - Success
-- Performing Test COMPILER_HAS_AVX2_FLAG
-- Performing Test COMPILER_HAS_AVX2_FLAG - Success
-- universal -> universal
-- include_install_dir         = include
-- include_install_dir_full    = include/universal
-- config_install_dir          = share/universal
-- include_install_dir_postfix = universal
-- PROJECT_SOURCE_DIR          = /home/stillwater/dev/clones/universal
-- PROJECT_VERSION             = 2.1.41
-- CMAKE_CURRENT_SOURCE_DIR    = /home/stillwater/dev/clones/universal
-- CMAKE_CURRENT_BINARY_DIR    = /home/stillwater/dev/clones/universal/build
--
-- ****************** Universal Arithmetic Library Configuration Summary␣
↪******************
-- General:
--   Version                     :   2.1.41
--   System                      :   Linux
--   C compiler                  :   /usr/bin/cc
--   Release C flags             :   -O3 -DNDEBUG -Wall -Wpedantic -Wno-narrowing -
↪Wno-deprecated
--   Debug C flags               :   -g -Wall -Wpedantic -Wno-narrowing -Wno-
↪deprecated
--   C++ compiler                :   /usr/bin/c++
--   Release CXX flags           :   -O3 -DNDEBUG -std=c++14  -Wall -Wpedantic -Wno-
↪narrowing -Wno-deprecated -std=c++14  -Wall -Wpedantic -Wno-narrowing -Wno-
↪deprecated
--   Debug CXX flags             :   -g -std=c++14  -Wall -Wpedantic -Wno-narrowing -
↪Wno-deprecated -std=c++14  -Wall -Wpedantic -Wno-narrowing -Wno-deprecated
--   Build type                  :   Release
--
--   BUILD_CI_CHECK              :   OFF
--
--   BUILD_STORAGE_CLASSES       :   OFF
--   BUILD_NATIVE_TYPES          :   OFF
--   BUILD_INTEGERS              :   OFF
--   BUILD_DECIMALS              :   OFF
--   BUILD_FIXPNTS               :   OFF
--   BUILD_LNS                   :   OFF
--   BUILD_UNUM_TYPE_1           :   OFF
--   BUILD_UNUM_TYPE_2           :   OFF
--   BUILD_POSITS                :   OFF
--   BUILD_VALIDS                :   OFF
--   BUILD_REALS                 :   OFF
--
--   BUILD_C_API_PURE_LIB        :   OFF
--   BUILD_C_API_SHIM_LIB        :   OFF
--   BUILD_C_API_LIB_PIC         :   OFF
--
```

(continues on next page)

(continued from previous page)

```
--   BUILD_CMD_LINE_TOOLS          :   ON
--   BUILD_EDUCATION               :   ON
--   BUILD_APPLICATIONS            :   ON
--   BUILD_NUMERICAL               :   OFF
--   BUILD_FUNCTIONS               :   OFF
--   BUILD_PLAYGROUND              :   ON
--
--   BUILD_CONVERSION_TESTS        :   OFF
--
--   BUILD_PERFORMANCE_TESTS       :   OFF
--
--   BUILD_IEEE_FLOAT_QUIRES       :   OFF
--
--   BUILD_DOCS                    :   OFF
--
-- Dependencies:
--   SSE3                          :   NO
--   AVX                           :   NO
--   AVX2                          :   NO
--   Pthread                       :   NO
--   TBB                           :   NO
--   OMP                           :   NO
--
-- Utilities:
--   Serializer                    :   NO
--
-- Install:
--   Install path                  :   /usr/local
--
-- Configuring done
-- Generating done
-- Build files have been written to: /home/stillwater/dev/clones/universal/build
```

The build options are enabled/disabled as follows:

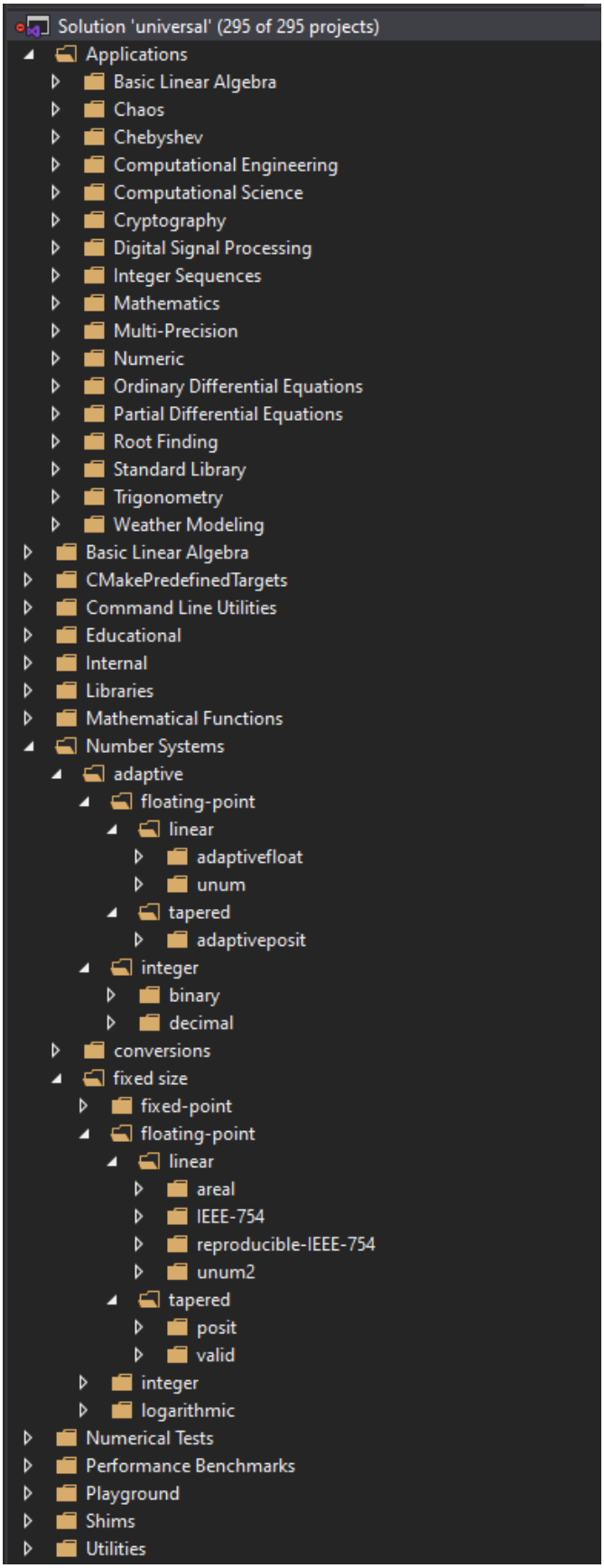
```
> cmake -DBUILD_EDUCATION=OFF -DBUILD_POSITS=ON ..
```

After building, issue the command *make test* to run the complete test suite of all the enabled components, as a regression capability when you are modifying the source code. This will take several minutes but will touch all the corners of the code.

```
> git clone https://github.com/stillwater-sc/universal
> cd universal
> mkdir build
> cd build
> cmake ..
> make -j $(nproc)
> make test
```

For Windows and Visual Studio, there are `CMakePredefinedTargets` that accomplish the same tasks:

```
    - ALL_BUILD will compile all the projects
    - INSTALL   will install the Universal library
    - RUN_TESTS will run all tests
```


Solution 'universal' (295 of 295 projects)
Applications
Basic Linear Algebra
Chaos
Chebyshev
Computational Engineering
Computational Science
Cryptography
Digital Signal Processing
Integer Sequences
Mathematics
Multi-Precision
Numeric
Ordinary Differential Equations
Partial Differential Equations
Root Finding
Standard Library
Trigonometry
Weather Modeling
Basic Linear Algebra
CMakePredefinedTargets
Command Line Utilities
Educational
Internal
Libraries
Mathematical Functions
Number Systems
adaptive
floating-point
linear
adaptivefloat
unum
tapered
adaptiveposit
integer
binary
decimal
conversions
fixed size
fixed-point
floating-point
linear
areal
IEEE-754
reproducible-IEEE-754
unum2
tapered
posit
valid
integer
logarithmic
Numerical Tests
Performance Benchmarks
Playground
Shims
Utilities

1.2 Quick start

If you just want to experiment with the number system tools and test suites, and don't want to bother cloning and building the source code, there is a Docker container to get started:

```
> docker pull stillwater/universal
> docker run -it --rm stillwater/universal bash
stillwater@b3e6708fd732:~/universal/build$ ls
CMakeCache.txt       Makefile      cmake-uninstall.cmake  playground  universal-
↪config-version.cmake
CMakeFiles           applications  cmake_install.cmake    tests       universal-
↪config.cmake
CTestTestfile.cmake  c_api         education              tools       universal-
↪targets.cmake
```

From the build directory, it is convenient to run any of the regression test suites:

```
stillwater@b3e6708fd732:~/universal/build$ tests/posit/specialized/fast_posit_8_0
Fast specialization posit<8,0> configuration tests
 posit<  8,0> useed scale     1     minpos scale         -6     maxpos scale
↪6
Logic operator tests
 posit<8,0>     ==          (native)   PASS
 posit<8,0>     !=          (native)   PASS
 posit<8,0>     <           (native)   PASS
 posit<8,0>     <=          (native)   PASS
 posit<8,0>     >           (native)   PASS
 posit<8,0>     >=          (native)   PASS
Assignment/conversion tests
 posit<8,0> integer assign (native)   PASS
 posit<8,0> float assign   (native)   PASS
Arithmetic tests
 posit<8,0> add            (native)   PASS
 posit<8,0> +=             (native)   PASS
 posit<8,0> subtract       (native)   PASS
 posit<8,0> -=             (native)   PASS
 posit<8,0> multiply       (native)   PASS
 posit<8,0> *=             (native)   PASS
 posit<8,0> divide         (native)   PASS
 posit<8,0> /=             (native)   PASS
 posit<8,0> negate         (native)   PASS
 posit<8,0> reciprocate    (native)   PASS
Elementary function tests
 posit<8,0> sqrt           (native)   PASS
 posit<8,0> exp                       PASS
 posit<8,0> exp2                      PASS
 posit<8,0> log                       PASS
 posit<8,0> log2                      PASS
 posit<8,0> log10                     PASS
 posit<8,0> sin                       PASS
 posit<8,0> cos                       PASS
 posit<8,0> tan                       PASS
 posit<8,0> atan                      PASS
 posit<8,0> asin                      PASS
 posit<8,0> acos                      PASS
 posit<8,0> sinh                      PASS
 posit<8,0> cosh                      PASS
```

(continues on next page)

(continued from previous page)

```
 posit<8,0> tanh                     PASS
 posit<8,0> atanh                    PASS
 posit<8,0> acosh                    PASS
 posit<8,0> asinh                    PASS
ValidatePower has been truncated
 posit<8,0> pow                      PASS
```

In /usr/local/bin there are a set of command line utilities to inspect floating point encodings.

```
stillwater@b3e6708fd732:~/universal/build$ ls /usr/local/bin
compd  compf  compfp  compieee  compld  complns  compp  compsi  compui  float2posit
↪propenv  propp

stillwater@b3e6708fd732:~/universal$ compieee 1.234567890123456789012345
compiler              : 7.5.0
float precision       : 23 bits
double precision      : 52 bits
long double precision : 63 bits

Representable?        : maybe

Decimal representations
input value:  1.234567890123456789012345
      float:                1.23456788
     double:        1.2345678901234567
long double:    1.2345678901234567899

Hex representations
input value:  1.234567890123456789012345
      float:                1.23456788    hex: 0.7f.1e0652
     double:        1.2345678901234567    hex: 0.3ff.3c0ca428c59fb
long double:    1.2345678901234567899    hex: 0.3fff.1e06521462cfdb8d

Binary representations:
      float:                1.23456788    bin: 0.01111111.00111100000011001010010
     double:        1.2345678901234567    bin: 0.01111111111.
↪0011110000001100101001000001010001100010110011111011
long double:    1.2345678901234567899    bin: 0.011111111111111.
↪0011110000001100101001000010100011000101
↪10011111101101110001101

Native triple representations (sign, scale, fraction):
      float:                1.23456788    triple: (+,0,00111100000011001010010)
     double:        1.2345678901234567    triple: (+,0,
↪0011110000001100101001000010100011000101100111111011)
long double:    1.2345678901234567899    triple: (+,0,
↪0011110000001100101001000010100011000101100111111011
↪1101110001101)

Scientific triple representation (sign, scale, fraction):
input value:  1.234567890123456789012345
      float:                1.23456788    triple: (+,0,00111100000011001010010)
     double:        1.2345678901234567    triple: (+,0,
↪0011110000001100101001000010100011000101100111111011)
long double:    1.2345678901234567899    triple: (+,0,
↪0011110000001100101001000010100011000101100111111011
↪1101110001101)
```

(continues on next page)

(continued from previous page)

```
      exact: TBD
```

Or posit encodings:

```
stillwater@b3e6708fd732:~/universal/build$ compp 1.234567890123456789
posit< 8,0> = s0 r10 e f01000 qNE v1.25
posit< 8,1> = s0 r10 e0 f0100 qNE v1.25
posit< 8,2> = s0 r10 e00 f010 qNE v1.25
posit< 8,3> = s0 r10 e000 f01 qNE v1.25
posit<16,1> = s0 r10 e0 f001111000001 qNE v1.234619140625
posit<16,2> = s0 r10 e00 f00111100000 qNE v1.234375
posit<16,3> = s0 r10 e000 f0011110000 qNE v1.234375
posit<32,1> = s0 r10 e0 f00111100000011001010010000011 qNE v1.2345678918063641
posit<32,2> = s0 r10 e00 f00111100000011001010010000001 qNE v1.2345678880810738
posit<32,3> = s0 r10 e000 f00111100000011001010010000001 qNE v1.2345678955316544
posit<48,1> = s0 r10 e0 f001111000000110010100100000101000110001011010 qNE v1.
↪2345678901234578
posit<48,2> = s0 r10 e00 f001111000000110010100100000101000110001011010 qNE v1.
↪2345678901234578
posit<48,3> = s0 r10 e000 f001111000000110010100100000101000110001011010 qNE v1.
↪2345678901233441
posit<64,1> = s0 r10 e0 f00111100000011001010010000010100011000101100111111011000000000␣
↪qNE v1.2345678901234567
posit<64,2> = s0 r10 e00 f00111100000011001010010000010100011000101100111111011000000000␣
↪qNE v1.2345678901234567
posit<64,3> = s0 r10 e000 f00111100000011001010010000010100011000101100111111011000000000␣
↪qNE v1.2345678901234567
posit<64,4> = s0 r10 e0000 f00111100000011001010010000010100011000101100111111011000000␣
↪qNE v1.2345678901234567
```

The following two educational examples are pretty informative when you are just starting out learning about posits: edu_scales and edu_tables.

```
stillwater@b3e6708fd732:~/universal/build$ education/posit/edu_scales
Experiments with the scale of posit numbers
Posit specificiation examples and their ranges:
Scales are represented as the binary scale of the number: i.e. 2^scale

Small, specialized posit configurations
nbits = 3
 posit<  3,0> useed scale     1     minpos scale         -1     maxpos scale
↪1
 posit<  3,1> useed scale     2     minpos scale         -2     maxpos scale
↪2
 posit<  3,2> useed scale     4     minpos scale         -4     maxpos scale
↪4
 posit<  3,3> useed scale     8     minpos scale         -8     maxpos scale
↪8
 posit<  3,4> useed scale    16     minpos scale        -16     maxpos scale
↪16
nbits = 4
 posit<  4,0> useed scale     1     minpos scale         -2     maxpos scale
↪2
 posit<  4,1> useed scale     2     minpos scale         -4     maxpos scale
↪4
 posit<  4,2> useed scale     4     minpos scale         -8     maxpos scale
↪8
```

(continues on next page)

(continued from previous page)

```
 posit<  4,3> useed scale     8     minpos scale        -16     maxpos scale
↪16
 posit<  4,4> useed scale    16     minpos scale        -32     maxpos scale
↪32
nbits = 5
 posit<  5,0> useed scale     1     minpos scale         -3     maxpos scale
↪3
 posit<  5,1> useed scale     2     minpos scale         -6     maxpos scale
↪6
 posit<  5,2> useed scale     4     minpos scale        -12     maxpos scale
↪12
 posit<  5,3> useed scale     8     minpos scale        -24     maxpos scale
↪24
 posit<  5,4> useed scale    16     minpos scale        -48     maxpos scale
↪48
...
```

The command `edu_tables` generates tables of full posit encodings and their constituent parts:

```
stillwater@b3e6708fd732:~/universal/build$ education/posit/edu_tables | more
Generate posit configurations
Generate Posit Lookup table for a POSIT<2,0> in TXT format
   #           Binary         Decoded       k    sign   scale          regime
↪exponent        fraction
     value    posit_format
   0:               00              00      -1       0       0               0
↪       ~               ~
          0           2.0x0p
   1:               01              01       0       0       0               1
↪       ~               ~
          1           2.0x1p
   2:               10              10       1       1       0               0
↪       ~               ~
        nar           2.0x2p
   3:               11              11       0       1       0               1
↪       ~               ~
         -1           2.0x3p
Generate Posit Lookup table for a POSIT<3,0> in TXT format
   #           Binary         Decoded       k    sign   scale          regime
↪exponent        fraction
     value    posit_format
   0:              000             000      -2       0      -1              00
↪       ~               ~
          0           3.0x0p
   1:              001             001      -1       0      -1              01
↪       ~               ~
        0.5           3.0x1p
   2:              010             010       0       0       0              10
↪       ~               ~
          1           3.0x2p
   3:              011             011       1       0       1              11
↪       ~               ~
          2           3.0x3p
   4:              100             100       2       1      -1              00
↪       ~               ~
        nar           3.0x4p
   5:              101             111       1       1       1              11
↪       ~               ~
```

(continues on next page)

(continued from previous page)

```
          -2            3.0x5p
   6:               110              110        0        1        0               10
↪        ~                ~
          -1            3.0x6p
   7:               111              101       -1        1       -1               01
↪        ~                ~
       -0.5            3.0x7p
...
```

1.3 Where to find help

- GitHub Issue: bug reports, feature requests, etc.
- Forum: discussion of alternatives to IEEE-754 for computational science.
- Slack: online chats, discussions, and collaboration with other users, researchers and developers.

1.4 Citation

Please cite our work if you use *Universal*.

```
@article{Omtzigt2020,
    author    = {E. Theodore L. Omtzigt and Peter Gottschling and Mark Seligman and
↪William Zorn},
    title     = {{Universal Numbers Library}: design and implementation of a high-
↪performance reproducible number systems library},
    journal   = {arXiv:2012.11011},
    year      = {2020},
}
```

1.5 Number systems

The _Universal_ library provides a broad set of parameterized number system specialized to the bit-level.

1.5.1 integer

integer is a fixed-precision arbitrary configuration 2's complement integer.

1.5.2 decimal

decimal is an adaptive precision decimal number system.

1.5.3 fixpnt

fixpnt is a fixed-size arbitrary configuration fixed-point number system with either Modulo or Saturating arithmetic.

1.5.4 areal

areal is a linear floating-point with an uncertainty bit and interval arithmetic behavior.

1.5.5 bfloat

bfloat is a linear floating-point with gradual underflow and overflow.

1.5.6 posit

posit is a tapered floating-point system.

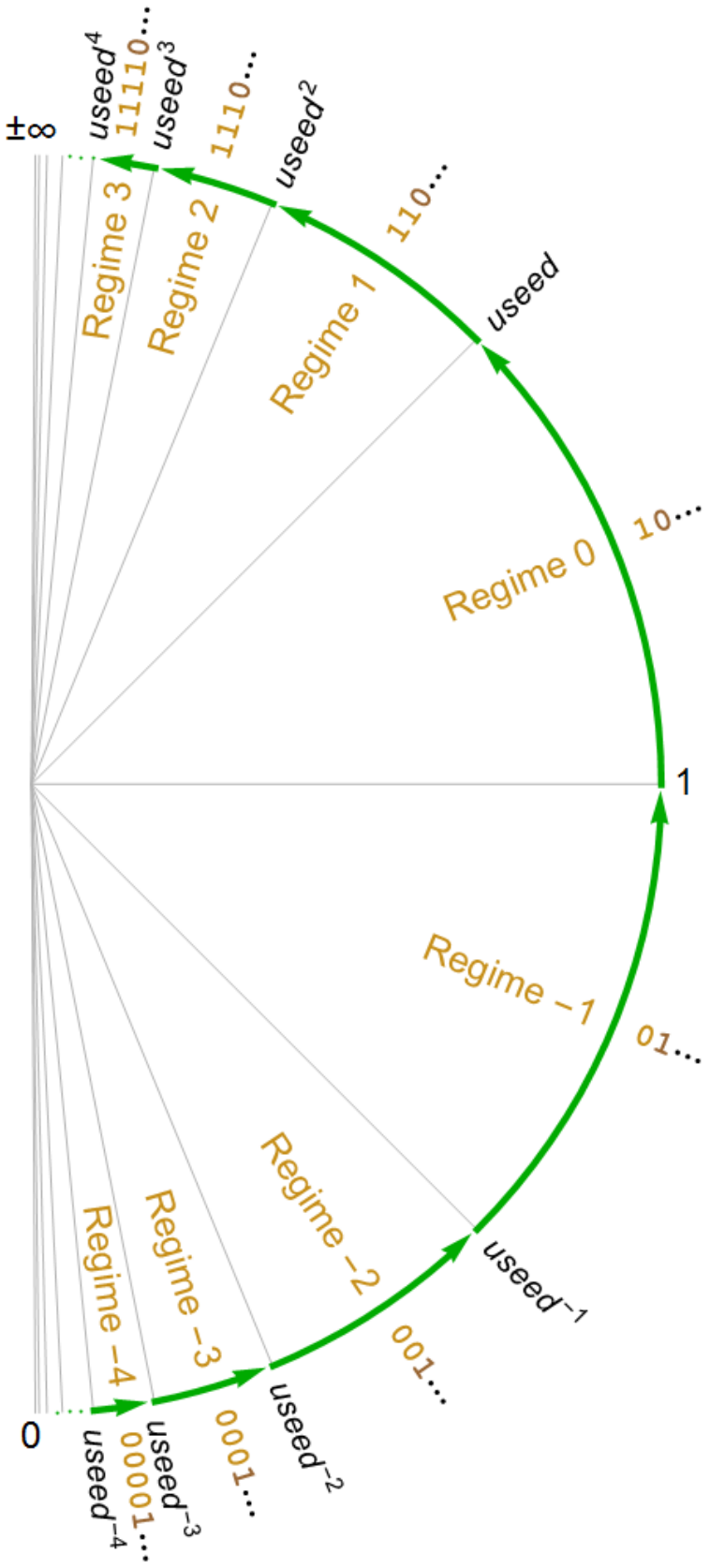
Posit Refinement Visualization

Universal numbers, unums for short, are for expressing real numbers, and ranges of real numbers. There are two modes of operation, selectable by the programmer, *posit* mode, and *valid* mode.

In *posit* mode, a unum behaves much like a floating-point number of fixed size, rounding to the nearest expressible value if the result of a calculation is not expressible exactly. A posit offers more accuracy and a larger dynamic range than floats with the same number of bits.

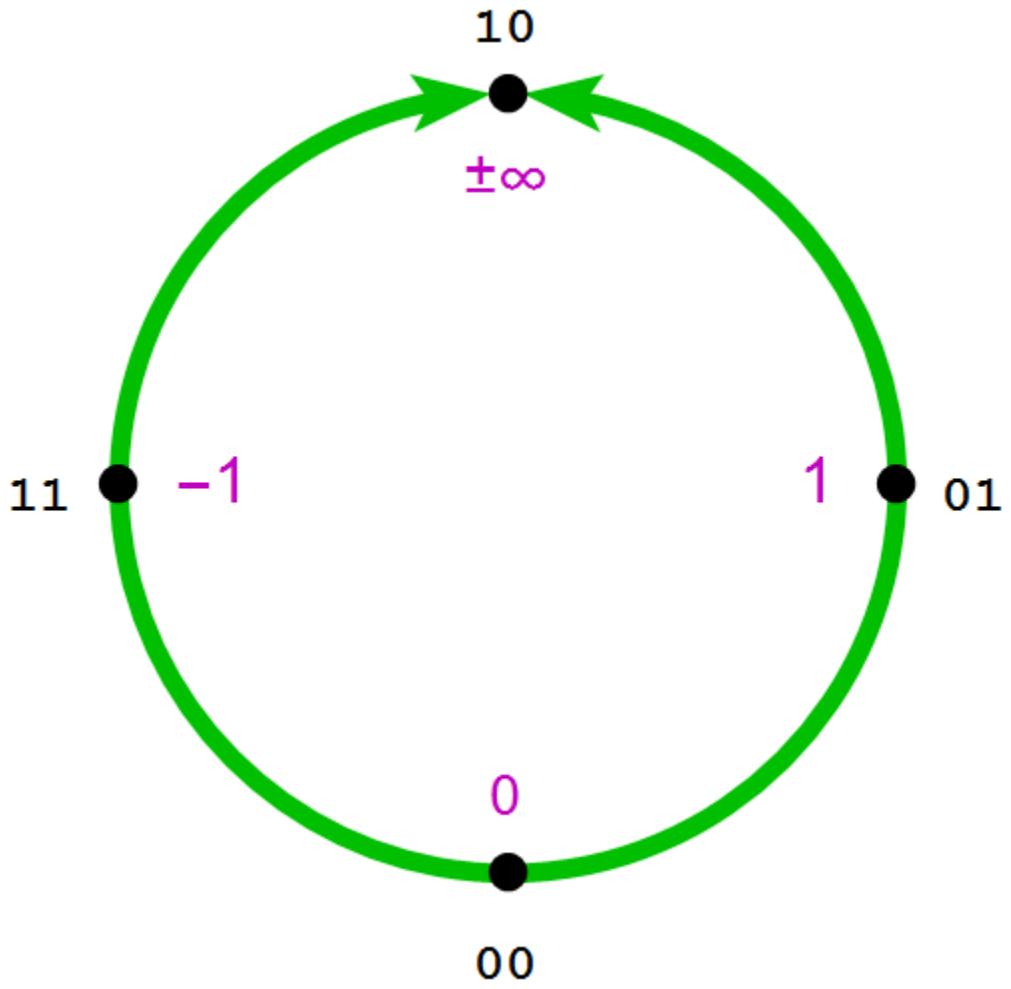
In *valid* mode, a unum represents a range of real numbers and can be used to rigorously bound answers much like interval arithmetic does.

The positive regime for a posit shows a very specific structure, as can be seen in the image blow:

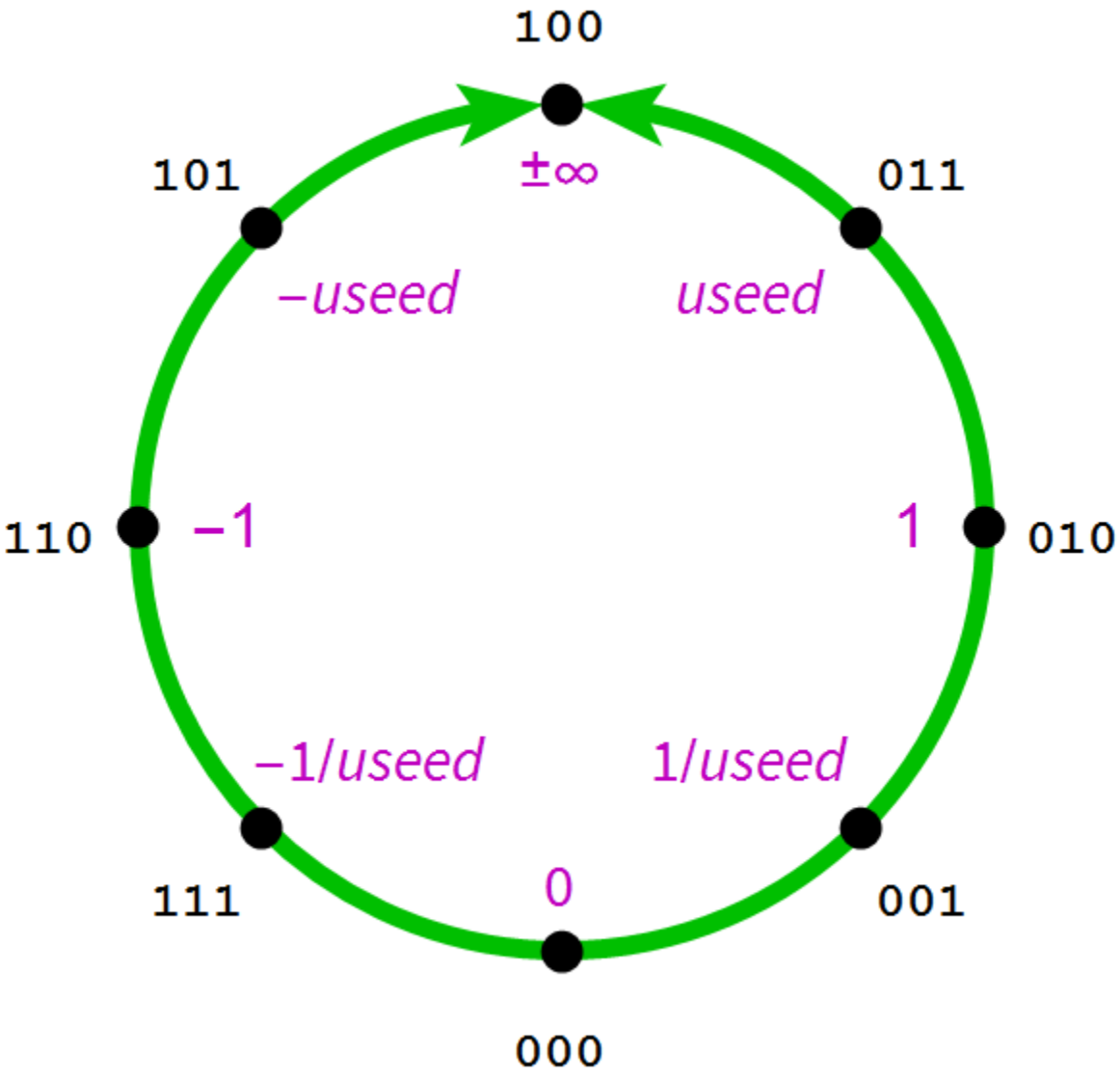

±∞
useed^4
11110…
useed^3
1110…
useed^2
110…
useed
10…
1
01…
useed^-1
001…
useed^-2
0001…
useed^-3
00001…
useed^-4
0
Regime 3
Regime 2
Regime 1
Regime 0
Regime −1
Regime −2
Regime −3
Regime −4

Posit configurations have a very specific relationship to one another. When expanding a posit, the new value falls 'between' the old values of the smaller posit. The new value is the arithmetic mean of the two numbers if the expanding bit is a fraction bit, and it is the geometric mean of the two numbers if the expanding bit is a regime or exponent bit. Here is the starting progression from *posit<2,0>* to *posit<7,1>*:

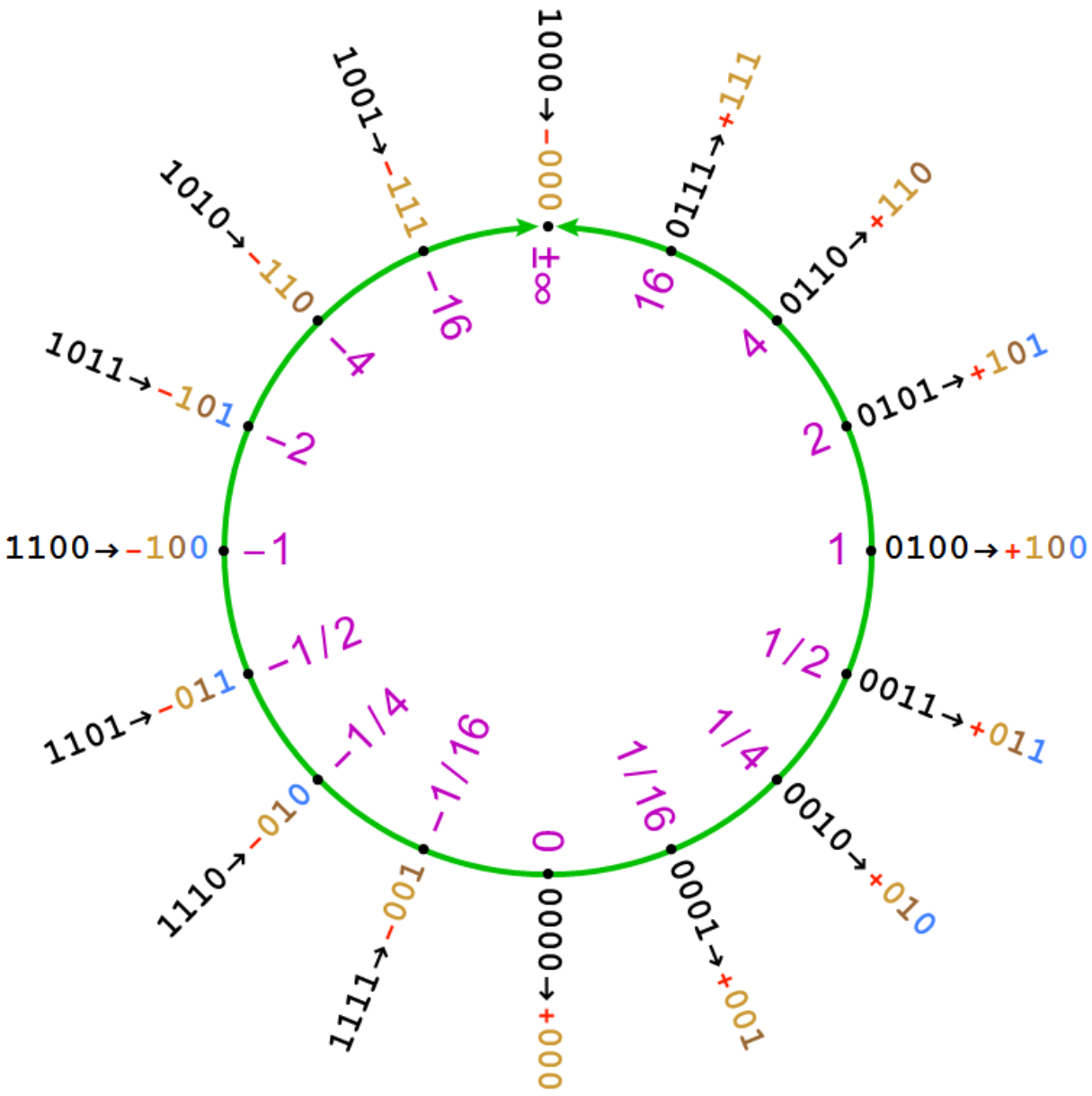
The *seed* posit:

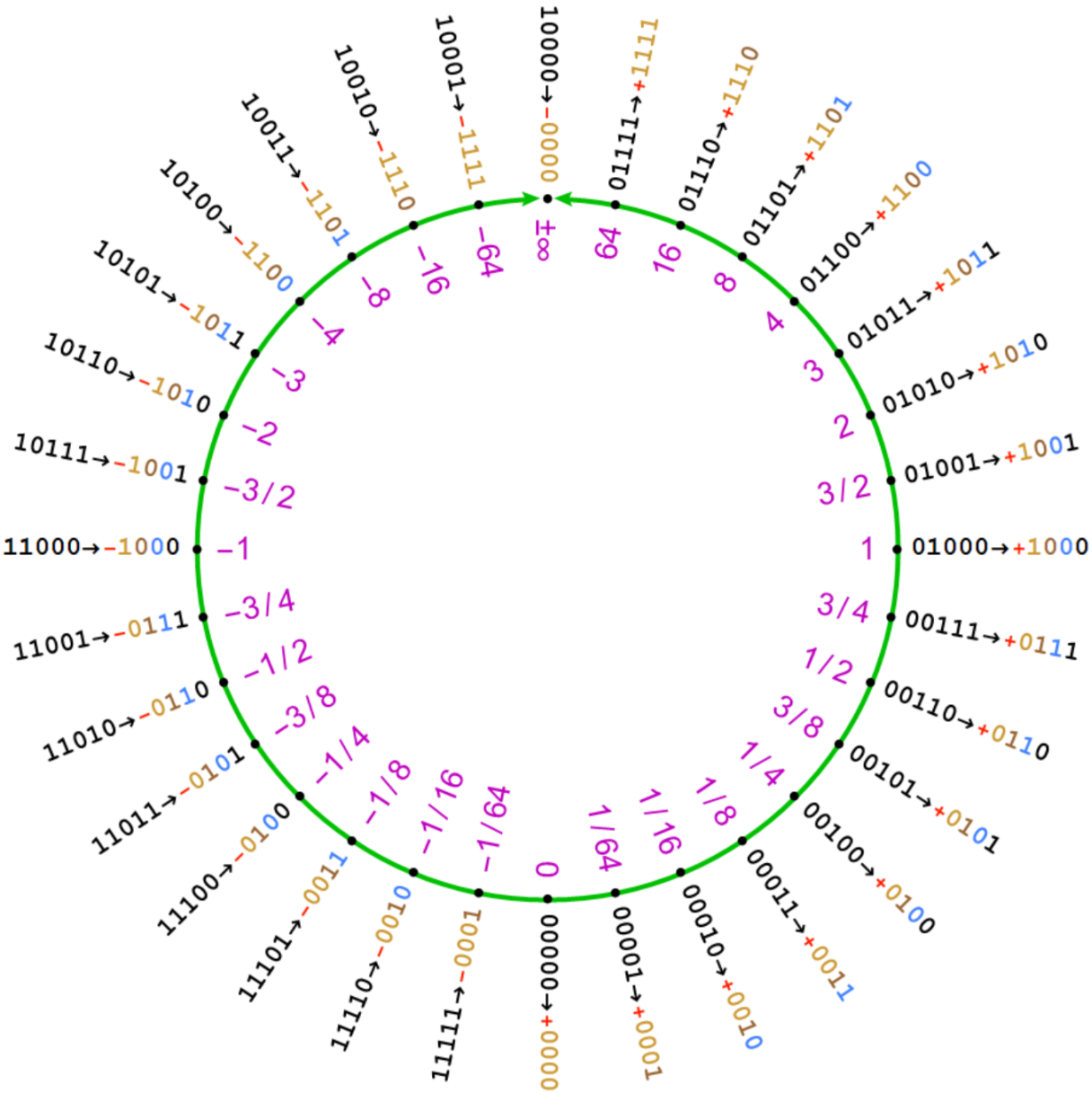


posit<3,0>:

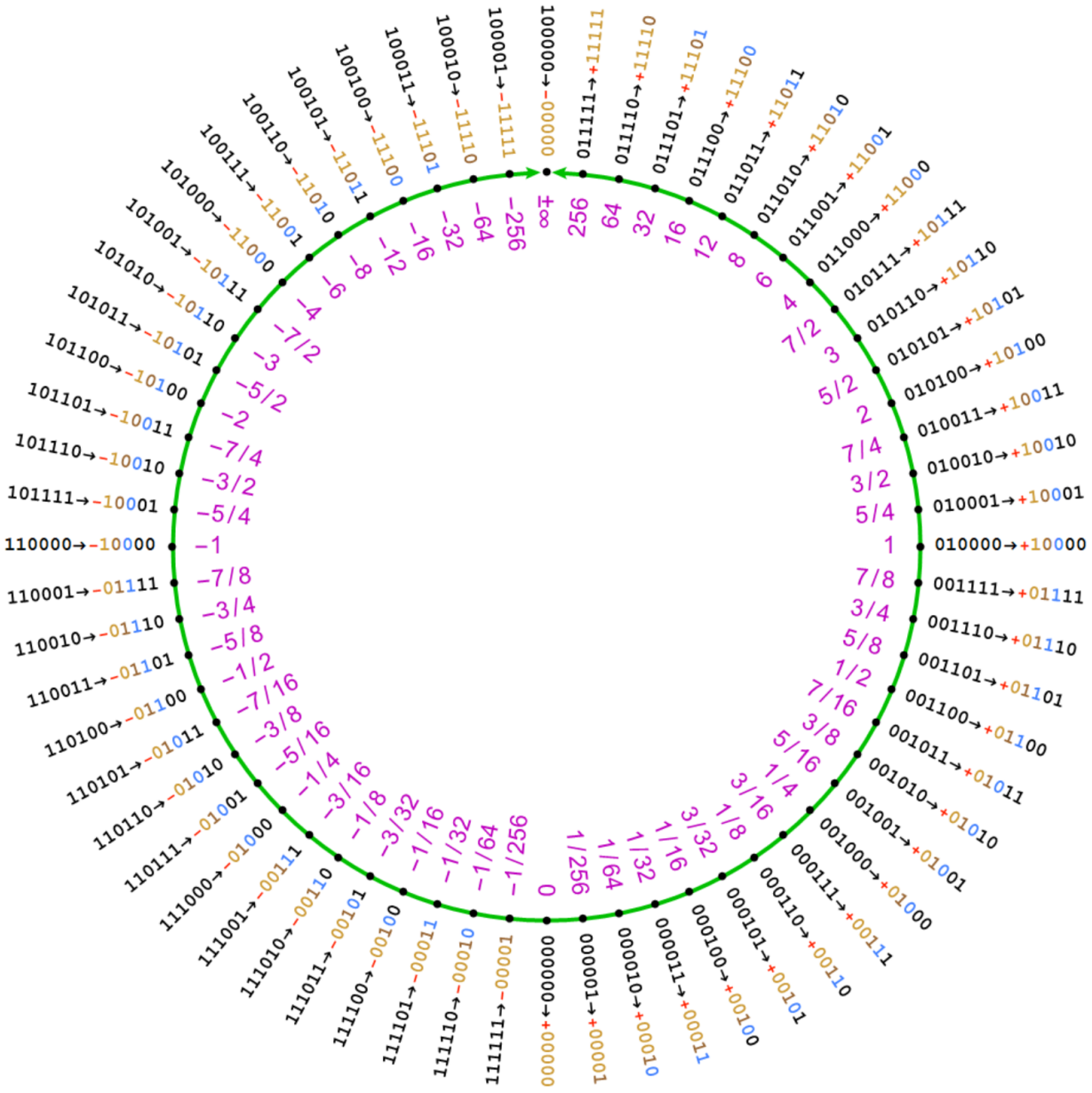


posit<4,1>:

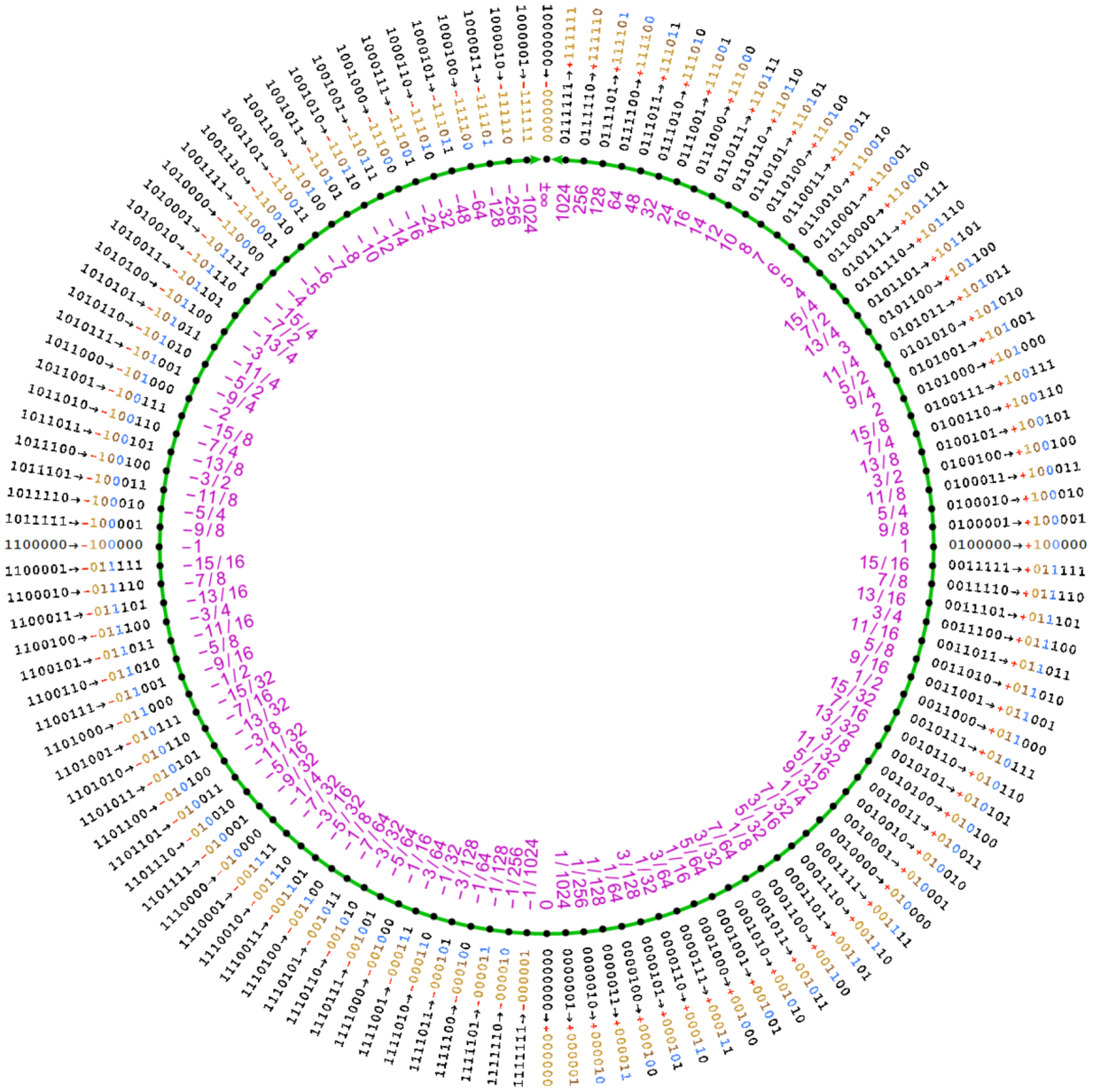

1000→-000
±∞
0111→+111
16
0110→+110
4
0101→+101
2
0100→+100
1
0011→+011
1/2
0010→+010
1/4
0001→+001
1/16
0000→+000
0
1111→-001
-1/16
1110→-010
-1/4
1101→-011
-1/2
1100→-100
-1
1011→-101
-2
1010→-110
-4
1001→-111
-16


10000→-0000
01111→+1111
01110→+1110
01101→+1101
01100→+1100
01011→+1011
01010→+1010
01001→+1001
01000→+1000
00111→+0111
00110→+0110
00101→+0101
00100→+0100
00011→+0011
00010→+0010
00001→+0001
00000→+0000
11111→-0001
11110→-0010
11101→-0011
11100→-0100
11011→-0101
11010→-0110
11001→-0111
11000→-1000
10111→-1001
10110→-1010
10101→-1011
10100→-1100
10011→-1101
10010→-1110
10001→-1111
±∞
64
16
8
4
3
2
3/2
1
3/4
1/2
3/8
1/4
1/8
1/16
1/64
0
-1/64
-1/16
-1/8
-1/4
-3/8
-1/2
-3/4
-1
-3/2
-2
-3
-4
-8
-16
-64

posit<5,1>:


100000→-00000
011111→+11111
011110→+11110
011101→+11101
011100→+11100
011011→+11011
011010→+11010
011001→+11001
011000→+11000
010111→+10111
010110→+10110
010101→+10101
010100→+10100
010011→+10011
010010→+10010
010001→+10001
010000→+10000
001111→+01111
001110→+01110
001101→+01101
001100→+01100
001011→+01011
001010→+01010
001001→+01001
001000→+01000
000111→+00111
000110→+00110
000101→+00101
000100→+00100
000011→+00011
000010→+00010
000001→+00001
000000→+00000
100001→-11111
100010→-11110
100011→-11101
100100→-11100
100101→-11011
100110→-11010
100111→-11001
101000→-11000
101001→-10111
101010→-10110
101011→-10101
101100→-10100
101101→-10011
101110→-10010
101111→-10001
110000→-10000
110001→-01111
110010→-01110
110011→-01101
110100→-01100
110101→-01011
110110→-01010
110111→-01001
111000→-01000
111001→-00111
111010→-00110
111011→-00101
111100→-00100
111101→-00011
111110→-00010
111111→-00001
±∞
256
64
32
16
12
8
6
4
7/2
3
5/2
2
7/4
3/2
5/4
1
7/8
3/4
5/8
1/2
7/16
3/8
5/16
1/4
3/16
1/8
3/32
1/16
1/32
1/64
1/256
0
-1/256
-1/64
-1/32
-1/16
-3/32
-1/8
-3/16
-1/4
-5/16
-3/8
-7/16
-1/2
-5/8
-3/4
-7/8
-1
-5/4
-3/2
-7/4
-2
-5/2
-3
-7/2
-4
-6
-8
-12
-16
-32
-64
-256

posit<6,1>:



posit<7,1>:

Posit value tables

The program "education/tables/edu_tables_posits" will generate the posit encodings and associated real values for reference. These tables are a great aid in understanding posit arithmetic and rounding.

```
>:~/dev/universal/build$ education/tables/edu_tables_posits::
    Generate posit configurations
    -128         100101         111011 Sign : -1 Regime :    1 Exponent :      8␣
↪Fraction :         1 Value :              -128 11
```

```
   Generate Posit Lookup table for a POSIT<3,0>
      #           Binary         Decoded       k    sign   scale          regime    ␣
↪   exponent        fraction                          value
```

(continues on next page)

(continued from previous page)

```
      0:              000             000      -2       1      -2              00 ␣
↪            ~                ~                                 0
      1:              001             001      -1       1      -1              01 ␣
↪            ~                ~                               0.5
      2:              010             010       0       1       0              10 ␣
↪            ~                ~                                 1
      3:              011             011       1       1       1              11 ␣
↪            ~                ~                                 2
      4:              100             100       2      -1       2              00 ␣
↪            ~                ~                               NaR
      5:              101             111       1      -1       1              11 ␣
↪            ~                ~                                -2
      6:              110             110       0      -1       0              10 ␣
↪            ~                ~                                -1
      7:              111             101      -1      -1      -1              01 ␣
↪            ~                ~                              -0.5
```

```
   Generate Posit Lookup table for a POSIT<4,0>
      #           Binary         Decoded       k    sign   scale          regime ␣
↪   exponent        fraction                          value
      0:             0000            0000      -3       1      -3             000 ␣
↪            ~                -                                 0
      1:             0001            0001      -2       1      -2             001 ␣
↪            ~                -                              0.25
      2:             0010            0010      -1       1      -1             01- ␣
↪            ~                0                               0.5
      3:             0011            0011      -1       1      -1             01- ␣
↪            ~                1                              0.75
      4:             0100            0100       0       1       0             10- ␣
↪            ~                0                                 1
      5:             0101            0101       0       1       0             10- ␣
↪            ~                1                               1.5
      6:             0110            0110       1       1       1             110 ␣
↪            ~                -                                 2
      7:             0111            0111       2       1       2             111 ␣
↪            ~                -                                 4
      8:             1000            1000       3      -1       3             000 ␣
↪            ~                -                               NaR
      9:             1001            1111       2      -1       2             111 ␣
↪            ~                -                                -4
     10:             1010            1110       1      -1       1             110 ␣
↪            ~                -                                -2
     11:             1011            1101       0      -1       0             10- ␣
↪            ~                1                             -1.5
     12:             1100            1100       0      -1       0             10- ␣
↪            ~                0                                -1
     13:             1101            1011      -1      -1      -1             01- ␣
↪            ~                1                             -0.75
     14:             1110            1010      -1      -1      -1             01- ␣
↪            ~                0                              -0.5
     15:             1111            1001      -2      -1      -2             001 ␣
↪            ~                -                             -0.25
```

```
   Generate Posit Lookup table for a POSIT<4,1>
      #           Binary         Decoded       k    sign   scale          regime ␣
↪   exponent        fraction                          value
```

(continues on next page)

(continued from previous page)

```
      0:              0000            0000     -3       1     -6              000 ␣
↪           -               ~                               0
      1:              0001            0001     -2       1     -4              001 ␣
↪           -               ~                          0.0625
      2:              0010            0010     -1       1     -2              01- ␣
↪           0               ~                            0.25
      3:              0011            0011     -1       1     -1              01- ␣
↪           1               ~                             0.5
      4:              0100            0100      0       1      0              10- ␣
↪           0               ~                               1
      5:              0101            0101      0       1      1              10- ␣
↪           1               ~                               2
      6:              0110            0110      1       1      2              110 ␣
↪           -               ~                               4
      7:              0111            0111      2       1      4              111 ␣
↪           -               ~                              16
      8:              1000            1000      3      -1      6              000 ␣
↪           -               ~                             NaR
      9:              1001            1111      2      -1      4              111 ␣
↪           -               ~                             -16
     10:              1010            1110      1      -1      2              110 ␣
↪           -               ~                              -4
     11:              1011            1101      0      -1      1              10- ␣
↪           1               ~                              -2
     12:              1100            1100      0      -1      0              10- ␣
↪           0               ~                              -1
     13:              1101            1011     -1      -1     -1              01- ␣
↪           1               ~                            -0.5
     14:              1110            1010     -1      -1     -2              01- ␣
↪           0               ~                           -0.25
     15:              1111            1001     -2      -1     -4              001 ␣
↪           -               ~                         -0.0625
```

```
   Generate Posit Lookup table for a POSIT<5,0>
      #          Binary         Decoded       k    sign   scale           regime   ␣
↪  exponent        fraction                          value
      0:             00000           00000     -4       1     -4              0000␣
↪            ~             --                                 0
      1:             00001           00001     -3       1     -3              0001␣
↪            ~             --                             0.125
      2:             00010           00010     -2       1     -2              001-␣
↪            ~             0-                              0.25
      3:             00011           00011     -2       1     -2              001-␣
↪            ~             1-                             0.375
      4:             00100           00100     -1       1     -1              01--␣
↪            ~             00                               0.5
      5:             00101           00101     -1       1     -1              01--␣
↪            ~             01                             0.625
      6:             00110           00110     -1       1     -1              01--␣
↪            ~             10                              0.75
      7:             00111           00111     -1       1     -1              01--␣
↪            ~             11                             0.875
      8:             01000           01000      0       1      0              10--␣
↪            ~             00                                 1
      9:             01001           01001      0       1      0              10--␣
↪            ~             01                              1.25
```

(continues on next page)

(continued from previous page)

```
      10:            01010          01010       0       1        0                10--␣
↪              ~               10                              1.5
      11:            01011          01011       0       1        0                10--␣
↪              ~               11                             1.75
      12:            01100          01100       1       1        1                110-␣
↪              ~               0-                                2
      13:            01101          01101       1       1        1                110-␣
↪              ~               1-                                3
      14:            01110          01110       2       1        2                1110␣
↪              ~               --                                4
      15:            01111          01111       3       1        3                1111␣
↪              ~               --                                8
      16:            10000          10000       4      -1        4                0000␣
↪              ~               --                              NaR
      17:            10001          11111       3      -1        3                1111␣
↪              ~               --                               -8
      18:            10010          11110       2      -1        2                1110␣
↪              ~               --                               -4
      19:            10011          11101       1      -1        1                110-␣
↪              ~               1-                               -3
      20:            10100          11100       1      -1        1                110-␣
↪              ~               0-                               -2
      21:            10101          11011       0      -1        0                10--␣
↪              ~               11                            -1.75
      22:            10110          11010       0      -1        0                10--␣
↪              ~               10                             -1.5
      23:            10111          11001       0      -1        0                10--␣
↪              ~               01                            -1.25
      24:            11000          11000       0      -1        0                10--␣
↪              ~               00                               -1
      25:            11001          10111      -1      -1       -1                01--␣
↪              ~               11                           -0.875
      26:            11010          10110      -1      -1       -1                01--␣
↪              ~               10                            -0.75
      27:            11011          10101      -1      -1       -1                01--␣
↪              ~               01                           -0.625
      28:            11100          10100      -1      -1       -1                01--␣
↪              ~               00                             -0.5
      29:            11101          10011      -2      -1       -2                001-␣
↪              ~               1-                           -0.375
      30:            11110          10010      -2      -1       -2                001-␣
↪              ~               0-                            -0.25
      31:            11111          10001      -3      -1       -3                0001␣
↪              ~               --                           -0.125
```

```
    Generate Posit Lookup table for a POSIT<5,1>
       #           Binary         Decoded       k    sign   scale          regime     ␣
↪   exponent         fraction                          value
       0:            00000          00000      -4       1      -8                0000␣
↪              -               -                                0
       1:            00001          00001      -3       1      -6                0001␣
↪              -               -                          0.015625
       2:            00010          00010      -2       1      -4                001-␣
↪              0               -                          0.0625
       3:            00011          00011      -2       1      -3                001-␣
↪              1               -                           0.125
```

(continues on next page)

(continued from previous page)

```
      4:          00100          00100      -1       1      -2          01--␣
↪             0              0                        0.25
      5:          00101          00101      -1       1      -2          01--␣
↪             0              1                       0.375
      6:          00110          00110      -1       1      -1          01--␣
↪             1              0                         0.5
      7:          00111          00111      -1       1      -1          01--␣
↪             1              1                        0.75
      8:          01000          01000       0       1       0          10--␣
↪             0              0                           1
      9:          01001          01001       0       1       0          10--␣
↪             0              1                         1.5
     10:          01010          01010       0       1       1          10--␣
↪             1              0                           2
     11:          01011          01011       0       1       1          10--␣
↪             1              1                           3
     12:          01100          01100       1       1       2          110-␣
↪             0              -                           4
     13:          01101          01101       1       1       3          110-␣
↪             1              -                           8
     14:          01110          01110       2       1       4          1110␣
↪             -              -                          16
     15:          01111          01111       3       1       6          1111␣
↪             -              -                          64
     16:          10000          10000       4      -1       8          0000␣
↪             -              -                         NaR
     17:          10001          11111       3      -1       6          1111␣
↪             -              -                         -64
     18:          10010          11110       2      -1       4          1110␣
↪             -              -                         -16
     19:          10011          11101       1      -1       3          110-␣
↪             1              -                          -8
     20:          10100          11100       1      -1       2          110-␣
↪             0              -                          -4
     21:          10101          11011       0      -1       1          10--␣
↪             1              1                          -3
     22:          10110          11010       0      -1       1          10--␣
↪             1              0                          -2
     23:          10111          11001       0      -1       0          10--␣
↪             0              1                        -1.5
     24:          11000          11000       0      -1       0          10--␣
↪             0              0                          -1
     25:          11001          10111      -1      -1      -1          01--␣
↪             1              1                       -0.75
     26:          11010          10110      -1      -1      -1          01--␣
↪             1              0                        -0.5
     27:          11011          10101      -1      -1      -2          01--␣
↪             0              1                      -0.375
     28:          11100          10100      -1      -1      -2          01--␣
↪             0              0                       -0.25
     29:          11101          10011      -2      -1      -3          001-␣
↪             1              -                      -0.125
     30:          11110          10010      -2      -1      -4          001-␣
↪             0              -                     -0.0625
     31:          11111          10001      -3      -1      -6          0001␣
↪             -              -                  -0.015625
```

```
    Generate Posit Lookup table for a POSIT<5,2>
       #            Binary         Decoded       k     sign    scale          regime      ␣
↪   exponent          fraction                           value
       0:             00000           00000      -4        1      -16               0000␣
↪              --               ~                                 0
       1:             00001           00001      -3        1      -12               0001␣
↪              --               ~                 0.000244140625
       2:             00010           00010      -2        1       -8               001-␣
↪              0-               ~                      0.00390625
       3:             00011           00011      -2        1       -6               001-␣
↪              1-               ~                       0.015625
       4:             00100           00100      -1        1       -4               01--␣
↪              00               ~                        0.0625
       5:             00101           00101      -1        1       -3               01--␣
↪              01               ~                         0.125
       6:             00110           00110      -1        1       -2               01--␣
↪              10               ~                          0.25
       7:             00111           00111      -1        1       -1               01--␣
↪              11               ~                           0.5
       8:             01000           01000       0        1        0               10--␣
↪              00               ~                             1
       9:             01001           01001       0        1        1               10--␣
↪              01               ~                             2
      10:             01010           01010       0        1        2               10--␣
↪              10               ~                             4
      11:             01011           01011       0        1        3               10--␣
↪              11               ~                             8
      12:             01100           01100       1        1        4               110-␣
↪              0-               ~                            16
      13:             01101           01101       1        1        6               110-␣
↪              1-               ~                            64
      14:             01110           01110       2        1        8               1110␣
↪              --               ~                           256
      15:             01111           01111       3        1       12               1111␣
↪              --               ~                          4096
      16:             10000           10000       4       -1       16               0000␣
↪              --               ~                           NaR
      17:             10001           11111       3       -1       12               1111␣
↪              --               ~                         -4096
      18:             10010           11110       2       -1        8               1110␣
↪              --               ~                          -256
      19:             10011           11101       1       -1        6               110-␣
↪              1-               ~                           -64
      20:             10100           11100       1       -1        4               110-␣
↪              0-               ~                           -16
      21:             10101           11011       0       -1        3               10--␣
↪              11               ~                            -8
      22:             10110           11010       0       -1        2               10--␣
↪              10               ~                            -4
      23:             10111           11001       0       -1        1               10--␣
↪              01               ~                            -2
      24:             11000           11000       0       -1        0               10--␣
↪              00               ~                            -1
      25:             11001           10111      -1       -1       -1               01--␣
↪              11               ~                          -0.5
      26:             11010           10110      -1       -1       -2               01--␣
↪              10               ~                         -0.25
```

(continues on next page)

(continued from previous page)

```
     27:           11011          10101      -1      -1      -3                01--␣
↪             01               ~                        -0.125
     28:           11100          10100      -1      -1      -4                01--␣
↪             00               ~                       -0.0625
     29:           11101          10011      -2      -1      -6                001-␣
↪             1-               ~                     -0.015625
     30:           11110          10010      -2      -1      -8                001-␣
↪             0-               ~                  -0.00390625
     31:           11111          10001      -3      -1     -12                0001␣
↪             --               ~               -0.000244140625
```

1.5.7 valid

valid is an interval arithmetic using posits and uncertainty bits.

1.5.8 unum Type 1

unum is a tapered, variable-length floating-point format with uncertainty interval semantics.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search