Table of Contents

uint64

The moonbitlang/core/uint64 package provides functionality for working with 64-b
it unsigned integers. This package includes constants, operators, and conversion
s for UInt64 values.

Constants

The package defines the minimum and maximum values for UInt64:

```

  test "UInt64 constants" {

    inspect(@uint64.min_value, content="0")


    inspect(@uint64.max_value, content="18446744073709551615")
  }
```

Arithmetic Operations

UInt64 supports standard arithmetic operations:

```

  test "UInt64 arithmetic" {
    let a : UInt64 = 100UL
    let b : UInt64 = 50UL


    inspect(a + b, content="150")


    inspect(a - b, content="50")


    inspect(a * b, content="5000")


    inspect(a / b, content="2")


    inspect(@uint64.max_value + 1UL, content="0")
    inspect(@uint64.min_value - 1UL, content="18446744073709551615")
  }
```

Bitwise Operations

UInt64 supports various bitwise operations:

```
1
2   test "UInt64 bitwise operations" {
3     let a : UInt64 = 0b1010UL
4     let b : UInt64 = 0b1100UL
5
6
7     inspect(a & b, content="8")
8
9
10    inspect(a | b, content="14")
11
12
13    inspect(a ^ b, content="6")
14
15
16    inspect(a << 1, content="20")
17    inspect(a << 2, content="40")
18
19
20    inspect(a >> 1, content="5")
21    inspect(b >> 2, content="3")
22  }
```

Comparison and Equality

UInt64 supports comparison and equality operations:

```
1
2   test "UInt64 comparison and equality" {
3     let a : UInt64 = 100UL
4     let b : UInt64 = 50UL
5     let c : UInt64 = 100UL
6
7
8     inspect(a == c, content="true")
9     inspect(a != b, content="true")
10
11
12    inspect(a > b, content="true")
13    inspect(b < a, content="true")
14    inspect(a >= c, content="true")
15    inspect(c <= a, content="true")
16  }
```

Byte Conversion

UInt64 provides methods for converting to bytes in both big-endian and little-endian formats:

```

  test "UInt64 byte conversion" {

    let be_bytes = 0x123456789ABCDEF0UL.to_be_bytes()
    inspect(
      be_bytes,
      content=(
        #|b"\x12\x34\x56\x78\x9a\xbc\xde\xf0"
      ),
    )


    let le_bytes = 0x123456789ABCDEF0UL.to_le_bytes()
    inspect(
      le_bytes,
      content=(
        #|b"\xf0\xde\xbc\x9a\x78\x56\x34\x12"
      ),
    )
  }
```

Default Value and Hashing

UInt64 implements the Default trait:

```

  test "UInt64 default value" {

    let a : UInt64 = 0UL
    inspect(a, content="0")


    let value : UInt64 = 42UL
    inspect(value.hash(), content="-1962516083")
  }
```

Type Conversions

UInt64 works with various conversions to and from other types:

```
1
2   test "UInt64 conversions" {
3
4     inspect((42).to_uint64(), content="42")
5
6
7     let value : UInt64 = 100UL
8     inspect(value.to_int(), content="100")
9     let as_double = value.to_double()
10    inspect(as_double, content="100")
11
12
13    inspect((-1).to_uint64(), content="18446744073709551615")
14
15
16    let from_double = 42.0.to_uint64()
17    inspect(from_double, content="42")
18  }
```

Working with Large Numbers

UInt64 is especially useful for applications requiring large unsigned integers:

```
1
2   test "UInt64 for large numbers" {
3
4     let large_number : UInt64 = (1UL << 63) - 1UL
5
6
7     inspect(large_number > (1UL << 32) - 1UL, content="true")
8
9
10    let result = large_number * 2UL
11    inspect(result, content="18446744073709551614")
12  }
```

Working with Hexadecimal Literals

UInt64 works well with hexadecimal literals for clarity when working with bit pa
tterns:

```
1
2   test "UInt64 hexadecimal literals" {
3
4     let value = 0xDEADBEEFUL
5
6
7     let ad = (value >> 16) & 0xFFUL
8     inspect(ad.to_byte(), content="b'\\xAD'")
9
10
11    let bytes = value.to_be_bytes()
12    inspect(
13      bytes,
14      content=(
15        #|b"\x00\x00\x00\x00\xde\xad\xbe\xef"
16      ),
17    )
18  }
```