Table of Contents

uint64

The moonbitlang/core/uint64 package provides functionality for working with 64-bit unsigned integers. This package includes constants, operators, and conversions 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:

```

test "UInt64 bitwise operations" {
  let a : UInt64 = 0b1010UL
  let b : UInt64 = 0b1100UL


  inspect(a & b, content="8")


  inspect(a | b, content="14")


  inspect(a ^ b, content="6")


  inspect(a << 1, content="20")
  inspect(a << 2, content="40")


  inspect(a >> 1, content="5")
  inspect(b >> 2, content="3")
}
```

Comparison and Equality

UInt64 supports comparison and equality operations:

```

test "UInt64 comparison and equality" {
  let a : UInt64 = 100UL
  let b : UInt64 = 50UL
  let c : UInt64 = 100UL


  inspect(a == c, content="true")
  inspect(a != b, content="true")


  inspect(a > b, content="true")
  inspect(b < a, content="true")
  inspect(a >= c, content="true")
  inspect(c <= a, content="true")
}
```

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:

```

test "UInt64 conversions" {

  inspect((42) to_uint64(), content="42")


  let value : UInt64 = 100UL
  inspect(value to_int(), content="100")
  let as_double = value to_double()
  inspect(as_double, content="100")


  inspect((-1) to_uint64(), content="18446744073709551615")


  let from_double = 42.0 to_uint64()
  inspect(from_double, content="42")
}
```

Working with Large Numbers

UInt64 is especially useful for applications requiring large unsigned integers:

```

test "UInt64 for large numbers" {

  let large_number : UInt64 = (1UL << 63) - 1UL


  inspect(large_number > (1UL << 32) - 1UL, content="true")


  let result = large_number * 2UL
  inspect(result, content="18446744073709551614")
}
```

Working with Hexadecimal Literals

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

```

test "UInt64 hexadecimal literals" {

  let value = 0xDEADBEEFUL


  let ad = (value >> 16) & 0xFFUL
  inspect(ad to_byte(), content="b’\\xAD’")


  let bytes = value to_be_bytes()
  inspect(
    bytes,
    content=(
      #|b"\x00\x00\x00\x00\xde\xad\xbe\xef"
    ),
  )
}
```