```
   btd =: monad define
  digits =. # y.
   if. 1 = 1 take y.
    do. ((digits copy 2) base x: y. ) - 2^ x: digits
    else. (digits copy 2) base x: y.
   end.
)
   0 7 rep 23
3 2
   alu_64 =: monad define
NB. a and b are each 64-bit summands
NB. the result is a 64-bit sum
('a' ; 'b' ; 'sub') =. y.
if. sub
  do. (64#2) rep (base x: a) - base x: b
  else. (64#2) rep (base x: a) + base x: b
end.
)
   a =: (64#2)rep _2
   a
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
   b =: (64#2)rep 2
   b
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
   [a =: (64#2)rep _2
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
   [c=:(64#2)rep _1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
   alu_64 a;b;0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   alu_64 a;b;1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
   0 7 rep _23
_4 5
   0 _7 rep 23
_4 _5
   div1 =: monad define
('dividend' ; 'divisor') =. y.
divisor =. divisor, 32#0
quotient =. 32#0
remainder=: dividend
count=. 33
while. 0 not_equal count
  do. remainder =: alu_64 remainder ; divisor ; 1
      control =. first remainder
      if. control
        do. remainder =: alu_64 remainder ; divisor ; 0
            quotient =. (1 drop quotient) , 0
        else. quotient =. (1 drop quotient) , 1
        end.
      divisor =. 0 , _1 drop divisor
      count =. <: count
```

```
  end.
quotient
)
   div1 ((62#0), 1 1) ; (30#0) , 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
   remainder
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
   div1 ((61#0), 1 1 1) ; (30#0) , 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
   remainder
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
   div1 ((61#0),1 1 0) ; (30#0) , 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
   remainder
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   alu_32 =: monad define
NB. a and b are each 32-bit summands
NB. the result is a 32-bit sum
NB. sub = 0 for addition
NB. sub = 1 for subtraction
('a' ; 'b' ; 'sub') =. y.
if. sub
  do. (32#2) rep (base x: a) - base x: b
  else. (32#2) rep (base x: a) + base x: b
end.
)
   div2 =: monad define
('dividend' ; 'divisor') =. y.
quotient =. 32#0
remainder=: (32#0) , dividend
count=. 32
remainder =: (1 drop remainder) , 0
while. 0 not_equal count
      do. remainder =: (alu_32 (32 take remainder) ; divisor ; 1) , 32 drop remainder
      if. control =. first remainder
        do. remainder =: (alu_32 (32 take remainder) ; divisor ; 0) , 32 drop remainder
            quotient =. (1 drop quotient) , 0
        else. quotient =. (1 drop quotient) , 1
        end.
      remainder =: (1 drop remainder) , 0
      count =. <: count
  end.
remainder =: (0 , _1 drop 32 take remainder) , 32 drop remainder
quotient
)
   div2 ((30#0), 1 1) ; (30#0) , 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
   remainder
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   _+1
_
   -_
```

```
__
   _¯_
_.
   _. + 2
_.
   _ = _
1
   __ = _
0
   _. = _.
1
```