```
functor MkBigNumAdd(structure U : BIGNUM_UTIL) : BIGNUM_ADD =
struct
  structure Util = U
  open Util
  open Seq

  infix 6 ++

  local
    datatype carry = GEN | PROP | STOP

    (* Initial carry bits *)
    fun init (ONE, ONE) = GEN
      | init (ZERO, ZERO) = STOP
      | init _ = PROP

    (* Scan binary operator to propagate carries *)
    fun propagate (x, PROP) = x
      | propagate (_, y) = y

    (* Conversions from carry to bit *)
    val G1 = fn GEN => ONE | _ => ZERO
    val P1 = fn PROP => ONE | _ => ZERO

    infix 6 :+: (* XOR *)
    fun x :+: y = (P1 o init) (x, y)

    fun nth' s i = (nth s i) handle Range => ZERO
  in
    fun x ++ y =
        let
          (* Normalize the lengths *)
          val N = Int.max (length x, length y)
          val (x, y) = (tabulate (nth' x) N, tabulate (nth' y) N)

          (* Generate and propagate carry bits with scan *)
          val (carries, last) = scan propagate STOP (map2 init x y)

          (* Do addition with final carry state *)
          fun result i =
              let fun ith s = nth s i
              in (ith x) :+: (ith y) :+: (G1 (ith carries))
              end handle Range => ONE

          (* Add a bit if the last one carried *)
        in tabulate result (if last = GEN then N+1 else N)
        end
  end

  val add = op++
end
```

```
functor MkBigNumSubtract(structure BNA : BIGNUM_ADD) : BIGNUM_SUBTRACT =
struct
  structure Util = BNA.Util
  open Util
  open Seq

  infix 6 ++ --
  fun x ++ y = BNA.add (x, y)

  local
    val flip = fn ONE => ZERO | ZERO => ONE

    (* removes trailing ZEROs from s *)
    fun trim s =
        let
          val flag = fn (i, ONE) => i+1 | _ => 0
          val toTake = reduce Int.max 0 (mapIdx flag s)
        in take (s, toTake)
        end

    fun nth' s i = (nth s i) handle Range => ZERO
  in
    fun x -- y =
        let
          val n = Int.max (length x, length y) + 1
          (* negates y by flipping all bits and adding ONE *)
          val negy = tabulate (flip o (nth' y)) n
          val result = x ++ negy ++ %[ONE]
        in trim (take (result, length result - 1))
        end
  end

  val sub = op--
end
```

```
functor MkBigNumMultiply(structure BNA : BIGNUM_ADD
                         structure BNS : BIGNUM_SUBTRACT
                         sharing BNA.Util = BNS.Util) : BIGNUM_MULTIPLY =
struct
  structure Util = BNA.Util
  open Util
  open Seq

  infix 6 ++ --
  fun x ++ y = BNA.add (x, y)
  fun x -- y = BNS.sub (x, y)

  infix 7 **

  val test = Util.fromIntInf o Util.toIntInf

  local
    (* removes trailing ZEROs from s *)
    fun trim s =
        let
          val flag = fn (i, ONE) => i+1 | _ => 0
          val toTake = reduce Int.max 0 (mapIdx flag s)
        in take (s, toTake)
        end

    fun nth' s i = (nth s i) handle Range => ZERO

    infix 6 << (* bitshift left : multiply by 2^n *)
    fun s << n = trim (tabulate (fn x => nth' s (x - n)) (length s + n))
  in
    fun x ** y =
        case (length x, length y)
          of ((0, _) | (_, 0)) => empty ()
           | (1, _) => y
           | (_, 1) => x
           | (lx, ly) =>
             let
               (* Normalize the lengths *)
               val N = Int.max (lx, ly)
               val (x', y') = (tabulate (nth' x) N, tabulate (nth' y) N)

               val (NODE (q, p), NODE (s, r)) = (showt x', showt y')
               val [p, q, r, s] = List.map trim [p, q, r, s]
               val (x, y) = (p ++ q, r ++ s)

               val (pr, xy, qs) = Primitives.par3 (fn () => p ** r,
                                                   fn () => x ** y,
                                                   fn () => q ** s)
               val offset = N div 2
               val mid = xy -- pr -- qs
             in (pr << offset * 2) ++ (mid << offset) ++ qs
             end
  end

  val mul = op**
end
```