

**课程实验报告**

**课程名称：串行与并行数据结构及算法实验**

**专业班级： ACM1701**

**学号： U201714850**

**姓名： 丁文隆**

**指导教师： 陆枫**

**报告日期： 2018.11.28**

**计算机科学与技术学院**

## Lab3bignumlab

### 1. 实验要求

实现n位二进制大整数的加法运算。输入a, b和输出s都是二进制位的串。要求算法的时间复杂度满足work=O(n)，span=O(log n)。

### 2. 实验思路

|  |
| --- |
| 2.1．加法求解思路  1.我们对于两个bignum，我们对于每一位产生一个pairs，pairs内容包含产生的本位值和进位值。例如pairs (ONE,ONE) = (ZERO,GEN)。  2.我们对于进位值进行scan操作确定到底有哪一位产生了进位。  3.将进位值与本位值进行合并，即若本位值碰到GEN即翻转，1变为0, 0变为1，其他情况保持不变。  4.最后去掉高位多余的0  2.2．减法求解思路  1.我们把y的值翻转加一，再加上x的值。  2.去掉高于x长度的位数，得到减法所得的的数  2.3.乘法求解思路  1．首先通过补充0，将两个bignum变得等长且为偶数长度  2．若是两个空串，则返回空串；若补充之后两个串的长度为1，则判定若两个数都是ONE，则为ONE，否则为空串。  3.对于其他情况，我们将串x分成p、q相同长度的两份，将串y分成r、s两份，按照公式A·B = pr·2n +(ps+rq)·2n/2 +qs求得结果，其中ps+rq =(p+q)∗(r +s)−pr−qs求解。（其中我们并行计算(p+q)∗(r+s),p∗r,q∗s这三个式子）  （具体实现参照下面代码） |

### 3. 回答问题

#### 3.1提供加法计算的代码和注释

Task 4.1 (35%). Implement the addition function

++ : bignum \* bignum -> bignum

in the functor MkBigNumAdd in MkBigNumAdd.sml. For full credit, on input with *m* and *n* bits, yoursolution must have *O*(*m*+*n*) work and *O*(lg(*m*+*n*)) span. Our solution has under 40 lines with comments.

|  |
| --- |
| functor MkBigNumAdd(structure U : BIGNUM\_UTIL) : BIGNUM\_ADD =  struct  structure Util = U  open Util  open Seq  infix 6 ++  exception NotYetImplemented  datatype carry = GEN | PROP | STOP  fun x ++ y =  let  val min = Int.min(length x, length y)  val max = Int.max(length x, length y)  (\*定义一个pairs，模拟二进制数每一位的加法，得到的结果也是一个数对，第一个是本位的结果，第二个判断是否进位\*)  fun pairs (ZERO,ZERO) = (ZERO,STOP)  | pairs (ONE,ONE) = (ZERO,GEN)  | pairs (ZERO,ONE) = (ONE,PROP)  | pairs (ONE,ZERO) = (ONE,PROP)  (\*定义first和second函数，分别用于取pairs数对的第一个和第二个，用于后续分成两部分计算本位值和进位值\*)  fun first (i,\_) = i  fun second (\_,i) = i  (\*定义addbits函数，遍历两个bignum，使得加法的每一位都形成一个pairs，较小的数高位用ZERO补齐\*)  fun addbits a =  if(a < min) then pairs ((nth x a),(nth y a))  else if (a<length x) then pairs ((nth x a),ZERO)  else if (a<length y) then pairs (ZERO,(nth y a))  else (ZERO,STOP)  (\*计算x,y经过addbits的结果\*)  val addresult0 = tabulate addbits max  (\*下面我们首先来计算进位，判断整个加法完成时候到底有哪些位置会产生进位  首先定义一个carrystep判断进位关系的产生、传递和停止关系  然后carryseq取用每一位单独的进位结果，即addresult0每一项的第二位进行map  然后执行scan操作，判断carry之后到底哪些位产生了真实进位  最后很关键的一步，由于产生进位是从第二位开始的，是对下一位产生进位，所以进位关系是对于下一位而言，这里要在最低位补充一个PROP或者STOP  (代码中最低位补充的是PROP)  \*)  fun carrystep (a,PROP) = a  | carrystep (\_,GEN) = GEN  | carrystep (\_,STOP) = STOP  val carryseq = map second addresult0  val carryresult0 = scani carrystep PROP carryseq  val carryresult = append(singleton(PROP), carryresult0)  (\*下面来计算每一位本位产生的值  首先先取addresult0的每一项的第一个，即为本位值，同时为防止最高位向下一位进位，这里在给最高位留出一个位子，用ZERO补充  \*)  val addresult1 = map first addresult0  val addresult2 = append(addresult1, singleton(ZERO))  (\*下面将本位和进位加起来，  定义一个反转函数，如果碰到进位就调用这个函数，ZERO变为ONE，ONE变为ZERO(因为这一位多加了一个ONE)  然后是主要的addtwoparts函数，对于本位的每一个数字，如果相应的碰到GEN，就产生位翻转，否则不变  最后对每一位使用这个函数，用tabulate并行实现  \*)  fun reverse ZERO = ONE  | reverse ONE = ZERO  fun addtwoparts i =  if (i >= length addresult2) then ZERO  else if (nth carryresult i = GEN) then reverse (nth addresult2 i)  else (nth addresult2 i)    val result = tabulate addtwoparts (length addresult2)  in  (\*最后判断最高位是否为ONE，若不为ONE，则为ZERO，去掉多余的零\*)  if(nth result ((length result)-1) = ONE) then result  else subseq result (0, length result - 1)  end    val add = op++  end |

#### 3.2提供减法计算的代码和注释

Task 4.2 (15%). Implement the subtraction function

-- : bignum \* bignum -> bignum

in the functor MkBigNumSubtract in MkBigNumSubtract.sml, where *x* -- *y* computes the number

obtained by subtracting *y* from *x*. We will assume that *x*≥*y*; that is, the resulting number will alwaysbe non-negative. You should also assume for this problem that ++ has been implemented correctly. Forfull credit, if *x* has *n* bits, your solution must have *O*(*n*) work and *O*(lg*n*) span. Our solution has fewerthan 20 lines with comments.

|  |
| --- |
| functor MkBigNumSubtract(structure BNA : BIGNUM\_ADD) : BIGNUM\_SUBTRACT =  struct  structure Util = BNA.Util  open Util  open Seq  exception NotYetImplemented  infix 6 ++ --  fun x ++ y = BNA.add (x, y)  fun x -- y =  let  (\*编写反转函数，将串y中的元素0变成1,1变成0，其他位用0补齐\*)  fun reverse y i =  if i < 0 orelse i >= length y orelse nth y i = ZERO  then ONE  else ZERO  (\*将y的值翻转加一得到负数的值，用x加上y的负数，然后舍弃最高位得到减法的值\*)  val revadd = (singleton ONE) ++ (tabulate (reverse y) (length x)) ++ x  val result0 = take (revadd , length x)  in  result0  end    val sub = op--  end |

#### 3.3提供乘法计算的代码和注释

Task 4.3 (30%). Implement the function

\*\* : bignum \* bignum -> bignum

in MkBigNumMultiply.sml. For full credit, if the larger number has n bits, your solution must satisfy*W∗∗*(*n*) = 3 · *W*∗∗(*n*/2)+ O(*n*)and have O (lg2*n* ) span. You should use the following function in the Primitives structure:

val par3 : (unit -> ’a) \* (unit -> ’b) \* (unit -> ’c) -> ’a \* ’b \* ’c

to indicate three-way parallelism in your implementation of \*\*. You should assume for this problem that++ and -- have been implemented correctly, and meet their work and span requirements. Our solution has 40 lines with comments.

|  |
| --- |
| 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  open Primitives  exception NotYetImplemented  infix 6 ++ --  infix 7 \*\*  fun x ++ y = BNA.add (x, y)  fun x -- y = BNS.sub (x, y)  fun x \*\* y =  let  (\*我们的思路是先把x,y变得同样长且长度为偶数，方便后续divide  不够的位数我们用0补，定义addzero往高位补n个0  makeaqual函数把两个bignum变得一样长，即若谁短，谁补差值个0  makeiteven函数即在两个穿长度相同情况下，若长度为奇数，则高位加0补为偶数  \*)  fun addzero (x,n) = append(x,(tabulate (fn i => ZERO) n))  fun makeequal (x,y) =  case Int.compare(length x, length y) of  LESS =>((addzero(x,length y - length x)),y)  | GREATER => (x,addzero(y,length x - length y))  | \_ => (x,y)  fun makeiteven (x,y) =  case(length x) mod 2 of  0 => (x,y)  | 1 => (addzero(x,1),addzero(y,1))  (\*定义pow2意思为x乘以2的n次方，实际上就是往左移动n位，低位用0补齐\*)  fun pow2 (x,n) =  case length x of  0 => empty()  | \_ => append((tabulate(fn i => ZERO) n), x)  in  let  (\*先把两个串变得一样长，若长度是0，就是空，若长度是1且两个都是ONE，证明为ONE，否则为ZERO  其余的，执行divide操作，先把串长n变成偶数，定义p、q、r、s分别为x1的高n/2位，低n/2位，x2的高n/2位，低n/2位  并行计算p \*\* r，q \*\* s，(p ++ q) \*\* (r ++ s)  计算ps+rq=(p ++ q)(r ++ s)-pr-qs  根据公式计AB=pr\*2^n+(ps+qr)\*2^(n/2)+qs算出两大数乘法\*)  val (x1,x2) = makeequal(x,y)  in  case length x1 of  0 => empty()  | 1=> (case (nth x1 0, nth x2 0) of  (ONE, ONE) => singleton(ONE)  | \_ => empty())  | \_ => let  val (x0,y0) = makeiteven(x1,x2)  val n = length x0  val p = drop(x0, n div 2)  val q = take(x0, n div 2)  val r = drop(y0, n div 2)  val s = take(y0, n div 2)  val (pr,qs,pqrs) = par3(fn() => p \*\* r, fn() => q \*\* s, fn() => ((p ++ q) \*\* (r ++ s)))  val psrq = pqrs -- pr --qs  val AB = pow2(pr, n) ++ pow2(psrq, n div 2) ++ qs  in  AB  end  end  end  val mul = op\*\*  end |

#### 3.4迭代计算复杂度分析

Task 5.1 (15%). Determine the complexity of the following recurrences. Give tight *Θ*-bounds, and

justify your steps to argue that your bound is correct. Recall that *f*∈*Θ*(*g*) if and only if *f*∈*O*(*g*) and*g*∈*O*(*f*). You may use any method (brick method, tree method, or substitution) to show that your bound is correct, except that you must use the substitution method for problem 3.

1. T (*n*) = 3T(*n*/2) + *Θ*(*n*)

2. T (*n*) = 2T(*n*/4) +*Θ*()

3. T (*n*) = 4T(*n*/4) +*Θ*() (Prove by substitution.

|  |
| --- |
| 1. T (*n*) = *Θ*()，证明如下：  首先，明显这棵树的树高是，每一层的值为，即我们有，这棵树的总和为：  2. , 证明如下：  首先，这棵数的树高是，每一层的值为，即我们有，这和棵树的额总和为：    3.，证明如下：  要证明.，即证明存在使得。令  对于base case：i=0，有，即即可  对于induction：i=k时首先若存在，使得  i=k+1时，有  即  即  即，存在使得等式成立。  根据上述论述，有.成立 |