# 基数排序算法

## 算法

核心思想：利用除法和取余即可得到某个数值的某一位。基数排序就是根据数字的位（个位、十位、百位…）来进行的排序算法，有点类似于哈希表，基数排序首先应该找到数字中最大的数，然后算出这个数有几位，其次依次根据个位、十位、百位、千位、…、最大位来进行排序

假设对以下8个数进行基数排序：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| arr | 33 | 58 | 999 | 0 | 233 | 54 | 133 | 7 |

首先对其个位排序，将其按照个位大小放入对应的桶中：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 桶号 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 元素 | 0 |  |  | 33 | 54 |  |  | 7 | 58 | 999 |
| 元素 |  |  |  | 233 |  |  |  |  |  |  |
| 元素 |  |  |  | 133 |  |  |  |  |  |  |

按照从左到右，从上到下的顺序以此将桶中的元素放回到arr中：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| arr | 0 | 33 | 233 | 133 | 54 | 7 | 58 | 999 |

对个位的排序结束。

接下来对其十位进行排序，将其按照十位大小依次放入对应的桶中：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 桶号 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 元素 | 0 |  |  | 33 |  | 54 |  |  |  | 999 |
| 元素 | 7 |  |  | 233 |  | 58 |  |  |  |  |
| 元素 |  |  |  | 133 |  |  |  |  |  |  |

按照从左到右，从上到下的顺序以此将桶中的元素放回到arr中：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| arr | 0 | 7 | 33 | 233 | 133 | 54 | 58 | 999 |

对十位的排序结束。

接下来对其百位进行排序，将其按照百位大小依次放入对应的桶中：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 桶号 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 元素 | 0 | 133 | 233 |  |  |  |  |  |  | 999 |
| 元素 | 7 |  |  |  |  |  |  |  |  |  |
| 元素 | 33 |  |  |  |  |  |  |  |  |  |
| 元素 | 54 |  |  |  |  |  |  |  |  |  |
| 元素 | 58 |  |  |  |  |  |  |  |  |  |

最后，按照从左到右，从上到下的顺序以此将桶中的元素放回到arr中：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| arr | 0 | 7 | 33 | 54 | 58 | 133 | 233 | 999 |

至此，排序结束。

总结：由上述过程可知，基数排序就是重复进行“置入”与“取出”的过程，其整体“置入”与“取出”的次数只与待排序数据的最大值有关。为什么先对“个位”进行排序，而不是我们通常比较两个数时先比较最高位？因为最后排序的位数对结果的影响最大，在此算法中先排最低位后排最高位，即最低位对最终结果影响最小最高位对最终结果影响最大，符合客观逻辑。

## C语言设计

|  |
| --- |
| #include"stdio.h"  #define length 8  // 汇编搭配，半字=两字节=16位  // 为了方便向汇编移植， barrel没用二维数组  //int arr[length] = {3,8,1,5,2,4,6,7}; // 待排序数组，i，arr[i]对应的某一位是j  int arr[length] = {33,58,999,0,233,54,133,7}; // 待排序数组，i，arr[i]对应的某一位是j  int record[10] = {0}; // 桶内元素个数记录，j，record[j]是k  int barrel[length\*10] = {0}; // 桶，k，barrel[j\*len+k]是桶中的元素  void Radix\_Sort(int \*arr, int \*record, int \*barrel, int len, int max\_scale)  {  int i,j,k;  int min\_scale = 1;  while(min\_scale<max\_scale) // 从低位向高位递进  {  int mid\_scale = min\_scale \* 10;  for(i=0;i<len;i++) // 遍历待排序数组  {  j = arr[i] % mid\_scale; // 确定某个待排序数arr[i]的某一位j  j = j / min\_scale;  k = record[j]; // 将待排序数arr[i]放入对应的桶中  barrel[j\*len+k] = arr[i];    record[j] ++ ; // 记录对应桶中元素数量+1  }  j = 0; // 记录点复位  for(i=0;i<len;) // 遍历空位  {  for(;record[j] ==0;j++) ; // 检测哪个桶非空  for(k=0;k<record[j];k++) // 从桶中取出放入空位  {  arr[i] = barrel[j\*len+k];  i++;  }  record[j] =0 ; // 取完一个桶则清空记录  }  min\_scale = mid\_scale;  }  }  int main()  {  int i;  // 显示排序前的序列  for(i=0;i<=7;i++)  printf("%d ",arr[i]);  printf("\n");  // 排序  Radix\_Sort(arr,record,barrel,8,1000);  // 显示排序后的序列  for(i=0;i<=7;i++)  printf("%d ",arr[i]);  printf("\n");  } |

## 汇编语言设计

用LoongIDE新建一个工程。

移除core文件夹下的bsp\_start.c源文件。

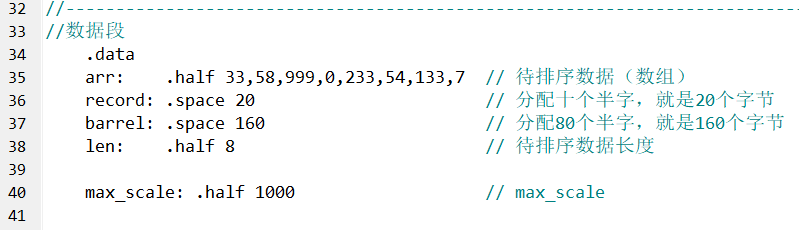
移除main.c源文件。

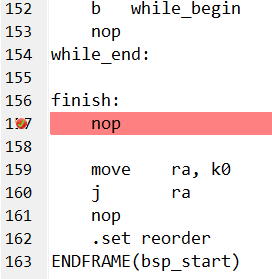
添加bsp\_start.S源文件。

在bsp\_start.S下添加如下代码：

|  |
| --- |
| /\*  \* bsp\_start.S  \*  \* created: 2022/1/28  \* author: Li TianLing  \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* v0 存储器寻址辅助寄存器  \* v1 跳转辅助寄存器  \* s0 参数len（也是length） （常量）  \* s1 待排序数据arr基地址 （常量）  \* s2 桶内元素个数记录数组record基地址（常量）  \* s3 桶barrel基地址 （常量）  \* s4 参数max\_scale （常量）  \* s5 参数mid\_scale  \* s6 参数min\_scale  \* t1 哨兵i  \* t2 哨兵j  \* t3 哨兵k  \* t4 待排序数组遍历arr[i]  \* t5 某桶内元素记录record[j]  \* t6 桶barrel[j\*len+k]  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "regdef.h"  #include "cpu.h"  #include "asm.h"  //-----------------------------------------------------------------------------  //数据段  .data  arr: .half 33,58,999,0,233,54,133,7 // 待排序数据（数组）  record: .space 20 // 分配十个半字，就是20个字节  barrel: .space 160 // 分配80个半字，就是160个字节  len: .half 8 // 待排序数据长度  max\_scale: .half 1000 // max\_scale  //-----------------------------------------------------------------------------  //代码段  .text  FRAME(bsp\_start,sp,0,ra)  .set noreorder  move k0, ra /\* 返回地址 \*/  la v0,len // 将长度len加载到s0（常量）  lh s0,(v0)  sll s0,1  la s1,arr // 将arr基地址加载到s1（常量）  la s2,record // 将record基地址加载到s2（常量）  la s3,barrel // 将barrel基地址加载到s3（常量）  la v0,max\_scale // 将max\_scale加载到s4（常量）  lh s4,(v0)  la s6,1 // 将min\_scale赋值为1  while\_begin:  sub v1,s6,s4 // min\_scale>=max\_scale就结束循环  bgez v1,while\_end  nop  mul s5,s6,10 // mid\_scale初始化为min\_scale\*10  la t1,0 // 哨兵i初始化为0  for\_pre\_arr\_begin:  sub v1,t1,s0  bgez v1,for\_pre\_arr\_end  nop    add v0,s1,t1 // 将arr[i]的值加载到t4中  lh t4,(v0)  div t4,s5 // 将arr[i]/mid\_scale的余数赋值给哨兵j  mfhi t2  div t2,s6 // 将哨兵j/min\_scale的商赋值给哨兵j  mflo t2  mul t2,2 // 哨兵j翻倍，【注】j是余数不是地址要翻倍    add v0,s2,t2 // 将record[j]的值加载到t5中  lh t5,(v0)  move t3,t5 // 将record[j]赋值给哨兵k，【注】recor[j]本来就是偶数地址，不用翻倍  add v0,s1,t1 // 将arr[i]的值加载到t4中  lh t4,(v0)  move t6,t4 // 将arr[i]赋值给t6  mul v0,t2,s0 // 将t6的值保存到barrel[j\*len+k]  srl v0,1 // 【注】j和len乘积是四倍，所以要右移一位除以二  add v0,s3  add v0,t3  sh t6,(v0)    add t5,2 // record[j]+2  add v0,s2,t2 // 保存  sh t5,(v0)  add t1,2 // 哨兵i+2  b for\_pre\_arr\_begin  nop  for\_pre\_arr\_end:    la t2,0 // 哨兵j复位  la t1,0 // 哨兵i复位    for\_after\_arr\_begin:  sub v1,t1,s0  bgez v1,for\_after\_arr\_end  nop    for\_record\_begin:  add v0,s2,t2 // 将record[j]加载到t5  lh t5,(v0)  bne t5,zero,for\_record\_end // 非空就结束  nop    add t2,2 // 哨兵j+2  b for\_record\_begin  nop  for\_record\_end:    la t3,0 // 哨兵k初始化为0  for\_barrel\_begin:  // record[j]不用重新加载  sub v1,t3,t5 // 哨兵k>=record[j]结束  bgez v1,for\_barrel\_end  nop    mul v0,t2,s0 // barrel[j\*len+k]加载到t6  srl v0,1 // 【注】j和len乘积是四倍，所以要右移一位除以二 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*srl  add v0,s3  add v0,t3  lh t6,(v0)  move t4,t6 // 将t6赋值给t4  add v0,s1,t1 // 将t4保存到arr[i]  sh t4,(v0)    add t1,2 // 哨兵i+2  add t3,2 // 哨兵k+2  b for\_barrel\_begin  nop  for\_barrel\_end:    la t5,0 // 将record[j]清除  add v0,s2,t2  sh t5,(v0)    b for\_after\_arr\_begin  nop  for\_after\_arr\_end:    move s6,s5 // min\_scale = mid\_scale  b while\_begin  nop  while\_end:  finish:  nop  move ra, k0  j ra  nop  .set reorder  ENDFRAME(bsp\_start) |

## 调试运行

初始数据如下所示：

如下图所示，在finish后打上断点：

联合调试后，双击s1寄存器，查看对应存储器数据：

如上图所示，排序成功。