ECE 154A Fall 2012
University of California, Santa Barbara

ECE 154A Introduction to Computer Architecture

Project #1: Radix Sort

Due at 11:00 PM November 2nd, 2012

1. Introduction

In this project you are asked to implement radix sort using MIPS assembly code. In particular we will consider a particular flavor of such algorithm – using the least significant digit radix sort with counting pass on each digit. Note that this project involves understanding of arrays and pointers. These concepts are introduced in the context of data flow instructions. Another, more elegant implementation of radix sort is based on linked lists which will be covered later in class.

Suppose that initially you are given array (“array”) with unsorted unsigned numbers (keys), e.g. 4-bit number array shown on Figure 1. During the first pass, keys in the array are grouped based on the least significant digit, e.g. for radix 2 keys are grouped by looking only at last bit, for radix 4 looking at 2 bits etc. During the second pass keys are grouped again using the last but one least significant digit (i.e. next bit for radix 2), but otherwise keeping the original order of keys. This operation is continued until keys are grouped for all digits with the final result being the sorted array.

To implement sorting for any single step on figures 1 and 2 counting sort is performed. In the first phase the number of occurrences for each digit at the corresponding position is counted and stored in the dedicated array “count”. Then using count array the absolute position for each group of digits within array (“abspos”) is calculated. Finally, in the last phase, keys are placed into temporary array (“arraytemp”) at the locations specified by corresponding values of abspos with the values of abspos being incremented after each use. For example, let’s assume that first step of radix-2 sorting of 4-bit numbers is being performed and the key value to be sorted is “1001”. The least significant digit is “1” so that program looks into abspos[1] value. Let’s assume that current value of abspos[1] = 4. So the program will execute arraytemp[4] = “1001” and then increment abspos[1]=abspos[1]+1. The full C code of the algorithm is presented below.

2. Project Tasks

Your code should be written as a procedure which works for different radixes and different word width. Assume that the size of the array and the base are passed in registers $a0, and $a1, correspondingly, while the logarithm base 2 of the radix is in $a2 and the word width in $a3. Your code has to preserve (spill) registers if necessary (according to the standard rules for procedures) but not the initial array. The sorted array should replace the initial one. You may create your own test array to test your code or use skeleton code.

3. Turn-in

Once your code is completed and works correctly place your procedure inside skeleton code “sort.s” (which can be downloaded from the class website), name your file as lastname_sort.s (e.g. Britney Spears would have to name her files as spears_sort.s) and mail it to: ece154fall2012@gmail.com.
<table>
<thead>
<tr>
<th>Initial array</th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>array[0]</td>
<td>0011</td>
<td>1110</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>array[1]</td>
<td>0101</td>
<td>0000</td>
<td>0101</td>
<td>1001</td>
</tr>
<tr>
<td>array[2]</td>
<td>1001</td>
<td>0010</td>
<td>1001</td>
<td>0010</td>
</tr>
<tr>
<td>array[3]</td>
<td>1110</td>
<td>0011</td>
<td>1110</td>
<td>0011</td>
</tr>
<tr>
<td>array[4]</td>
<td>0011</td>
<td>0101</td>
<td>0010</td>
<td>0101</td>
</tr>
<tr>
<td>array[5]</td>
<td>0000</td>
<td>1001</td>
<td>0011</td>
<td>1001</td>
</tr>
<tr>
<td>array[6]</td>
<td>0010</td>
<td>0111</td>
<td>0011</td>
<td>1110</td>
</tr>
</tbody>
</table>

**phase 1:**
- count[0] = 3
- count[1] = 4
- count[2] = 2
- count[3] = 2

**phase 2:**
- abspos[0] = 0
- abspos[1] = 3
- abspos[2] = 2
- abspos[3] = 5

Figure 1. Example of least significant digit radix-2 sort with counting passes for 4-bit numbers.

<table>
<thead>
<tr>
<th>Initial array</th>
<th>Step 1</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>array[0]</td>
<td>0011</td>
<td>0000</td>
</tr>
<tr>
<td>array[1]</td>
<td>0101</td>
<td>0101</td>
</tr>
<tr>
<td>array[2]</td>
<td>1001</td>
<td>1001</td>
</tr>
<tr>
<td>array[3]</td>
<td>1110</td>
<td>1110</td>
</tr>
<tr>
<td>array[4]</td>
<td>0011</td>
<td>0010</td>
</tr>
<tr>
<td>array[5]</td>
<td>0000</td>
<td>0011</td>
</tr>
<tr>
<td>array[6]</td>
<td>0010</td>
<td>0011</td>
</tr>
</tbody>
</table>

**phase 1:**
- count[0] = 1
- count[1] = 2
- count[2] = 2
- count[3] = 2

**phase 2:**
- abspos[0] = 0
- abspos[1] = 1
- abspos[2] = 3
- abspos[3] = 5

Figure 2. Example of least significant digit radix-4 sort with counting passes for 4-bit numbers.
4. Radix sort C pseudocode example

```
#define N 1000;       // the length of the array
#define logradix 1;  
#define wordwidth 32; // the number of bits for each element in the array

unsigned int array[N];
unsigned int arraytemp[N];
int count[2^logradix];
int abspos[2^logradix];

int i, j, k;
radixmask = 2^logradix -1;

for(j=0; j<wordwidth; j=j+logradix) {
   /* phase 1: counting */
   for (k=0; k<2^logradix; k++) count[k] = 0;         /* initialize values of count array to zero */
   for (i=0; i < N; i++) count[(array[i]>>j)&radixmask] ++;  /* count number of times each digit occur */

   /* phase 2: calculating absolute position for each digit */
   abspos[0] = 0;                                       /* initialize the first element of abspos array to zero */
   for (k=0; k<2^logradix-1; k++) abspos[k+1]=abspos[k]+count[k];  /* calculate absolute position */

   /* phase 3: sorting phase */
   for (i=0; i < N; i++) {
      curradix = (array[i]>>j)&radixmask;
      arraytemp[abspos[curradix]] = array[i];    /* place into arraytemp to the corresponding position */
      abspos[curradix]++;
   }

   /* copy arraytemp to array (this step could be avoided but added for simplicity) */
   for (i=0; i < N; i++) array[i] = arraytemp[i];
}
```