Lab 5: Bubble Sort

黃永廣
助教：劉柏廷、郭柏興、林嘉偉、陳昶宏
雲科電子
2014/12/22~26

Download New .sof File for DE0

- A new LC3_data_path.sof is placed in the lab folder
- Reminder: address for LEDs is 0xFFFD
- Encoding for HEX10 (address 0xFFFF) and Hex32 (address 0xFFFE) is already done by the Verilog code of the soft core LC3
- AND R0,R0,#0
- ADD R1,R0,#15
- STR R1,R0,#-1 ; show 0F on Hex10
- STR R1,R0,#-2 ; show 0F on Hex32

Check HEXs, LEDs with This Program

```
        .ORIG x3000
        AND R0,R0,#0
        LD  R1,n255
loop    STR R1,R0,#-1  ; write to HEX10
        LD  R2,time
delay   ADD R2,R2,#-1
        BRp delay
        ADD R1,R1,#-1
        BRzp loop
        LD  R1,n255
loop2   STR R1,R0,#-2  ; write to HEX32
        LD  R2,time
delay2  ADD R2,R2,#-1
        BRp delay2
        ADD R1,R1,#-1
        BRzp loop2
        LD  R1,n255
loop3   STR R1,R0,#-3  ; write to LEDs
        LD  R2,time
delay3  ADD R2,R2,#-1
        BRp delay3
        ADD R1,R1,#-1
        BRzp loop3
bye     BRnzp bye
n255    .FILL x00FF
time    .FILL x0080
        .END
```

Task 1: Bubble Sort

- Write the code to bubble sort a list of 10 numbers
- At the end, all numbers should be sorted in increasing order
- Show the number of swaps on LED9_0
- Show the number of comparisons on Hex3_0
- Repeat experiment with 20 numbers and 30 numbers (or more if needed) with data of decreasing numbers
- Record #swaps and #comparisons for the three experiments in a table

bubble0.c

```
#include <stdio.h>

void swap_next(int* array, int j)
{
  int temp;
  temp = array[j];
  array[j] = array[j+1];
  array[j+1] = temp;
}
```

```
int main(void) {
  int size=10, i, j, temp;
  int nums[size];

  for (i = 0;  i<size;  i = i+1)
    nums[i] = size-i-1;

  for (i = size-1;  i>0;  i = i-1) {
    for (j = 0;  j<i;  j = j+1)
      if (nums[j]>nums[j+1])
        swap_next(nums, j);
  }
}
```

bubble1.c

```
int main(void) {
  int size=10, i, j, temp, c=0, swaps=0;
  int nums[size];

  for (i = 0;  i<size;  i = i+1)
    nums[i] = size-i-1;
  for (i = size-1;  i>0;  i = i-1) {
    counter_j = i;
    for (j = 0;  j<i;  j = j+1) {
      c = c + 1;
      if (nums[j] > nums[j+1]) {
        swap_next(nums, j);
        swaps = swaps+1;
      }
      counter_j = counter_j-1;
      if (counter_j<0) exit j loop;
    }
    if (i=0) exit i loop;
  }
}
```

Getting from C to Assembly Language

```
for (i = size-1;  i>0;  i = i-1) {
  for (j = 0;  j<i;  j = j+1) {
    if (nums[j] > nums[j+1]) {
      swap_next(nums, j);
    }
  }
}
```

```
for (i = size-1;  i>0;  i = i-1) {
  for (k = i;  k>0;  k = k-1) {
    j = size-1-k;
    if (nums[j] > nums[j+1]) {
      swap_next(nums, j);
    }
  }
}
; LC-3: BRp inLoop
; exit inLoop when k=0
```

Task 1: Example

- i = 4
- j =0,1,2,3,4
- a[j]=4,3,2,1,0
- At j=0, a[0]=4> a[1]=3, swap to get a[]: 3,4,2,1,0
- At j=1, a[1]=4 > a[2]=2, swap to get a[]: 3,2,4,1,0
- At j=2, a[2]=4 > a[3]=1, swap to get a[]: 3,2,1,4,0
- At j=3, a[3]=4 > a[4]=0, swap to get a[]: 3,2,1,0, 4
- Ignore a[4]=4 from now on because it is the max no. correctly placed

- i = 3, j=0,1,2,3; result a[]: 2,1,0, 3,4
- i = 2, j=0,1,2; result a[]: 1,0, 2,3,4
- i = 1, j=0; result a[]: 0, 1,2,3,4
- SORTING DONE!

Bubble.asm

- .orig x3000
- …

- Size .FILL #10
- Mtable .FILL #9
- .FILL #8
- .FILL #7
- .FILL #6
- .FILL #5
- .FILL #4
- .FILL #3
- .FILL #2
- .FILL #1
- .FILL #0

Time Complexity of Bubble Sort

- For arrays of different sizes, count the accumulated no. of swaps and no. of comparisons
- Record in a table of array size and accumulated no. of swaps and total no. of comparisons
- Write down:
- s = f(n)
- c = g(n)

Array Size n	#Swaps s	#Comparisons c	Time t (sec)
10			
20			
30			

Task 2: Put Maximum Element at the End

- Improve bubble sort
- Do not swap two neighbor elements
- i = size-1, size-2, …, 1
  - Check all elements and find the maximum no. in the remaining elements
  - Then swap nums[i] with the maximum element
  - This puts the max. element in nums[i]
- Why would this reduce the no. of swaps?
- Repeat the experiment for size of 10, 20, 30 (or more if needed).
- Record #swaps and #comparisons in table.
- In this new algorithm, is #comparisons a better or worse measure of time complexity than #swaps? Why?
- Is this more efficient than bubble sort? Why?

Task 2: Example

- i =0,1,2,3,4,5,6,7,8,9
- a[i]=4,3,2,1,0,9,8,7,6,5
- Max=4, i of max=0
- At i=1, max=4 > a[1]=3, so no update
- At i=2, max=4 > a[2]=2, so no update
- At i=3, max=4 > a[3]=1, so no update
- At i=4, max=4 > a[4]=0, so no update
- At i=5, max=4 < a[5]=9, so update max := 9, i_max := 5
- At i=6, max=9 > a[6]=8, so no update
- …
- At i=9, max=9 > a[9]=5, so no update

Example

- i: 0,1,2,3,4,5,6,7,8,9
- Data: 9,8,7,6,5,4,3,2,1,0
- Check i=0 to 9: max=a[0]=9, i_max=0
- Swap: 0,8,7,6,5,4,3,2,1, 9
- Check i=0 to 8: max=a[1]=8, i_max=1
- Swap: 0,1,7,6,5,4,3,2, 8,9
- Check i=0 to 7: max=a[2]=7, i_max=2
- Swap: 0,1,2,6,5,4,3, 7,8,9
- Check i=0 to 6: max=a[3]=6, i_max=3
- Swap: 0,1,2,3,5,4, 6,7,8,9
- Check i=0 to 5: max=a[4]=5, i_max=4
- Swap: 0,1,2,3,4, 5,6,7,8,9
- …

swap_max.c

```
void swap(int* array, int j, int k) {
  int temp;
  temp = array[j];
  array[j] = array[k];
  array[k] = temp;
}

int main(void) {
  int size=10, i, j, k, temp, swaps=0, max, i_max;
  int nums[size];

  for (i = 0;  i<size;  i = i+1)
    nums[i] = size-i-1;

  for (i = size-1;  i>0;  i = i-1) {
    max = nums[0];
    i_max = 0;
    for (j = 1;  j <= i;  j = j+1) {
      if (nums[j] > max) {
        max = nums[j];
        i_max = j;
      }
    }
    swap(nums, i, i_max);
    swaps = swaps + 1;
  }
}
```

Record Experimental Data and Explain

- Record the experimental data in a table like the one on the right
- Compare the performance of this algorithm to the previous bubble sort algorithm
- Difference in #swaps?
- Difference in #comparisons?

Array Size n	#Swaps s	#Comparisons c	Time t (sec)
10			
20			
30			