Homework 3

CDA 3104 Computer Organization and Assembly Language Programming
Due date: Oct. 28, 2013   11:59pm

Requirements:

1. This assignment as well as other assignments in this class must be finished on Windows operating system.
2. Zip your program and submit the zip file on Canvas.
3. You should add enough comments in your programs.
4. ***Please make your own test drivers.*** I will test your program with my own test drivers. So do not assume the existence of any variable or constant.

Assignments:

Please implement the following procedures:

```
; Receives: ESI: offset of byte array, ECX: len of array, AL: search key
; Returns: EBP: index of search key in array (-1 when search fails)
; Assumes: array is sorted
binarySearchB PROC uses EAX EBX ECX ESI EDX
        …
        ret
binarySearchB ENDP
```

```
; Receives: ESI: offset of first array element,
;           EDI: offset of last array element,
;           AL: partition pivot
;
; Returns: EBP: array elements with an index, greater than EBP, are
;           greater than the pivot (AL).
;          EBP = -1 when all array elements are greater than the pivot
;
; Description: Partitions a byte array into two portions: left and right.
;         Elements in the left portion are less than or equal to
;         the pivot (AL). The ones in the right are greater than.
;
;Note: only works for byte arrays
partitionB PROC uses ESI EDI EAX EBX
        …
        ret
partitionB ENDP
```

Here is a C-like program, implementing binary search and partition:

```
int binarySearch(int array[], int size, int key) {

  int first = 0, last = size-1;
  while (first <= last) {
      int mid = (first + last) / 2;  // find the middle element.
      if (key > array[mid])  // not in the lower half
          first = mid + 1;
      else if (key < array[mid]) // not in the upper half
          last = mid - 1;
      else
          return mid;   // search succeeds
  }
  return -1;    // search fails
}
```

```
int partition(int [ ] array, int size, int pivot){
        int down=0, up=size-1;
        while(down<up){
                // finds the first, from left, element that is greater than pivot
                while(down<=up && array [down]<=pivot){
                        down++;
                }

                // finds the first, from right, element that is less than or equal to pivot
                while(up>=down && array [up]>pivot){
                        up--;
                }

                if(down<up){
                        // exchange array[down] and array[up]
                        int temp= array [down];
                        array [down]= array [up];
                        array [up]=temp;
                }
        }
        return up;
}
```

<u>Grading Policies:</u>

Program readability	10%
Successfully implement procedure binary search	35%
Successfully implement procedure partition	35%
Successfully implement test drivers	20%