Digital Architectures – Ex 10

Prepared by Aviv Shukrun

1. Starting from address 0x10000100 resides a sorted array of integers (no repeating values). The size of the array is written in address 0x10000000. In address 0x10000004 an integer element is written. You should find the index of the element in the array, and write it to address 0x10000008. If the element is not in the array – write (-1) to that address. The search algorithm should have time complexity of log(n), where n is the number of elements in the array.
    a. Your code should run on the **QTPSIM** program, which was discussed in class. Use an example of an array with the following values:
    1, 5, 7, 9, 0xb, 0xd, 0x10, 0x4000, 0x50000, 0x700000,
    and look for the value 0xd.
    b. You can assume that the value we look for is not at the last position.
    c. Document your program. **Undocumented exercises will not be checked**.
    d. You can find a C version of the exercise below.
    e. You should submit a zipped folder containing your **code (.asm file)** and a **screenshot** showing the simulator in action when you finish the program.
    f. Submission can be done in **pairs**. In this case a **txt file** containing the IDs of the group should be added to the ZIP file. Only one of the pair should submit the exercise, **not both!**
    g. You can use the div command in the assembly language:
    div $s7, $t0, 2

Guidelines:
Implement a binary search algorithm:

```
first = 0;
last = size – 1;
while( last – first > 1 )
{
        mid = (last – first) / 2 + first;
        if( A[mid] == val )
                break;

        if( A[mid] > val )
                last = mid;
        else
                first = mid;
}
```

Solution (.asm code)

```
#-------------------------------------------------------
# This program finds an element in a sorted array of "arr_size"
# starting from the address written in "strt_adrs"
# and writes the index of the element to "index"

# We implement a binary search algorithm:
#
# first = 0
# last = size -1
# while (last - first > 1) {
#       mid = (last-first)/2 + first
#       if A[mid] == val
#               break;
#       if A[mid] > val
#               last = mid;
#               continu
#       else
#               first = mid;
#               continue;
# }
#

#-------------------------------------------------------
# Here the data of this program starts
#-------------------------------------------------------

                .data   0x10000000


arr_size:       .word   0x0000000a      # array size = 10
element:        .word 0x0000000d# element to search
index:          .word 0xffffffff      # index of element in the array
strt_adrs:      .word   0x10000100      # start address of array

        .data   0x10000100      # put the next data block
                        # starting from 0x10000100
        .word 0x00000001
   .word       0x00000005
        .word 0x00000007
        .word 0x00000009
        .word 0x0000000b
        .word 0x0000000d
   .word       0x00000010
   .word       0x00004000
   .word       0x00050000
```

```
        .word 0x00700000

#---------------------------------------------------------
#
#  Here is the program itself:
#
        .text   0x0400000
#---------------------------------------------------------
main:
        lui     $gp, 0x1000
        # $gp points at 0x10000000
        lw      $s0, 0($gp)     # $s0 = array_size value
   lw     $s1, 12($gp)         # $s1 = strt_adrs value
                        # $s1 will be our ptr from start
        lw      $s2, 4($gp)     # $s2 = element to search

        add             $s3, $0, $0             # $s3 = first index
        lw              $s4, 0($s1)             # first element
        addi    $s5, $s0, -1    # $s5 = arr_size-1 = last index

        add     $t9, $s5, $s5   # $t9 = 2*$s5
        add     $t9, $t9, $t9   # $t9= 4*(arr_size-1)
        add     $t9, $t9, $s1   # $t9= address of last element
        lw              $s6, 0($t9)             # last element
#---------------------------------------------------------
loop:
#       First test if last - first > 1 If so, exit.
        sub     $t0, $s5, $s3#
        slti    $t1, $t0, 2             # if $t0 < 2 then $t1 = 1
        bne     $t1,$0, endlp           # if $t0 < 2 go to endlp
        ori     $0,$0,0

        div             $s7,$t0,2               # calculate mid index
        add             $s7,$s7,$s3             # mid index= first+(last-first)/2

        add     $t3, $s7, $s7   # $t3 = 2*$s5
        add     $t3, $t3, $t3   # $t3= 4*mid
        add     $t3, $t3, $s1   # $t3= address of mid element

        lw              $t2, 0($t3)              # $t2 = A[mid]

        beq             $t2,$s2,end_srch    #if element = A[mid]
        ori     $0,$0,0
        slt             $t4, $s2, $t2           # if element < A[mid] then $t4 = 1
if:
        beq             $t4, $zero, else
```

```
        ori     $0,$0,0
        add             $s5, $s7, $zero         # last index = mid index
        #move           $s5, $s7                # last index = mid index
        add             $s6, $t2, $zero         # last val = mid val
        #move           $s6, $t2                # last val = mid val
        j               loop
        ori             $0,$0,0
else:
        add             $s3, $s7, $zero         # first index = mid index
        #move           $s3, $s7                # first index = mid index
        add             $s4, $t2, $zero
        #move           $s4, $t2
        j               loop
        ori             $0,$0,0
#-------------------------------------------------------
end_srch:

        sw              $s7, 8($gp)         # index = mid index

#-------------------------------------------------------
#
endlp:
   beq     $zero, $zero, endlp
        ori      $0,$0,0
#
#-------------------------------------------------------
```