

## CS F342 Computer Architecture

Semester 1 – 2021-22

### Lab Sheet 10 & 11

Goals for the Lab: We build up on prior labs and Exploring sorting techniques using MIPS

#### Exercise 1:

Write a program to implement C bubble sort program given below in MIPS.

C program code:

```
int main()
{
    int Sz = 10;
    int List[10] = {17, 5, 92, 87, 41, 10, 23, 55, 72, 36} ;
    int Stop, // $s3: upper limit for pass
    Curr, // $s0: index of current value in comparison
    Next, // $s1: index of successor to current value
    Temp; // $s2: temp storage for swap
    for (Stop = Sz-1; Stop > 0; Stop)
    {
        for (Curr = 0; Curr < Stop; Curr++)
        {
            Next = Curr + 1;
            if ( List[Curr] > List[Next] )
            {
                Temp = List[Curr];
                List[Curr] = List[Next];
                List[Next] = Temp;
            }
        }
    }
    printf("Sorted list in ascending order:\n");
    for (Curr = 0; Curr < Stop; Curr++)
    printf("%d\n", List[Curr]);
}
```

**Hint ::** To convert Curr to offset you can use sll \$t4, \$t2, 2 or similar where \$t2 is Curr, \$t4 is offset from starting address of buffer and shift of 2 implies multiplying by 4.

Partial assembly code: (Highlighted part is complete)

```
.data
list: .word 17, 5, 92, 87, 41, 10, 23, 55, 72, 36
space: .asciiz " "
.text
main:

li $s7, 10 #size of the list(sz)
addi $s3, $s7, -1 # $s3 = Stop = sz-1
```

*#Write the loop, swap code here*

```

exit:    #print the array

la $t0,list
li $t2,0 #as a counter while printing the list

print:

lw $a0,($t0)    #load current word in $a0

li $v0,1
syscall         #print the current word
la $a0,space
li $v0,4
syscall         #print space in b/w words
addi $t0,$t0,4  #point to next word
addi $t2,$t2,1  #counter++
blt $t2,$s7,print

li $v0,10
syscall

```

### **Exercise 2:**

Write a program to implement above program but store floating point numbers instead of integer.

*Hint: Use commands swc1, lwc1, c.le.s, bc1f, bc1t*

Comparison of FP values sets a code in a special register and Branch instructions jump depending on the value of the code:

```

c.le.s $f2, $f4 #if $f2 <= $f4 then code = 1 else code = 0

bc1f label #if code == 0 then jump to label

bc1t label # if code == 1 then jump to label

```

### **Exercise 3:**

Write a program to implement C Insertion sort program given below in MIPS.

#### **C program code:**

```

int main()
{
    int n = 5;
    int array[5] = { 5, 3, 4, 2, 1 };
    int c = 0;
    int d = 0;
    int t = 0;

    for (c = 1 ; c <= n - 1; c++) {
        d = c;
        while (d > 0 && array[d] < array[d - 1]) {
            t = array[d];

```

```

        array[d] = array[d - 1];
        array[d - 1] = t;
        d--;
    }
}

for (c = 0; c <= n - 1; c++) {
    printf("%d\n", array[c]);
}

return 0;
}

```

### Partial assembly code:

```

.data
array: .word 0:1000      # an array of word, for storing values.
size:  .word 5           # actual count of the elements in the array.

sort_prep:
    la    $t0, array      # load array to $t0.
    lw    $t1, size       # load array size to $t1.
    li    $t2, 1          # loop runner, starting from 1.

sort_xloop:
    la    $t0, array      # load array to $t0.
    bge   $t2, $t1, sort_xloop_end # while (t2 < $t1).
    move  $t3, $t2        # copy $t2 to $t3.

sort_iloop:
    la    $t0, array      # load array to $t0.
    mul   $t5, $t3, 4      # multiply $t3 with 4, and store in $t5
    add   $t0, $t0, $t5    # add the array address with $t5, which is the index multiplied with 4.
    ble   $t3, $zero, sort_iloop_end # while (t3 > 0).
    lw    $t7, 0($t0)      # load array[$t3] to $t7.
    lw    $t6, -4($t0)     # load array[$t3 - 1] to $t6.
    bge   $t7, $t6, sort_iloop_end # while (array[$t3] < array[$t3 - 1]).
    lw    $t4, 0($t0)
    sw    $t6, 0($t0)
    sw    $t4, -4($t0)
    subi  $t3, $t3, 1
    j     sort_iloop      # jump back to the beginning of the sort_iloop.

sort_iloop_end:
    addi  $t2, $t2, 1      # increment loop runner by 1.
    j     sort_xloop      # jump back to the beginning of the sort_xloop.

sort_xloop_end:
    li    $v0, 4           # 4 = print_string syscall.
    la    $a0, sorted_array_string # load sorted_array_string to argument register $a0.
    syscall
    li    $v0, 4           # 4 = print_string syscall.
    la    $a0, line        # load line to argument register $a0.
    syscall
    jal   print            # call print routine.

```

#### Exercise 4:

Observe the sample code given below and Write a program to implement merge sort program in MIPS.

#### partial code:

mergesort:

```
addi    $sp, $sp, -16      # Adjust stack pointer
sw      $ra, 0($sp)        # Store the return address on the stack
sw      $a0, 4($sp)        # Store the array start address on the stack
sw      $a1, 8($sp)        # Store the array end address on the stack

sub      $t0, $a1, $a0     # Calculate the difference between the start and end address (i.e. number of elements * 4)

ble      $t0, 4, mergesortend # If the array only contains a single element, just return

srl      $t0, $t0, 3        # Divide the array size by 8 to half the number of elements (shift right 3 bits)
sll      $t0, $t0, 2        # Multiple that number by 4 to get half of the array size (shift left 2 bits)
add      $a1, $a0, $t0      # Calculate the midpoint address of the array
sw      $a1, 12($sp)        # Store the array midpoint address on the stack

jal      mergesort          # Call recursively on the first half of the array

lw      $a0, 12($sp)        # Load the midpoint address of the array from the stack
lw      $a1, 8($sp)         # Load the end address of the array from the stack

jal      mergesort          # Call recursively on the second half of the array

lw      $a0, 4($sp)         # Load the array start address from the stack
lw      $a1, 12($sp)        # Load the array midpoint address from the stack
lw      $a2, 8($sp)         # Load the array end address from the stack

jal      merge              # Merge the two array halves
```

mergesortend:

```
lw      $ra, 0($sp)        # Load the return address from the stack
addi    $sp, $sp, 16       # Adjust the stack pointer
jr      $ra                # Return
```

#### **Exercise 5:**

Write a MIPS Program to implement Quick sort (Home Work)

### Exercise 6:

Observe the sample code given below and Write a program to implement Binary search program in MIPS.

#### partial code:

```
.data
msg_inputList:  .asciiz "Please enter positive numbers in ascending order and a 0 to terminate\n"
msg_searchList: .asciiz "Please enter a number to initSearch for\n"

initSearchList:
    li    $v0, 4          # syscall 4 (print_str)
    la    $a0, msg_searchList # load the search items input message
    syscall                # execute message print

    li    $s2, 0          # set search items counter to 0

searchList:
    li    $v0, 5          # syscall 5 (read_int)
    syscall                # execute int reading
    move   $t1, $v0        # move int to $t1
    blez   $v0, initSearch # start search if 0 was entered

    li    $v0, 9          # syscall 4 (sbrk)
    la    $a0, 4          # 4 bytes allocated for ints
    syscall                # execute memory allocation

    li    $t0, 4          # 4 bytes for an int
    add    $t2, $s4, $s2   # length of the list is counter1 + counter 2
    mul    $t0, $t2, $t0   # length of the input storage address space
    add    $t0, $t0, $s1   # calculate end of address spaces
    move   $s3, $t0        # store end of address space
    sw     $t1, ($t0)      # store input on the heap
    addi   $s2, $s2, 1     # counter++

    j      searchList     # take next input

initSearch:
    move   $t6, $s5        # store end address of input items
    move   $t7, $s3        # store end address of search items

search:
    move   $t5, $s5        # store end address of input items
    beq    $t7, $t6, exit  # if there's nothing to search, exit
```

### Exercise 7:

Observe the sample code given below and Write a program to implement Heap sort program in MIPS.

#### partial code:

```
.text
.globl main
main:

    la $a0, array      # a0 = &array

    la $t0, size
    lw $a1, 0($t0)     # a1 = size(array)

    jal heapsort       # print the array
    move $t0, $a0
    add $t1, $zero, $zero

heapsort:              # a0 = &array, a1 = size(array)
    addi $sp, $sp, -12
    sw $a1, 0($sp)     # save size
    sw $a2, 4($sp)     # save a2
    sw $ra, 8($sp)     # save return address

heapsort_loop:        # swap(array[0],array[n])
    lw $t0, 0($a0)
    sll $t1, $a2, 2     #t1 = bytes(n)
    add $t1, $t1, $a0
    lw $t2, 0($t1)
    sw $t0, 0($t1)
    sw $t2, 0($a0)

    addi $a2, $a2, -1   # n--
    jal bubble_down    # a0 = &array, a1 = 0, a2 = n

    bnez $a2, heapsort_loop
make_heap:            # a0 = &array, a1 = size
    addi $sp, $sp, -12
    sw $a1, 0($sp)
    sw $a2, 4($sp)
    sw $ra, 8($sp)

    addi $a2, $a1, -1   # a2 = size - 1

    addi $a1, $a1, -1   # start_index = size - 1
    srl $a1, $a1, 1     # start_index /= 2

    blt $a1, $zero, end_make_heap # if(start_index < 0) return
make_heap_loop:
    jal bubble_down    # a0 = &array, a1 = start_index, a2 = size-1
    addi $a1, $a1, -1
    ble $zero, $a1, make_heap_loop
```

### **Exercise 8:**

Write a program to implement C Selection sort program given below in MIPS.

#### **C program code:**

```
int main() {  
    int arr[10]={6,12,0,18,11,99,55,45,34,2};  
    int n=10;  
    int i, j, position, swap;  
    for (i = 0; i < (n - 1); i++) {  
        position = i;  
        for (j = i + 1; j < n; j++) {  
            if (arr[position] > arr[j])  
                position = j;  
        }  
        if (position != i) {  
            swap = arr[i];  
            arr[i] = arr[position];  
            arr[position] = swap;  
        }  
    }  
    for (i = 0; i < n; i++)  
        printf("%d\t", arr[i]);  
    return 0;  
}
```