Computer Architecture Assignment 1

Chosen MIPS Assembly Program:

Sorting Algorithm (Using Insertion Sort, sorts in ascending order at the output address) implemented using C++. Instruction and Data memory are 32 bits wide similar to MIPS. Here, Memory is Byte addressable.

Submission Details:

The zip file contains the following files.

1. **IMT2022527_InsertionSort.asm**: This file contains the MIPS assembly code for performing Insertion Sort. The code itself has basic MIPS instructions and is commented appropriately.
2. **IMT2022527_Report.pdf**: This pdf contains a detailed explanation of the working of Insertion Sort Algorithm in MIPS assembly code and it's implementation. It also explains how the assembler works and has screenshots of the output of the algorithm.

3. **IMT2022527_Assembler.cpp**: This is the C++ code for the MIPS Assembler that generates and writes the machine code (of the MIPS assembly code) into the file 'SelfGeneratedBinary.txt'.
4. **MarsGeneratedBinary.txt**: This file contains the machine code generated by the Mars Simulator for the same assembly code.
5. **SelfGeneratedBinary.txt**: This file contains the machine code generated by the MIPS Assembler written in C++.
6. **IMT2022527_CheckFiles.cpp**: This file contains C++ code to check if the content of the files 'SelfGeneratedBinary.txt' and 'MarsGeneratedBinary.txt' are the same and throws an appropriate message for the same on execution.

➔ On Verification, it can be seen that the machine code generated by self-written assembler and the MARS simulator are the same.

Insertion Sort:

In Insertion Sort, we iterate from left to right, evaluating each element and comparing it with the elements to its left. If the current element has a lower value than the one to its left, we swap them, ensuring the element is 'inserted' into the appropriate position in the array. Throughout this process, the array is divided into sorted and unsorted partitions. The sorted partition expands as items are correctly placed, while the unsorted section diminished.

The Insertion Sort algorithm in C++:

```
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        // Move elements of arr[0..i-1],
        // that are greater than key,
        // to one position ahead of their
        // current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

The MIPS program for Insertion Sort algorithm:

->The register $t1 contains the number of integers ‘n’. (Given)

->The register $t2 contains the input address. (Given)

->The register $t3 contains the output address where the sorted array is to be stored at. (Given)

Code	**Comments**

add $t0,$zero,$zero
 	
copytooutput:
slt $s0,$t0,$t1
beq $s0,$zero,insertionsort
 	
sll $t9,$t0,2
add $s6,$t2,$t9
lw $s5,0($s6)
add $s6,$t3,$t9
	Functionality: Copying the integers from input address to output address
	
	$t0 is being used as a counter to implement output[i]=input[i] where ‘I’ is iterated n times.
	
	$t9 stored i*4 since memory is byte addressable, $s6 stores $t2+$t4 $s5 stores M($s6).

sw $s5,0($s6)
 	
addi $t0,$t0,1
j copytooutput
	Now $s6 is assigned $t3+$t9 and $s5 is written into M($s6).
	
	Counter is incremented by one
	Jump back to the copytooutput label.

insertionsort:
 	
addi $t5,$zero,1
add $t6,$zero,$zero
add $t7,$zero,$zero
	Functionality: Initializing counters and necessary variables for implementing the sorting algorithm
	$t5 is 'l'
	$t6 is 'key'
	$t7 is 'j'

outerloop:
 	
slt $s0,$t5,$t1
beq $s0,$zero,finishsort
 	
sll $s1,$t5,2
add $s2,$s1,$t3
lw $t6,0($s2)
 	
addi $t9,$zero,1
sub $t7,$t5,$t9
 	
jal innerloop
 	
sll $s1,$t7,2
add $s2,$s1,$t3
sw $t6,4($s2)
 	
addi $t5,$t5,1
j outerloop
	Functionality: Perform the outer for loop in the Insertion Sort algorithm.
	
	$s0 is used to check if i<n. It ends the sorting algorithm(i.e.,jumps to finishsort label) if $s0 evaluates to zero.
	
	$s1 stores i*4 since memory is byte addressable, $s2 stores $s1+$t3
	$t6 stores M($s2) i.e., key has been assigned arr[i].
	Assigning 'j' the value of 'l' decremented by one.
	
	A procedure call to innerloop.
	
	$s1 stores j*4, $s2 stores $s1+$t3
	Now, $t6 i.e., key, is assigned to M(4+$s2). This is the same as arr[j+1]=key.
	Incrementing 'l' by one.
	Jumping to the outerloop.

innerloop:	Functionality: Performs the inner while loop of the insertion sort which is to swap as long as the element on

	the left has a higher value than the element itself.
slt $s0,$t7,$zero bne $s0,$zero,goback	$s0 is used to check if j<0. If $s0 evaluates to 1, then branches to goback.

sll $s1,$t7,2
add $s2,$s1,$t3
lw $s4,0($s2)
	$s1 stores j*4, $s2 stores $s1+$t3
	Now, $s4 stores M($s2) i.e., $s4 stores arr[j]

slt $s0,$t6,$s4 beq $s0,$zero,goback	$s0 is used to check if key<arr[j]. If $s0 evaluates to zero, then it branches to goback.
sw $s4,4($s2)	Now, $s4 is assigned to M(4+$s2). This is the same as arr[j+1]=arr[j].
addi $t9,$zero,1 sub $t7,$t7,$t9	Decrementing the value of 'j' by 1.
j innerloop	Jumping to the innerloop.
goback: jr $ra	Functionality: To return from the procedure call
finishsort:	This label marks the end of the sorting algorithm.

Registers usage:

The registers not mentioned in this table are either being used only for getting inputs from terminal and printing them or are not being used at all.

Register	Stored Value	Used For/As
$t0	Initialized to '0' and increments by '1' till it gets value 'n'.	A counter to copy the elements stored in the input address to the output address.

$t1	A 32 bit address location entered in it's decimal equivalent.	Storing the number of integers to be taken as input from terminal.
$t2	A 32 bit address location entered in it's decimal form.	Storing the starting address of the input numbers.
$t3	A 32 bit address location entered in it's decimal form.	Storing the starting address of the output numbers.
$t4	A 32 bit address/data in it's decimal form or 'n'.	A temporary placeholder for storing 'n' or a 32 bit address, or the data in a memory location.
$t5	Initialized to '1' and increments by 1 till it gets value 'n-1'.	A counter for the outer loop of insertion sort. $t5 is the equivalent for 'I' in Insertion sort.
$t6	Initialized to '0' at the starting of the algorithm. Gets initialized to 'i-1' inside the loop.	A conditional for the termination of the while loop. $t6 is the equivalent for 'j' in insertion sort.
$t7	Initialized to '0' at the starting of the algorithm. Gets assigned different values throughout the algorithm.	An quivalent for 'key' in insertion sort.
$t8	A 32 bit address.	A temporary placeholder for storing a 32 bit address.
$t9	$t0*4 or stores '1'.	An intermediate placeholder for access arr[i] etc. and is used for performing subtraction by '1'.
$s0	'0' or '1'.	A placeholder to check whether a condition evaluates to true or false.
$s1	'i*4' or 'j*4'.	Accessing arr[j] is equivalent to loading from M(j*4+$t3). $s1 stores i*4 or j*4 for this purpose.
$s2	$s1+$t3 or $s1+$t2.	Storing values like 'j*4+$t3' so that the values at that location can be loaded/stored into using 'lw', 'sw'.
$s4	Data at a memory location.	Storing/loading M(x) to access arr[j] etc. to work with later.

$s5	Data at a memory location.	A temporary placeholder to read from M(x) and later write this value into memory at a different/same location. This is used only during copying of input elements into output address.
$s6	$t9+$t3 or $t9+$t2.	Storing values like ‘i*4+$t9’. This is used only during copying of input elements into output address.
$ra	Stores pc+4 at the time of encounter of this instruction, where pc refers to the address of this instruction.	Procedure calls.

Working of the self-written Assembler: The C++ file contains code to parse assembly instructions and generate corresponding machine code just like an assembler. It uses unordered maps like R_umap, I_umap, J_umap to map instructions to their formats and registers which maps each register to it's register number in binary form of 5 bits. First, it creates a map between labels and their addresses by reading through the entire assembly code once. This is used for instructions like 'beq', 'bne'. Now, the assembly code is read again, generating machine code for each instruction encountered based on it's type and the registers used. The generated machine code for the instructions are stored in a 'machine_code' vector. Finally, the machine code is written to the file 'SelfGeneratedBinary.txt'.

Screenshots of Outputs:

1. Input: The input array is {34}.

```
Enter No. of integers to be taken as input: 1
Enter starting address of inputs(in decimal format): 268501184
Enter starting address of outputs (in decimal format): 268501216
Enter the integer: 34
```

Output: The sorted array is {34}.

```
34

-- program is finished running --
```

2. Input: The input array is {34,12}.

```
Enter No. of integers to be taken as input: 2
Enter starting address of inputs(in decimal format): 268501184
Enter starting address of outputs (in decimal format): 268501292
Enter the integer: 34
Enter the integer: 12
```

Output: The sorted array is {12,34}.

```
12
34

-- program is finished running --
```

3. Input: The input array is {23,89}.

```
Enter No. of integers to be taken as input: 2
Enter starting address of inputs(in decimal format): 268501184
Enter starting address of outputs (in decimal format): 268501216
Enter the integer: 23
Enter the integer: 89
```

Output: The sorted array is {23,89}.

```
23
89

-- program is finished running --
```

4. Input: The input array is {45,89,76}.

```
Enter No. of integers to be taken as input: 3
Enter starting address of inputs(in decimal format): 268501184
Enter starting address of outputs (in decimal format): 268501216
Enter the integer: 45
Enter the integer: 89
Enter the integer: 76
```

Output: The sorted array is {45,76,89}.

```
45
76
89

-- program is finished running --
```

5. Input: The input array is {98,97,96,95,93,92,87,83,81,76,75,1,8}.

```
Enter No. of integers to be taken as input: 13
Enter starting address of inputs(in decimal format): 268501184
Enter starting address of outputs (in decimal format): 268501280
Enter the integer: 98
Enter the integer: 97
Enter the integer: 96
Enter the integer: 95
Enter the integer: 93
Enter the integer: 92
Enter the integer: 87
Enter the integer: 83
Enter the integer: 81
Enter the integer: 76
Enter the integer: 75
Enter the integer: 1
Enter the integer: 8
```

Output: The sorted array is {1,8,75,76,81,83,87,92,93,95,96,97,98}.

```
1
8
75
76
81
83
87
92
93
95
96
97
98

-- program is finished running --
```