

Assignment-1 Report

By:

- Divyam Sareen (IMT2022010)
- Aryan Mishra (IMT2022502)

Q1. We have implemented the Bubble sort algorithm to sort the elements in ascending order.

- Firstly, let's look at the pseudocode of Bubble sort algorithm:

```
bubbleSort(arr){  
  n = length(arr)  
  for i from 0 to n-1{  
    for j from 0 to n-i-1{  
      if arr[j] > arr[j+1]  
        swap(arr[j], arr[j+1])  
    }  
  }  
}
```

This basically compares the adjacent elements of the given array. If the adjacent elements are out of order(i.e increasing/decreasing depends on what the user chooses to have), they are swapped. This process is repeated iteratively throughout the array and so at the end of all iterations, we find the elements at their correct positions in order.

- Explanation of the code we wrote(for step by step detailed explanation please refer to the comments of the code):

It can be divided into several parts:

1. Initialisation:
 - Set \$t5 to $n - 1$ (assuming \$t1 contains n)
 - Initialize outer loop iterator to 0
 - Store the input address in \$t8
2. Outer loop(loop 2):
 - Check if outer loop iterator is equal to $n - 1$, exit if true
3. Inner loop(loop 3) within outer loop:
 - Set \$t7 to $n - i - 1$ (where i is the outer loop iterator)
 - Initialize inner loop iterator to 0
 - Restore the original input address for the inner loop
4. Inner loop body(loop 3):
 - Check if inner loop iterator is equal to $n - i - 1$, exit if true
 - Load $arr[j]$ into \$s4
 - Load $arr[j+1]$ into \$s5
 - If $arr[j] > arr[j+1]$, jump to swap
 - Increment j
 - Increment inner loop iterator
 - Jump back to the start of inner loop
5. Swap function:
 - move $arr[j]$ to a temporary register for swapping if the condition $arr[j] > arr[j+1]$ is satisfied
 - move $arr[j+1]$ to $arr[j]$
 - move value of $arr[j]$ which was stored in a temporary register into $arr[j+1]$ and hence swapping is completed
 - store the value of $arr[j]$ in the memory
 - store the value of $arr[j+1]$ in the memory
 - increment j
 - increment the iterator of inner loop
 - jump back to the start of inner loop
6. End of outer loop (loop 2):
 - Restore the original input address

- Store the original output address in \$t6
- Initialize iterator for the next loop to 0
- 7. Copy result to output(loop 4):
 - Check if iterator is equal to n, exit if true
 - Load the first element of the input array into \$t5
 - Store that element in the memory at the given output
 - Move to the next element of the output array
 - Move to the next element of the input array
 - Increment iterator
 - Jump back to the start of this loop
- 8. End of loop 4:
 - Restore the original output address into \$t3

- Sample outputs:

1.

The screenshot shows the MARS MIPS simulator interface. The assembly code window displays the following instructions:

```

14: jal print_int_statement
15: jal input_int
16: move $t1,$t4
20: jal print_int_statement
21: jal input_int
22: move $t2,$t4
26: jal print_out_int_statement
27: jal input_int
28: move $t3,$t4
32: move $t8,$t2
33: move $t5,$t3
34: loop1: beq $t1,$t1,loop1end
  
```

The Data Segment window shows memory addresses and values for registers \$t0 through \$t10. The Registers window shows the current values of registers \$t0 through \$t10. The Mars Messages window shows the following input/output messages:

```

Enter No. of integers to be taken as input: 4
Enter starting address of inputs(in decimal format): 268501216
Enter the integer: 45
Enter the integer: 23
Enter the integer: 34
Enter the integer: 1
  
```

The screenshot shows the Mars Messages window with the following input/output messages:

```

Enter the integer: 45
Enter the integer: 23
Enter the integer: 34
Enter the integer: 1
  
```

2.

C:\Users\T761\Downloads\mips1.asm - MARS 4.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Performance Overlay (Alt+F8)

GPU: NVIDIA GeForce RTX 305...

FPS: N/A

MSFS FPS: N/A

Number Latency: N/A

Registers Coproc 1 Coproc 2

Name	Number	Value
\$0 (\$zero)	0	0
\$12 (\$status)	12	61509
\$13 (\$cause)	13	0
\$14 (\$epc)	14	0

Text Segment

Addr	Code	Basic	Source
4194310	0x0c100054	jal 4194640	20: jal print inp int statement
4194320	0x0c100042	jal 4194568	21: jal print int
4194324	0x00000021	addu \$10,\$0,\$12	22: move \$t2,\$t4
4194328	0x0c100059	jal 4194660	26: jal print out int statement
4194332	0x0c100042	jal 4194568	27: jal print int
4194336	0x00000021	addu \$11,\$0,\$12	28: move \$t3,\$t4
4194340	0x00000021	addu \$24,\$0,\$10	32: move \$t8,\$t2
4194344	0x00000021	addu \$23,\$0,\$0	33: move \$s7,\$zero \$i = 0
4194348	0x12e90006	beq \$23,\$5,\$6	34: loop1: beq \$s7,\$t1,loop1end
4194352	0x0c10005e	jal 4194680	35: jal print enter int
4194356	0x0c100042	jal 4194568	36: jal print int
4194360	0xad4c0000	sw \$12,0(\$10)	37: sw \$t4,0(\$t2)

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268501088	540682612	1953383680	1931506277	1953653108	543645385	1915181921	544437093	1864394351
268501120	1970304117	673215348	1679847017	1835623269	1713400529	1634562671	540682612	1953383680
268501152	1948283493	1763730752	1734702190	540701285	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	1	25	34	44	45	56	98	667
268501248	0	0	0	0	0	0	0	0
268501280	0	0	0	0	0	0	0	0
268501312	0	0	0	0	0	0	0	0
268501344	1	25	34	44	45	56	98	667
268501376	0	0	0	0	0	0	0	0

Mars Messages Run I/O

Enter No. of integers to be taken as input: 8
Enter starting address of inputs(in decimal format): 268501216
Enter starting address of outputs (in decimal format): 268501344

Clear

Enter the integer: 45
Enter the integer: 667
Enter the integer: 34
Enter the integer: 25

Mars Messages Run I/O

Enter the integer: 667
Enter the integer: 34
Enter the integer: 25
Enter the integer: 1
Enter the integer: 56
Enter the integer: 98
Enter the integer: 44

Clear

Mars Messages Run I/O

Enter the integer: 44
1
25
34
44
45
56

Clear

Mars Messages Run I/O

34
44
45
56
98
667

Clear

Q2. Assembler for the sorting algorithm:

{NOTE: We have only created the code which gives the machine code of the sorting algorithm part not input and outputs. So, the machine code generated by the code will be the subset of the actual code generated by MARS)

We have named the code generated by assembler as
“machine_code.txt”

We have named the code generated by assembler as
“machinecode7.txt”

Explanation:

- Step 1: Initialization and Definitions

→ The code begins by defining dictionaries for opcodes, function codes, register codes, numeric values, and label addresses. These dictionaries provide mappings between assembly mnemonics and their binary representations.

→ Additionally, the code initializes an empty list (`machine_code`) to store the generated machine code. Another dictionary (`label_dict`) is created to store labels and their corresponding addresses.

- Step 2: Function Definitions

The code defines several functions that will be used to assemble different types of MIPS instructions:

→ `assemble_r_type(parts)`: Assembles R-type instructions.

→ `assemble_i_type(parts)`: Assembles I-type instructions.

→ `assemble_lw_sw_type(parts)`: Assembles LW/SW-type instructions.

→ `assemble_j_type(parts)`: Assembles J-type instructions.

→ `process_line(line, lineno, pass_no)`: Processes each line of the assembly code during both passes, collecting labels in the first pass and generating machine code in the second pass.

- Step 3: First Pass (Label Collection)

The code opens the input file (`mips.txt`) and iterates through each line. In the first pass, it collects labels and their corresponding addresses:

- It skips empty lines and removes comments and whitespace.
- For lines with labels, it extracts the label and stores its address.
- The line number is incremented.

- Step 4: Second Pass (Code Generation)

The code clears the `machine_code` list and opens the input file again. During the second pass, it generates machine code:

- For each line, it skips empty lines and removes comments and whitespace.
- It calls the `process_line` function to assemble instructions based on their types.
- Special handling is provided for the `move` and `bgt` instructions as they are pseudo-instructions consisting of more than one instruction.

- `move` is implemented as an `addu` instruction.
 - `bgt` is implemented as `slt` followed by `bne`.

- Step 5: Output

The generated machine code is written to an output file (`machine_code.txt`).

VS Code editor interface showing a file explorer on the left with 'machinecode7' selected. The main editor displays a list of 84 lines of binary code (0s and 1s). The terminal at the bottom shows a command prompt with 'python3 temp1.py'.

```
43 000010000010000000000000000011011
44 00100010111011100000000000000001
45 000010000010000000000000000010111
46 000000000011000010100000100001
47 00000000000101101100000100001
48 00000000000000101100000100001
49 000100101100100100000000000110
50 10001101010011010000000000000000
51 10101101011011010000000000000000
52 0010000101101011000000000000100
53 0010000101001010000000000000100
54 0010001011010110000000000000001
55 00001000001000000000000000110000
56 000000000001110010110000100001
57 00000000000000101110000100001
58 000100101110100100000000000110
59 10001101011011000000000000000000
60 0000110000100000000000001000110
61 0000110000100000000000001001010
62 00100001011011000000000000100
63 001000101110111000000000000001
64 000010000010000000000000111001
65 001001000000010000000000001010
66 000000000000000000000000001100
67 0010010000000100000000000000101
68 000000000000000000000000001100
69 00000000000001001100000100001
70 000000111110000000000000001000
71 001001000000010000000000000001
72 00000000000110001000000100001
73 000000000000000000000000001100
74 000000111110000000000000001000
75 00100100000001000000000000100
76 00111000000001000100000000001
77 001101000100100000000000000000
78 000000000000000000000000001100
79 000000111110000000000000001000
80 00100100000001000000000000100
81 00111000000001000100000000001
82 001101000010010000000000000010
83 000000000000000000000000001100
84 000000111110000000000000001000
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● aryan@Aryans-MacBook-Pro assignment-1 % python3 temp1.py
○ aryan@Aryans-MacBook-Pro assignment-1 %
```

VS Code editor interface showing a file explorer on the left with 'machinecode7' selected. The main editor displays a list of 100 lines of binary code (0s and 1s). The terminal at the bottom shows a command prompt with 'python3 temp1.py'.

```
63 001000101110111000000000000001
64 000010000010000000000000111001
65 001001000000010000000000001010
66 000000000000000000000000001100
67 00100100000001000000000000101
68 000000000000000000000000001100
69 00000000000001001000000100001
70 000000111110000000000000001000
71 001001000000010000000000000001
72 000000000001100001000000100001
73 000000000000000000000000001100
74 000000111110000000000000001000
75 00100100000001000000000000100
76 00111000000001000100000000001
77 001101000010010000000000000000
78 000000000000000000000000001100
79 000000111110000000000000001000
80 00100100000001000000000000100
81 00111000000001000100000000001
82 0011010000100100000000000010
83 000000000000000000000000001100
84 000000111110000000000000001000
85 00100100000001000000000000100
86 00111000000001000100000000001
87 00110100001001000000000010111
88 000000000000000000000000001100
89 000000111110000000000000001000
90 00100100000001000000000000100
91 00111000000001000100000000001
92 00110100001001000000000011001
93 000000000000000000000000001100
94 000000111110000000000000001000
95 00100100000001000000000000100
96 00111000000001000100000000001
97 0011010000100100000000001001101
98 000000000000000000000000001100
99 000000111110000000000000001000
100
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● aryan@Aryans-MacBook-Pro assignment-1 % python3 temp1.py
○ aryan@Aryans-MacBook-Pro assignment-1 %
```


