

Brian Humphries

David Roster

10/25/2018

ECE 156A

## Homework 1: Report

### Algorithm Code:

We did all algorithm code by hand by first writing down our pseudo algorithms into Assembly and then translating into binary and finally hex. Pictured below is all algorithms we used. This part of the lab was challenging because we made many translation errors, but our handwritten steps made it easier to debug because we had written evidence of each instruction made and the steps taken to get each answer.

Handwritten assembly code and diagrams for Bubble Sort. The code is written in MIPS assembly, with binary representations of instructions and comments in English. The diagrams show the state of registers and memory during the sorting process.

**Assembly Code:**

```
addi $t0, $0, 0
addi $t1, $0, 0
addi $t2, $0, 0
addi $t3, $0, 0
addi $t4, $0, 0
addi $t5, $0, 0
addi $t6, $0, 0
addi $t7, $0, 0
addi $t8, $0, 0
addi $t9, $0, 0
addi $t10, $0, 0
addi $t11, $0, 0
addi $t12, $0, 0
addi $t13, $0, 0
addi $t14, $0, 0
addi $t15, $0, 0
addi $t16, $0, 0
addi $t17, $0, 0
addi $t18, $0, 0
addi $t19, $0, 0
addi $t20, $0, 0
addi $t21, $0, 0
addi $t22, $0, 0
addi $t23, $0, 0
addi $t24, $0, 0
addi $t25, $0, 0
addi $t26, $0, 0
addi $t27, $0, 0
addi $t28, $0, 0
addi $t29, $0, 0
addi $t30, $0, 0
addi $t31, $0, 0
```

**Diagrams:**

The diagrams show the state of registers and memory during the sorting process. The registers are labeled with their values, and the memory is shown as a sequence of bytes. The diagrams are used to track the progress of the sorting algorithm and to debug any errors.

Bubble Sort Assembly Code

## AVG Value Assembly Code: Part 1 and Part 2

add: \$2 \$0 \$190

0001 1001 0000 0000 0000 0001 0001 1001 11  
0 0 0 0 0 1 1 3

add \$3 \$0 \$1AC

0001 1011 0000 0000 0000 0001 0011 0010 11  
1 6 0 0 0 1 4 3

Ar

add \$6 \$0 \$0

0000 0000 0000 0000 0000 0000 0010 0100 11  
0 0 0 0 0 0 3 1 3

add ~~\$7 \$0 \$4~~  
\$7 \$0 \$0

0 0 0 0 0 0 11 0011 0011  
2 9 3

add \$4 \$0 \$4

0 0 4 0 0 0 100 0010011  
2 1 3

~~add \$7 \$0 \$3~~

add \$9 \$0 \$182 store bp

0000 0000 0000 0000 0100 1 0010 011  
0 2 0 0 4 9 3

add \$20 \$0 \$1D8 store r10

0000 0000 0000 0000 0100 0010 0011 0011  
0 8 0 0 5 1 3

bay \$2 \$3 \$96 1100000

0000 0011 0001 0000 0000 0000 1100 011  
0 6 2 1 8 0 6 3

1w \$5 \$2 0

0000 0000 0000 00010 010 0010 0000011  
0 0 0 1 2 2 3

Bank Address 25

add: \$6 \$6 \$5

000 000 0010 0011 000 00110 0110011  
0 0 1 2 3 3 3

add: \$2 \$2 \$4

000 000 0000 0100 000 00010 0110011  
0 0 2 2 0 1 3

add: \$7 \$7 1

0000 0000 0001 00111 000 0011 0010011  
0 0 1 3 5 3 7

JALR \$11 \$9 0

0 0 0 01001 000 01010 110011  
0 0 24 8 5 2 7

~~SLR \$8 \$8~~

add: \$12 \$0 3

0000 0000 0011 0010 000 0100 010011  
0 0 3 0 0 1 3

~~SLR \$8 \$6 \$12~~

rd \$8 \$2

0000000 01100 0010 101 0100 010011  
0 0 3 3 4 3 3

[illegible]

~~1v~~ addi \$2 \$0 0 → initialize base address  
 2v addi \$3 \$0 4  
 3v addi \$6 \$0 0 → initialize mem var  
 4v lw \$5 \$2 0 → store first val as mem  
 5v ~~add \$2 \$2 \$3~~ → jump here  
 6v addi \$5 \$0 0 → ~~initialize comparator~~  
 7v lw \$5 \$2 0  
 8v slti \$6 \$5  
 9v beq \$1 \$0 150  
 10v beq \$6 \$5 90  
 11v beq \$1 \$0 172  
 12v beq \$1 \$0 172

2) base Addr  
 3) Addr inc  
 4) Jump addr  
 5) next  
 6) begin

JAL \$4 0  
 Addi \$4 \$0 5

mem < next  
 JAL \$4 0

The above is the naïve assembly code. The code on the left are the ~~cases~~ three cases after comparing two values.

The naïve algorithm is exactly the same except line 9 branches to d'92 and line 11 branches d'60

Minimum/Maximum Assembly Code
-------------------------------

## Insertion Sort Assembly Code

```
for_init:
    addi $2, $0, 1
    addi $7, $0, 8
    addi $7, $7, -1
    addi $11, $0, 4
    addi $8, $0, 1000
    addi $15, $0, 10
    sw $15, 0($8)
    lw $14, 0($8)
    addi $15, $0, 2834
    sw $15, 4($8)
    addi $15, $0, 2883
    sw $15, 8($8)
    addi $15, $0, 947
    sw $15, 12($8)
    addi $15, $0, 99
    sw $15, 16($8)
    addi $15, $0, 2
    sw $15, 20($8)

    addi $8, $0, 1050
    addi $15, $0, 10
    sw $15, 0($8)
    addi $15, $0, 2834
    sw $15, 4($8)
    addi $15, $0, 2883
    sw $15, 8($8)
    addi $15, $0, 947
    sw $15, 12($8)
    addi $15, $0, 99
    sw $15, 16($8)
    addi $15, $0, 2
    sw $15, 20($8)

for_loop:
    blt $7, $2, end_for
    addi $6, $2, -1
    mul $6, $2, $11
    add $6, $8, $6
    lw $4, 0($6)
```

```

while:
    blt $3, $0, endwhile
    mul $6, $3, $11
    add $6, $8, $6
    lw $5, 0($6)
    blt $5, $4, endwhile
    sw $5, 4($6)
    addi $3, $3, -1
    jal $0, while

endwhile:
    mul $6, $3, $11
    add $6, $8, $6
    sw $4, 4($6)
    addi $2, $2, 1
    jal $0 for_loop

end_for:
    jal $0, end_for

```

### 32 Bit-Machine Code:

Our process was transcribing our assembly code on paper into machine code. Once our machine code was ready, we inserted it line by line into a text file and debugged it as we went by running simulations. This process allowed us to quickly identify when new errors would pop up and their most likely reason for occurrence. Some issues we did have during this method is by hardcoding data into our text file, deletions and insertions into our file changed our memory mapping and resulted in bugs.

Our machine codes will be uploaded in a Zip file which will be attached along with this report in an email to our designated TA.

### Testbench File:

Our Testbench will be uploaded in a Zip file which will be attached along with this report in an email to our designated TA.

### Simulation Results:

Our simulation results are displayed below. Any more detailed results of simulations can be given at a moment's notice at the request of the TA.

Bubble sort :

- On every iteration you find a largest number in the array and bubble it out to the last index of the array.

Insertion sort :

- Here you have two regions sorted and unsorted. At Every Iteration you pick up an element from unsorted region and insert at proper location in Sorted region.

Insertion Sort and Bubble sort both have Worst Case  $O(N^2)$ . But if the array is mostly sorted Insertion Sort will perform better. Insertion sort is often used with Quick sort: In Quick Sort after some level of recursion (When array is mostly sorted) Insertion sort is used.

AVG WaveForms
---------------

94	00000000
95	00000000
96	00000000
97	00000000
98	00000000
99	00000000
100	00000000
101	00000001
102	00000113
103	00000010
104	00000004
105	00000400
106	00008005
107	00007300
108	000f0010

Wave - Default									
		Msgs							
+ /vscale_hex_tb/...	0001ff07	00f3e301							
								0001ff07	

MIN WaveForms

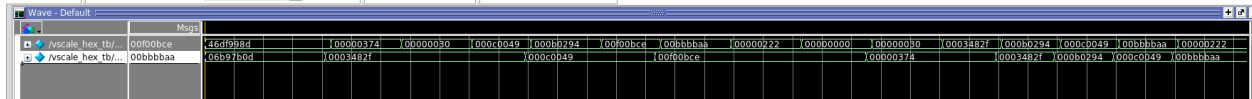
Wave - Default									
		Msgs							
+ /vscale_hex_tb/...	00010010	{b2c28465}{00010010}{00000555}{0000000f}							

MAX WaveForms

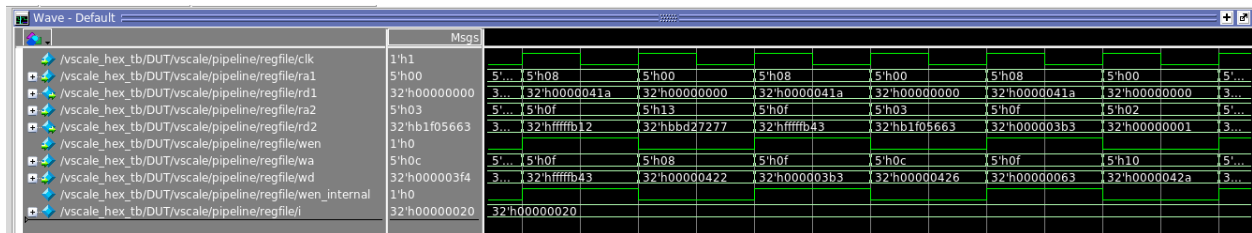
Objects		
Name	Value	Kind
[29]	b2a72665	Packed Array
[28]	96ab582d	Packed Array
[27]	de8e28bd	Packed Array
[26]	2e58495c	Packed Array
[25]	e2ca4e5	Packed Array
[24]	f4007ae8	Packed Array
[23]	e77696ce	Packed Array
[22]	793069f2	Packed Array
[21]	47ecdb8f	Packed Array
[20]	8932d612	Packed Array
[19]	bbd27277	Packed Array
[18]	72aff7e5	Packed Array
[17]	d513d2aa	Packed Array
[16]	e2f784c5	Packed Array
[15]	00000030	Packed Array
[14]	7cfe999	Packed Array
[13]	462df78c	Packed Array
[12]	76d457ed	Packed Array
[11]	1e8dcd3d	Packed Array
[10]	3b23f176	Packed Array
[9]	06d7cd0d	Packed Array
[8]	00000424	Packed Array
[7]	0000009c	Packed Array
[6]	00c03cba	Packed Array
[5]	xxxxxxxx	Packed Array
[4]	00000014	Packed Array
[3]	00000004	Packed Array
[2]	00000424	Packed Array
[1]	0000000X	Packed Array
[0]	12153524	Packed Array
wen_internal	1'h0	Net
i	32'h00000020	Integer

Wave - Default									
		Msgs							
+ /vscale_hex_tb/...	-No...	{b2c28465}{00010...}{00b73e80}{00c03cba}							

## BubbleSort WaveForms



## INSERTION SORT Waveform



## Feedback:

Lab 01 was challenging and tedious due to my partner and I have little experience using Modelsim. The lab however gave us an excellent understanding how valuable a processor is and just how vital its role is in translating higher level code into lower level code. We understood why the creators of MIPS emphasized its 4 principles that governed the conditions on which MIPS worked.

The lab could be better implemented by having a couple workshops on using MIPS and Modelsim in general. Many of our classmates were struggling to finish the lab on time. If the lab were not extended, it would have resulted in less productive work overall. Many of the underclassmen are taking ECE 156A and ECE 154A concurrently so while Modelsim was also introduced in ECE 154, we have only been working with this software for less than two weeks.

## CODE:

19000113

1b000193

00000313

00000393

00400213

02000493

08000513

06218063

00000293

00012283

00530333

00220133

00138393

000485e7

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000



00000000

00300613

00c35433

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000001

00000113

00000010

00000004

00000400

00008005

00007300

000f0010