2021-Eg 211-Computer Architecture

Assignment 2

Submitted By Vihan Vashishth(Imt2021065) And Vineet Priyadarshi(Imt2021018)

Instructions to run the MIPS program files

- Make sure you are in the directory which contains the file

```
IMT2021018 IMT2021065_Q1A/B.asm
```

- Execute the following command in terminal:

java -jar Mars4_5.jar nc <file_name>.asm

OR

- In MARS-MIPS, use the `F3` to assemble the code and `F5` to run it.

Bubble Sort

C equivalent algorithm:

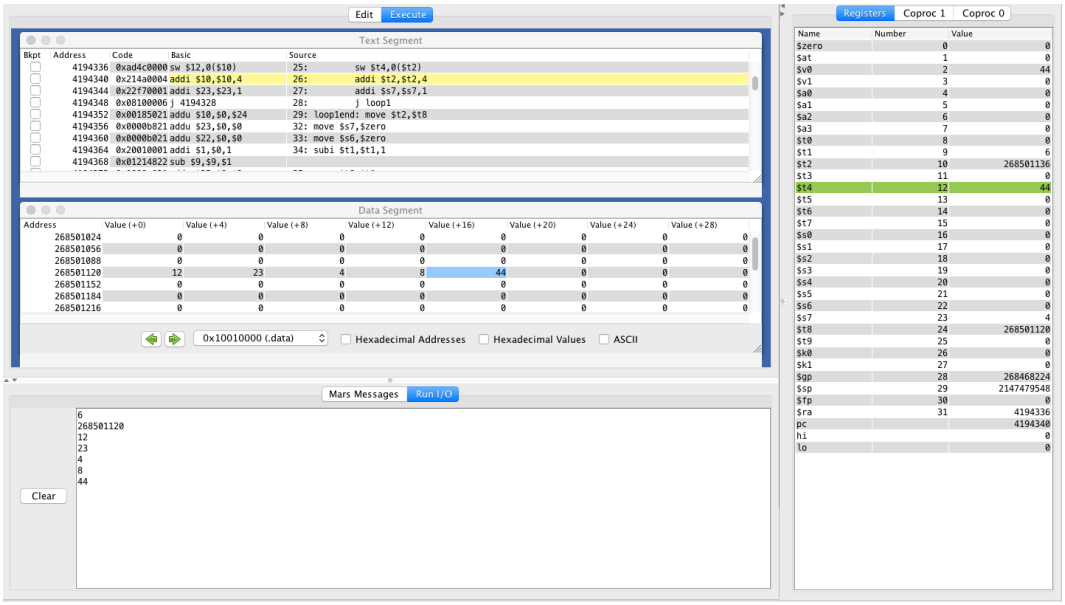
```
for (i = 0; i < n - 1; i++){
    for (j = 0; j < n - i - 1; j++)
    {
        if (arr[j] > arr[j + 1])
        {
            swap(arr[j], arr[j + 1]);
        }
    }
}
```

The MIPS code:

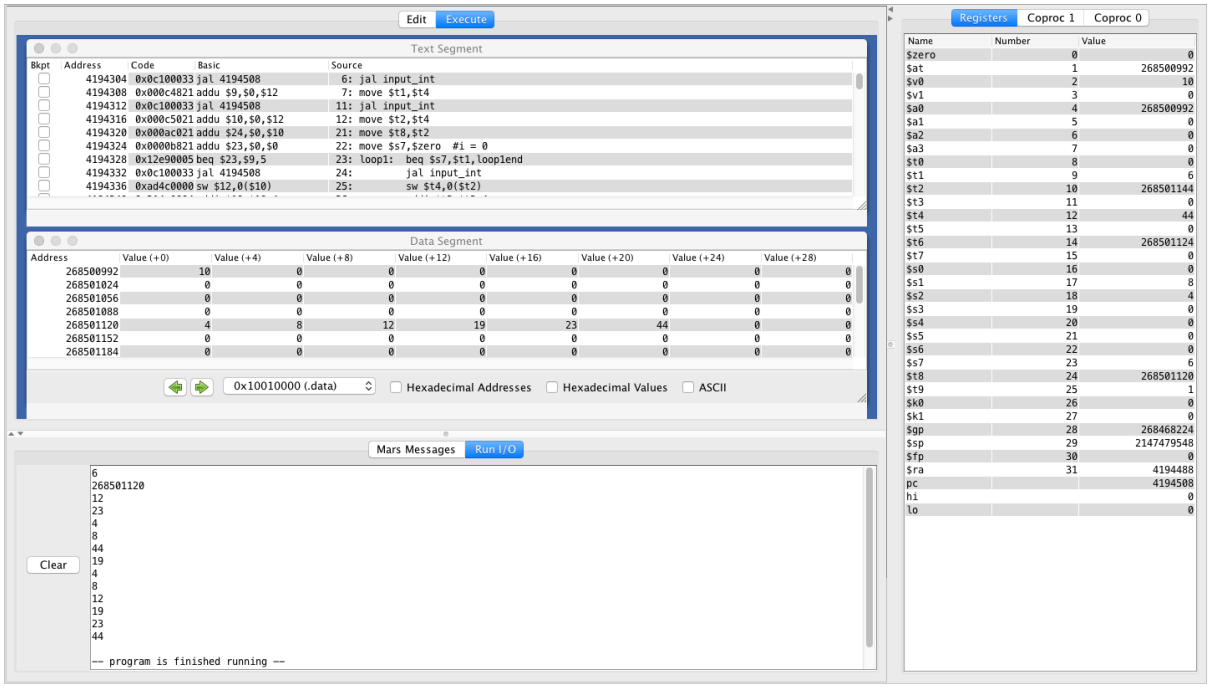
- The program first takes ’n’ as the input where n is the number of integers in our array. This ’n’ is stored in $t1.
- Then input the address of the base of our input array. It’ll be stored in $t2.

- While i<n-1, the program continues inside the outerloop, incrementing the value of ‘i’ by 1. Else it branches to outerloppend and exits the loop.
- While j<n-i-1, the program continues inside the innerloop, incrementing the value of ‘j’ by 1. Else it branches to the innerloopend and exits the inner loop until the ‘i’ is incremented.
- The swap function is called if arr[j]>arr[j+1]. It swaps the numbers stored int j and j+1 location in the input array.
- The numbers are sorted in the same input array and our original array is printed out as a result.

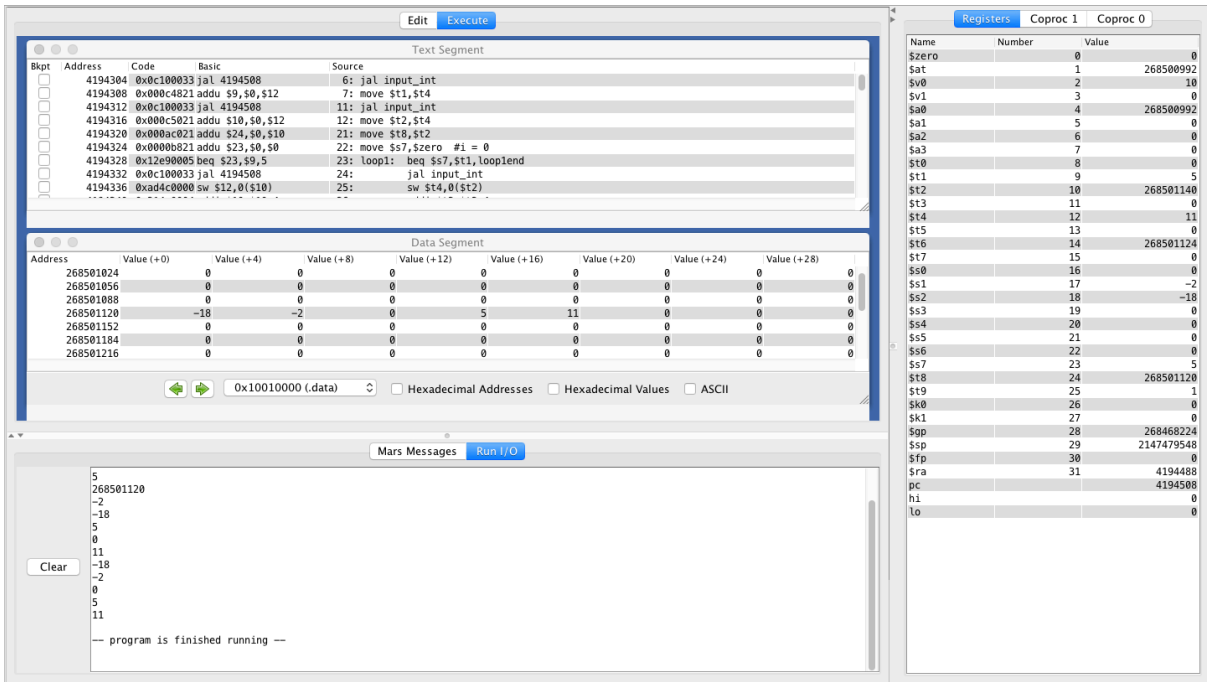
Storing the inputs(12, 23, 4, 8, 44) in the memory:



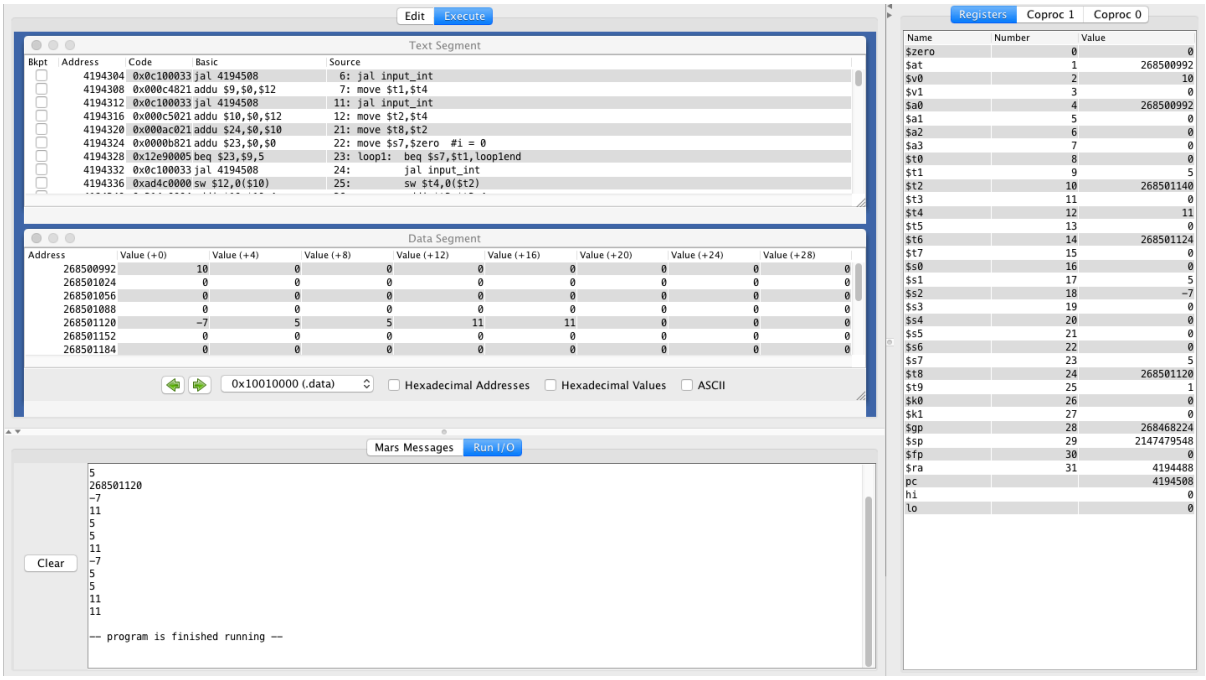
The output:



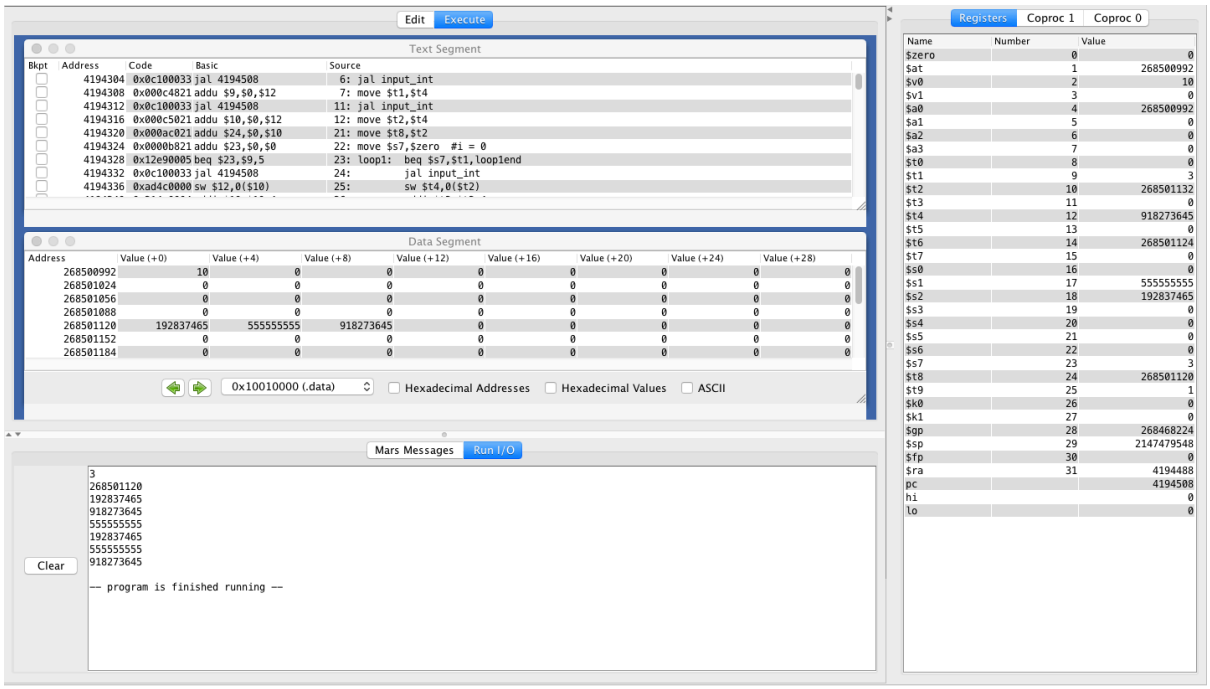
Input With Negative Numbers (-2, -18, 5, 0, 11):



Input With Duplicate Numbers (-7, 11, 5, 5, 11):



Input With Large Numbers (192837465,918273645,555555555):



Sum Of Three Consecutive Elements In An Array

C equivalent algorithm:

```
int A[3*N] = {a list of 3*N values}; int B[N];
int i, j=0;
void main (void)
{
for (i=0; i<N; i++){ B[i] = res_triplet(A,j); j=j+3;
}

}
```

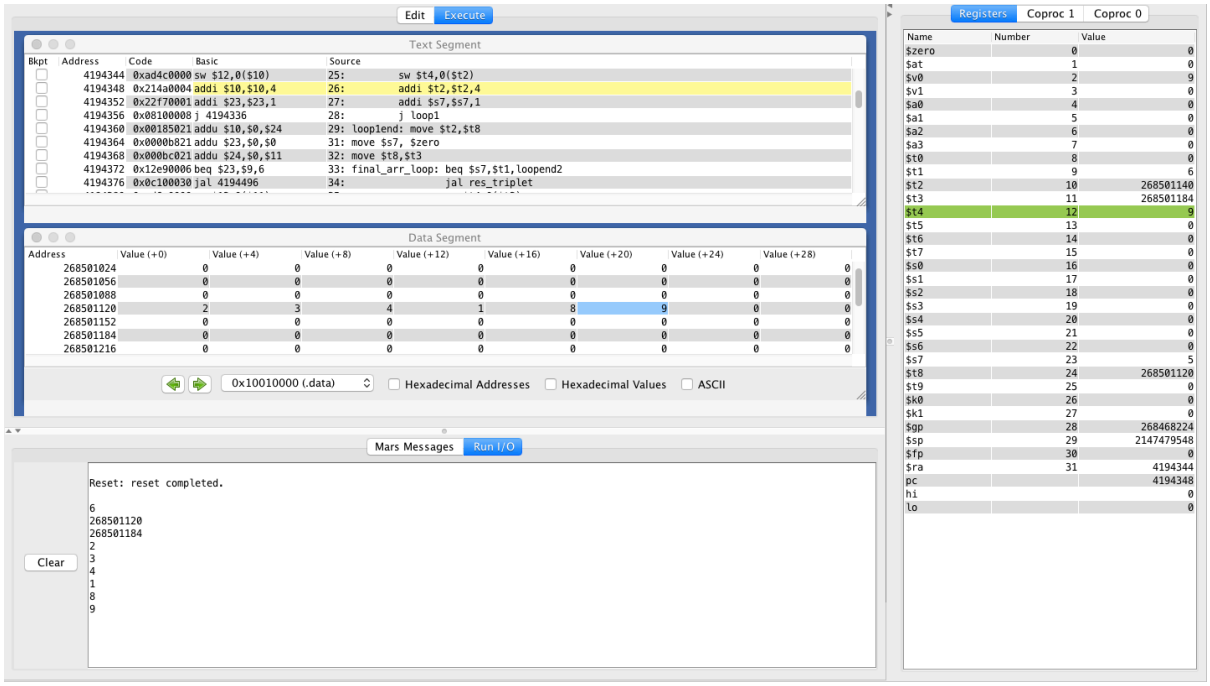
```
int res_triplet(int V[ ], int pos) {
int i, sum=0;
for (i=0; i<3; i++)

sum = sum + V[pos+i]; sum=abs(sum);
return sum;
}
```

• Function abs(int x) returns the absolute value of its input argument.

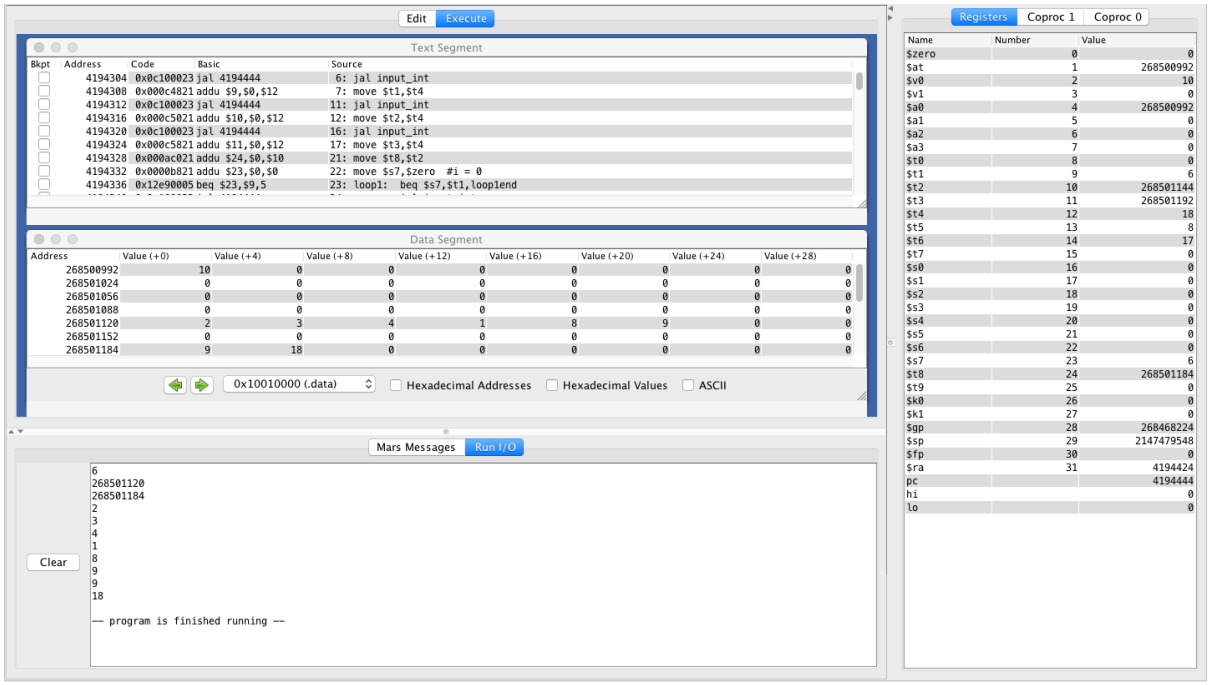
The MIPS code:

- The program first takes ’n’ as the input where n is the number of integers in our array. This ’n’ is stored in $t1.
- Then input the address of the base of our input array. It’ll be stored in $t2.
- Then input the address of the base of our output array. It’ll be stored in $t3.
- After taking in the input final_arr_loop is executed till ‘i’ is less than ’n’.
- res_triplet function is called in the final_arr_loop in each iteration. It sums three consecutive elements and returns the sum’s absolute value.
- The absolute values are stored in $t3’s value’s memory location.
- After the completion of the final_arr_loop, the n numbers who’s starting address is given by $t3 are printed.

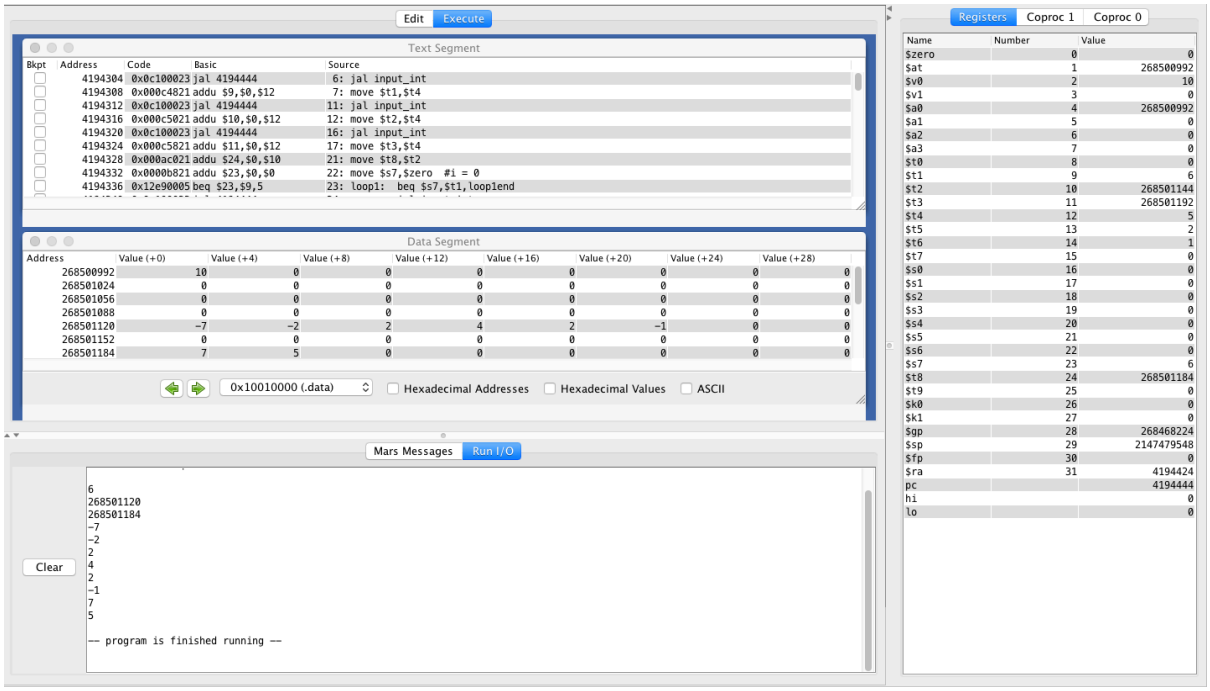
Storing the inputs(2, 3, 4, 1, 8, 9) in the memory:


Edit
Execute
Text Segment
Bkpt Address Code Basic Source
4194344 0xad4c0000 sw $12,0($10) 25: sw $t4,0($t2)
4194348 0x214a0004 addi $10,$10,4 26: addi $t2,$t2,4
4194352 0x22f70001 addi $23,$23,1 27: addi $s7,$s7,1
4194356 0x08100008 j 4194336 28: j loop1
4194360 0x00185021 addu $10,$0,$24 29: loop1end: move $t2,$t8
4194364 0x0000b821 addu $23,$0,$0 31: move $s7, $zero
4194368 0x000bc021 addu $24,$0,$11 32: move $t8,$t3
4194372 0x12e90006 beq $23,$9,6 33: final_arr_loop: beq $s7,$t1,loopend2
4194376 0x0c100030 jal 4194496 34: jal res_triplet
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+12) Value (+16) Value (+20) Value (+24) Value (+28)
268501024 0 0 0 0 0 0 0 0
268501056 0 0 0 0 0 0 0 0
268501088 0 0 0 0 0 0 0 0
268501120 2 3 4 1 8 9 0 0
268501152 0 0 0 0 0 0 0 0
268501184 0 0 0 0 0 0 0 0
268501216 0 0 0 0 0 0 0 0
0x10010000 (.data)
Hexadecimal Addresses
Hexadecimal Values
ASCII
Mars Messages
Run I/O
Reset: reset completed.
6
268501120
268501184
2
3
4
1
8
9
Clear
Registers
Coproc 1
Coproc 0
Name Number Value
$zero 0 0
$at 1 0
$v0 2 9
$v1 3 0
$a0 4 0
$a1 5 0
$a2 6 0
$a3 7 0
$t0 8 0
$t1 9 6
$t2 10 268501140
$t3 11 268501184
$t4 12 9
$t5 13 0
$t6 14 0
$t7 15 0
$s0 16 0
$s1 17 0
$s2 18 0
$s3 19 0
$s4 20 0
$s5 21 0
$s6 22 0
$s7 23 5
$t8 24 268501120
$t9 25 0
$k0 26 0
$k1 27 0
$gp 28 268468224
$sp 29 2147479548
$fp 30 0
$ra 31 4194344
pc 4194348
hi 0
lo 0

The output:


Edit
Execute
Text Segment
Bkpt Address Code Basic Source
4194304 0x0c100023 jal 4194444 6: jal input_int
4194308 0x000c4821 addu $9,$0,$12 7: move $t1,$t4
4194312 0x0c100023 jal 4194444 11: jal input_int
4194316 0x000c5021 addu $10,$0,$12 12: move $t2,$t4
4194320 0x0c100023 jal 4194444 16: jal input_int
4194324 0x000c5821 addu $11,$0,$12 17: move $t3,$t4
4194328 0x000ac021 addu $24,$0,$10 21: move $t8,$t2
4194332 0x0000b821 addu $23,$0,$0 22: move $s7,$zero #i = 0
4194336 0x12e90005 beq $23,$9,5 23: loop1: beq $s7,$t1,loop1end
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+12) Value (+16) Value (+20) Value (+24) Value (+28)
268500992 10 0 0 0 0 0 0 0
268501024 0 0 0 0 0 0 0 0
268501056 0 0 0 0 0 0 0 0
268501088 0 0 0 0 0 0 0 0
268501120 2 3 4 1 8 9 0 0
268501152 0 0 0 0 0 0 0 0
268501184 9 18 0 0 0 0 0 0
0x10010000 (.data)
Hexadecimal Addresses
Hexadecimal Values
ASCII
Mars Messages
Run I/O
6
268501120
268501184
2
3
4
1
8
9
9
18
-- program is finished running --
Clear
Registers
Coproc 1
Coproc 0
Name Number Value
$zero 0 0
$at 1 268500992
$v0 2 10
$v1 3 0
$a0 4 268500992
$a1 5 0
$a2 6 0
$a3 7 0
$t0 8 0
$t1 9 6
$t2 10 268501144
$t3 11 268501192
$t4 12 18
$t5 13 8
$t6 14 17
$t7 15 0
$s0 16 0
$s1 17 0
$s2 18 0
$s3 19 0
$s4 20 0
$s5 21 0
$s6 22 0
$s7 23 6
$t8 24 268501184
$t9 25 0
$k0 26 0
$k1 27 0
$gp 28 268468224
$sp 29 2147479548
$fp 30 0
$ra 31 4194424
pc 4194444
hi 0
lo 0

Input With Negative Numbers (-2, -18, 5, 0, 11):



Instructions to run the RV-FPGA program files

- Make sure you are in the directory which contains the file

```
IMT2021018 IMT2021065_Q2B.C
IMT2021018 IMT2021065_Q2B.S
```

- Execute the following command in VSCODE:

```
run on VSCODE
```

- ADD ‘debug_tool=whisper’ in .ini file.

Bubble Sort

C equivalent algorithm:

```
for (i = 0; i < n - 1; i++){

    for (j = 0; j < n - i - 1; j++)
    {
        if (arr[j] > arr[j + 1])
        {

            swap(arr[j], arr[j + 1]);
        }
    }
}
```

The RV-FPGA code:

- We implement bubbleSort for N=10 numbers.

  V={ 3 , 1 , 4 , 2 , 5 , 10 , 7 , 6 , 9 , 8}

- We define a memory location:

  ```
  #define GPIO_LEDs 0x80001404
  ```

  This is the base address of array V.

- We put array inputs in memory location using WRITE_GPIO.

```
#address = 0x80001404

#WRITE_GPIO(address,V[i])
```

The command stores V[i] in address location 'address'

- After sorting, store the numbers in memory location '80001414'.

```
#address = 0x80001414

#WRITE_GPIO(address,V[i])
```

- We print array in terminal using printfNexys( ).

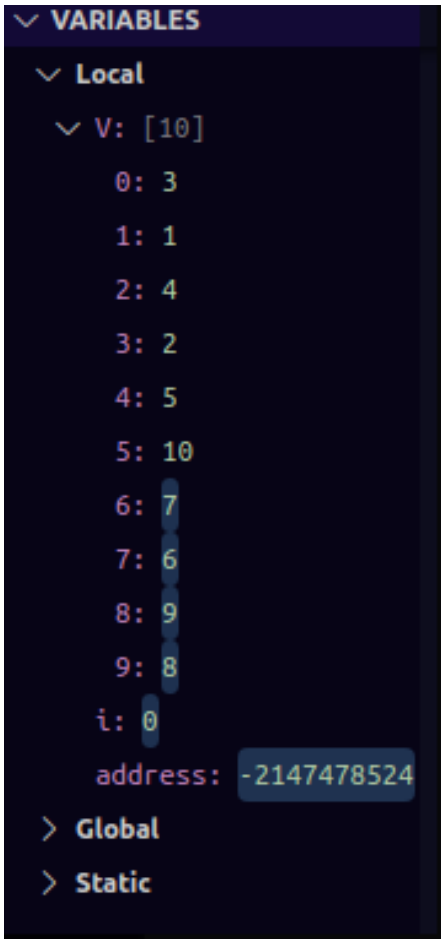
Storing the inputs in the memory

```
1   Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
2  80001400: 00 00 00 00 03 01 04 02 05 0A 07 06 09 08 00 00
3  80001410: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
4  80001420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
5  80001430: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
6  80001440: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
7  80001450: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
8  80001460: 00 00 00 00
9
```

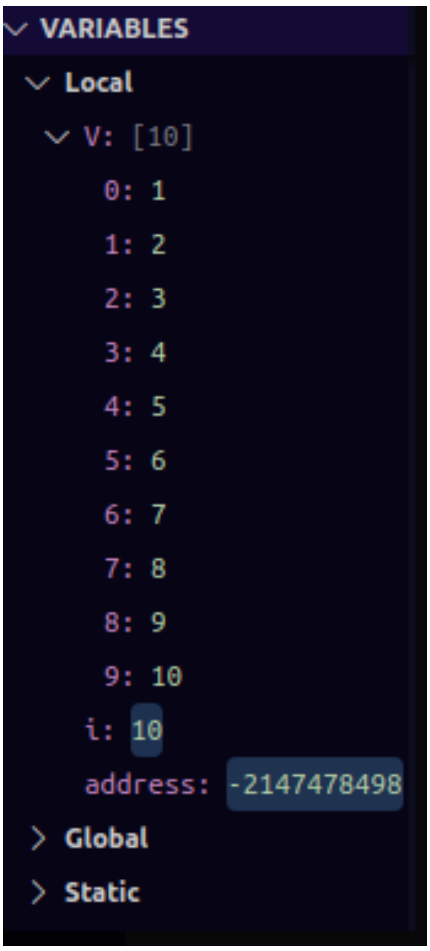
After sorting, the sorted numbers are stored in memory location with base address 80001414.

```
 Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
80001400: 00 00 00 00 03 01 04 02 05 0A 07 06 09 08 00 00
80001410: 00 00 00 00 01 02 03 04 05 06 07 08 09 0A 00 00
80001420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80001430: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80001440: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80001450: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80001460: 00 00 00 00
```

Local Variables before sorting



Local Variables after sorting



Finding Gcd

The RV-FPGA code:

- We first take 10 inputs. t0 register stores the memory location(PC value). Register t1 and t2 store the inputs a and b on which we need to perform the operation.

- We define a memory location:

```
        #li t0 0x80001400
```

```
The value a is stored in 0(t0) and value b is
stored in 4(t0).
Then we update the value of t0 by incrementing
with 10.
```

- After all the inputs are done, we reset t0 back to initial address 0x80001400.

- We then enter a for loop which iterates 10 times. t5 register stores 0 and s1 register stores value 10. We increment t5 after calculating GCD of each pair.

```
#addi t5, t5, 0
```

- Inside the for loop, we have a while loop which calculates the remainder of a and b and stores it in another register t3.

```
#rem t3, t1, t2
```

- If the remainder (in t3) is equal to 0, we jump outside the while loop and the GCD is the value stored in t1. This is then stored in memory location 8(t0), else we continue.

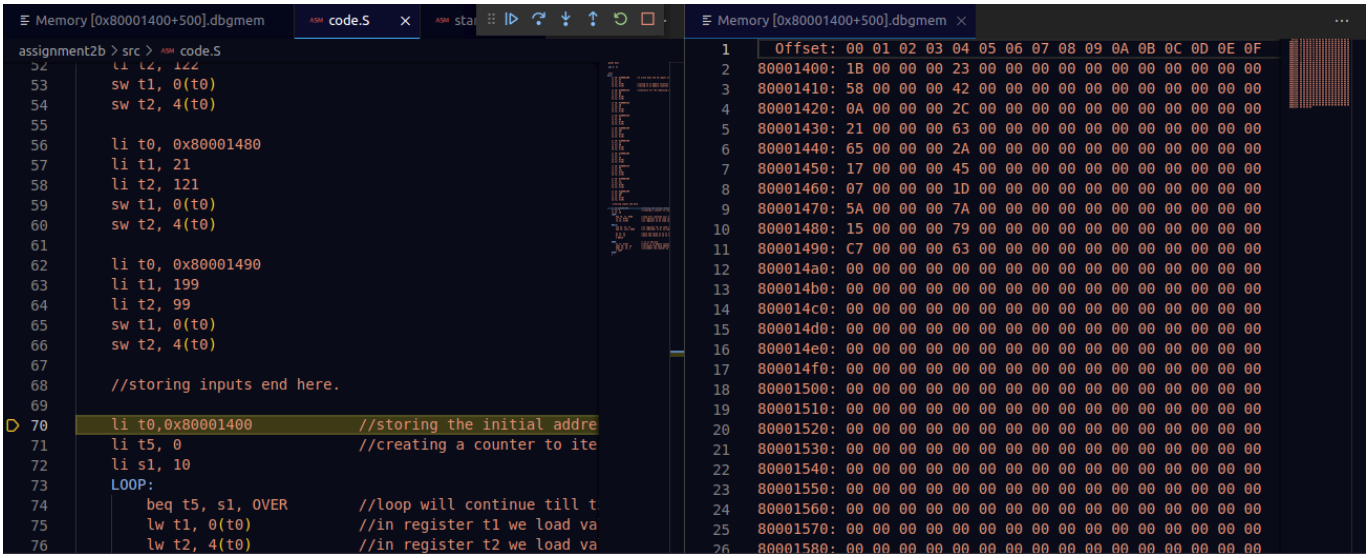
```
#beq t3, zero, DONE
```

- We store the value of t2 register in t1 and t3(remainder) in t2.

```
#mv t1, t2      stores value of t2 in t1

#mv t2, t3      stores value of t3 in t2
```

- We repeat the loop till we reach satisfy the bee condition. Once we find the GCD, we increment PC by 10 and do the same for the next set of inputs in 0(t0) and 4(t0) and save it in 8(t0).

Storing The Input In Memory.



Storing The Input In Memory.