Computer Architecture
Assignment-2

Submitted by:
Madhav Sood (IMT2021009) and Nilay Kamat (IMT2021096)

Q2a.

Instructions to run codes :

- We open terminal in the directory containing the .asm files for Q1 and Q2, i.e., Bubblesort_IMT2021096_IMT2021009.asm and Add3Ele_IMT2021096_IMT2021009.asm respectively.

- Execute the following command : java -jar Mars4_5.jar
  This will open the MARS-MIPS simulator. Open the respective files for Q1 and Q2. The F3 key assembles the code and F5 key is used to run it all in one go.

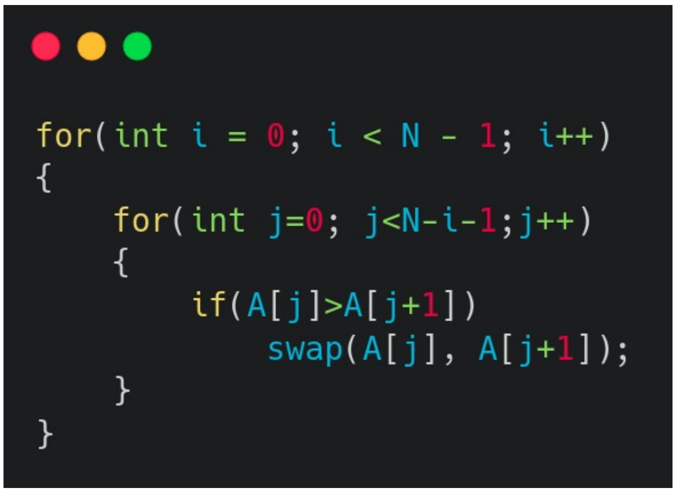
- The first line of input is the number of elements to be stored in the array.

- The second and third line of input should be the starting address of input array and output array respectively.

- The rest of the inputs are the elements of the array.
  (All inputs are to be given in decimal format.)

Q1 :

The algorithm used to sort the numbers is **Bubble sort**.



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

Logic used in the code :

- The program stores the input numbers starting from the starting address provided by the user. (Let total number of input numbers = 'n' and name of the input array be 'A')

- First, we have created the outer loop of bubble sort algorithm which is named as **outer_loop** in the code.

- In the outer_loop, value in $s6 register is used as the loop counter 'i' for the outer loop and we have also simultaneously stored the value of upper bound for the inner loop i.e (n-i-1) in $s0 register.

- After executing the above outer_loop, we jump to the inner loop named '**loopi**'.

- In the inner loop, value in $s4 is used as the loop counter 'j'. The value of A[j] is stored in $t4 and the value of A[j+1] is stored in $s3 registers respectively. The condition for swapping is checked by using the 'slt' instruction followed by the 'beq' instruction. If (value in $s3 < value in $t4) i .e A[j+1]<A[j], then we move into the **swap** function, else we continue the inner loop.

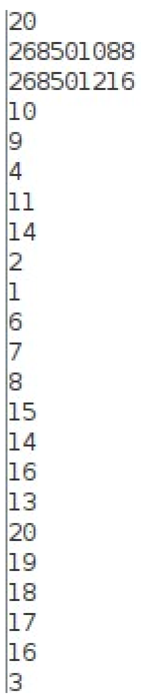
- If (value in $s3 < value in $t4), then **swap** function executes. In this we swap the values in the registers and also swap the values in the respective memory locations. Basically after execution of the swap function, the value of A[j+1] will be in 'j'th memory location and the value of A[j] will be in 'j+1'th memory location.

- We keep executing the above logic till we break out of the outer_loop with the help of the loop counter. After this whole execution, the input array 'A' will be sorted.

- Following this, the loop named **'loop_ou'** has been implemented in the code to store the values of the sorted input array into the output array.

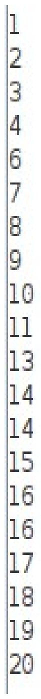
EXAMPLE 1 : GENERIC EXAMPLE

INPUTS:

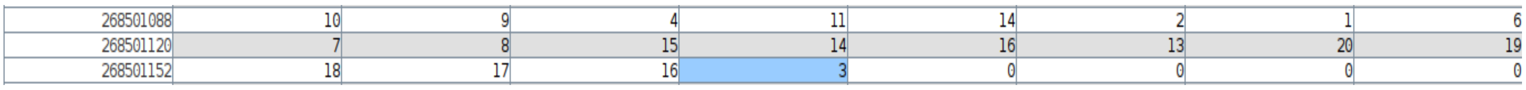


```
20
268501088
268501216
10
9
4
11
14
2
1
6
7
8
15
14
16
13
20
19
18
17
16
3
```

OUTPUT:

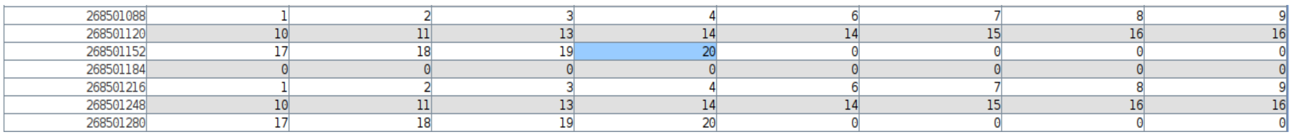


```
1
2
3
4
6
7
8
9
10
11
13
14
14
15
16
16
17
18
19
20
```



268501088	10	9	4	11	14	2	1	6
268501120	7	8	15	14	16	13	20	19
268501152	18	17	16	3	0	0	0	0

UNSORTED INPUT ARRAY IN MEMORY

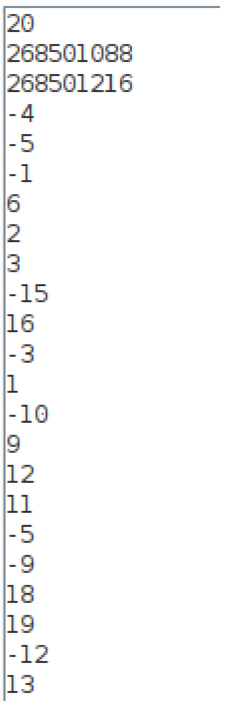


268501088	1	2	3	4	6	7	8	9
268501120	10	11	13	14	14	15	16	16
268501152	17	18	19	20	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	1	2	3	4	6	7	8	9
268501248	10	11	13	14	14	15	16	16
268501280	17	18	19	20	0	0	0	0

SORTED INPUT AND OUTPUT ARRAYS
IN MEMORY

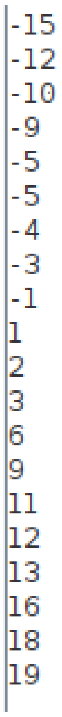
EXAMPLE 2: NEGATIVE NUMBERS

INPUTS:

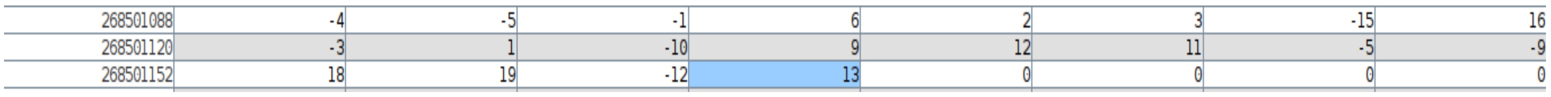


```
20
268501088
268501216
-4
-5
-1
6
2
3
-15
16
-3
1
-10
9
12
11
-5
-9
18
19
-12
13
```

OUTPUT:

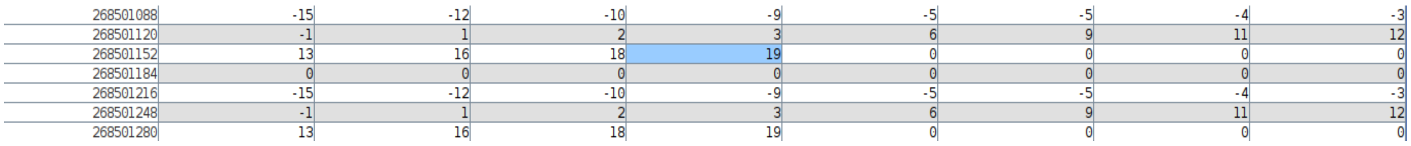


```
-15
-12
-10
-9
-5
-5
-4
-3
-1
1
2
3
6
9
11
12
13
16
18
19
```



268501088	-4	-5	-1	6	2	3	-15	16
268501120	-3	1	-10	9	12	11	-5	-9
268501152	18	19	-12	13	0	0	0	0

UNSORTED INPUT ARRAY IN MEMORY



268501088	-15	-12	-10	-9	-5	-5	-4	-3
268501120	-1	1	2	3	6	9	11	12
268501152	13	16	18	19	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	-15	-12	-10	-9	-5	-5	-4	-3
268501248	-1	1	2	3	6	9	11	12
268501280	13	16	18	19	0	0	0	0

SORTED INPUT AND OUTPUT ARRAYS
IN MEMORY

Q2:

Logic used in the code :

- The number of elements to be stored in the input array should be a multiple of 3 (To be taken care by the user when inputting the number of elements) (Let number of elements in input array 'A' be 3*n)

- We have taken a loop named '**loop_ou**' for executing the logic in the question. In this loop the sum of 3 consecutive elements is stored in the output array 'B'.

- Value in $s0 is used as the loop counter. Value in $s1 register is used as counter for printing the output array. For storing absolute value of sum, 'bltz' instruction is used which jumps to '**negate**' if the sum of the 3 consecutive elements is negative.

- In the negate function, 'negu' instruction makes the sum of the 3 consecutive elements positive and stores it in the output array 'B' while also incrementing all the counters and jumping back to the loop.

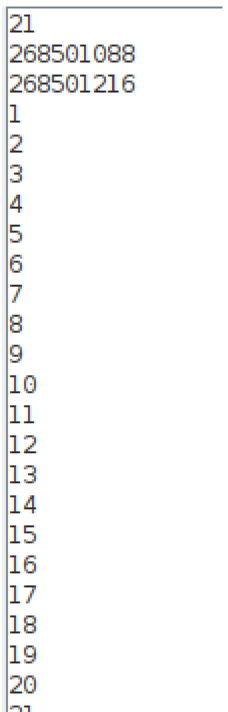
- The exit condition is checked by using the value in $s0 (loop counter) and (3*n-3) which is stored in $t1 register. ($t1 register originally had 3*n elements, but was decremented by 3 to store 3*n-3 for upper bound for the loop)

- After exiting the loops, all the original values of $t1,$t2 and $t3 are restored.

- In the printing loop, the upper bound is changed to the value in register $s1.

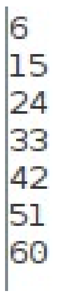
EXAMPLE 1 : GENERIC EXAMPLE

INPUTS:

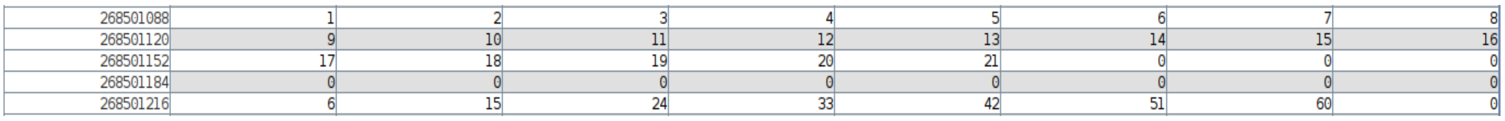


```
21
268501088
268501216
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

OUTPUT:



```
6
15
24
33
42
51
60
```

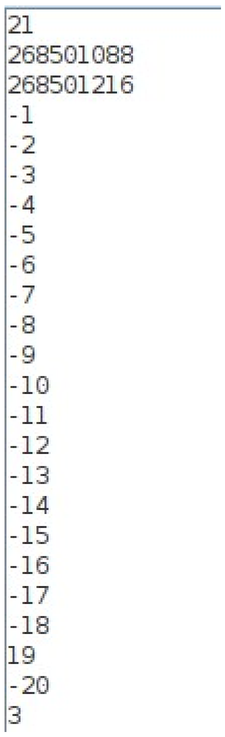


268501088	1	2	3	4	5	6	7	8
268501120	9	10	11	12	13	14	15	16
268501152	17	18	19	20	21	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	6	15	24	33	42	51	60	0

INPUT AND OUTPUT ARRAYS
IN MEMORY

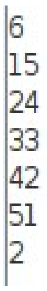
EXAMPLE 2 : WITH NEGATIVE NUMBERS

INPUTS:

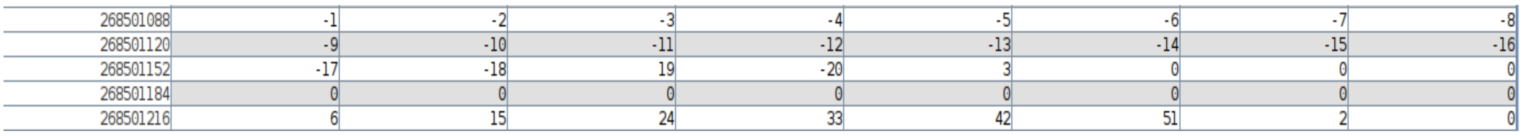


```
21
268501088
268501216
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
-11
-12
-13
-14
-15
-16
-17
-18
19
-20
3
```

OUTPUT:



```
6
15
24
33
42
51
2
```



268501088	-1	-2	-3	-4	-5	-6	-7	-8
268501120	-9	-10	-11	-12	-13	-14	-15	-16
268501152	-17	-18	19	-20	3	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	6	15	24	33	42	51	2	0

INPUT AND OUTPUT ARRAYS
IN MEMORY

Q2b. (RV-FPGA)

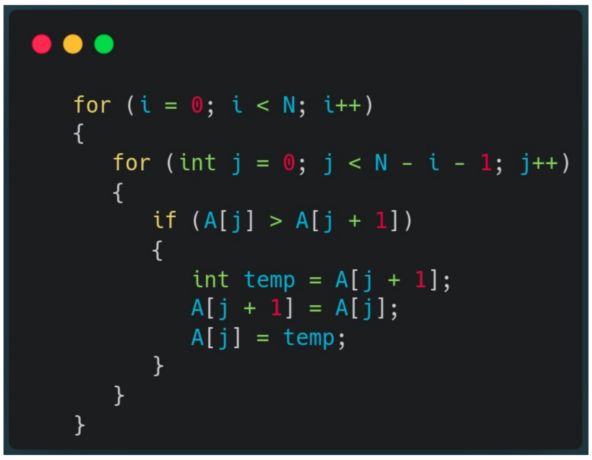
Instructions to run codes :

- To run the codes for this question, we will need to use Visual Studio Code.
- In VSCode, we need the PlatformIO extension wherein we create a new project and place the code file in the **sys** folder.
- We then use "**Run and Debug**" to see how the code works.
- We add memory locations to see the changes that take place in the memory. Memory locations to be added are:
    - 0x80001400 - 64 bytes for Q1 and Q2.
    - 0x80001500 - 64 bytes for Q1 and Q2.

Q1 :

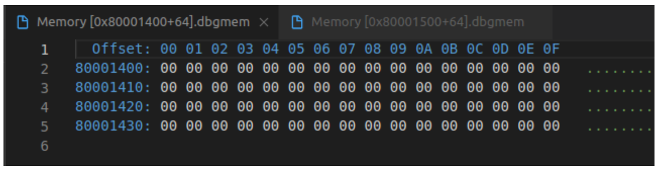
Logic used in the code :

- We have created an array A of N elements. (N has been defined before the main() function) and written the values to the memory, starting from memory location x80001400.
- We then used the bubble sort algorithm to rearrange the elements of the array in ascending order.
- We then wrote the sorted array into memory, starting from memory location x80001500.
- Writing to memory is done by using **GPIO_LED's** and **GPIO_Sorted** which have been defined in the start of the code file.

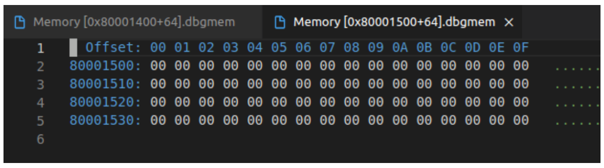


```
for (i = 0; i < N; i++)
{
    for (int j = 0; j < N - i - 1; j++)
    {
        if (A[j] > A[j + 1])
        {
            int temp = A[j + 1];
            A[j + 1] = A[j];
            A[j] = temp;
        }
    }
}
```

Memory locations x80001400(Unsorted array) and x80001500(Array after sorting) before code is run.

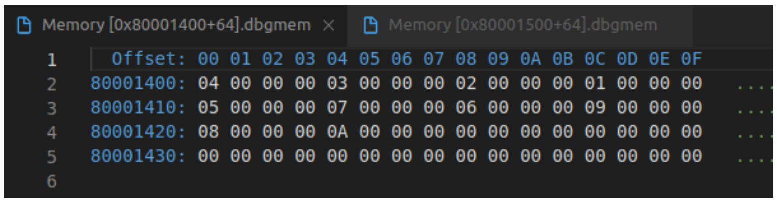


```
  Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
80001400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ........
80001410: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ........
80001420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ........
80001430: 00 00 00 00 00 00 00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

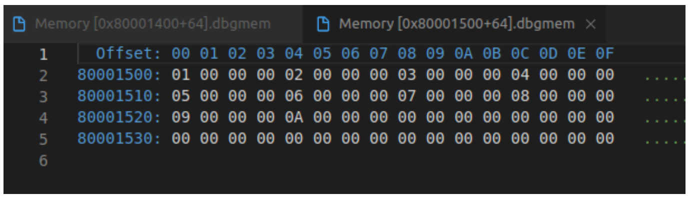


```
  Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
80001500: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ......
80001510: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ......
80001520: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ......
80001530: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ......
```

Memory locations x80001400(Unsorted array) and x80001500(Array after sorting) after code is run.



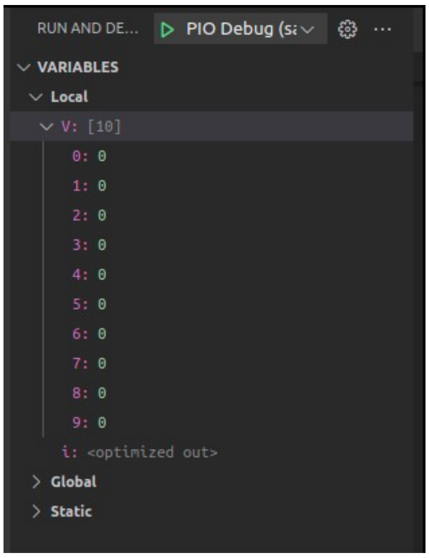
```
  Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
80001400: 04 00 00 00 03 00 00 00 02 00 00 00 01 00 00 00    ...
80001410: 05 00 00 00 07 00 00 00 06 00 00 00 09 00 00 00    ...
80001420: 08 00 00 00 0A 00 00 00 00 00 00 00 00 00 00 00    ...
80001430: 00 00 00 00 00 00 00 00 00 00 00 00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



```
  Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
80001500: 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00    ....
80001510: 05 00 00 00 06 00 00 00 07 00 00 00 08 00 00 00    ....
80001520: 09 00 00 00 0A 00 00 00 00 00 00 00 00 00 00 00    ....
80001530: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ....
```

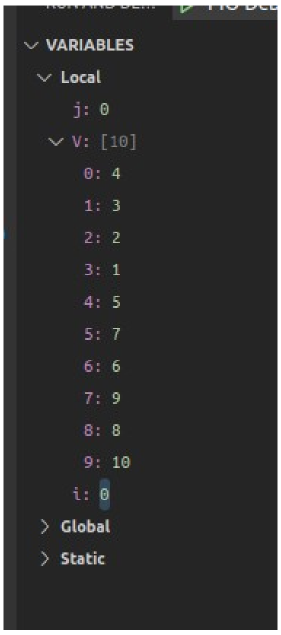
Array contents in variables section :

Pic A - before array is created.
Pic B - after array creation but before bubble sort algorithm is implemented.
Pic C - Sorted array.



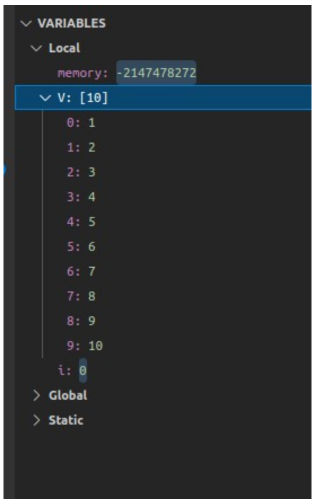
```
RUN AND DE...  PIO Debug (s
VARIABLES
  Local
    v: [10]
      0: 0
      1: 0
      2: 0
      3: 0
      4: 0
      5: 0
      6: 0
      7: 0
      8: 0
      9: 0
    i: <optimized out>
  > Global
  > Static
```

Pic - A



```
VARIABLES
  Local
    j: 0
    v: [10]
      0: 4
      1: 3
      2: 2
      3: 1
      4: 5
      5: 7
      6: 6
      7: 9
      8: 8
      9: 10
    i: 0
  > Global
  > Static
```

Pic - B



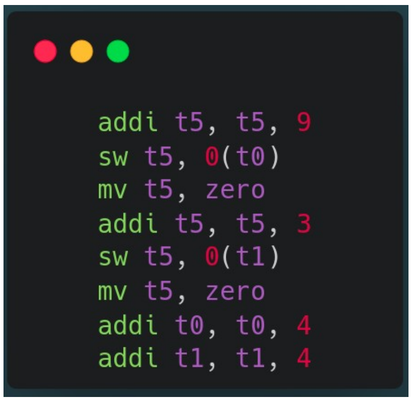
```
VARIABLES
  Local
    memory: -2147478272
    v: [10]
      0: 1
      1: 2
      2: 3
      3: 4
      4: 5
      5: 6
      6: 7
      7: 8
      8: 9
      9: 10
    i: 0
  > Global
  > Static
```

Pic - C

Q2 :

Logic used in the code :

- We have used 3 arrays to implement the code to find GCD.
  - Array A stores the 'a' elements. (Base address 0x80001400 is stored in register t0)
  - Array B stores the 'b' elements. (Base address 0x80001500 is stored in register t1)
  - Array C stores the gcd value of 'a' and 'b'. (Base address 0x80001600 is stored in register t2)
- We have used the following code to add elements in arrays A and B. The elements used in our test-case for the arrays are:
  - Array A - 9, 12, 2, 13, 30, 69, 3, 4, 16, 25.
  - Array B - 3, 7, 8, 26, 29, 23, 2, 16, 30, 75.



```
addi t5, t5, 9
sw t5, 0(t0)
mv t5, zero
addi t5, t5, 3
sw t5, 0(t1)
mv t5, zero
addi t0, t0, 4
addi t1, t1, 4
```

- Using the Euclidian algorithm to solve for the GCD value, we check for the greater of the 2 values and reduce it to the remainder obtained by dividing it by the smaller number. When any one of them becomes 0, we print the other value as the GCD.

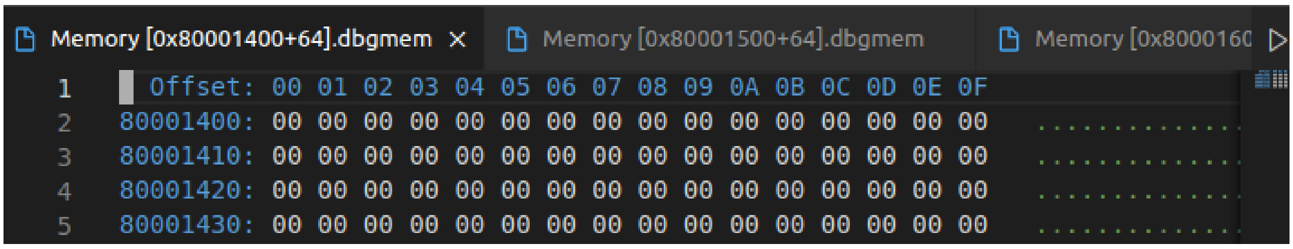
- For every a[i] and b[i] whose GCD is to be calculated, we use **inp:** to store a[i] and b[i] in registers s6 and s1 respectively.

- We then proceed to **loop:** where we check if either of these values are 0. if Yes, we jump to end1 or end2 to store the GCD conditionally.
- If none are 0, we check the condition of a[i]>=b[i]. If true, we jump to L1 where a[i] is updated to a[i]%b[i] - remainder obtained on division of a[i] by b[i].
  If false, we update b[i] to b[i]%a[i].

  We continue to update these values until one of the numbers becomes 0, in which we go back to previous step.

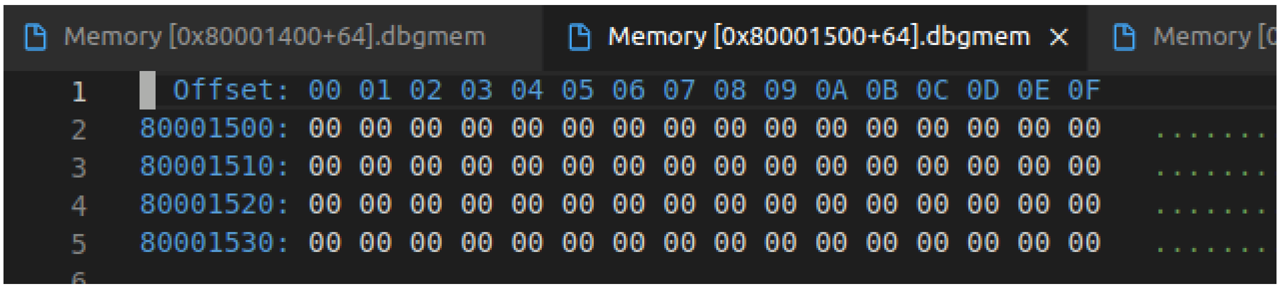
- We have also used a counter to help in termination of the loop after calculation of GCD's of all numbers in the arrays. This counter is stored in register s5 and is compared with register s3 which stores the value of N(number of elements in array).

Memory locations and memory for all arrays before code implementation:



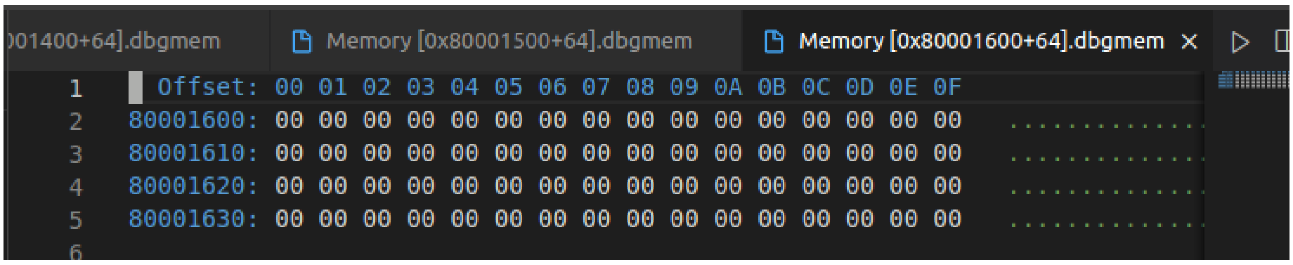
```
Memory [0x80001400+64].dbgmem    Memory [0x80001500+64].dbgmem    Memory [0x8000160
 Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
80001400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .............
80001410: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .............
80001420: 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   .............
```

Elements of array A



```
Memory [0x80001400+64].dbgmem    Memory [0x80001500+64].dbgmem    Memory [(
 Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
80001500: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .......
80001510: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .......
80001520: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .......
80001530: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .......
```

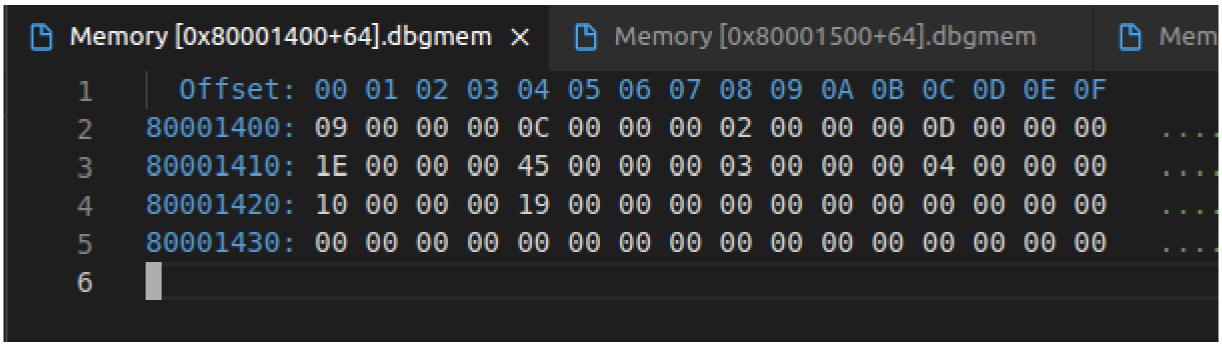
Elements of array B



```
)01400+64].dbgmem    Memory [0x80001500+64].dbgmem    Memory [0x80001600+64].dbgmem
 Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
80001600: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .............
80001610: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .............
80001620: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .............
80001630: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .............
```

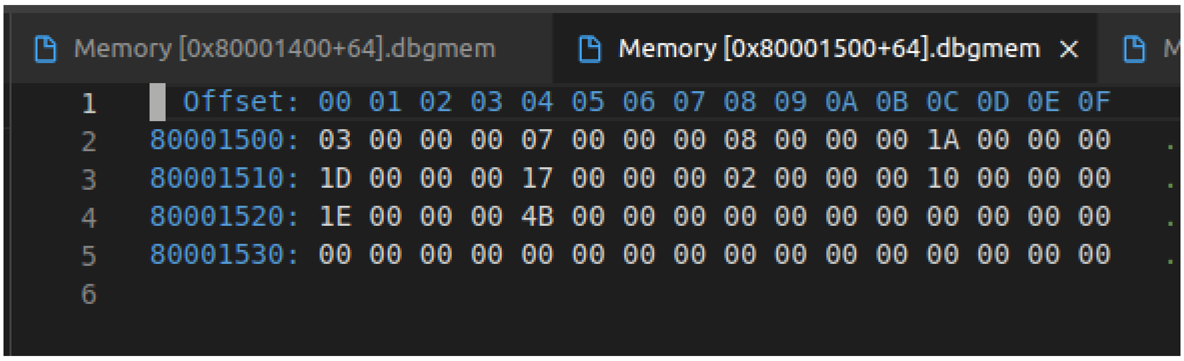
Elements of array C

Memory locations and memory for all arrays before code implementation:



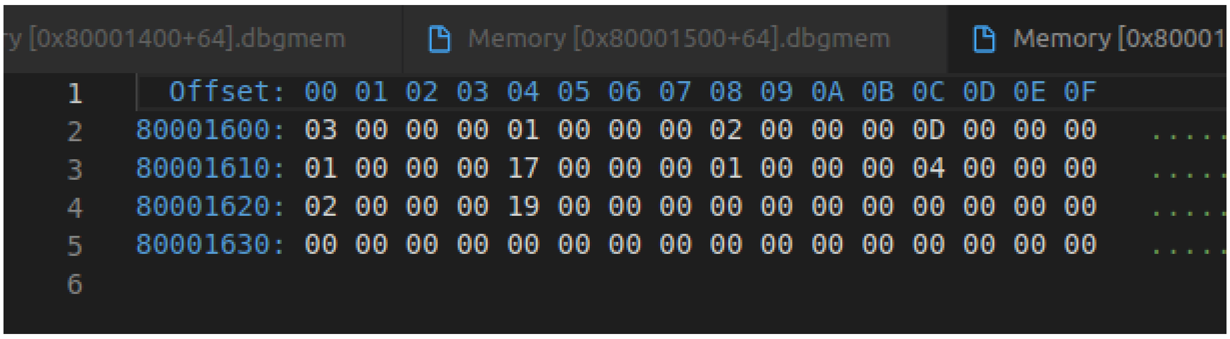
```
 Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
80001400: 09 00 00 00 0C 00 00 00 02 00 00 00 0D 00 00 00   ....
80001410: 1E 00 00 00 45 00 00 00 03 00 00 00 04 00 00 00   ....
80001420: 10 00 00 00 19 00 00 00 00 00 00 00 00 00 00 00   ....
80001430: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ....
```

Elements of array A



```
 Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
80001500: 03 00 00 00 07 00 00 00 08 00 00 00 1A 00 00 00   .
80001510: 1D 00 00 00 17 00 00 00 02 00 00 00 10 00 00 00   .
80001520: 1E 00 00 00 4B 00 00 00 00 00 00 00 00 00 00 00   .
80001530: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .
```

Elements of array B



```
 Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
80001600: 03 00 00 00 01 00 00 00 02 00 00 00 0D 00 00 00   .....
80001610: 01 00 00 00 17 00 00 00 01 00 00 00 04 00 00 00   .....
80001620: 02 00 00 00 19 00 00 00 00 00 00 00 00 00 00 00   .....
80001630: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .....
```

Elements of array C
