Computer Architecture Assignment 1

Niranjan Gopal (IMT2022543)

Akash Sridhar (IMT2022501)

Question 1

Program implemented:

1)Sorting (ascending order)

2)Matrix Multiplication

Sorting:

We have implemented bubble sort algorithm. Below is the algorithm in python.

```python
def bubble_sort(arr):
    N = len(arr)

    for i in range(N):
        for j in range(0, N - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

    return arr
```

Mars compiler output:

```
Enter choice ( 1/2/3 )  :1
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): 268501280
Enter the integer: 1
Enter the integer: 8
Enter the integer: 4
Enter the integer: 6
Enter the integer: 4
Enter the integer: 5
Enter the integer: 13
Enter the integer: 99
1
4
4
5
6
8
13
99
```

Working:

- The program starts by defining some registers:
- **$t0**: Number of inputs (N)
- **$t1**: Input array address
- **$t2**: Output array address
- The program enters a loop to read user input, store it in arrays, and increment the array addresses and a loop variable **s0**.
- After reading all inputs, it enters an outer loop, tracking iterations with **s0**.
- Inside the outer loop, an inner loop is initialized to compare adjacent elements and swap them if necessary. This continues until the inner loop completes.
- After each inner loop iteration, **s1** is incremented, and the outer loop iterates until **s0** equals **t0**.
- After each outer loop iteration, the next largest element will be in it's correct position in the array.
- Once the outer loop finishes, it enters a loop to print the sorted array values one by one.

Matrix Multiplication:

The following algorithm was used for multiplying 2 square matrices of ANY ORDER.

```
for k in range(N*N):

    if k and k%N == 0:
        row_offset += 1
        clm_offset = 0

    for j in range(N):
        C[k] = A[row_offset*N + j] * B[clm_offset + j*N]

    clm_offset +=1
```

Since there is no concept of 2D arrays in MIPS, traditional matrix multiplication would not work. Therefore, since the only way to represent matrix is 1 dimensional array, we used offsets for each row.

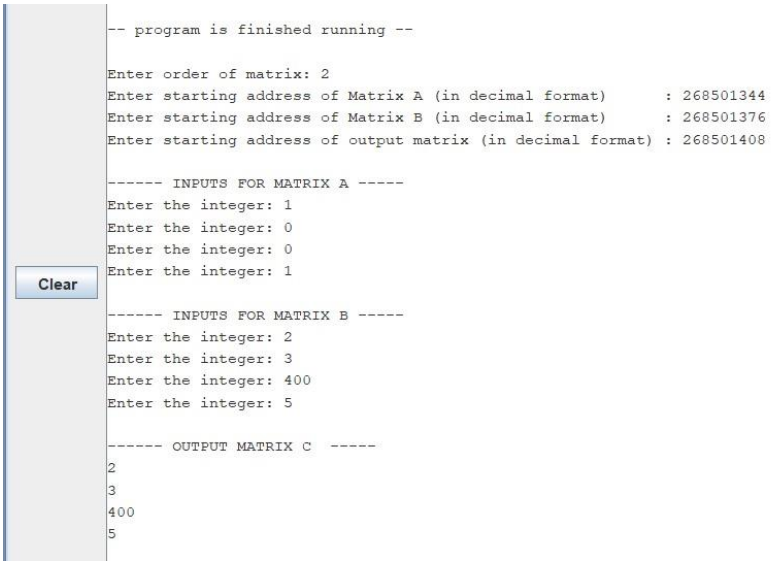
Each matrix is an array of N squared elements.

Meaning, the first N elements will be the first row of the matrix. Next N elements will be the second row of the matrix and so on.

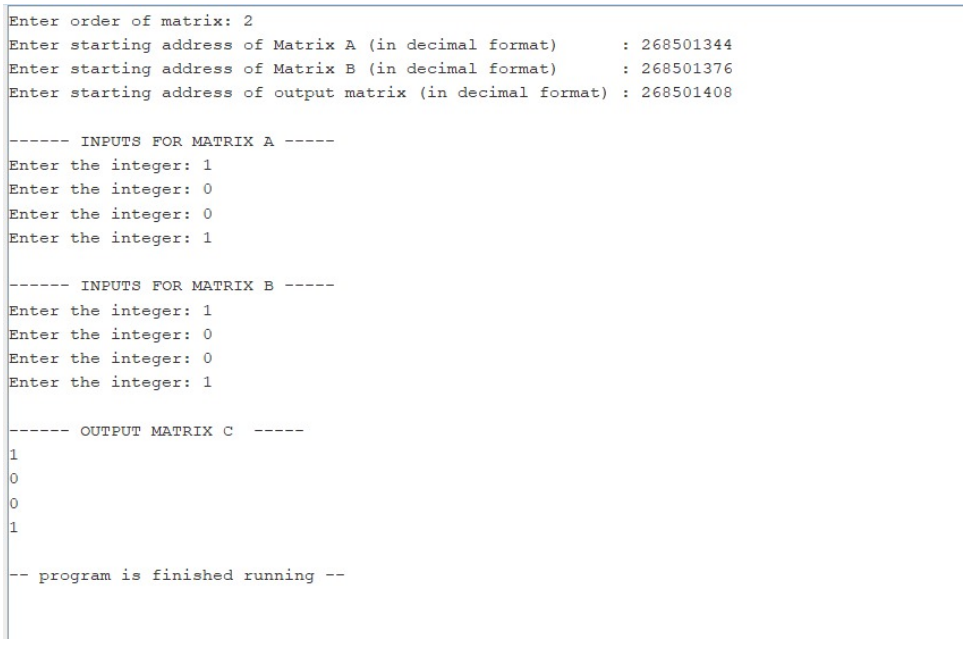
Since there is no concept of mod in MIPS, this was another obstacle we faced. We found that there was a concept of hi and lo register and how it works with mul and div operations which we used to solve this problem.

Mars compiler output:

Multiplying square matrices of order 2



Multiplying 2 identity matrices



Multiplying 2 4x4 matrices !

Clear

```
1
Enter choice ( 1/2/3 )  :2
Enter order of matrix: 4
Enter starting address of Matrix A (in decimal format)      : 268501536
Enter starting address of Matrix B (in decimal format)      : 268501632
Enter starting address of output matrix (in decimal format) : 268501728

------ INPUTS FOR MATRIX A -----
Enter the integer: 1
Enter the integer: 2
Enter the integer: 3
Enter the integer: 4
Enter the integer: 5
Enter the integer: 6
Enter the integer: 7
Enter the integer: 8
Enter the integer: 9
Enter the integer: 10
Enter the integer: 11
Enter the integer: 12
Enter the integer: 13
Enter the integer: 14
Enter the integer: 15
Enter the integer: 16

------ INPUTS FOR MATRIX B -----
Enter the integer: 1
Enter the integer: 0
Enter the integer: 0
Enter the integer: 0
Enter the integer: 0
Enter the integer: 1
Enter the integer: 0
Enter the integer: 0
Enter the integer: 0
Enter the integer: 0
Enter the integer: 0
Enter the integer: 1
```

Clear

```
------ INPUTS FOR MATRIX B -----
Enter the integer: 1
Enter the integer: 0
Enter the integer: 0
Enter the integer: 0
Enter the integer: 0
Enter the integer: 1
Enter the integer: 0
Enter the integer: 0
Enter the integer: 0
Enter the integer: 0
Enter the integer: 0
Enter the integer: 1
Enter the integer: 0
Enter the integer: 0
Enter the integer: 0
Enter the integer: 1

------ OUTPUT MATRIX C  -----
1
2
0
7
5
6
0
15
5
6
0
15
5
6
0
15
Enter choice ( 1/2/3 )  :3

-- program is finished running --
```

Error Encountered and Progress

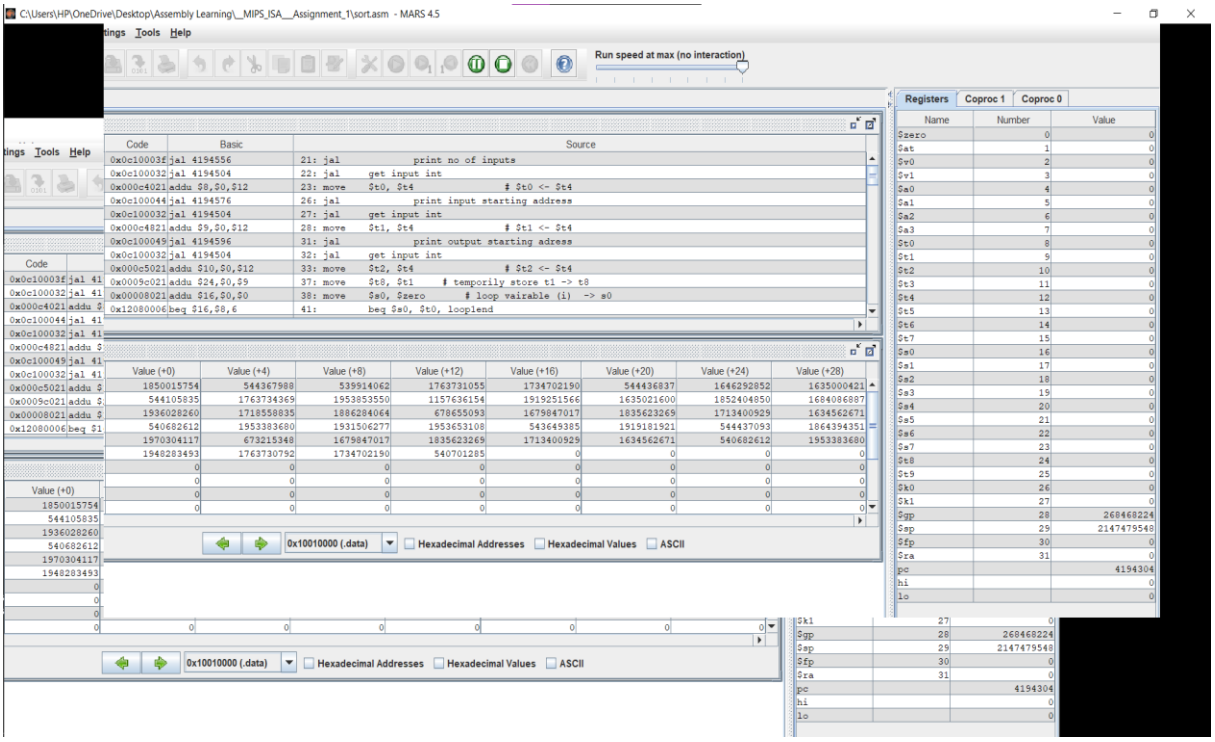
```
Runtime exception at 0x004000bc: invalid integer input
```

```
Runtime exception at 0x00400038: address out of range 0x00000000
```

```
: Runtime exception at 0x00400100: fetch address not aligned on word boundary 0x10010141
```

- move is equivalent to add

```
1  move  $t8, $t1
2  add   $t8, $t1, $0
3
4
5
6  # both are the same thing
```

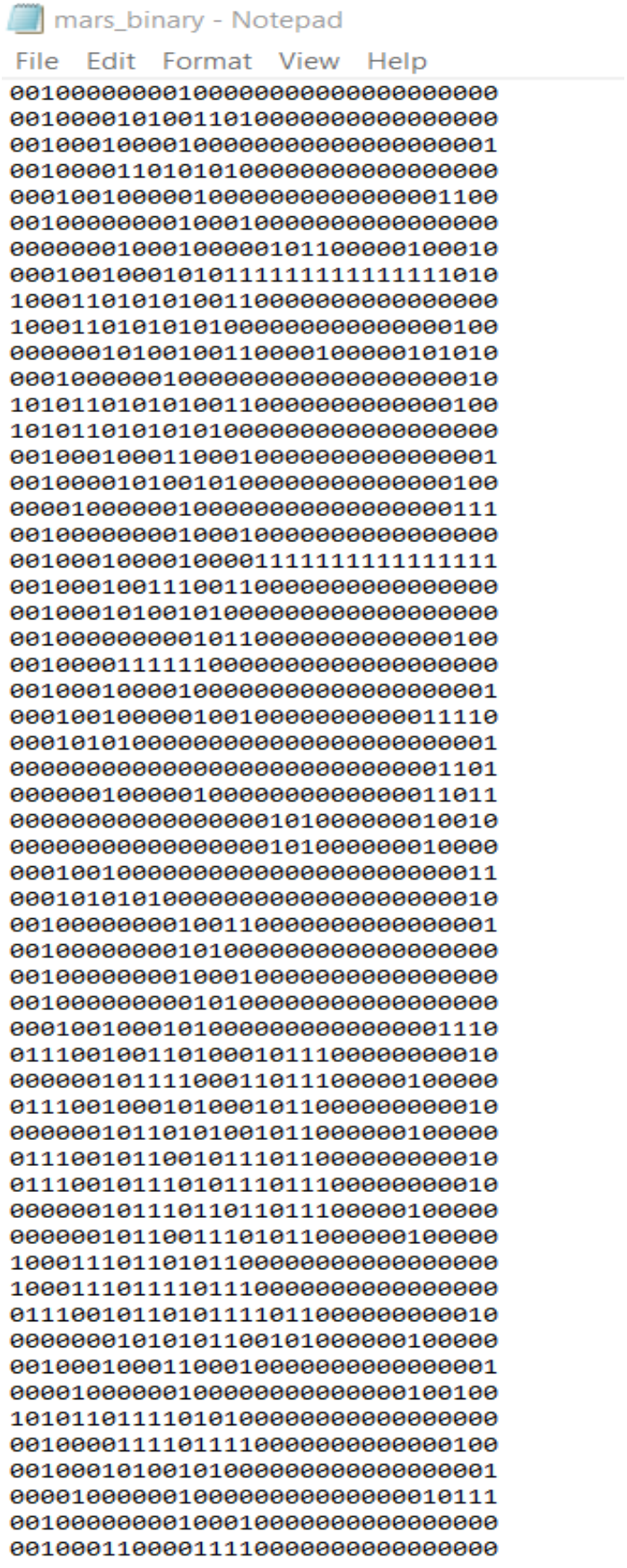


Question 2:

The assembler was written in python.

The next two pages contains the binaries generated by MARS and by the assembler we made.

```
PS C:\Users\akash\Desktop\CA> & C:/U
00100000000100000000000000000000
00100001010011010000000000000000
00100010000100000000000000000001
00100001101010100000000000000000
00010010000010000000000000001100
00100000000100010000000000000000
00000001000100000101100000100010
00010010001010111111111111111010
10001101010100110000000000000000
10001101010101000000000000000100
00000010100100110000100000101010
00010000001000000000000000000010
10101101010100110000000000000100
10101101010101000000000000000000
00100010001100010000000000000001
00100001010010100000000000000100
00001000000100000000000000000111
00100000000100010000000000000000
00100010000100001111111111111111
00100010011100110000000000000000
00100010100101000000000000000000
00100000000010110000000000000100
00100001111110000000000000000000
00100010000100000000000000000001
00010010000010010000000000011110
00010101000000000000000000000001
00000000000000000000000000001101
00000010000010000000000000011011
00000000000000000010100000010010
00000000000000000010100000010000
00010010000000000000000000000011
00010101010000000000000000000010
00100000000100110000000000000001
00100000000101000000000000000000
00100000000100010000000000000000
00100000000010100000000000000000
00010010001010000000000000001110
01110010011010001011100000000010
00000010111100011011100000100000
01110010001010001011000000000010
00000010110101001011000000100000
01110010110010111011000000000010
01110010111010111011100000000010
00000010111011011011100000100000
00000010110011101011000000100000
10001110110101100000000000000000
10001110111101110000000000000000
01110010110101111011000000000010
00000001010101100101000000100000
00100010001100010000000000000001
00001000000100000000000000100100
10101101111101010000000000000000
00100001111011110000000000000100
00100010100101000000000000000001
00001000000100000000000000010111
00100000000100010000000000000000
00100011000011110000000000000000
PS C:\Users\akash\Desktop\CA>
```


mars_binary - Notepad
File Edit Format View Help
00100000001000000000000000000000
00100001010011010000000000000000
00100010000100000000000000000001
00100001101010100000000000000000
00010010000010000000000000001100
00100000000100010000000000000000
00000001000100000101100000100010
00010010001010111111111111111010
10001101010100110000000000000000
10001101010101000000000000000100
00000010100100110000100000101010
00010000001000000000000000000010
10101101010100110000000000000100
10101101010101000000000000000000
00100010001100010000000000000001
00100001010010100000000000000100
00001000000100000000000000000111
00100000000100010000000000000000
00100010000100001111111111111111
00100010011100110000000000000000
00100010100101000000000000000000
00100000000010110000000000000100
00100001111110000000000000000000
00100010000100000000000000000001
00010010000010010000000000011110
00010101000000000000000000000001
00000000000000000000000000001101
00000010000010000000000000011011
00000000000000000010100000010010
00000000000000000010100000010000
00010010000000000000000000000011
00010101010000000000000000000010
00100000001001100000000000000001
00100000000101000000000000000000
00100000001000100000000000000000
00100000000010100000000000000000
00010010001010000000000000001110
01110010011010001011100000000010
00000010111100011011100000100000
01110010001010001011000000000010
00000010110101001011000000100000
01110010110010111011000000000010
01110010111010111011100000000010
00000010111011011011100000100000
00000010110011101011000000100000
10001110110101100000000000000000
10001110111101110000000000000000
01110010110101111011000000000010
00000001010101100101000000100000
00100010001100010000000000000001
00001000000100000000000000100100
10101101111010100000000000000000
00100001111011110000000000000100
00100010100101000000000000000001
00001000000100000000000000010111
00100000000100010000000000000000
00100011000011110000000000000000