# COSS Group5 Assignment 1

## Problem Statement

A data analyst Mr. Ashok is working on information collected by a mobile service provider ByTel regarding monthly usage of data. The information collected is in the form of a Matrix (X axis - Total data downloaded in GB vs Y axis - Month of the year). He plans to use the Kibana (data visualization) tool to plot the data in charts. The tool prefers Months as X axis and data downloaded as Y axis. Hence he requires you to incorporate the following using CPU OS simulator based assembly language program:

* Accept the data collected in the form of a Matrix (X axis - Total data downloaded in GB vs. Y axis - Month of the year) from the keyboard.
* Transpose the matrix.
* Display the transposed matrix on console.

## Team

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Name** | **ID NO** |
| **1** | Anirudh Menon | 2018AB04132 |
| **2** | Amit Kumar | 2018AB04046 |
| **3** | Vengal O.G.T. | 2018AB04020 |

## Description of the solution

A matrix called ‘A’ can be of order m x n, represented as Amxn. Here m is the number or rows and n is the number of columns in the matrix. WhenAmxn is transposed, it will be of the format of Bnxm.

The input data was accepted directly into data memory of CPU-OS simulator. It was then displayed on the output console as Input Matrix. It was then transposed using assembly language operations, stored into memory and then displayed on the console of CPU-OS Simulator as Output Matrix.

## Design

Multiple design approaches were evaluated to consider the matrix transpose along with various pros and cons.

***Various pros and cons of the approaches are -***

Approach 1: A simple transpose of the A[i,j] to A[j,i] as typically programmed in a high level programming language like C or Java.

Problems with the approach: This approach was not possible since multidimensional array representation was not possible in the given CPU OS Simulator. Various options including coding in CPU-OS programming language in the compiler and running on the simulator was tried but did not yield good results during our research and experimentation.

Approach-2**:** Use of stack push and pop operations for storing and then moving it to transpose. This approach was more time consuming and high stack usage, since the writing logic was found to be first calculated and was considered complex.

Approach-3 **:** Store the value of the initial matrix in a single array or address location incrementing from A11, A12, A13…., A1n, A21, A22,………….. A2n,………………… Am1,………..until Amn. The stored values are then looped through to find the rows and columns and then interchanged displayed in the order of transposed matrix. That means, the first column is picked as the first row in the output.  
Approach-3 has been selected for implementation in CPU OS assignment solution.

There were three main parts of the assembly code implementation. They are as follows:

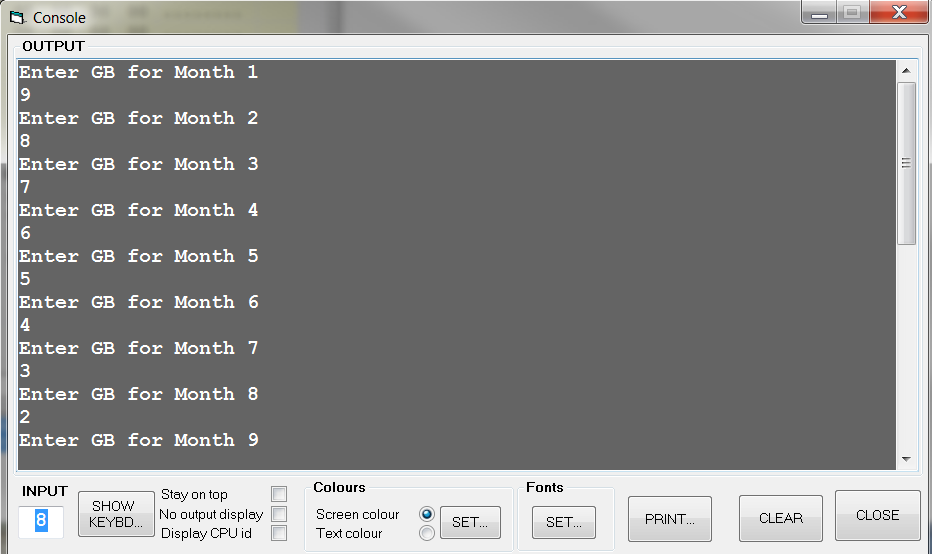
1. Reading Source Data from console for every month
2. Storing the month and data used in the memory
3. Transposing logic of the matrix and printing the final transpose data from the memory to console.

|  |  |  |
| --- | --- | --- |
| ***Sl No*** | ***Name of the data*** | ***Addressing mode used*** |
| *1* | *Read - Size of matrix from program.* | *Register Direct Addressing Mode* |
| *2* | *Read – Input data for each month from the Keyboard console and storing the input data along with the month* | *Memory Direct addressing*  *Memory Indirect Addressing*  *Register Direct Addressing Mode*  *Register Indirect Addressing Mode*  *Indexed Addressing Mode,*  *Base Register Addressing Mode* |
| *3* | *Print Input data for each month in 2x12 matrix format*  *First Row: Month of year*  *Second Row: GB data used (in single digit)* | *Memory Direct addressing*  *Memory Indirect Addressing*  *Register Direct Addressing Mode*  *Register Indirect Addressing Mode*  *Indexed Addressing Mode,*  *Base Register Addressing Mode* |
| *4* | *Write transposed matrix data from memory to register and then to output console.* | *Memory Direct Addressing,*  *Memory Indirect Addressing*  *Register Direct Addressing Mode*  *Register Indirect Addressing Mode*  *Index Addressing Mode,*  *Base Register Addressing Mode* |
| *5* | *Read Strings to be displayed as tags “Enter GB for Month”, “Input Matrix” and “Output Matrix:”* | *Memory Direct Addressing* |
| ***Sl No*** | ***Name of the function*** | ***Description*** |
| *1* | *Initial Assignment* | *This function is used to assign the initial data assignment including the matrix size* |
| *2* | *Input\_data* | *This function performs the operation of accepting data from keyboard. This accepts the number of GB for that month. Note: The keyboard can accept only single digit value due to limitation of keyboard* |
| *3* | *Input wait* | *This function waits for the input to be inserted from keyboard.* |
| *4* | *Prinit\_Input\_Matrix* | *This function has logic to read the matrix in row format and print in 2 row x12 column format* |
| *5* | *Transpose\_Print* | *This function has logic to read the matrix in 2 row x12 column format and then print in transpose 12 row x2 column format, ie transposing logic of the matrix. This function also has the logic to print the output data* |

## Procedure

### 4.1 Data Input

The input data is directly entered by the user for every month in the console. This data will be stored in memory for retrieval and display later.



Here are the details of the address location for Ashok’s Matrix which is a 2x12 matrix of the format Month in first row and Total data downloaded in GB in second row.

|  |  |  |
| --- | --- | --- |
| **LAddress #** | **Data Stored** | **Sample Value** |
| 0000 -0023 | String | “Enter GB for Month” |
| 0024 -0037 | String | “Input Matrix:” |
| 0040-0054 | String | “Output Matrix:” |
| 0056 | A1,1 (January Month – Row 1, Column 1) | 1 |
| 0064 | A2,1 (Data downloaded in January – Row 1, Column 1) | 9 |
| 0072 | A1,2 (Feb Month – Row 1, Column 2) | 2 |
| 0080 | A2,2 (Data downloaded in February – Row 2, Column 2) | 8 |
| 0088 | A13 (Mar Month – Row 1, Column 3) | 3 |
| 0096 | A2,3 (Data downloaded in March – Row 2, Column 3) | 7 |
| 0104 | A14 (April Month – Row 1, Column 3) | 4 |
| 0112 | A2,4 (Data downloaded in April – Row 2, Column 4) | 6 |
| 0120 | A1,5 (May Month – Row 1, Column 3) | 5 |
| 0128 | A2,5 (Data downloaded in May – Row 2, Column 5) | 5 |
| 0136 | A1,6 (June Month – Row 1, Column 3) | 6 |
| 0144 | A2,6 (Data downloaded in June – Row 2, Column 6) | 4 |
| 0152 | A1,7 (July Month – Row 1, Column 3) | 7 |
| 0160 | A2,7 (Data downloaded in July – Row 2, Column 7) | 3 |
| 0168 | A1,8 (August Month – Row 1, Column 3) | 8 |
| 0176 | A2,8 (Data downloaded in August – Row 2, Column 8) | 2 |
| 0184 | A1,9 (September Month – Row 1, Column 3) | 9 |
| 0192 | A2,9 (Data downloaded in September – Row 2, Column 9) | 1 |
| 0200 | A1,10 (October Month – Row 1, Column 3) | 10 |
| 0208 | A2,10 (Data downloaded in October – Row 2, Column 10) | 0 |
| 0216 | A1,11 (November Month – Row 1, Column 3) | 11 |
| 0224 | A2,11 (Data downloaded in November – Row 2, Column 11) | 9 |
| 0232 | A1,12 (December Month – Row 1, Column 3) | 12 |
| 0240 | A2,12 (Data downloaded in December – Row 2, Column 12) | 8 |

Table 1: Input Values

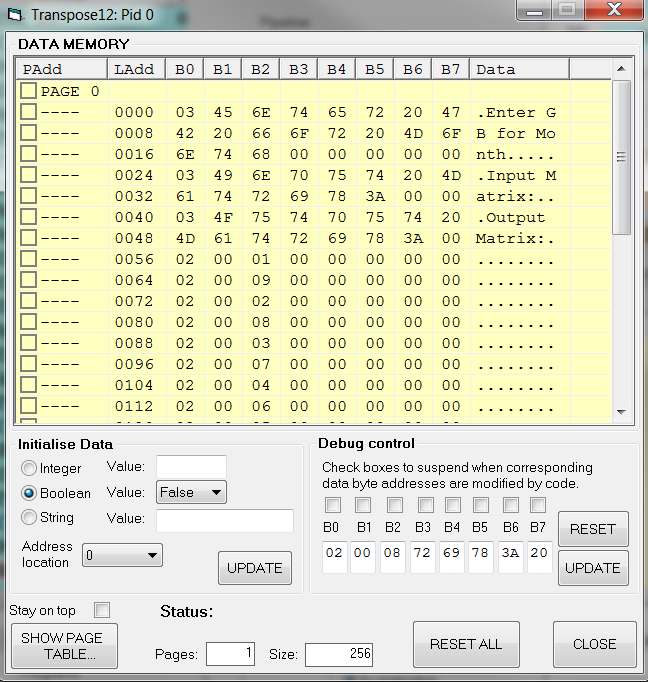


Fig 1a: Screenshot of Data Memory (Input Storage Matrix)

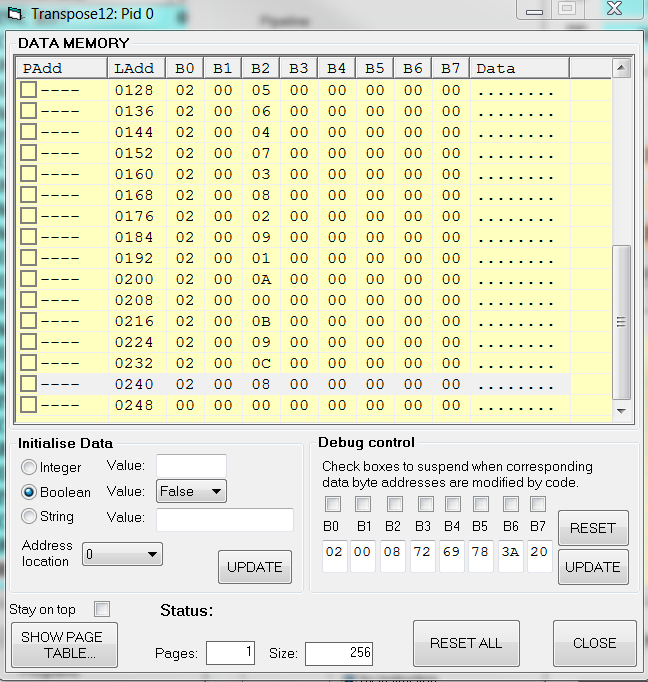


Fig 1b. Screenshot of Data Memory (Input Storage Matrix)

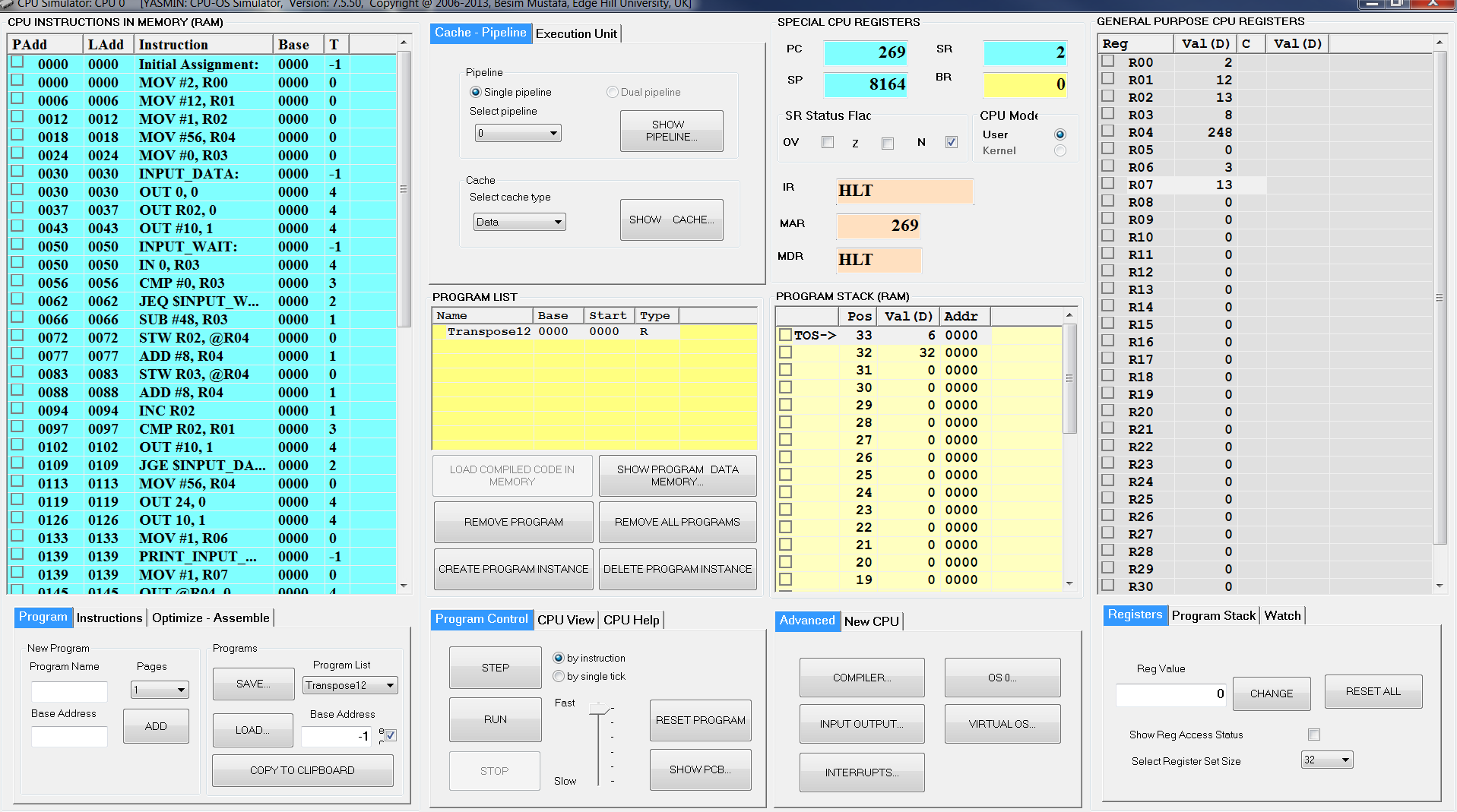


Fig 2a: Screenshot of the Input Program – Part 1

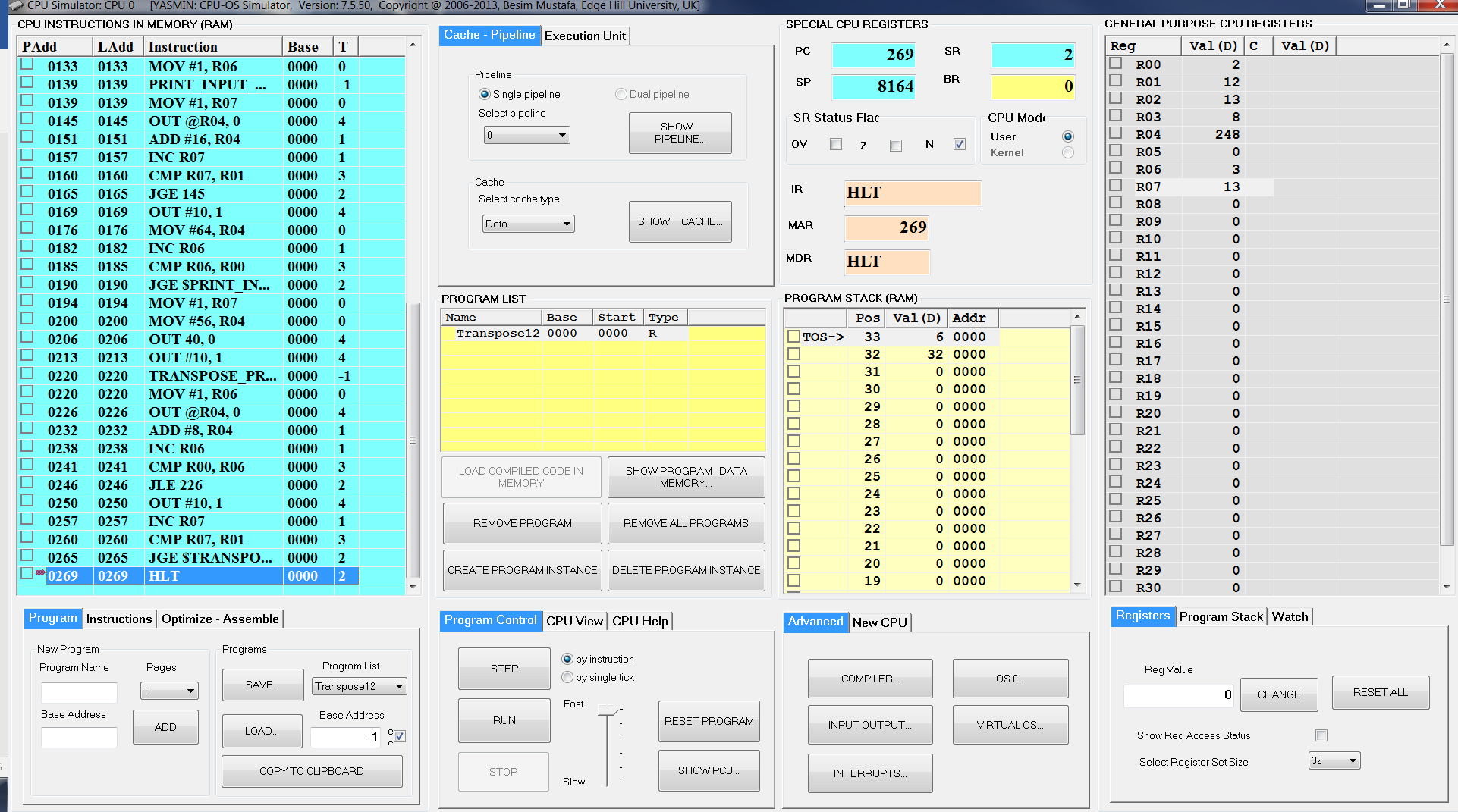


Fig 2b: Screenshot of the Input Program – Part 2

### 4.2 Expected Output Values

The expected address locations for the 12x2 Matrix B that is the transpose of Matrix A:

1 9

2 8

3 7

4 6

5 5

6 4

7 3

8 2

9 1

10 0

11 9

12 8

Table 2: Expected Output Values

**CPU OS Simulator Program**

### 4.3 Program

|  |
| --- |
| * 1,55,#FALSE#,#FALSE#,"Transpose12",0,0,0,270,1,6,1004915311 * 0,0,0,0,0,0 * 0 * 0 * 5 * "TRANSPOSE\_PRINT",220 * "INPUT\_WAIT",50 * "INPUT\_DATA",30 * "Initial Assignment",0 * "PRINT\_INPUT\_MATRIX",139 * 0,-1,"Initial Assignment",0,0,0,0,#FALSE# * 0,0,"MOV #2, R00",0,0,0,0,#FALSE# * 6,0,"MOV #12, R01",0,0,0,0,#FALSE# * 12,0,"MOV #1, R02",0,0,0,0,#FALSE# * 18,0,"MOV #56, R04",0,0,0,0,#FALSE# * 24,0,"MOV #0, R03",0,0,0,0,#FALSE# * 30,-1,"INPUT\_DATA",0,0,0,0,#FALSE# * 30,4,"OUT 0, 0",0,0,0,0,#FALSE# * 37,4,"OUT R02, 0",0,0,0,0,#FALSE# * 43,4,"OUT #10, 1",0,0,0,0,#FALSE# * 50,-1,"INPUT\_WAIT",0,0,0,0,#FALSE# * 50,4,"IN 0, R03",0,0,0,0,#FALSE# * 56,3,"CMP #0, R03",0,0,0,0,#FALSE# * 62,2,"JEQ $INPUT\_WAIT",0,0,0,0,#FALSE# * 66,1,"SUB #48, R03",0,0,0,0,#FALSE# * 72,0,"STW R02, @R04",0,0,0,0,#FALSE# * 77,1,"ADD #8, R04",0,0,0,0,#FALSE# * 83,0,"STW R03, @R04",0,0,0,0,#FALSE# * 88,1,"ADD #8, R04",0,0,0,0,#FALSE# * 94,1,"INC R02",0,0,0,0,#FALSE# * 97,3,"CMP R02, R01",0,0,0,0,#FALSE# * 102,4,"OUT #10, 1",0,0,0,0,#FALSE# * 109,2,"JGE $INPUT\_DATA",0,0,0,0,#FALSE# * 113,0,"MOV #56, R04",0,0,0,0,#FALSE# * 119,4,"OUT 24, 0",0,0,0,0,#FALSE# * 126,4,"OUT 10, 1",0,0,0,0,#FALSE# * 133,0,"MOV #1, R06",0,0,0,0,#FALSE# * 139,-1,"PRINT\_INPUT\_MATRIX",0,0,0,0,#FALSE# * 139,0,"MOV #1, R07",0,0,0,0,#FALSE# * 145,4,"OUT @R04, 0",0,0,0,0,#FALSE# * 151,1,"ADD #16, R04",0,0,0,0,#FALSE# * 157,1,"INC R07",0,0,0,0,#FALSE# * 160,3,"CMP R07, R01",0,0,0,0,#FALSE# * 165,2,"JGE 145",0,0,0,0,#FALSE# * 169,4,"OUT #10, 1",0,0,0,0,#FALSE# * 176,0,"MOV #64, R04",0,0,0,0,#FALSE# * 182,1,"INC R06",0,0,0,0,#FALSE# * 185,3,"CMP R06, R00",0,0,0,0,#FALSE# * 190,2,"JGE $PRINT\_INPUT\_MATRIX",0,0,0,0,#FALSE# * 194,0,"MOV #1, R07",0,0,0,0,#FALSE# * 200,0,"MOV #56, R04",0,0,0,0,#FALSE# * 206,4,"OUT 40, 0",0,0,0,0,#FALSE# * 213,4,"OUT #10, 1",0,0,0,0,#FALSE# * 220,-1,"TRANSPOSE\_PRINT",0,0,0,0,#FALSE# * 220,0,"MOV #1, R06",0,0,0,0,#FALSE# * 226,4,"OUT @R04, 0",0,0,0,0,#FALSE# * 232,1,"ADD #8, R04",0,0,0,0,#FALSE# * 238,1,"INC R06",0,0,0,0,#FALSE# * 241,3,"CMP R00, R06",0,0,0,0,#FALSE# * 246,2,"JLE 226",0,0,0,0,#FALSE# * 250,4,"OUT #10, 1",0,0,0,0,#FALSE# * 257,1,"INC R07",0,0,0,0,#FALSE# * 260,3,"CMP R07, R01",0,0,0,0,#FALSE# * 265,2,"JGE $TRANSPOSE\_PRINT",0,0,0,0,#FALSE# * 269,2,"HLT",0,0,0,0,#FALSE# * 3 * 0,18,"Enter GB for Month" * 24,13,"Input Matrix:" * 40,14,"Output Matrix:" * 0 * 0 * -1 |

## **Result**

The matrix is stored in the format of an array of 2x12 words spaces in the memory. The transposed matrix is created in memory and displayed in the format as follows:

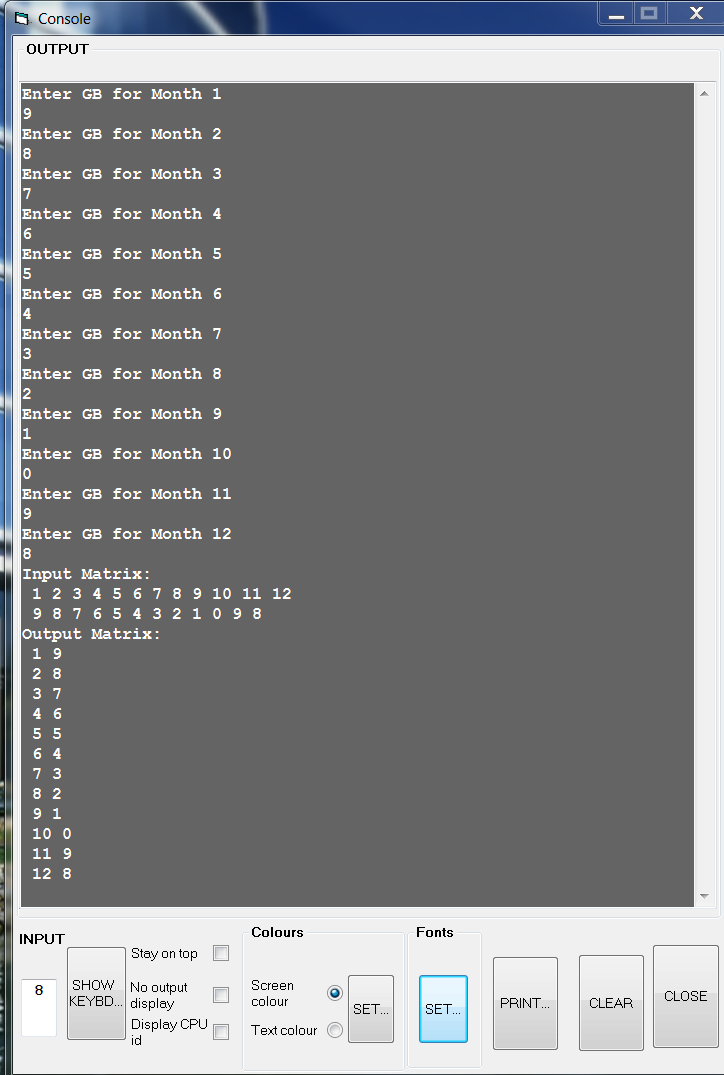


Fig 4: Screenshot of transposed matrix output

## Observations, Limitations and Workarounds

Observation 1:CPU OS Simulator does not wait for the user to provide the input when in RUN mode. The usability limitations on entering data in interactive GUI for a series of data input while in RUN mode.

1. **Potential cause:** The simulator probably does not have input interrupt feature available
2. **Workaround:**
3. User has to enter first data in the input and then run on “STEP” mode ONLY. It is painful to run on STEP mode and then enter the data manually for every loop.
4. Alternative solution of entering the data in the memory location directly is a simpler approach for larger data sets and when you want to execute in RUN Mode.
5. Accept the data into a register and compare whether this register is not null in a loop. This option was selected in our design.

Observation 2**:** It has been found that sometimes the CPU-OS Simulator gives an error “Data Descriptor Error: Expected xx, read yy” when in program is executed in continuous “RUN” mode instead of “STEP” by instruction mode.

1. **Potential cause:** This seems to be bug in the simulator.
2. **Workaround:** If this is error occurs, the work around would be to restart in “STEP” mode either from where it was last throwing error.

Observation 3**:** CPU OS Simulator can accept only single characters. It cannot accept more than one digit integer or a negative number.

1. **Potential cause:** The simulator probably does not have the capability to accept more than one character.
2. **Workaround:** User has to enter a single character and it has to be converted to integer based on ASCII conversions. If multiple character integer has to be accepted, then the data has either to be directly input into memory or accepted as individual integers based on the number of numbers that are in the integer and reconstructed to create the final integer.

## Conclusions

Data analyst Mr. Ashok’s use-case of transposing a 2x12 matrix into respective 12x2 transpose matrix can successfully be achieved using the given code and a sample of the same is provided above. We have successively accepted 12 inputs and computed the matrix with months and data and tested for transposing matrix of 2x12matrix

## Contributions

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Name** | **ID NO** | **Contribution** |
| 01 | Anirudh Menon | 2018AB04132 | Analysis, Design, Test, Debug, Documentation |
| 02 | Amit Kumar | 2018AB04046 | Design, Documentation |
| 03 | Vengal O.G.T. | 2018AB04020 | Research, Analysis, Design, Debug, Implementation, Test, Documentation |