AMRITA VISHWA VIDYAPEETHAM | School of Engineering

Elements Of Computing Systems 1

END - SEMESTER 1- PROJECT

NAME	ROLL NO
ADITHYA KRISHNA	AM.EN.U4AIE21005
ADITHYA S NAIR	AM.EN.U4AIE21006
ATHUL GIREESH	AM.EN.U4AIE21020
NAVNEETH KRISHNA	AM.EN.U4AIE21047
ANOOP BOBY MANUEL	AM.EN.U4AIE21015

PSEUDO CODE

```
import java.util.*;


class nTwoT

{

        public static void main(String[] args)

        { Scanner sc=new Scanner(System.in);

        int a[] = new int[8];

        for(int i=0; i<8; i++)

        {

        a[i]= sc.nextInt();


        }


        int k, j, temp;

   for(k = 0; k < 8; k++)

        {
```

```
        for(j = k+1; j < 8; j++)
        {
        if(a[j] < a[k])
        {
                temp = a[k];
                a[k] = a[j];
                a[j] = temp;
        }
        }
        }
        for(int t=0;t<8;t++)
        {
        System.out.print(a[t]+ " ");
        }
        }

}
```

The pseudo code is written in java programming language and it uses the bubble sort algorithm to sort 8 values entered by the user into an array. The algorithm uses two loops, one nested inside the other to execute the program.

HACK ASSEMBLY CODE

```
//This asm aims at sorting numbers manually entered into the registers 0-7 of the ram and then storing the sorted numbers to the registers 30-37

//We proceed by copying the manually entered numbers in the registers 0-7 to the registers 30-37

//Then we use the ram registers 30-37 to store the sorted numbers too


//RAM 40 acts as a pointer to the initial location of the first element to be copied i.e at RAM 0

//RAM 41 acts as a pointer to the initial location of the first register where the element must be copied to


//RAM 50 stores the array size or the total no. of values that must be sorted

//RAM 45 copies the value in RAM 50 for loop iterations
```

```
@0     //initializing the value of the first register for pointer
D=A
@40
M=D

@30    //initializing the destination register as pointer
D=A
@41
M=D

@8    //Putting the array size into the register
D=A
@50
M=D

@50   //copying the array size into another register for loop iterations
D=M
@45
```

```
M=D

(loop)  //loop label indicating the code start for the copy loop

//@45
//D=M

//transfering elements from 0-7 to 30-37
@40
A=M
D=M
@41
A=M
M=D

//Updating the location values
@40
M=M+1
```

```
@41

M=M+1


//condition for exiting or returning to the loop

@45

M=M-1

D=M

@loop   //reruns the loop

D;JGT

@init   //moves on to the next part of the code

D;JEQ



(init)


//Initialize the start value for sort(register number):A=30

@30

D=A
```

```
@20
M=D


// Load arraysize into both parent and child iteration pass
@50
D=M
@10
M=D
@12
M=D

@45
D=M
@CONDITION
D;JEQ
```

```
(CONDITION)

        // Check parent iteration
@10
D=M

(END)
@END
D; JEQ

        //Check child iteration within parent
@12
M=M-1      // M=5-1; M=4;
D=M        // D=4 ;
@RELOAD
D; JEQ

//moves first value into register for comparison
```

```
@20
A=M          //Load array[i]
D=M          //D = array[i]


//brings in the next value for comparison
@20
A=M+1        //Load array[i+1]
D=D-M        // D = array[i] - array[i+1]


             // If array[i] > array[i+1] then SWITCH{MAIN CONDITION
FOR SORT}
@SWITCH
D; JGT       // Jump to SWITCH
             // If array[i] <= array[i+1] then move to next iteration
@20
M=M+1        //Increment child iteration
@CONDITION
0; JMP       // Jump to CONDITION again on next iteration
```

```
(SWITCH)
//Store array[i]
@20
A=M   //load array[i]
D=M   //D = array[i]
@11
M=D   // Place array[i] for SWITCH

//Store array[i+1]
@20
A=M+1  //load array[i+1]
D=M    //store array[i+1] in D register

//Move i+1 into i
@20
A=M   //load array[i]
M=D   //Place i+1 into i
```

```
//Move i into i+1
@11
D=M   //Load i into D
@20
A=M+1 //Load array[i+1]
M=D   //Place i into i+1

// Increment child iteration and jump to next comparison
@20
M=M+1            //Increment child pass iteration
@CONDITION
0; JMP           // Jump to CONDITION again on next iteration

(RELOAD)
// Reset child iteration within pass
@50
D=M
```

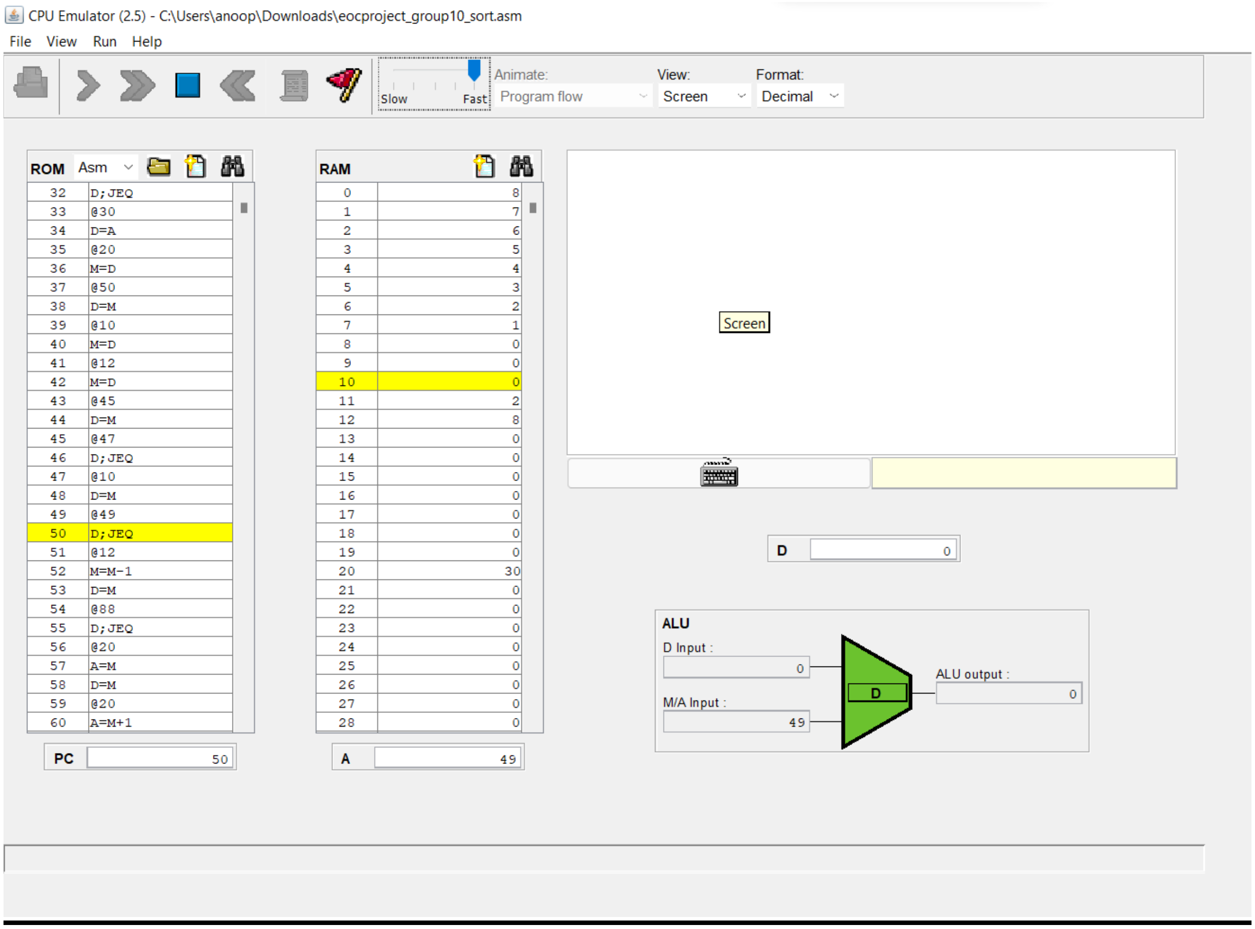
```
@12
M=D

// Decrement child iteration
@10
M=M-1

// Reset A-register place
@30
D=A
@20
M=D
@CONDITION
0; JMP
```

INPUT

The values to be sorted are placed in registers 0-7 manually.



CPU Emulator (2.5) - C:\Users\anoop\Downloads\eocproject_group10_sort.asm

Animate: Program flow

View: Screen

Format: Decimal

ROM Asm

32	D;JEQ
33	@30
34	D=A
35	@20
36	M=D
37	@50
38	D=M
39	@10
40	M=D
41	@12
42	M=D
43	@45
44	D=M
45	@47
46	D;JEQ
47	@10
48	D=M
49	@49
50	D;JEQ
51	@12
52	M=M-1
53	D=M
54	@88
55	D;JEQ
56	@20
57	A=M
58	D=M
59	@20
60	A=M+1

PC 50

RAM

0	8
1	7
2	6
3	5
4	4
5	3
6	2
7	1
8	0
9	0
10	0
11	2
12	8
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	30
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

A 49

D 0

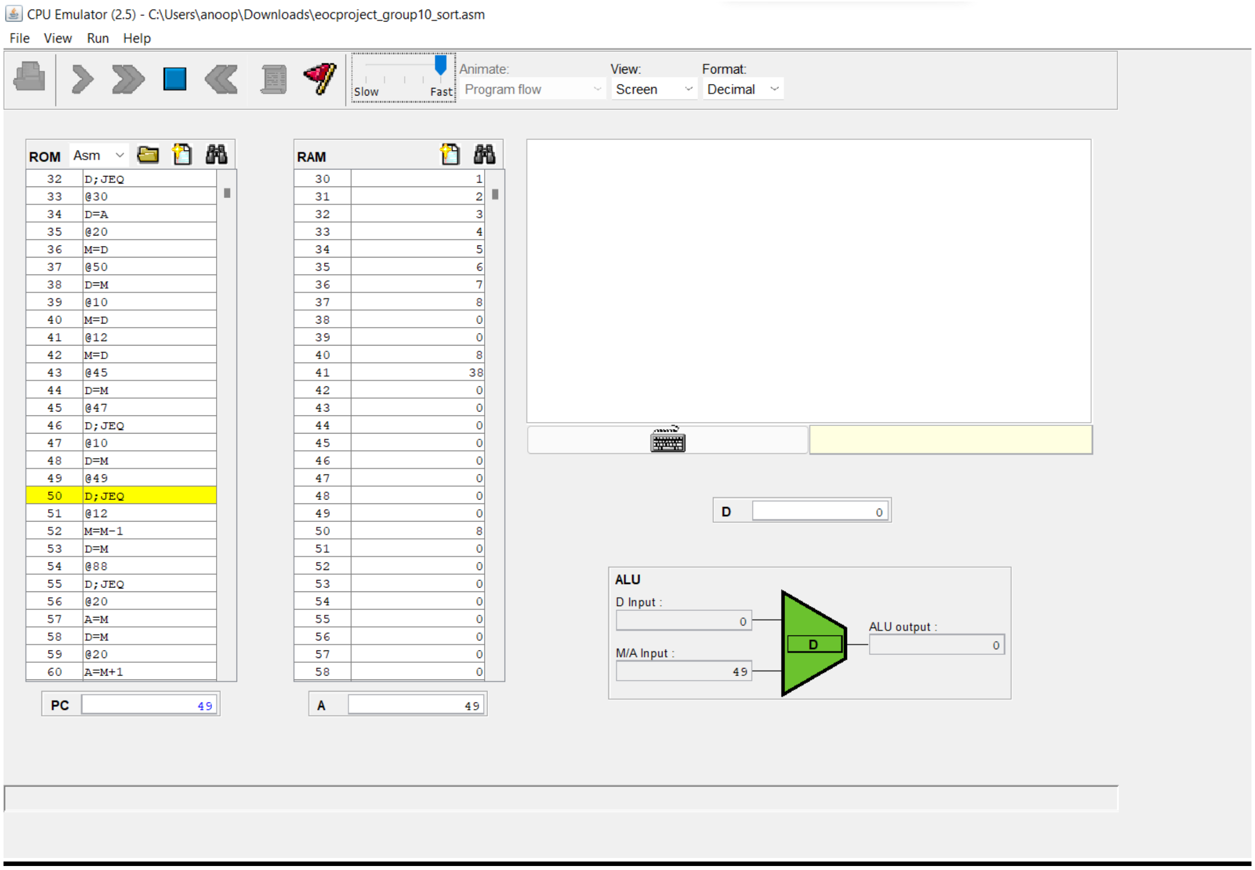
D Input : 0

M/A Input : 49

ALU output : 0

OUTPUT

The sorted values are displayed in the registers 30-37 automatically. The values are sorted in ascending order i.e lowest to highest.


CPU Emulator (2.5) - C:\Users\anoop\Downloads\eocproject_group10_sort.asm
File
View
Run
Help
Slow
Fast
ALU
D

Animate: Program flow

View: Screen

Format: Decimal

ROM Asm

```
32  D;JEQ
33  @30
34  D=A
35  @20
36  M=D
37  @50
38  D=M
39  @10
40  M=D
41  @12
42  M=D
43  @45
44  D=M
45  @47
46  D;JEQ
47  @10
48  D=M
49  @49
50  D;JEQ
51  @12
52  M=M-1
53  D=M
54  @88
55  D;JEQ
56  @20
57  A=M
58  D=M
59  @20
60  A=M+1
```

PC: 49

RAM

30	1
31	2
32	3
33	4
34	5
35	6
36	7
37	8
38	0
39	0
40	8
41	38
42	0
43	0
44	0
45	0
46	0
47	0
48	0
49	0
50	8
51	0
52	0
53	0
54	0
55	0
56	0
57	0
58	0

A: 49

D: 0

D Input : 0

M/A Input : 49

ALU output : 0

INSIGHTS

By doing this project we have learned many things about the hack assembly code some of which include:-

- Learning how the basic A,D,M registers work.
- How to copy information from registers and transfer them to other registers.
- How to implement and manipulate loops to our favour.
- How to exit a loop to continue the program.
- How to use pointers to point to registers where we extract values.
- How to work on extracted values in a temporary register.
- How to implement nested loops and how to check the child and parent loop conditions.
- Implementing labels to access specific parts of code whenever we want.
- Updating existing values inside a register for it to act as a counter of sorts.
- Using an infinite loop to terminate the program.

==