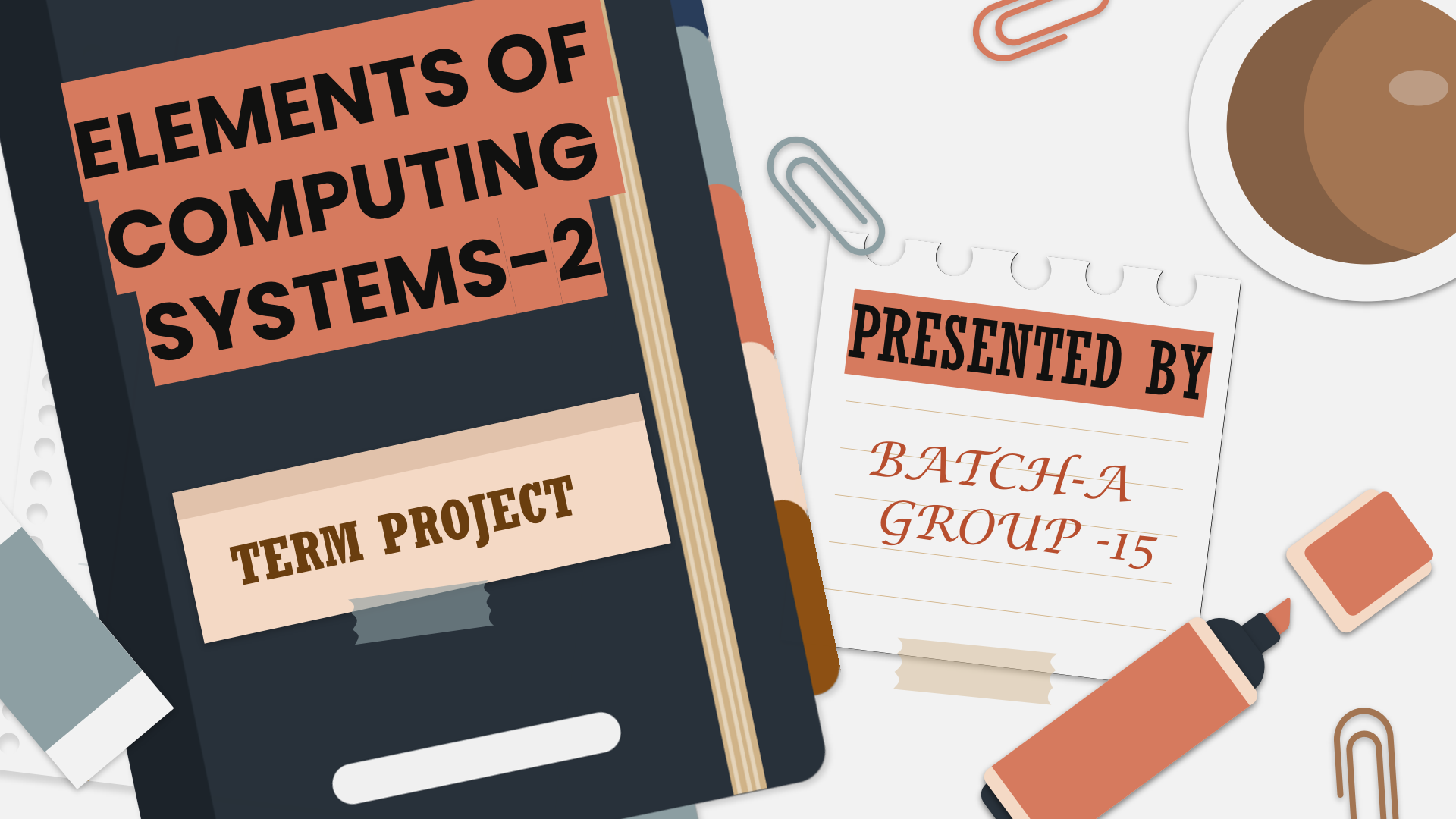
ELEMENTS OF COMPUTING SYSTEMS-2

TERM PROJECT



PRESENTED BY

BATCH-A
GROUP -15

TEAM MEMBERS

P.S.S.SAI.KEERTHANA(CB.EN.U4AIE21038)

P. KOMAL SAI ANURAG(CB.EN.U4AIE21039)

UDAYAGIRI VARUN(CB.EN.U4AIE21071)

SEJAL SINGH(CB.EN.U4AIE21061)

PART – 1

Write the Jack program of the given task and Compile and successfully simulate it on VM Emulator.

Implement the following String library functions in Jack

1. Sets the j'th character of this string to the given character.

2. Appends the given character to the end of this string, and returns the string.

3. Erases the last character from this string.

4. Returns the integer value of this string until the first non-numeric character.

JACK Code:

```
// Implementation of String library in Jack

class String2 {
    field Array arr;
    field int length;

    constructor String2 new(String str) {
        var int i;

        let length = str.length();
        let arr = Array.new(length);
        let i = 0;
        while (i < length) {
            let arr[i] = str.charAt(i);
            let i = i + 1;
        }
        return this;
    }
```

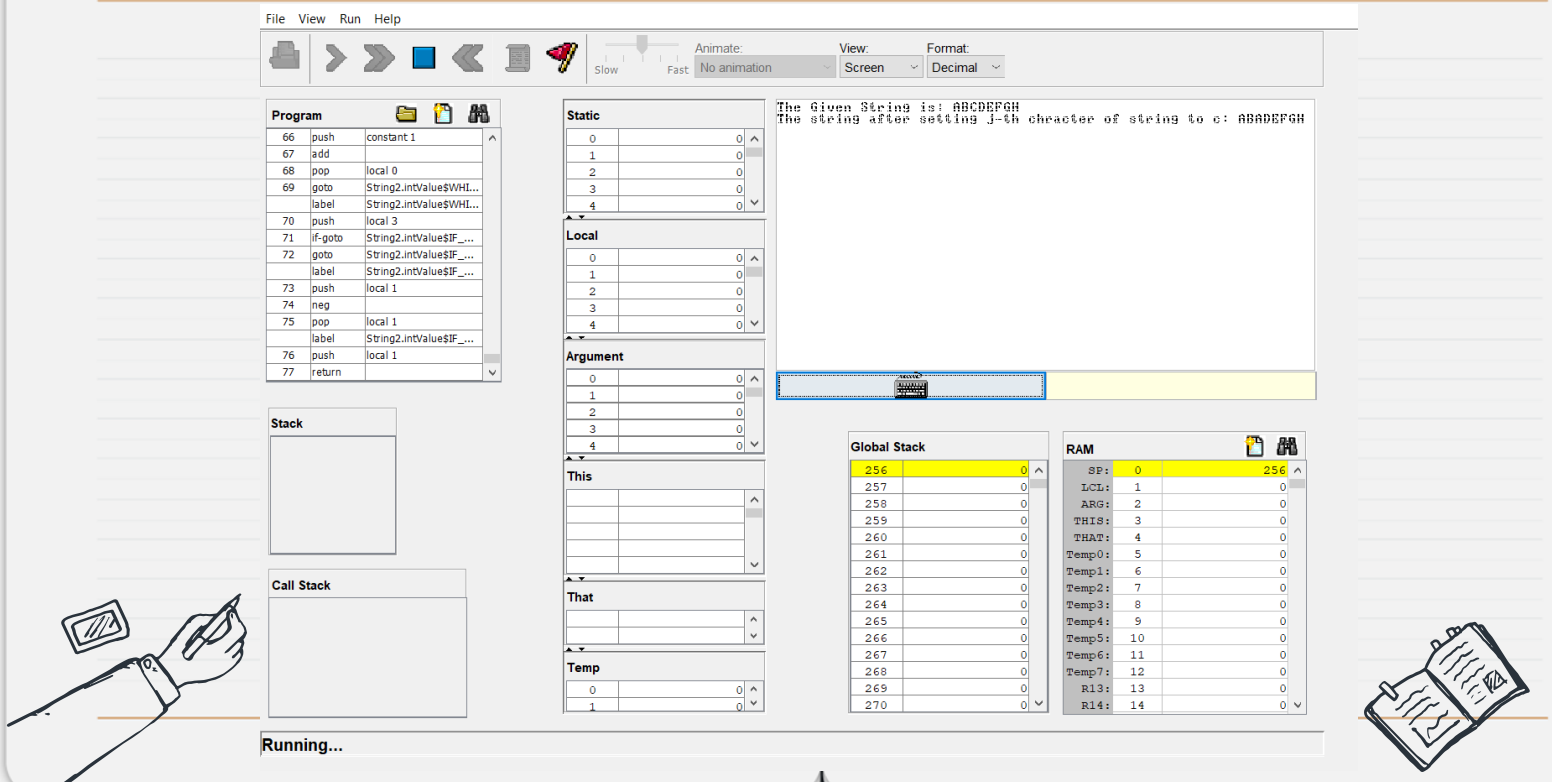
```
    method String get() {
        var String str;
        var int i;

        let str = String.new(length);
        let i = 0;
        while (i < length) {
            do str.appendChar(arr[i]);
            let i = i + 1;
        }
        return str;
    }
```

1. SetCharAt

```
44      /** Sets the j'th character of this string to be c */
45      method void setCharAt(int j, char c) {
46          var int i;
47
48          let i = 0;
49          while (i < length) {
50              if (i = j) {
51                  let arr[i] = c;
52              }
53              let i = i + 1;
54          }
55          return;
56      }
57
```

```
21          let str = "ABCDEFGH";
22          let s = String2.new(str);
23          do Output.println();
24          do s.setCharAt(1, 65);
25          do Output.printString(s.get());
26          do Output.println();
27
```

VM Emulator Output

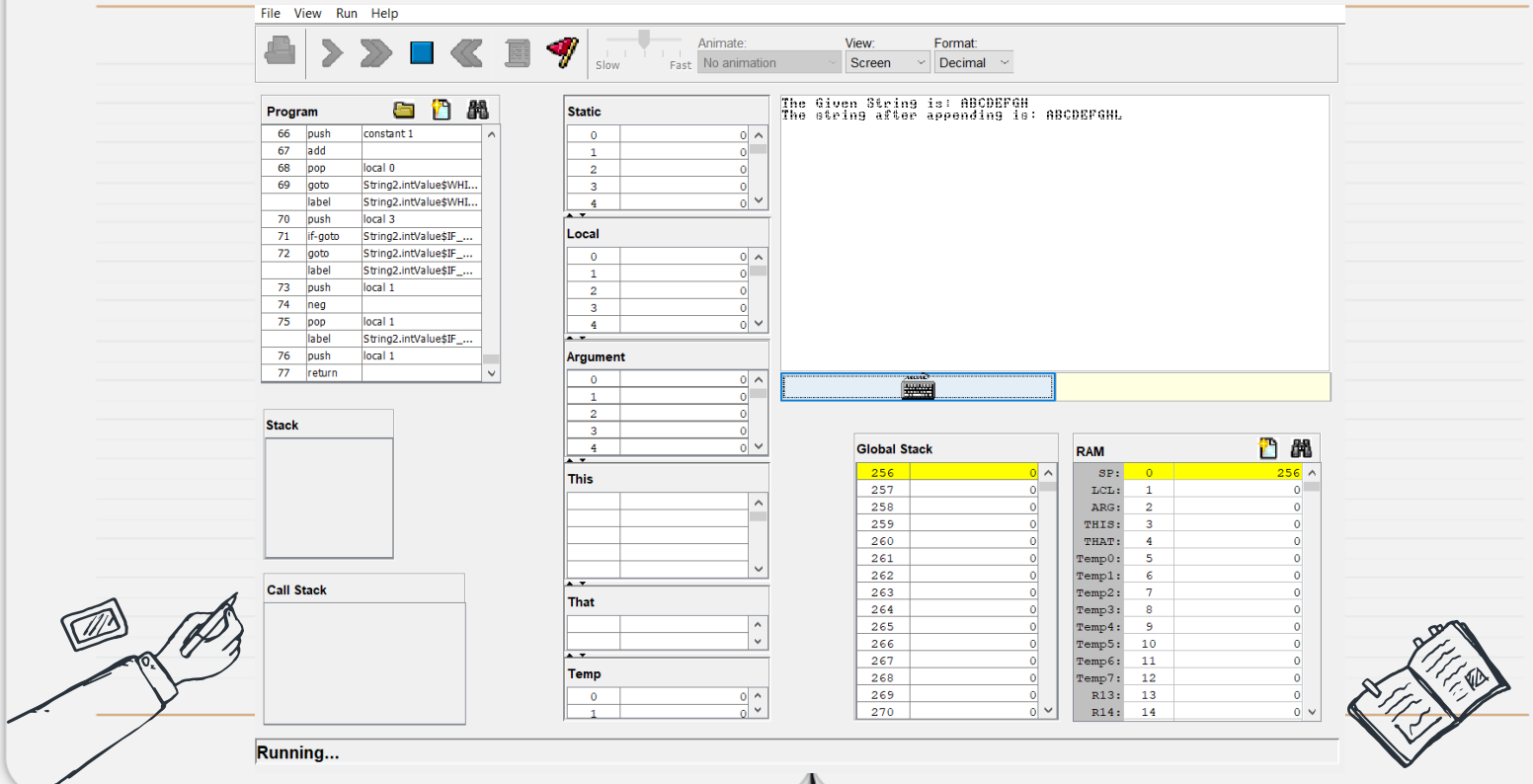


2. appendLastChar

```
58      /** Appends the character c to the end of this String. */
59      method void appendLastChar(char c) {
60          var Array array;
61          var int i;
62
63          let array = Array.new(length + 1);
64          let i = 0;
65          while (i < length) {
66              let array[i] = arr[i];
67              let i = i + 1;
68          }
69          let array[i] = c;
70          let arr = array;
71          let length = length + 1;
72          return;
73      }
```

```
        do Output.printString("The Given String is: ");
        do Output.printString(str);
        do Output.println();
        do Output.printString("The string after appending given last cahracter: ");
        do s.appendLastChar(69);
        do Output.printString(s.get());
        do Output.println();
```

VM Emulator Output

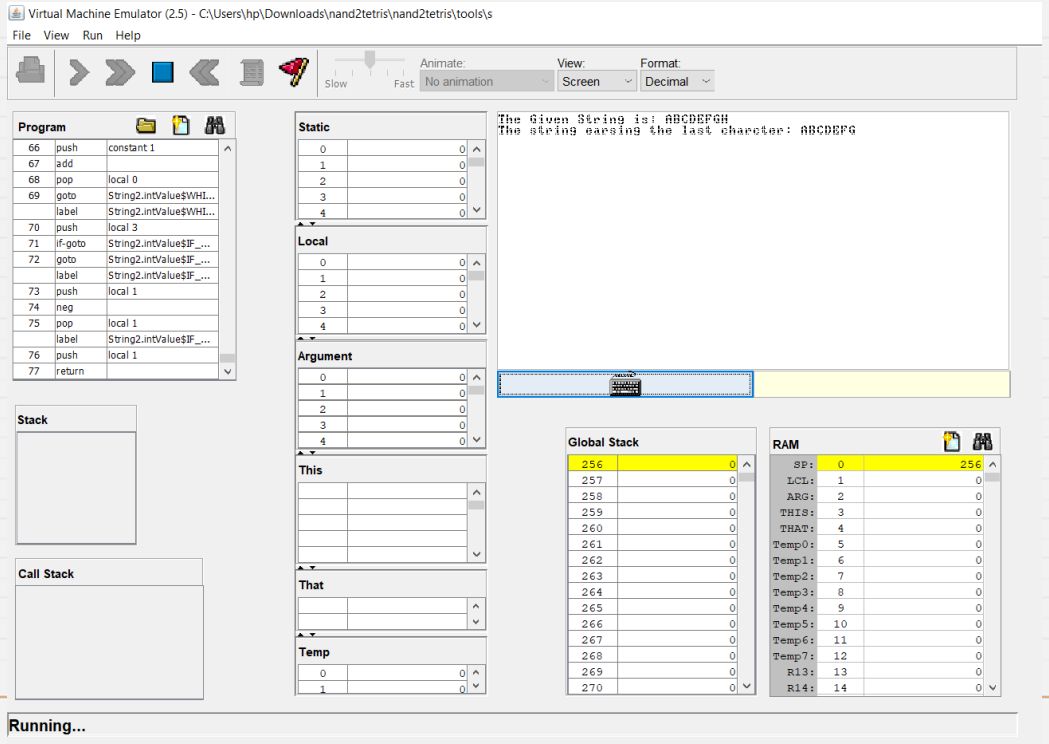


3. eraseLastChar

```
75      /** Erases the last character from this String. */
76      method void eraseLastChar() {
77          var Array array;
78          var int i;
79
80          let array = Array.new(length - 1);
81          let i = 0;
82          while (i < (length - 1)) {
83              let array[i] = arr[i];
84              let i = i + 1;
85          }
86          let arr = array;
87          let length = length - 1;
88          return;
89      }
```

```
45          do Output.printString("The Given String is: ");
46          do Output.printString(str);
47          do Output.println();
48          do Output.printString("The string after earsing the last charcter: ");
49          do s.eraseLastChar();
50          do Output.printString(s.get());
51          do Output.println();
```

VM Emulator Output



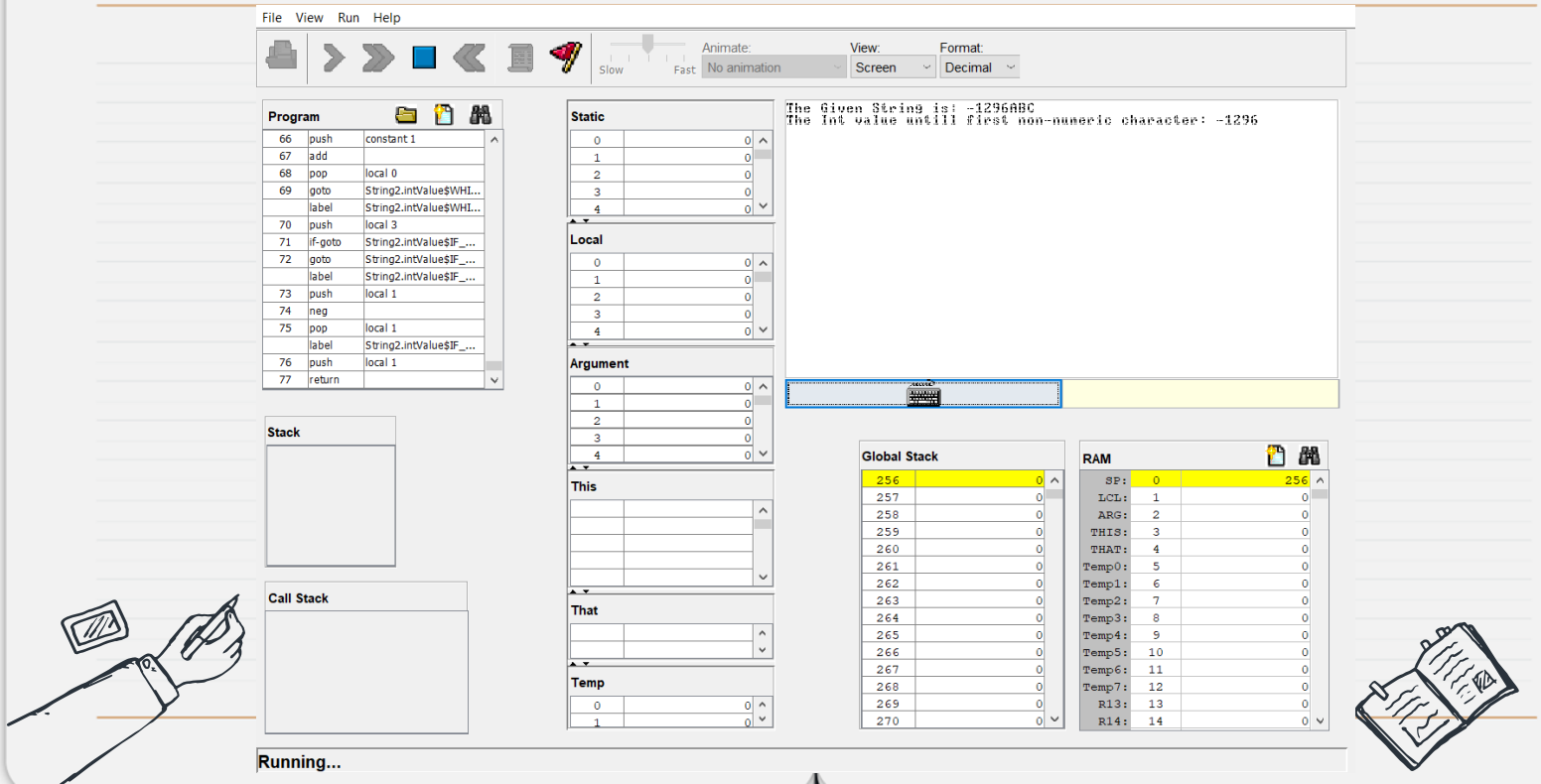
4. intValue

```
91      /** returns the integer value of this String until the first non-numeric character. */
92      method int intValue() {
93            var int i, result;
94            var boolean notfound, neg;
95
96            let neg = false;
97            let i = 0;
98            if (arr[0] = 45) {
99                let neg = true;
100               let i = 1;
101           }
102
103           let result = 0;
104           let notfound = true;
105           while (i < length) {
106               if ((arr[i] > 47 & arr[i] < 58) & notfound) {
107                   let result = result * 10 + (arr[i] - 48);
108               } else {
109                   let notfound = false;
110               }
111               let i = i + 1;
112           }
```

```
113
114           if (neg) {
115               let result = -result;
116           }
117           return result;
118       }
119   }
120
```

```
35          let str = "-1296ABC";
36          let s = String2.new(str);
37
38          do Output.printString("The Given String is: ");
39          do Output.printString(str);
40          do Output.println();
41          do Output.printString("The Int value untill first non-numeric character: ");
42          do Output.printInt(s.intValue());
```

VM Emulator Output



ASCII	Hex	Symbol
0	0	NUL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOT
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS
9	9	TAB
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI

ASCII	Hex	Symbol
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US

ASCII	Hex	Symbol
32	20	(space)
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/

ASCII	Hex	Symbol
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

ASCII	Hex	Symbol
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O

ASCII	Hex	Symbol
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

ASCII	Hex	Symbol
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o

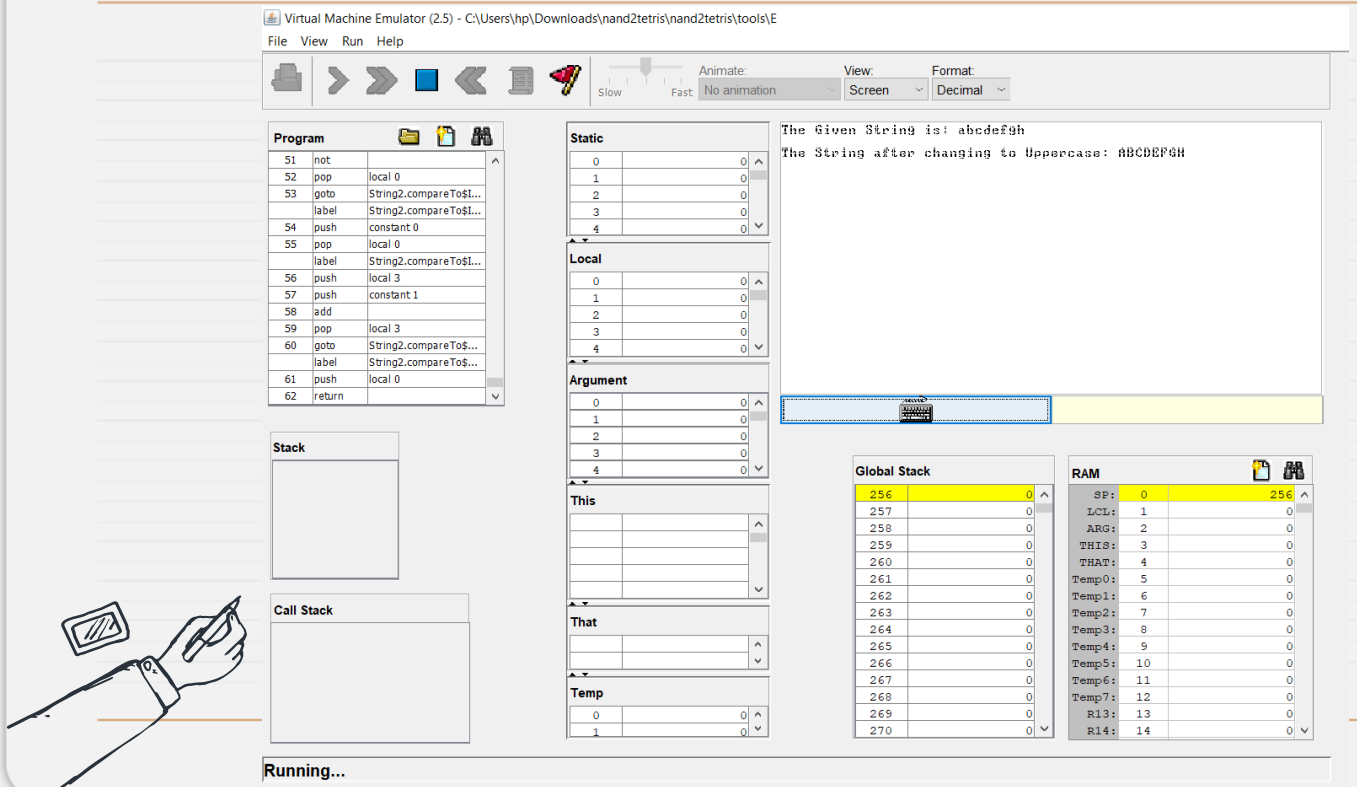
ASCII	Hex	Symbol
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	

5. toUpperCase

```
34      /** converts all the chracters of the string to upper case */
35      method void toUpperCase(){
36        var int i;
37        let i = 0;
38
39        while(i < length){
40          if((arr[i] > 96) & (arr[i] < 123)){
41            let arr[i] = (arr[i] - 32);
42          }
43          let i = i + 1;
44        }
45        return;
46      }
```

```
63      let str = "abcdefgh";
64      let s = String2.new(str);
65
66      do Output.printString("The Given String is: ");
67      do Output.printString(str);
68      do Output.println();
69      do Output.println();
70      do Output.printString("The String after changing to Uppercase: ");
71      do s.toUpperCase();
72      do Output.printString(s.get());
```

VM Emulator Output



6. toLowerCase

```
48      /** converts all the chracters of the string to lower case. */
49      method void toLowerCase(){
50        var int i;
51        let i = 0;
52
53        while(i < length){
54          if((arr[i] > 64) & (arr[i] < 91)){
55            let arr[i] = (arr[i] + 32);
56          }
57          let i = i + 1;
58        }
59        return;
60      }
```

```
71        let str = "ABCDEFGH";
72        let s = String2.new(str);
73
74        do Output.printString("The Given String is: ");
75        do Output.printString(str);
76        do Output.println();
77        do Output.println();
78        do Output.printString("The String after changing to Lowercase: ");
79        do s.toLowerCase();
80        do Output.printString(s.get());
```

VM Emulator Output

Virtual Machine Emulator (2.5) - C:\Users\hp\Downloads\nand2tetris\nand2tetris\tools\E

File View Run Help

Animate: No animation | View: Screen | Format: Decimal

Program

51	not	
52	pop	local 0
53	goto	String2.compareTo$I...
	label	String2.compareTo$I...
54	push	constant 0
55	pop	local 0
	label	String2.compareTo$I...
56	push	local 3
57	push	constant 1
58	add	
59	pop	local 3
60	goto	String2.compareTo$...
	label	String2.compareTo$...
61	push	local 0
62	return	

Stack

Call Stack

Static

0	0
1	0
2	0
3	0
4	0

Local

0	0
1	0
2	0
3	0
4	0

Argument

0	0
1	0
2	0
3	0
4	0

This

That

Temp

0	0
1	0

```
The Given String is: ABCDEFGH
The String after changing to Lowercase: abcdefgh
```

Global Stack

256	0
257	0
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

RAM

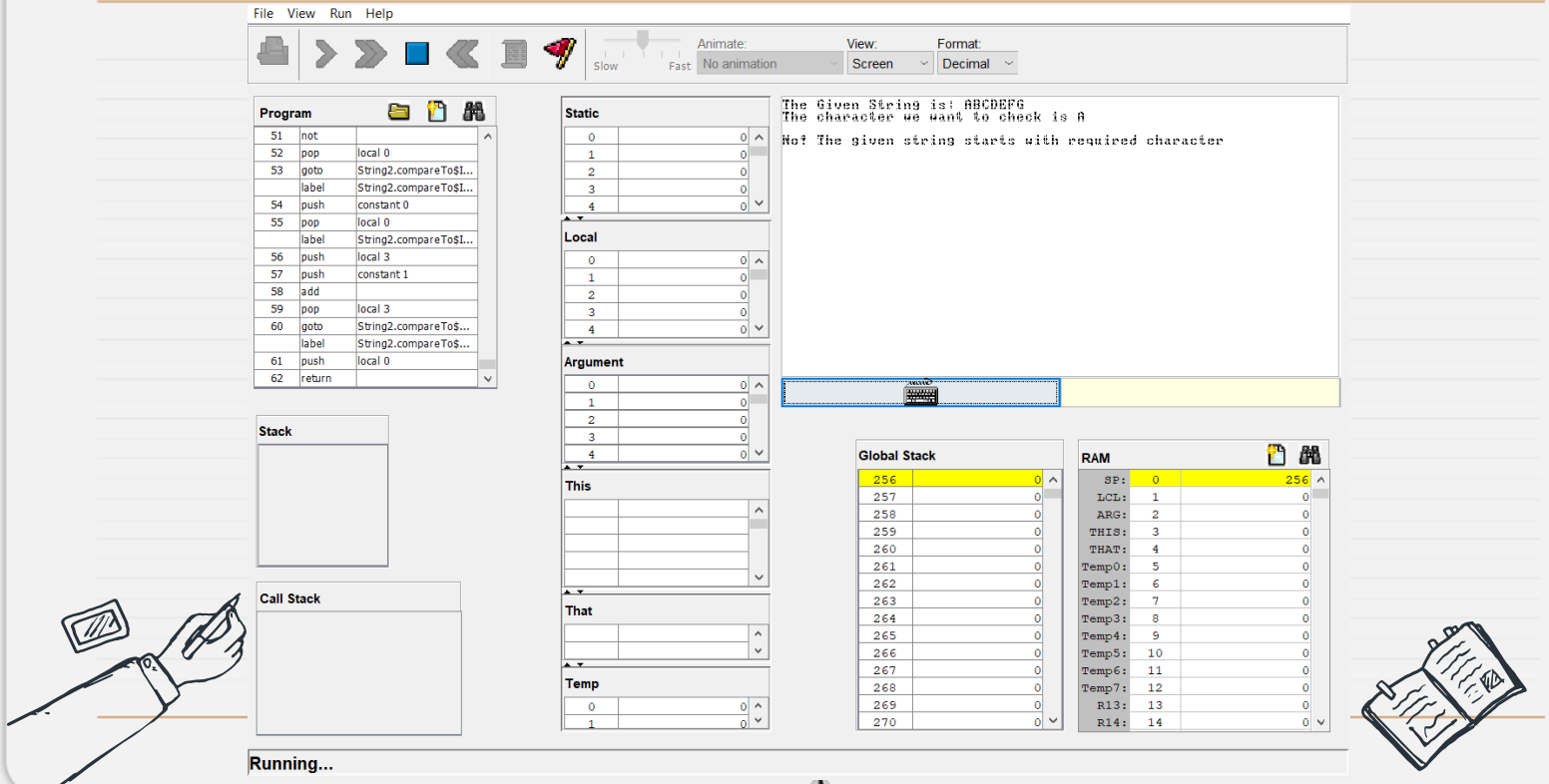
SP:	0	256
LCL:	1	0
ARG:	2	0
THIS:	3	0
THAT:	4	0
Temp0:	5	0
Temp1:	6	0
Temp2:	7	0
Temp3:	8	0
Temp4:	9	0
Temp5:	10	0
Temp6:	11	0
Temp7:	12	0
R13:	13	0
R14:	14	0

7. startwith

```
62      /** checks if string starts with the specified character. */
63      method boolean startwith(char ch){
64        var int i;
65        let i = 0;
66
67        if(arr[0] = ch){
68          return true;
69        }
70        else{
71          return false;
72        }
73      }
```

```
    let str = "ABCDEFGH";
    let s = String2.new(str);

    do Output.printString("The Given String is: ");
    do Output.printString(str);
    do Output.println();
    do Output.printString("The character we want to check is A ");
    do Output.println();
    do Output.println();

    if(s.startwith(65)){
      do Output.printString("Yes! The given string starts with required character");
    }
    else {
      do Output.printString("No! The given string starts with required character");
    }
```

VM Emulator Output

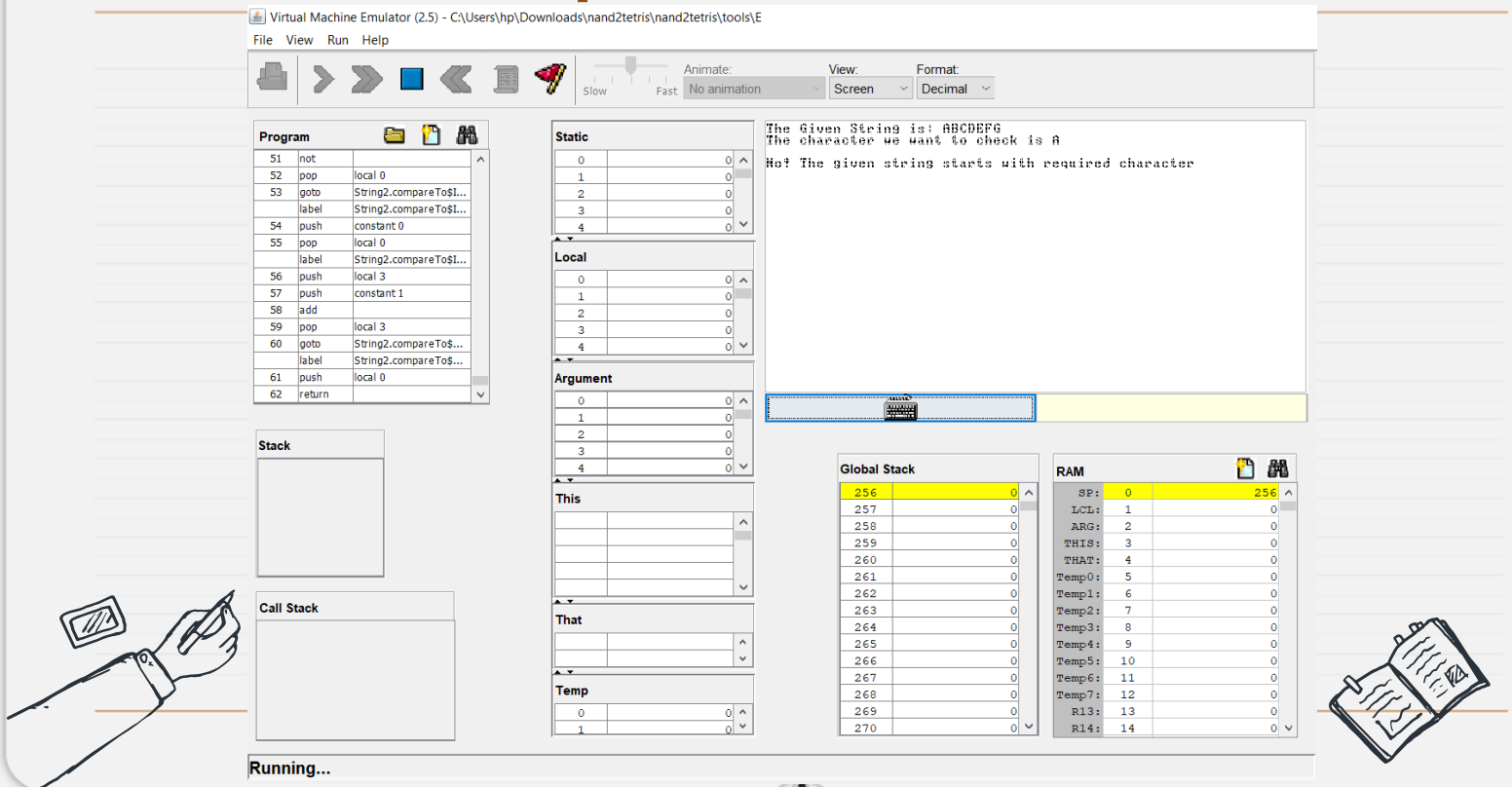


8. endswith

```
75      /** checks if string ends with the specified character. */
76      method boolean endwith(char ch){
77        var int i;
78        let i = 0;
79
80        if(arr[(length-1)] = ch){
81          return true;
82        }
83        else{
84          return false;
85        }
86      }
```

```
25      let str = "ABCDEFGH";
26      let s = String2.new(str);
27
28      do Output.printString("The Given String is: ");
29      do Output.printString(str);
30      do Output.println();
31      do Output.printString("The character we want to check is A ");
32      do Output.println();
33      do Output.println();
34
35      if(s.endwith(65)){
36        do Output.printString("Yes! The given string starts with required character");
37      }
38      else {
39        do Output.printString("No! The given string starts with required character");
40      }
```

VM Emulator Output



9. subString

```
103       /** returns the sub-string of the given index. */
104       method String subString(int start, int end){
105         var String str;
106         var int i;
107
108         let str = String.new(end - start);
109         let i = start;
110
111         while(i < end) {
112           do str.appendChar(arr[i]);
113           let i = i + 1;
114         }
115
116         return str;
117       }
```

```
40      let str = "ABCDEFGH";
41      let s = String2.new(str);
42
43      do Output.printString("The Given String is: ");
44      do Output.printString(str);
45      do Output.println();
46      do Output.printString("The required sub-string from index(1 to 4) is: ");
47      do Output.printString(s.subString(1,4));
```

VM Emulator Output

Virtual Machine Emulator (2.5) - C:\Users\hp\Downloads\nand2tetris\nand2tetris\tools\E

File View Run Help

Slow Fast Animate: No animation View: Screen Format: Decimal

Program

51	not	
52	pop	local 0
53	goto	String2.compareTo$I...
	label	String2.compareTo$I...
54	push	constant 0
55	pop	local 0
	label	String2.compareTo$I...
56	push	local 3
57	push	constant 1
58	add	
59	pop	local 3
60	goto	String2.compareTo$...
	label	String2.compareTo$...
61	push	local 0
62	return	

Stack

Call Stack

Static

0	0
1	0
2	0
3	0
4	0

Local

0	0
1	0
2	0
3	0
4	0

Argument

0	0
1	0
2	0
3	0
4	0

This

That

Temp

0	0
1	0

```
The Given String is: ABCDEFG
The required sub-string from index(1 to 4) is: BCD
```

Global Stack

256	0
257	0
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

RAM

SP:	0	256
LCL:	1	0
ARG:	2	0
THIS:	3	0
THAT:	4	0
Temp0:	5	0
Temp1:	6	0
Temp2:	7	0
Temp3:	8	0
Temp4:	9	0
Temp5:	10	0
Temp6:	11	0
Temp7:	12	0
R13:	13	0
R14:	14	0

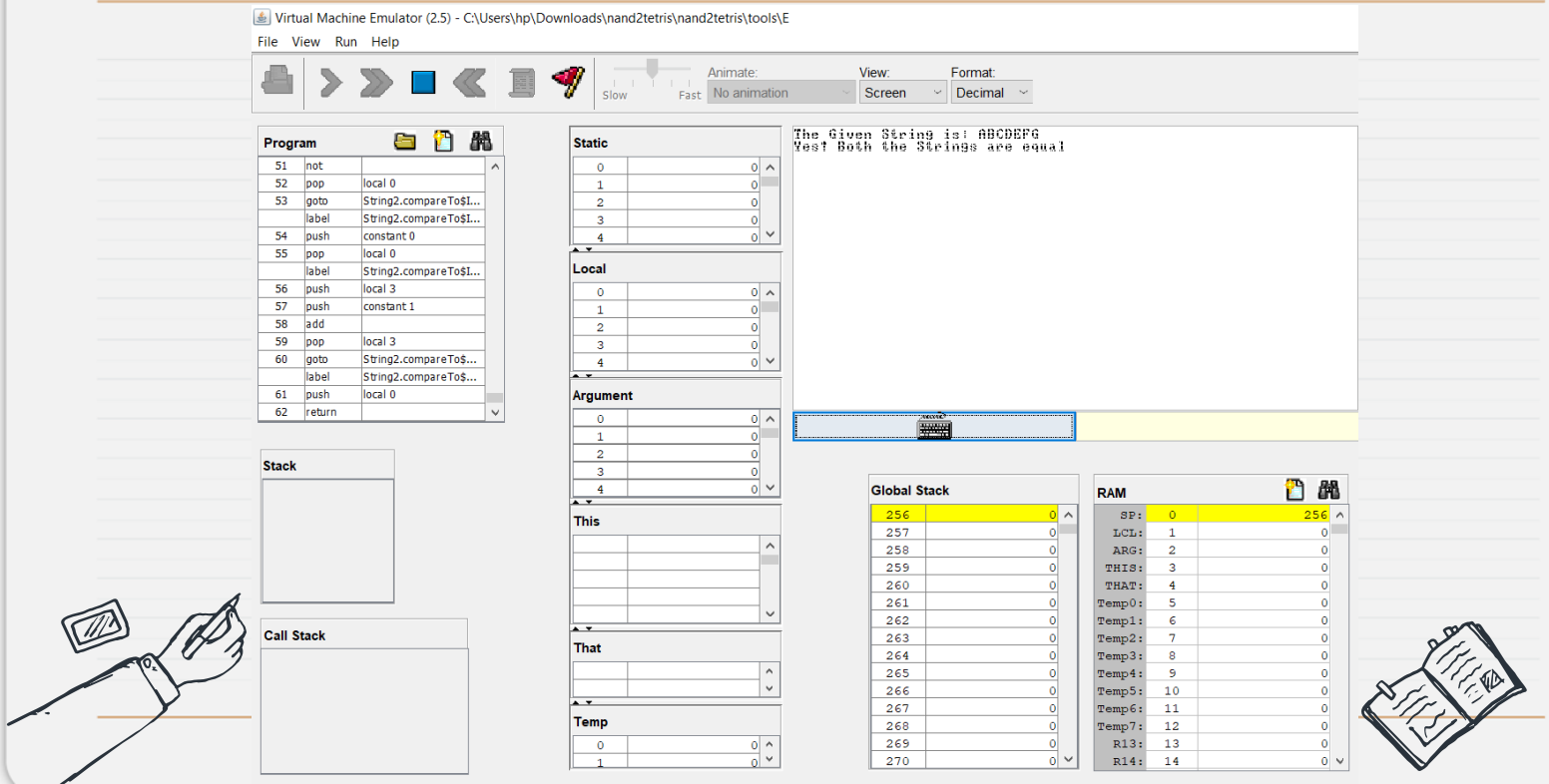
10. compareTo

```
118     /** compares two string and return true if they are equal */
119     method boolean compareTo(String p){
120         var boolean isCompare;
121         var Array array;
122         var int j;
123         var int i;
124
125         if(~(length = p.length())){
126           return false;
127         }
128
129         let j = p.length();
130         let array = Array.new(j);
131
132         while(i < j){
133           let array[i] = p.charAt(i);
134           let i = i + 1;
135         }
```

```
137         let i = 0;
138         while(i < p.length()){
139           if(arr[i] = array[i]){
140             let isCompare = true;
141           } else {
142             let isCompare = false;
143           }
144           let i = i + 1;
145         }
146         return isCompare;
147     }
```

```
46      let str = "ABCDEFGH";
47      let s = String2.new(str);
48
49      do Output.printString("The Given String is: ");
50      do Output.printString(str);
51      do Output.println();
52      if(s.compareTo("ABCDEFG")){
53        do Output.printString("Yes! Both the Strings are equal");
54      }
55      else {
56        do Output.printString("No! Both the Strings are not equal");
57      }
```

VM Emulator Output

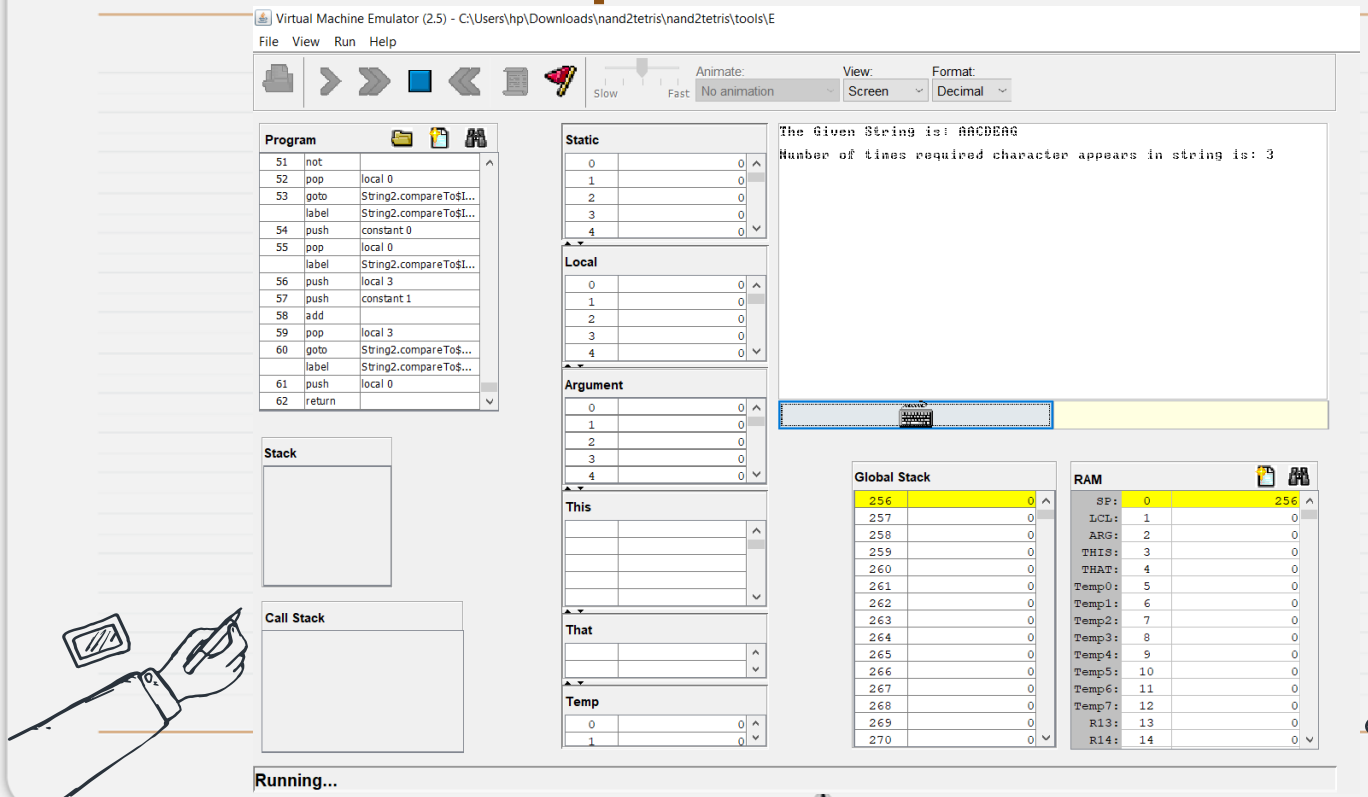


11. count

```
104     /** returns how many times the given chracter occur in the given string. */
105     method int count(char c){
106       var int i;
107       var int k;
108
109       while(i < length){
110         if(arr[i] = c){
111           let k = k + 1;
112         }
113         let i = i + 1;
114       }
115       return k;
116     }
```

```
56      let str = "AACDEAG";
57      let s = String2.new(str);
58
59      do Output.printString("The Given String is: ");
60      do Output.printString(str);
61      do Output.println();
62      do Output.println();
63      do Output.printString("Number of times required character appears in string is: ");
64      do Output.printInt(s.count(65));
```

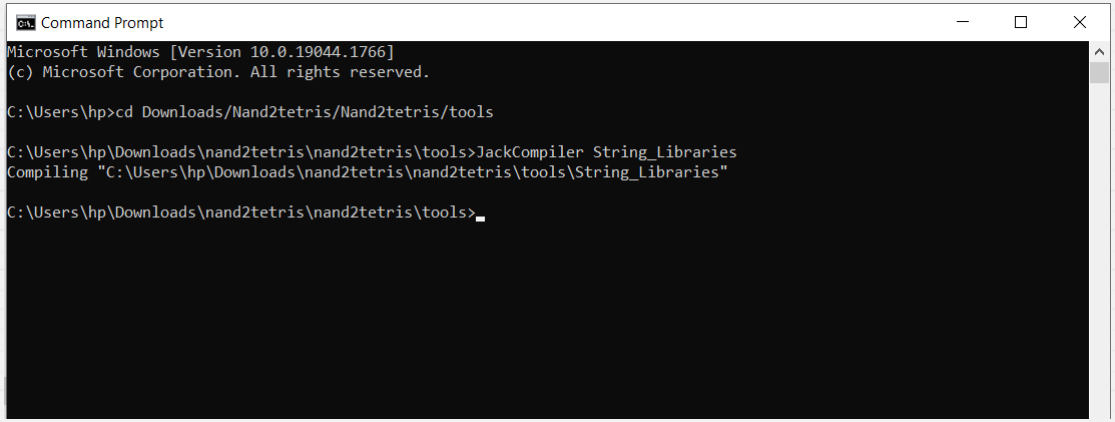
VM Emulator Output



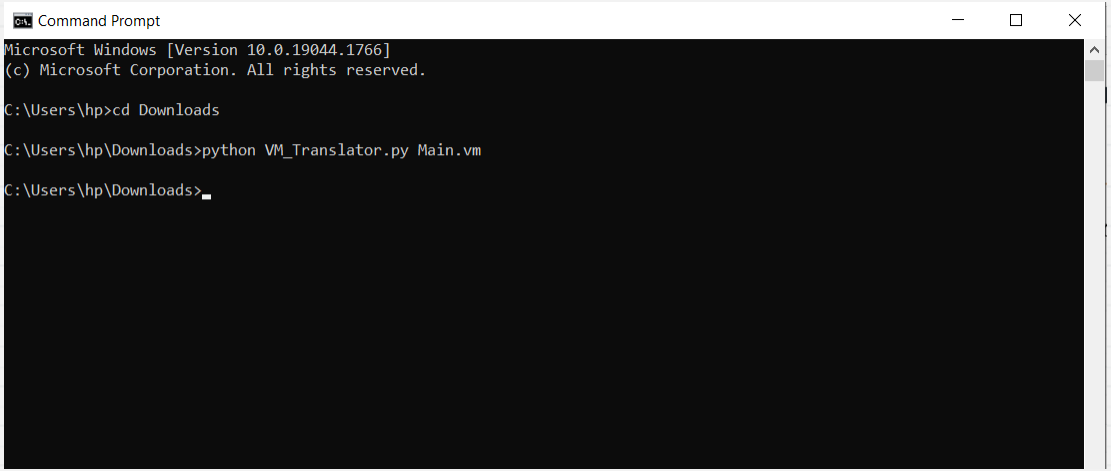
PART – 2

Write your own general purpose assembler to convert the assembly program into 16 bit machine codes also use the Nand2tetris assembler for conversion. (Your own assembler should be developed by you and is different from the assembler tool supplied along with the book)

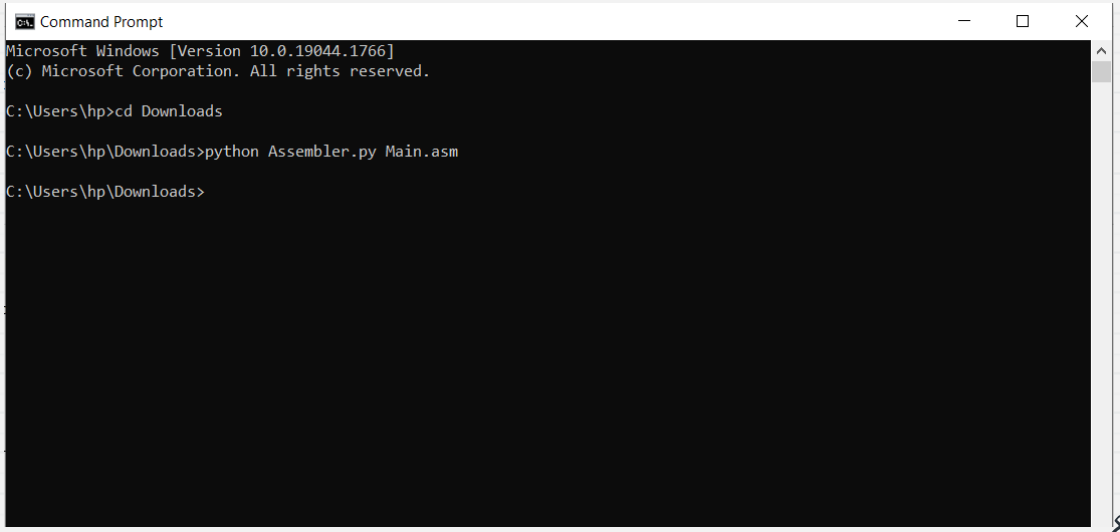
- The Jack Code we wrote is compiled into vm file using the Jack Compiler.



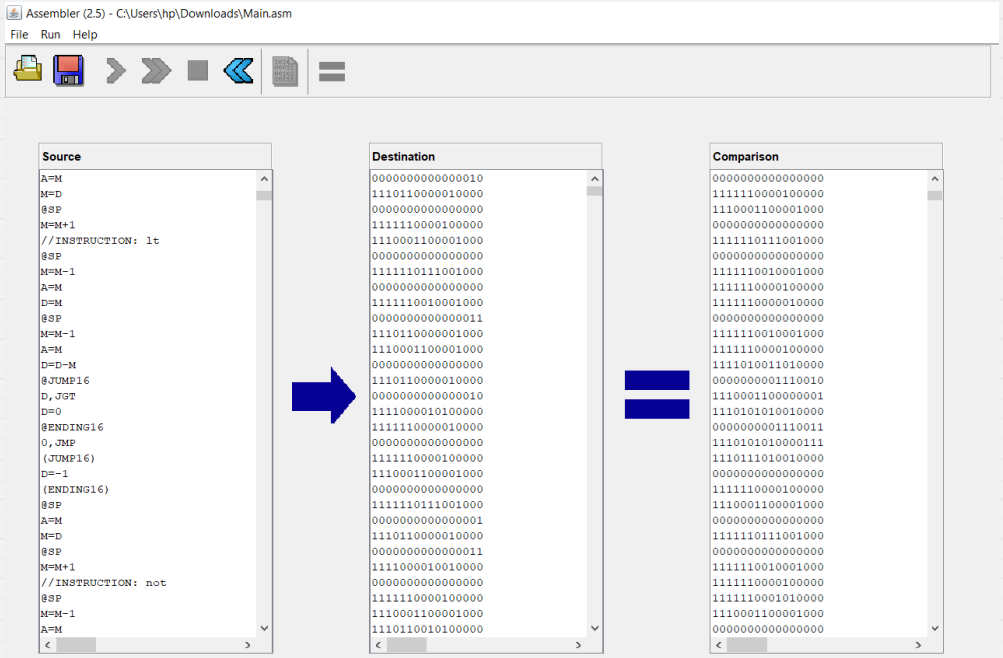
- This vm file is converted into the asm file by the VM translator we implemented in python.



- This asm file is converted into the hack file by the Assembler we implemented in python.



Comparision of .hack file generated by our own and assembler in Nand2tetris software suite



PART – 3

Dump the machine codes generated by your own assembler into the RAM.hdl. Then design the architecture for implementing your machine instructions to get the result. (Here I presume that you already have the hdl scripts of all the hardware's used by the CPU. Additional thing what you have to do is connecting the address and data buses during the execution).

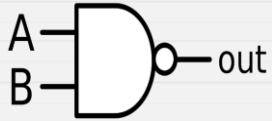
Hack CPU in Jack

NAND :

Jack Code:

```
// Implementation of Nand gate in jack

class Nand {
  function int nand(int a, int b){
     var int out;
     let out = 0;

     if(a = 1 & b = 1){
       let out = 0;
     }
     else{
       let out = 1;
     }
     return out;
   }
}
```

LOGIC CIRCUIT AND TRUTH TABLE :



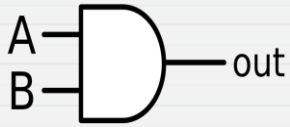
Input		Output
A	B	Y= $\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

AND :

Jack Code:

```
// Implementation of And-gate using Nand in jack

class And {
  function Int and(int a, int b){
     var int out;
     let out = Nand.nand(Nand.nand(a, b), Nand.nand(a, b));
     return out;
   }
}

```

LOGIC CIRCUIT AND TRUTH TABLE :



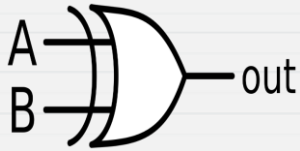
Inputs		Output
A	B	Y=A.B
0	0	0
0	1	0
1	0	0
1	1	1

XOR :

Jack Code:

```
// Implementation of Xor-gate using Nand gate in jack

class Xor {
  function Int xor(int a, int b){
     var int na, nb, c, d, out;

     let na = Not.not(a);
     let nb = Not.not(b);
     let c = And.and(na, b);
     let d = And.and(a, nb);
     let out = Or.or(c, d);

     return out;
   }
}
```

LOGIC CIRCUIT AND TRUTH TABLE :



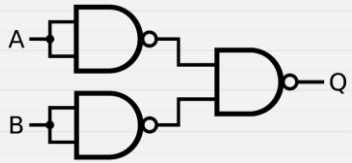
Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

OR :

Jack Code:

```
// Implementation Or-gate using nand gate in jack

class Or {
  function Int or(int a, int b){
     var int out;
     let out = Nand.nand(Nand.nand(a, a), Nand.nand(b, b));
     return out;
   }
}

```

LOGIC CIRCUIT AND TRUTH TABLE :



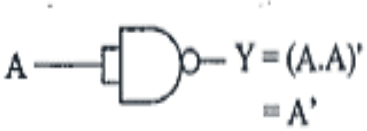
Inputs		Output
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT :

Jack Code:

```
// Implementation of Not-gate using Nand gate

class Not {
  function Int not(int in){
     var int out;
     let out = Nand.nand(in, in);

     return out;
   }
}
```

LOGIC CIRCUIT AND TRUTH TABLE :



Input	Output
A	$Y=\overline{A}$
0	1
1	0

MUX :

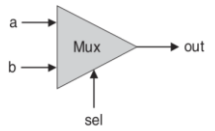
Jack Code:

```
// Implementation of Mux-gate using Nand gate

class Mux {
  function Int mux(int a, int b, int sel){
     var int out;

     let out = Nand.nand(Nand.nand(a,(Nand.nand(sel,sel))),Nand.nand(b,sel));

     return out;
   }
}
```

LOGIC CIRCUIT AND TRUTH TABLE :

a	b	sel	out
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

sel	out
0	a
1	b

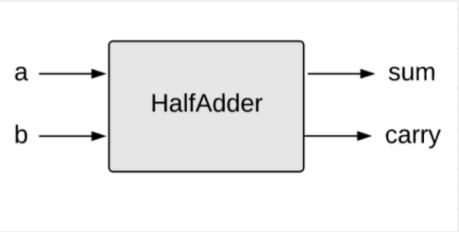


HALFADDER :

Jack Code:

```
// Implementation of Half-Adder in jack.

class HalfAdder {
  function int sum(int a, int b){
     var int out;
     let out = Xor.xor(a,b);
     return out;
   }

   function int carry(int a, int b){
      var int out;
      let out = And.and(a, b);
      return out;
    }
}
```

LOGIC CIRCUIT AND TRUTH TABLE :



a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

FULLADDER :

Jack Code:

```
// Implementation of Full-Adder in jack.

class FullAdder {
  function int sum(int a, int b, int c){
     var int out;
     let out = Xor.xor(Xor.xor(a,b),c);
     return out;
   }

   function int carry(int a, int b, int c){
      var int out;
      let out = Or.or(Or.or(And.and(a, b),And.and(b, c)),And.and(a, c));
      return out;
    }
}
```

LOGIC CIRCUIT AND TRUTH TABLE :

a	b	c	sum	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

to BINARY

Jack Code:

```
// Jack code to convert decimal into 16-bit binary representation.

class toBinary {
  function Array toBin(int a){
    var Array f;
    var int pos,i;

    let f = Array.new(16);


      let a = a;
      let pos = 15;
      while(a > 0){
        let f[pos - i] = toBinary.remainder(a, 2);
        let a = a / 2;
        let pos = pos - 1;
      }

    return f;
  }
```

```
23      function int remainder(int a, int b){
24        var int minus;
25        var int quoitent;
26        var int rem;
27
28        let minus = a-b;
29        let quoitent = 0;
30
31        while(minus > -1){
32          let minus = minus - b;
33          let quoitent = quoitent + 1;
34        }
35        let rem = minus + b;
36        return rem;
37      }
38
39  }
40
```

to BINARYALL

Jack Code:

```
// Jack code to convert 16-bit binary representation into corresponding decimal.

class toBinaryall {
  function Array toBinall(int a){
    var Array f, f1,f2,notf2,f3;
    var int pos,i,cf1,cf2,cnotf2,cf;
    var int c,d,e,g,h,i,j,k,l,m,n,o,p,q,z;

    let f = Array.new(16);
    let f1 = Array.new(16);
    let f2 = Array.new(16);

    if (a > 0){
      let f = toBinary.toBin(a);
    }

    if(a < 0){
      let a = -(a);
      let f1 = toBinary.toBin(a);

      let i = 0;
      while(i < 16){
        let f2[i] = Not.not(f1[i]);
        let i = i + 1;
      }
```

```
26          let f3 = toBinary.toBin(1);
27
28        let f[15] = HalfAdder.sum(f2[15],f3[15]);
29        let c = HalfAdder.carry(f2[15],f3[15]);
30
31        let f[14] = FullAdder.sum(f2[14],f3[14],c);
32        let d = FullAdder.carry(f2[14],f3[14],c);
33
34        let f[13] = FullAdder.sum(f2[13],f3[13],d);
35        let e = FullAdder.carry(f2[13],f3[13],d);
36
37        let f[12] = FullAdder.sum(f2[12],f3[12],e);
38        let z = FullAdder.carry(f2[12],f3[12],e);
39
40        let f[11] = FullAdder.sum(f2[11],f3[11],z);
41        let g = FullAdder.carry(f2[11],f3[11],z);
42
43        let f[10] =FullAdder.sum(f2[10],f3[10],g);
44        let h = FullAdder.carry(f2[10],f3[10],g);
45
46        let f[9] = FullAdder.sum(f2[9],f3[9],h);
47        let i = FullAdder.carry(f2[9],f3[9],h);
48
49        let f[8] = FullAdder.sum(f2[8],f3[8],i);
50        let j = FullAdder.carry(f2[8],f3[8],i);
```

```
52        let f[7] = FullAdder.sum(f2[7],f3[7],j);
53        let k  =  FullAdder.carry(f2[7],f3[7],j);
54
55        let f[6] = FullAdder.sum(f2[6],f3[6],k);
56        let l = FullAdder.carry(f2[6],f3[6],k);
57
58        let f[5] = FullAdder.sum(f2[5],f3[5],l);
59        let m = FullAdder.carry(f2[5],f3[5],l);
60
61        let f[4] = FullAdder.sum(f2[4],f3[4],m);
62        let n = FullAdder.carry(f2[4],f3[4],m);
63
64        let f[3] = FullAdder.sum(f2[3],f3[3],n);
65        let o = FullAdder.carry(f2[3],f3[3],n);
66
67        let f[2] = FullAdder.sum(f2[2],f3[2],o);
68        let p = FullAdder.carry(f2[2],f3[2],o);
69
70        let f[1] = FullAdder.sum(f2[1],f3[1],p);
71        let q = FullAdder.carry(f2[1],f3[1],p);
72
73        let f[0] = FullAdder.sum(f2[0],f3[0],q);
74
75      }
76      return f;
77    }
78
79  }
```

to Decimal

Jack Code:

```
// Jack code to convert 16-bit binary representation into corresponding decimal.

class toDecimal {
  function int toDec(Array n){
    var int i,j,num,sum;
    var Array s;
    let s = Array.new(16);
    let i = 0;
    while(i < 16){
      let s[i] = n[i];
      let i = i + 1;
    }

    let i = 15;
    let j = 1;
    while(i > -1){
      let num = (s[i]) * (j);
      let sum = sum + num;
      let i = i - 1;
      let j = j * 2;
    }

    return sum;
  }

}
```

to Decimal8

Jack Code:

```
// Jack code to convert 8-bit binary representation into corresponding decimal.

class toDecimal8 {
  function int toDec8(Array n){
    var int i,j,num,sum;
    var Array s;
    let s = Array.new(8);
    let i = 0;
    while(i < 8){
      let s[i] = n[i];
      let i = i + 1;
    }

    let i = 7;
    let j = 1;
    while(i > -1){
      let num = (s[i]) * (j);
      let sum = sum + num;
      let i = i - 1;
      let j = j * 2;
    }

    return sum;
  }

}
```

16-BIT AND :

Jack Code:

```
// Implementation of 16-bit And gate in Jack.

class And16 {
  function Array and16(int a, int b){
     var Array out;
     var int Bina,Binb,i;

     let Bina = toBinaryall.toBinall(a);
     let Binb = toBinaryall.toBinall(b);
     let out = Array.new(16);

     let i = 0;
     while(i < 16){
       let out[i] = And.and(Bina[i],Binb[i]);
       let i = i + 1;
     }
     return out;
   }
}
```

16-BIT ADDER:

Jack Code:

```
// Implementation of 16-bit Adder in jack.

class Add16 {
  function Array add16(int x, int y){
     var Array out,a,b;
     var int c,d,e,f,g,h,i,j,k,l,m,n,o,p,q;

     let a = toBinaryall.toBinall(x);
     let b = toBinaryall.toBinall(y);

     let out = Array.new(16);

     let out[15] = HalfAdder.sum(a[15],b[15]);
     let c = HalfAdder.carry(a[15],b[15]);

    let out[14] = FullAdder.sum(a[14],b[14],c);
    let d = FullAdder.carry(a[14],b[14],c);

    let out[13] = FullAdder.sum(a[13],b[13],d);
    let e = FullAdder.carry(a[13],b[13],d);

    let out[12] = FullAdder.sum(a[12],b[12],e);
    let f = FullAdder.carry(a[12],b[12],e);
```

```
    let out[11] = FullAdder.sum(a[11],b[11],f);
    let g = FullAdder.carry(a[11],b[11],f);

    let out[10] =FullAdder.sum(a[10],b[10],g);
    let h = FullAdder.carry(a[10],b[10],g);

    let out[9] = FullAdder.sum(a[9],b[9],h);
    let i = FullAdder.carry(a[9],b[9],h);

    let out[8] = FullAdder.sum(a[8],b[8],i);
    let j = FullAdder.carry(a[8],b[8],i);

    let out[7] = FullAdder.sum(a[7],b[7],j);
    let k  =  FullAdder.carry(a[7],b[7],j);

    let out[6] = FullAdder.sum(a[6],b[6],k);
    let l = FullAdder.carry(a[6],b[6],k);

    let out[5] = FullAdder.sum(a[5],b[5],l);
    let m = FullAdder.carry(a[5],b[5],l);

    let out[4] = FullAdder.sum(a[4],b[4],m);
    let n = FullAdder.carry(a[4],b[4],m);
```

16-BIT INCREMENTOR :

Jack Code:

```
// Implementation of 16-bit Incrementor in jack.

class Inc16 {
  function Array inc16(int a){
     var Array out;

     let out = Add16.add16(a,1);

     return out;
   }
}
```

16-BIT NOT :

Jack Code:

```
// 16-bit implementation of Not-gate in jack.

class Not16 {
  function Array not16(int a){
     var Array out;
     var int a1,i;

     let a1 = toBinaryall.toBinall(a);
     let out = Array.new(16);

     let i = 0;
     while(i < 16){
       let out[i] = Not.not(a1[i]);
       let i = i + 1;
     }
     return out;
   }
}
```

16-BIT OR :

Jack Code:

```
// Implementation of 16-bit Or-gate in jack

class Or16 {
  function Array or16(int a, int b){
     var Array out;
     var int Bina,Binb,i;

     let Bina = toBinary.toBin(a);
     let Binb = toBinary.toBin(b);
     let out = Array.new(16);

     let i = 0;
     while(i < 16){
       let out[i] = Or.or(Bina[i],Binb[i]);
       let i = i + 1;
     }
     return out;
   }
}

```

16-BIT MUX :

Jack Code:

```
// Implementation 16-bit of Mux-gate in jack.

class Mux16 {
  function Array mux16(int a, int b, int sel){
     var Array out,x,y;
     var int i;

     let x = toBinaryall.toBinall(a);
     let y = toBinaryall.toBinall(b);

     let out = Array.new(16);

     if(sel = 1){
       let i = 0;
       while(i < 16){
         let out[i] = y[i];
         let i = i + 1;
       }
     }
     if(sel = 0){
       let i = 0;
       while(i < 16){
         let out[i] = x[i];
         let i = i + 1;
       }
     }
     return out;
   }
}
```

OR3input :

Jack Code:

```
// Implementation of Or-gate with three inputs in jack.

class Or3Input {
  function Int or3input(int a, int b, int c){
     var int t1,out;

     let t1 = Or.or(a,b);
     let out = Or.or(t1,c);

     return out;
   }
}
```

OR-8WAY:

Jack Code:

```
// Implementation of Or-8-way in jack.

class Or8Way {
  function int or8way(int x){
     var int a,b,c,d,e,f,out;
     var Array l;
     let l = Array.new(8);

     let l = toBinary8.toBin8(x);

     let a = Or.or(l[0],l[1]);
     let b = Or.or(a,l[2]);
     let c = Or.or(b,l[3]);
     let d = Or.or(c,l[4]);
     let e = Or.or(d,l[5]);
     let f = Or.or(e,l[6]);
     let out = Or.or(f,l[7]);

     return out;
   }
}
```

DFF:

Jack Code:

```
// Implementation of Data Flip-Flop in jack.

class DFF{
  function Array dff(int data, int clk){
    var int Sout,Notout,Rout,notQ,Q;
     if(clk = 1){
       let Sout = Nand.nand(data,clk);
       let Notout = Not.not(data);

       let Rout = Nand.nand(clk,Notout);
       let notQ = Nand.nand(Rout,Q);
       let Q = Nand.nand(Sout,notQ);
     }
     return Q;
   }
}

```

DFF16:

Jack Code:

```
// Implementation of Data Flip-Flop in jack.

class DFF{
  function Array dff(int data, int clk){
    var int Sout,Notout,Rout,notQ,Q;
     if(clk = 1){
       let Sout = Nand.nand(data,clk);
       let Notout = Not.not(data);

       let Rout = Nand.nand(clk,Notout);
       let notQ = Nand.nand(Rout,Q);
       let Q = Nand.nand(Sout,notQ);
     }
     return Q;
   }
}

```

REGISTER :

Jack Code:

```
// Implementation of register in jack.

class Register {
  function Int register(int prevout, int in, int load){
     var int t;
     var int stage1;

     let stage1 = Mux.mux(prevout,in,load);
     let t = DFF.dff(stage1,1);
     return stage1;
   }
}
```

16-BIT REGISTER :

Jack Code:

```
// Implementation of 16-bit register in jack.

class Register16 {
  function Array register16(int prevout, int in, int load){
     var Array out,a,b;
     var int i;

     let out = Array.new(16);
     let a = toBinaryall.toBinall(prevout);
     let b = toBinaryall.toBinall(in);

     let i = 0;
     while(i < 16){
       let out[i] = Register.register(a[i],b[i],load);
       let i = i + 1;
     }

     return out;
   }
}
```

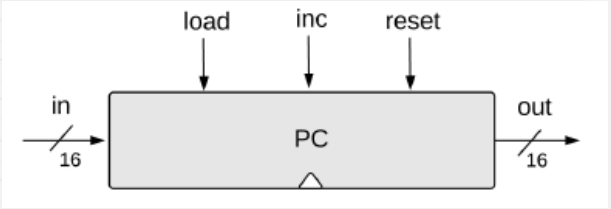
PROGRAM COUNTER :

- The Program Counter (PC) is a register that keep track of which instruction should be fetched and executed next.
- The PC supports three control operation.

  Reset : fetches the first instruction

  Next : fetches the next instruction

  Goto : fetches the instruction n



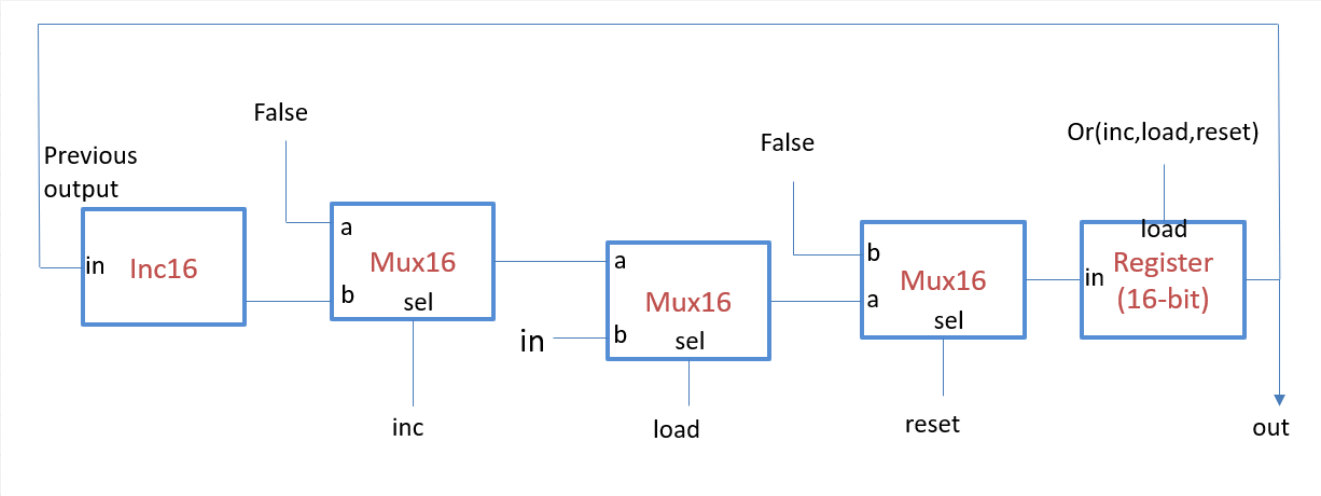
- PC contains one input, one output, and three control input pins.

  Input control bits :
  1. Reset
  2. Load
  3. Increment

- These input controls pins have a priority order associated with it. Depending on the priority, the program counter will behaves.

- **Priority order :**
  1. Reset
  2. Load
  3. Increment

- PC perform the operations in according to the priority order. If we asked to perform two operation at a time, it performs only the operation which has the highest priority order.


Previous
output
in
Inc16
False
a
b
Mux16
sel
inc
in
a
b
Mux16
sel
load
False
b
a
Mux16
sel
reset
Or(inc,load,reset)
load
in
Register
(16-bit)
out

Jack Code:

```
// Implementation Program Counter in jack.

class PC {
  function Array pc(int in, int load,int inc,int reset){
     var Array fl,prevout,FBout,incout,w0,w1,cout;
     var int regload,cFBout,cincout,cw0,cw1,cfl,cprevout,ccout;

     let fl = Array.new(16);
     let prevout = Array.new(16);
     let FBout = Array.new(16);

     let cFBout = toDecimal.toDec(FBout);

     let regload = Or3Input.or3input(inc,load,reset);
     let incout = Inc16.inc16(cFBout);

     let cincout = toDecimal.toDec(incout);

     let w0 = Mux16.mux16(cFBout,cincout,inc);

     let cw0 = toDecimal.toDec(w0);
     let w1 = Mux16.mux16(cw0,in,load);

```

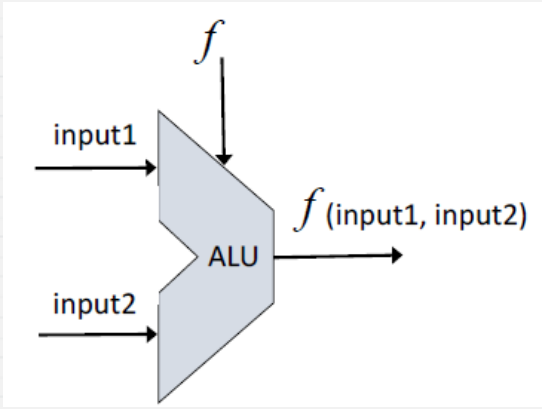
```
24        let cw1 = toDecimal.toDec(w1);
25        let cfl = toDecimal.toDec(fl);
26
27        let cout = Mux16.mux16(cw1,cfl,reset);
28
29        let cprevout = toDecimal.toDec(prevout);
30        let ccout = toDecimal.toDec(cout);
31
32
33        let FBout = Register16.register16(cprevout,ccout,regload);
34
35        return FBout;
36
37    }
38  }
```

ALU

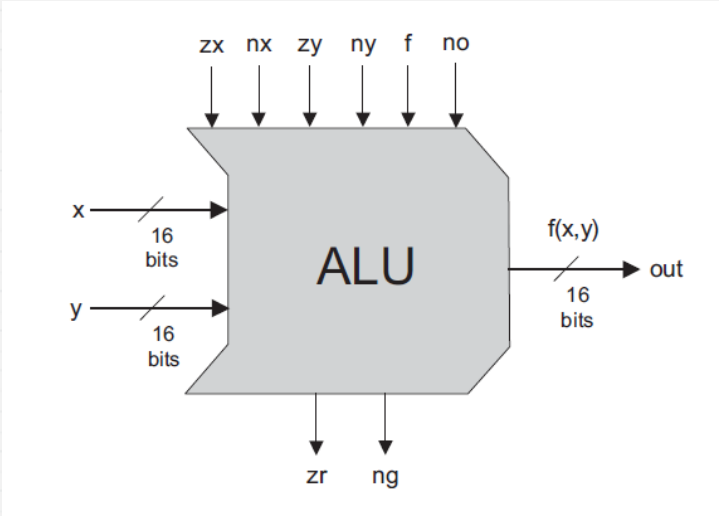
- The ALU computes a function on the two inputs, and outputs the result.
- f: one out of family of pre-defined arithmetic and logical function

  Arithmetic functions: integer addition, multiplication, division…...

  Logical function: And, Or, Xor



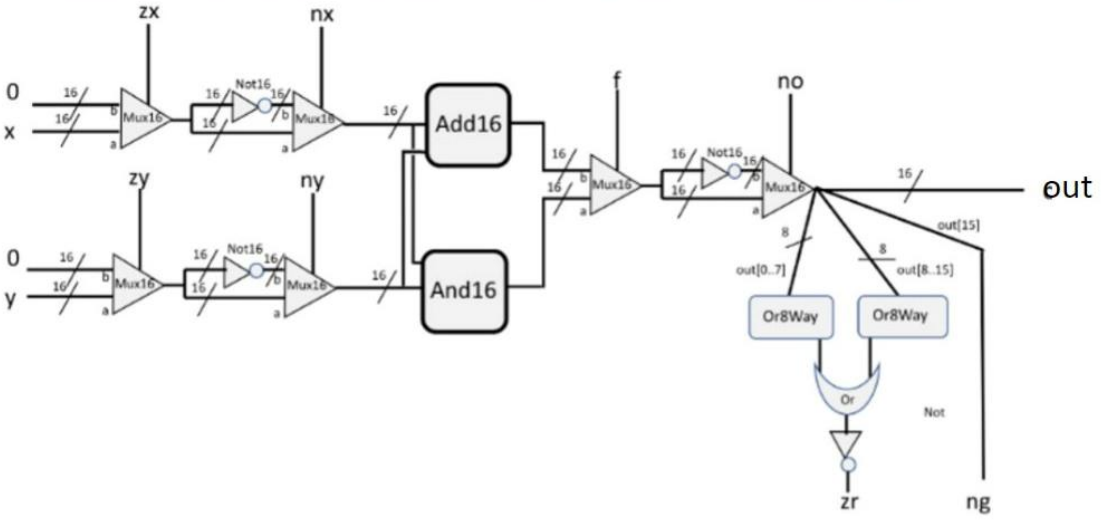
- To cause ALU to compute a function, set the control bits to one of the binary combinations listed below



control bits

zx	nx	zy	ny	f	no	out
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	!x
1	1	0	0	0	1	!y
0	0	1	1	1	1	-x
1	1	0	0	1	1	-y
0	1	1	1	1	1	x+1
1	1	0	1	1	1	y+1
0	0	1	1	1	0	x-1
1	1	0	0	1	0	y-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	1	1	1	y-x
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

pre-setting the x input		pre-setting the y input		selecting between computing + or &	post-setting the output	Resulting ALU output
zx	nx	zy	ny	f	no	out
if zx then x=0	if nx then x=!x	if zy then y=0	if ny then y=!y	if f then out=x+y else out=x&y	if no then out=!out	out(x,y)=



- The ALU contains two control bits (zr, ng)
  if (out = 0) then zr = 1, else zr = 0
  if (out < 0) then ng =1, else ng = 0

Jack Code:

```
// Implementation of Arithmetic Logic Unit in jack.

class ALU {
  function Array alu(int zx, int nx, int zy, int ny, int f, int no, int x, int y){
    var Array out, x1, y1, x2, y2,notx1, notx2, noty1,noty2;
    var Array addout, andout,fout, notfout;
    var Arary outfirst, outsecond;
    var int fls, tr, cx1,coutfirst,coutsecond,cnotx1,cy1,cx2,cy2,caddout,candout;
    var int i,j,a,cfout,cnotfout,cnoty1, final;
    var int ng,zr,zr0,zr1,zr2;

    let outfirst = Array.new(8);
    let outsecond = Array.new(8);

    let fls = 0;
    let tr = 0;

    let x1 = Mux16.mux16(x, fls, zx);
    let y1 = Mux16.mux16(y,fls,zy);

    let cx1 = toDecimal.toDec(x1);
    let notx1 = Not16.not16(cx1);
    let cnotx1 = toDecimal.toDec(notx1);
    let x2 = Mux16.mux16(cx1,cnotx1,nx);

    let cy1 = toDecimal.toDec(y1);
    let noty1 = Not16.not16(cy1);
    let cnoty1 = toDecimal.toDec(noty1);
    let y2 = Mux16.mux16(cy1,cnoty1,ny);

```

```
    let cx1 = toDecimal.toDec(x1);
    let notx1 = Not16.not16(cx1);
    let cnotx1 = toDecimal.toDec(notx1);
    let x2 = Mux16.mux16(cx1,cnotx1,nx);

    let cy1 = toDecimal.toDec(y1);
    let noty1 = Not16.not16(cy1);
    let cnoty1 = toDecimal.toDec(noty1);
    let y2 = Mux16.mux16(cy1,cnoty1,ny);
```

```
31      let cx2 = toDecimal.toDec(x2);
32      let cy2 = toDecimal.toDec(y2);
33
34      let addout = Add16.add16(cx2,cy2);
35      let andout = And16.and16(cx2,cy2);
36
37      let caddout = toDecimal.toDec(addout);
38      let candout = toDecimal.toDec(andout);
39
40      let fout = Mux16.mux16(candout,caddout,f);
41
42      let cfout = toDecimal.toDec(fout);
43      let notfout = Not16.not16(cfout);
44      let cnotfout = toDecimal.toDec(notfout);
45
46      let out = Mux16.mux16(cfout,cnotfout,no);
47
48
49      return out;
50    }
```

```
function Array zr(int zx, int nx, int zy, int ny, int f, int no, int x, int y){
  var Array out, x1, y1, x2, y2,notx1, notx2, noty1,noty2;
  var Array addout, andout,fout, notfout;
  var Arary outfirst, outsecond;
  var int fls, tr, cx1,coutfirst,coutsecond,cnotx1,cy1,cx2,cy2,caddout,candout;
  var int i,j,a,cfout,cnotfout,cnoty1, final;
  var int ng,zr,zr0,zr1,zr2;

  let outfirst = Array.new(8);
  let outsecond = Array.new(8);

  let fls = 0;
  let tr = 0;

  let x1 = Mux16.mux16(x, fls, zx);
  let y1 = Mux16.mux16(y,fls,zy);

  let cx1 = toDecimal.toDec(x1);
  let notx1 = Not16.not16(cx1);
  let cnotx1 = toDecimal.toDec(notx1);
  let x2 = Mux16.mux16(cx1,cnotx1,nx);
```

```
  let cy1 = toDecimal.toDec(y1);
  let noty1 = Not16.not16(cy1);
  let cnoty1 = toDecimal.toDec(noty1);
  let y2 = Mux16.mux16(cy1,cnoty1,ny);

  let cx2 = toDecimal.toDec(x2);
  let cy2 = toDecimal.toDec(y2);
```

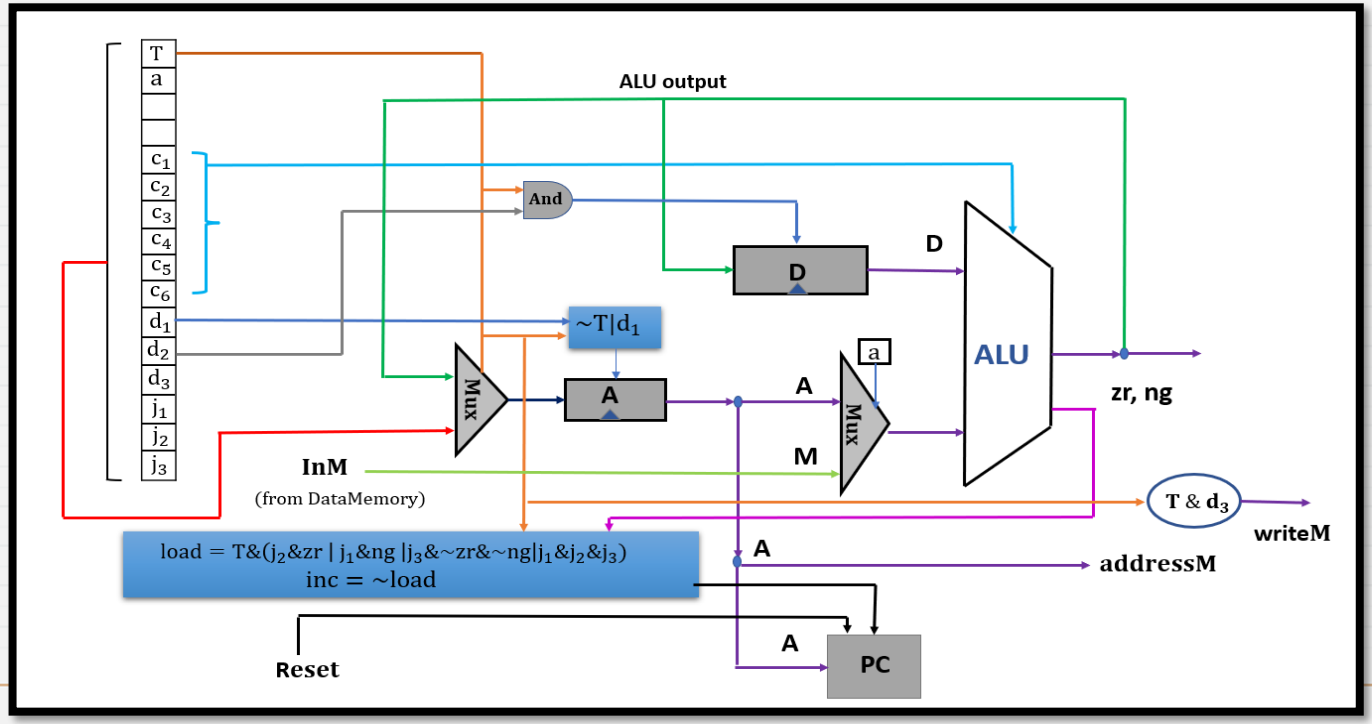
```
82      let addout = Add16.add16(cx2,cy2);
83      let andout = And16.and16(cx2,cy2);
84
85      let caddout = toDecimal.toDec(addout);
86      let candout = toDecimal.toDec(andout);
87
88      let fout = Mux16.mux16(candout,caddout,f);
89
90      let cfout = toDecimal.toDec(fout);
91      let notfout = Not16.not16(cfout);
92      let cnotfout = toDecimal.toDec(notfout);
93
94      let out = Mux16.mux16(cfout,cnotfout,no);
95
96      let i = 8;
97      while(i < 16){
98        let outfirst[i-8] = out[i];
99        let i = i + 1;
100     }
101
102     let i = 0;
103     while(i < 8){
104       let outsecond[i] = out[i];
105       let i = i + 1;
106     }
107
108     let coutfirst = toDecimal8.toDec8(outfirst);
109     let coutsecond = toDecimal8.toDec8(outsecond);
```

```
111     let zr0 = Or8Way.or8way(coutfirst);
112     let zr1 = Or8Way.or8way(coutsecond);
113     let zr2 = Or.or(zr1,zr0);
114     let zr = Not.not(zr2);
115
116     return zr;
117   }
```

```
119    function int ng(int zx, int nx, int zy, int ny, int f, int no, int x, int y){
120      var Array out, x1, y1, x2, y2,notx1, notx2, noty1,noty2;
121      var Array addout, andout,fout, notfout;
122      var Arary outfirst, outsecond;
123      var int fls, tr, cx1,coutfirst,coutsecond,cnotx1,cy1,cx2,cy2,caddout,candout;
124      var int i,j,a,cfout,cnotfout,cnoty1, final;
125      var int ng,zr,zr0,zr1,zr2;
126
127      let outfirst = Array.new(8);
128      let outsecond = Array.new(8);
129
130      let fls = 0;
131      let tr = 0;
132
133      let x1 = Mux16.mux16(x, fls, zx);
134      let y1 = Mux16.mux16(y,fls,zy);
135
136      let cx1 = toDecimal.toDec(x1);
137      let notx1 = Not16.not16(cx1);
138      let cnotx1 = toDecimal.toDec(notx1);
139      let x2 = Mux16.mux16(cx1,cnotx1,nx);
```

```
141      let cy1 = toDecimal.toDec(y1);
142      let noty1 = Not16.not16(cy1);
143      let cnoty1 = toDecimal.toDec(noty1);
144      let y2 = Mux16.mux16(cy1,cnoty1,ny);
145
146      let cx2 = toDecimal.toDec(x2);
147      let cy2 = toDecimal.toDec(y2);
```

CPU BLOCK DIAGRAM



The C-instruction specification

Symbolic syntax: *dest* = *comp* ; *jump*

Binary syntax: 1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3

comp		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D|A	D|M	0	1	0	1	0	1
a==0	a==1						

dest	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

jump	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Examples:

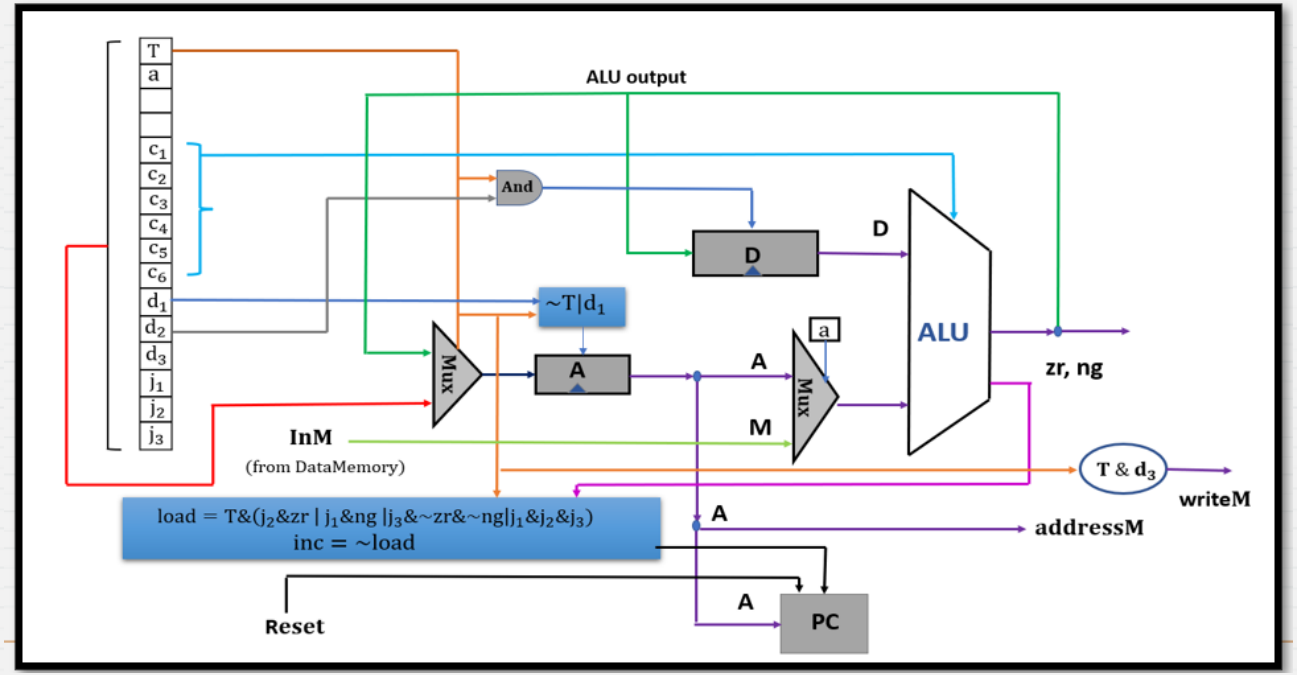
Symbolic: MD=D+1

Binary: 1110011111011000

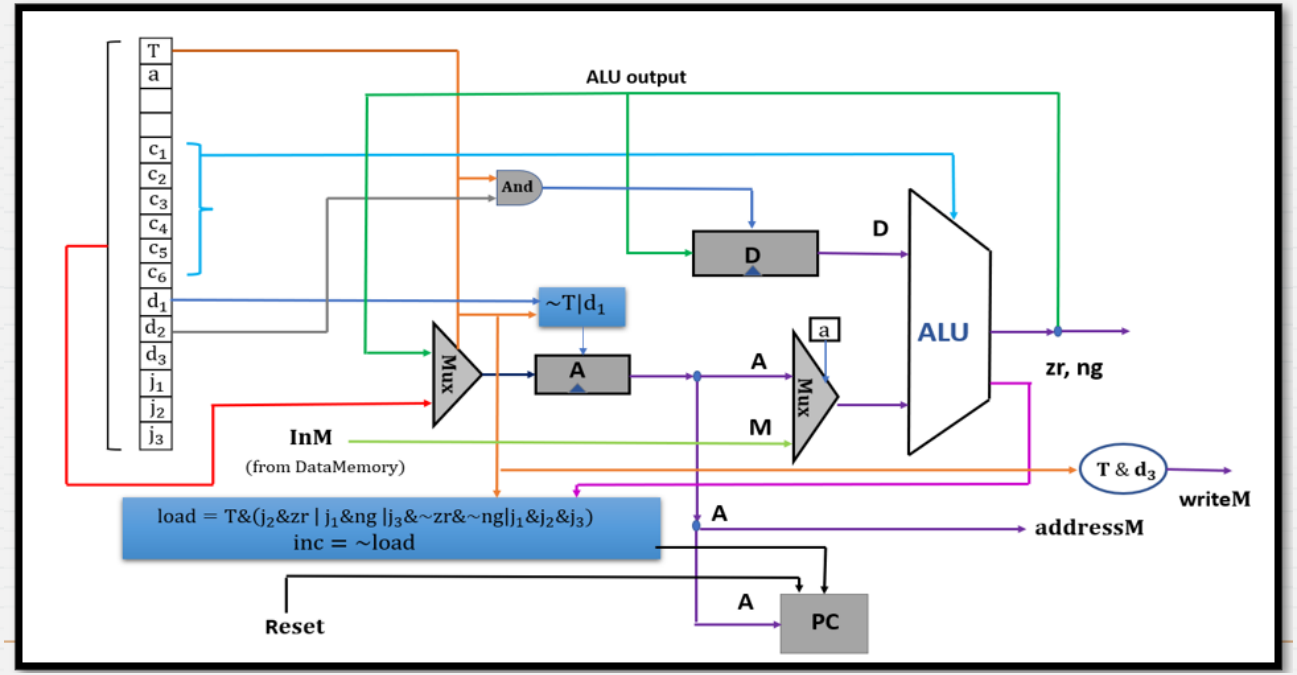
CPU

```
// Implementation of Hack CPU in jack.

class CPU {
  function int o1(int inM, int x,int reset){
    var Array outtM,prevout,i,A,AM,D,outM,instruction;
    var int ni,intoA,AorM,intoD,couttM,cins,cprevout,ci,cA,cinM,cAM,cD,zr,ng;

    let outtM = Array.new(16);
    let prevout = Array.new(16);
    let instruction = toBinaryall.toBinall(x);

    let ni = Not.not(instruction[0]);

    let couttM = toDecimal.toDec(outtM);
    let i = Mux16.mux16(couttM,x,ni);

    let intoA = Or.or(instruction[10],ni);
    let cprevout = toDecimal.toDec(prevout);
    let ci = toDecimal.toDec(i);
    let A = Register16.register16(cprevout,ci,intoA);

    let AorM = And.and(instruction[15],instruction[12]);
    let cA = toDecimal.toDec(A);
```

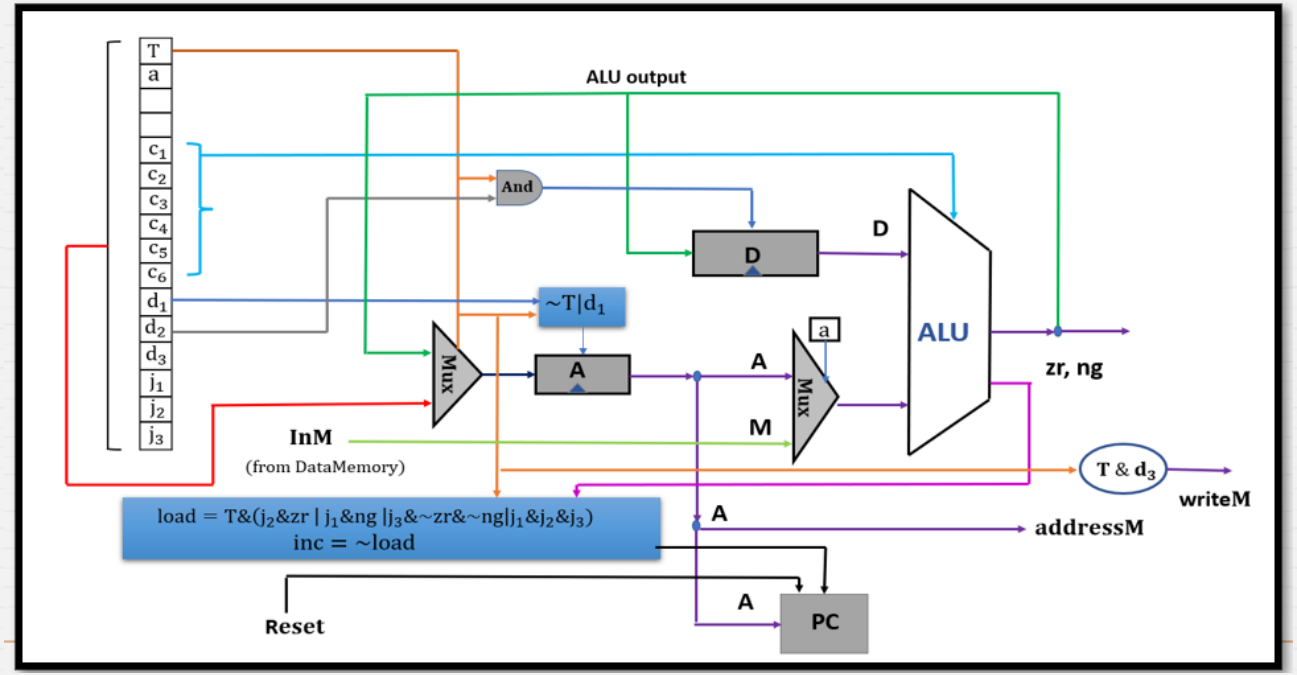
```
    let cA = toDecimal.toDec(A);

    let AM = Mux16.mux16(cA,inM,AorM);

    let intoD = And.and(instruction[0],instruction[12]);

    let D = Register16.register16(cprevout,couttM,intoD);

    let cAM = toDecimal.toDec(AM);
    let cD = toDecimal.toDec(D);

    let outM = ALU.alu(instruction[4],instruction[5],instruction[6],instruction[7],instruction[8],instruction[9],cD,cAM);

    return outM;

   }
```


T
a
c1
c2
c3
c4
c5
c6
d1
d2
d3
j1
j2
j3
ALU output
And
D
D
~T|d1
A
Mux
A
a
Mux
ALU
zr, ng
M
InM
(from DataMemory)
T & d3
writeM
load = T&(j2&zr | j1&ng |j3&~zr&~ng|j1&j2&j3)
inc = ~load
A
addressM
A
Reset
PC

```
40      function int o2(int inM,int x,int reset){
41        var int writeM;
42        var Array instruction;
43
44        let instruction = toBinaryall.toBinall(x);
45
46        let writeM = And.and(instruction[0],instruction[12]);
47
48        return writeM;
49      }
50
```


T
a
c1
c2
c3
c4
c5
c6
d1
d2
d3
j1
j2
j3
ALU output
And
D
D
~T|d1
A
Mux
A
a
Mux
ALU
zr, ng
M
InM
(from DataMemory)
T & d3
writeM
load = T&(j2&zr | j1&ng |j3&~zr&~ng|j1&j2&j3)
inc = ~load
A
addressM
A
Reset
PC

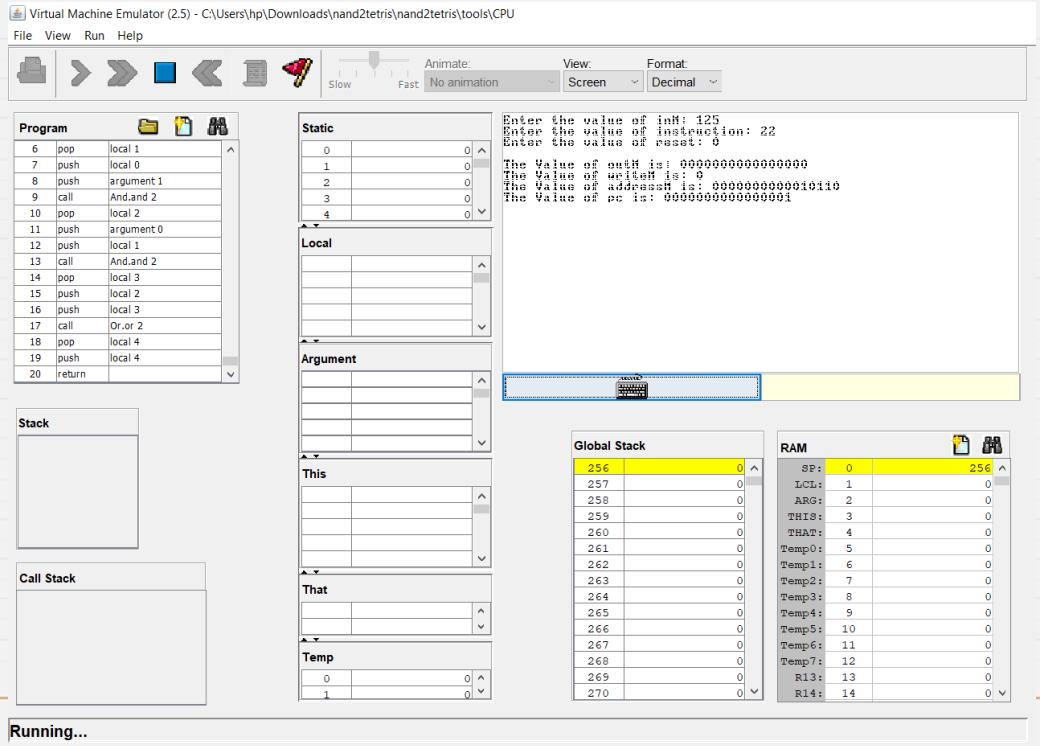
```
51      function Array addressm(int inM,int x,int reset){
52        var Array outtM,prevout,i,A,AM,D,outM,instruction;
53        var int ni,intoA,AorM,intoD,couttM,cins,cprevout,ci,cA,cinM,cAM,cD,zr,ng;
54
55        let outtM = Array.new(16);
56        let prevout = Array.new(16);
57        let instruction = toBinaryall.toBinall(x);
58
59        let ni = Not.not(instruction[0]);
60
61        let couttM = toDecimal.toDec(outtM);
62        let i = Mux16.mux16(couttM,x,ni);
63
64        let intoA = Or.or(instruction[10],ni);
65        let cprevout = toDecimal.toDec(prevout);
66        let ci = toDecimal.toDec(i);
67        let A = Register16.register16(cprevout,ci,intoA);
68
69        return A;
70      }
```


T
a
c1
c2
c3
c4
c5
c6
d1
d2
d3
j1
j2
j3
ALU output
And
D
D
~T|d1
A
Mux
A
a
Mux
ALU
zr, ng
M
InM
(from DataMemory)
T & d3
writeM
load = T&(j2&zr | j1&ng |j3&~zr&~ng|j1&j2&j3)
inc = ~load
A
addressM
A
Reset
PC

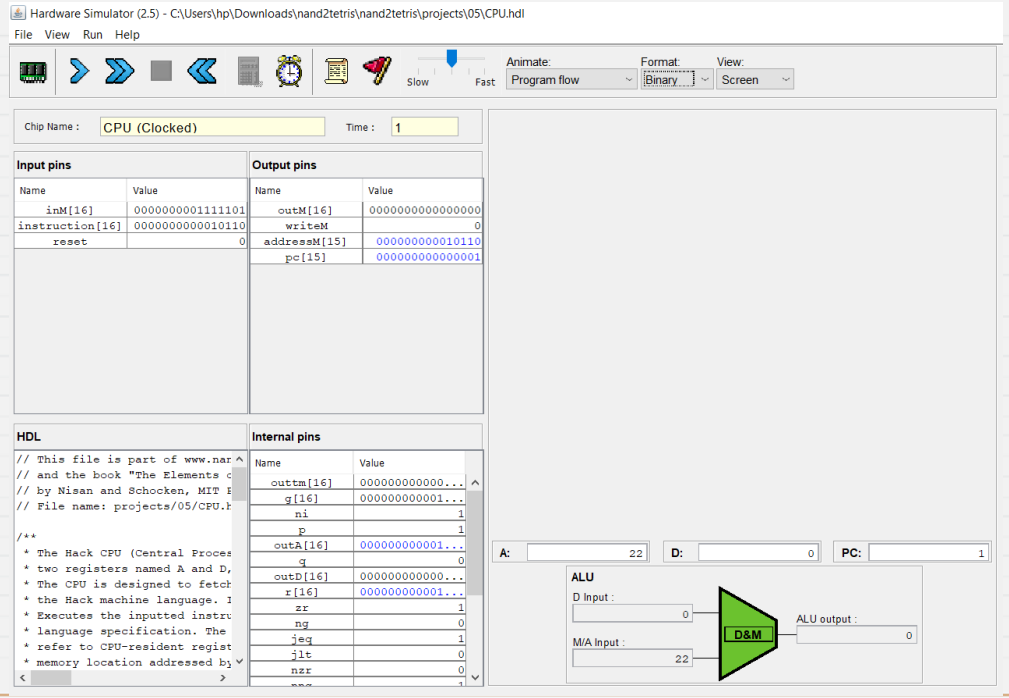
```
72      function Array o3(int inM,int x,int reset){
73        var Array outtM,prevout,i,A,AM,D,outM,instruction,out;
74        var int ni,intoA,AorM,intoD,couttM,cins,cprevout,ci,cA,cinM,cAM,cD,zr,ng;
75        var int writeM,pos,nzr,jgt,posnzr,ld1,jeq,ld2,jlt,ld3,ldt,ld;
76
77        let outtM = Array.new(16);
78        let prevout = Array.new(16);
79        let instruction = toBinaryall.toBinall(x);
80
81        let ni = Not.not(instruction[0]);
82
83        let couttM = toDecimal.toDec(outtM);
84        let i = Mux16.mux16(couttM,x,ni);
85
86        let intoA = Or.or(instruction[10],ni);
87        let cprevout = toDecimal.toDec(prevout);
88        let ci = toDecimal.toDec(i);
89        let A = Register16.register16(cprevout,ci,intoA);
90
91        let AorM = And.and(instruction[15],instruction[12]);
92        let cA = toDecimal.toDec(A);
93
94        let AM = Mux16.mux16(cA,inM,AorM);
95
96        let intoD = And.and(instruction[0],instruction[12]);
```

```
98        let D = Register16.register16(cprevout,couttM,intoD);
99
100       let cAM = toDecimal.toDec(AM);
101       let cD = toDecimal.toDec(D);
102
103       let outM = ALU.alu(instruction[4],instruction[5],instruction[6],instruction[7],instruction[8],instruction[9],cD,cAM);
104       let zr = ALU.zr(instruction[4],instruction[5],instruction[6],instruction[7],instruction[8],instruction[9],cD,cAM);
105       let ng = ALU.alu(instruction[4],instruction[5],instruction[6],instruction[7],instruction[8],instruction[9],cD,cAM);
106
107       let writeM = And.and(instruction[0],instruction[12]);
108
109       let pos  = Not.not(ng);
110       let nzr = Not.not(zr);
111       let jgt = And.and(instruction[0], instruction[15]);
112       let posnzr = And.and(pos, jgt);
113       let ld1 = And.and(jgt, posnzr);
114
115       let jeq = And.and(instruction[0], instruction[14]);
116       let ld2 = And.and(jgt, zr);
117
118       let jlt = And.and(instruction[0], instruction[13]);
119       let ld3 = And.and(jlt, ng);
120
121       let ldt = Or.or(ld1, ld2);
122       let ld = Or.or(ld3, ldt);
123
124       let out = PC.pc(cA,ld,1,reset);
```

Implementation Of CPU IN VM Emulator



Comparison of Output by using CPU implemented in HDL



Sudoku Solver in Jack

MultiArray.jack

```
// Implememntation of Multi-Dimensional Array in Jack

class MultiArray {
  field Array a; // first row
  field Array b; // second row
  field Array c; // third row
  field Array d; // forth row
  field Array e; // five row
  field Array f; // sixth row
  field Array g; // seventh row
  field Array h; // eight row
  field Array i; // ninth row

  field int length;
```

```
16    constructor MultiArray new(int numberOfColumns){
17      let length = numberOfColumns;
18      let a = Array.new(length);
19      let b = Array.new(length);
20      let c = Array.new(length);
21      let d = Array.new(length);
22      let e = Array.new(length);
23      let f = Array.new(length);
24      let g = Array.new(length);
25      let h = Array.new(length);
26      let i = Array.new(length);
27
28      return this;
29    }
30
```

```
31    method void add(int row, int column, int data){
32      if(column > length){
33        do Output.printString("Array Out Of Bounds Exception");
34        do Output.println();
35        do Output.printString("Index accessed: ");
36        do Output.printInt(column);
37        do Output.printString(", Array Length: ");
38        do Output.printInt(length);
39        return;
40      }
41      if(row = 0){
42        let a[column] = data;
43      }
44      if(row = 1){
45        let b[column] = data;
46      }
47
48      if(row = 2){
49        let c[column] = data;
50      }
51
52      if(row = 3){
53        let d[column] = data;
```

```
56        if(row = 4){
57          let e[column] = data;
58        }
59
60        if(row = 5){
61          let f[column] = data;
62        }
63
64        if(row = 6){
65          let g[column] = data;
66        }
67
68        if(row = 7){
69          let h[column] = data;
70        }
71
72        if(row = 8){
73          let i[column] = data;
74        }
```

```
76        if(~(MultiArray.or(row))) {
77          do Output.printString("Enter valid row number");
78        }
79        return;
80      }
81
82      method int get(int row, int column){
83        if(column > length){
84          do Output.printString("Array Out Of Bounds Exception");
85          do Output.println();
86          do Output.printString("Index accessed: ");
87          do Output.printInt(column);
88          do Output.printString(", Array Length: ");
89          do Output.printInt(length);
90          return -32767;
91        }
92        if(row = 0){
93          return a[column];
94        }
95        if(row = 1){
96          return b[column];
97        }
```

```
98
99        if(row = 2){
100         return c[column];
101       }
102
103       if(row = 3){
104         return d[column];
105       }
106
107       if(row = 4){
108         return e[column];
109       }
110
111       if(row = 5){
112         return f[column];
113       }
114
115       if(row = 6){
116         return g[column];
117       }
118
119       if(row = 7){
120         return h[column];
121       }
```

```
123       if(row = 8){
124         return i[column];
125       }
126
127        if(~(MultiArray.or(row))) {
128         do Output.printString("Enter valid row number");
129       }
130       return -32767;
131     }
132
133     method int lenght(){
134       return length;
135     }
136
137     function boolean or(int row){
138       if(row = 0){
139         return true;
140       }
141       if(row = 1){
142         return true;
143       }
144
145       if(row = 2){
146         return true;
```

```
146         return true;
147       }
148       if(row = 3){
149         return true;
150       }
151       if(row = 4){
152         return true;
153       }
154       if(row = 5){
155         return true;
156       }
157       if(row = 6){
158         return true;
159       }
160       if(row = 7){
161         return true;
162       }
163       if(row = 8){
164         return true;
165       }
166       return false;
167     }
168   }
169
```

```
// Implememntation Suduko Solver in Jack

class Main {

  function int remainder(int a, int b){
    var int minus;
    var int quiotent;
    var int rem;
    var int sqt;

    let minus = a - b;
    let quiotent = 0;

    while(minus > -1){
      let minus = minus - b;
      let quiotent = quiotent + 1;
    }
    let rem = minus+b;

    return rem;
  }

```

```
25    function boolean isNumberinRow(MultiArray a, int n, int r){
26      var int i;
27      let i = 0;
28
29      while(i < 9){
30        if(a.get(r,i) = n){
31          return true;
32        }
33        let i = i + 1;
34      }
35      return false;
36    }
37
```

```
38     /* checks whether the given number is in the given coloumn */
39
40    function boolean isNumberinColumn(MultiArray a, int n, int c){
41      var int i;
42      let i = 0;
43
44      while(i < 9){
45        if(a.get(i,c) = n){
46          return true;
47        }
48        let i = i + 1;
49      }
50      return false;
51    }
52
```

```
53     /* checks whether the given number is in the given 3x3 box  */
54
55    function boolean isinBox(MultiArray a, int n, int r, int c){
56      var int boxRowSt, boxColSt;
57      var int i,j;
58
59      let boxRowSt = (r - (Main.remainder(r,3)));
60      let boxColSt = (c - (Main.remainder(c,3)));
61
62      let i = boxRowSt;
63
64      while(i < (boxRowSt + 3)){
65            let j = boxColSt;
66        while(j < (boxColSt + 3)){
67          if(a.get(i,j) = n){
68            return true;
69          }
70          let j = j + 1;
71        }
72        let i = i + 1;
73      }
74      return false;
75    }
```

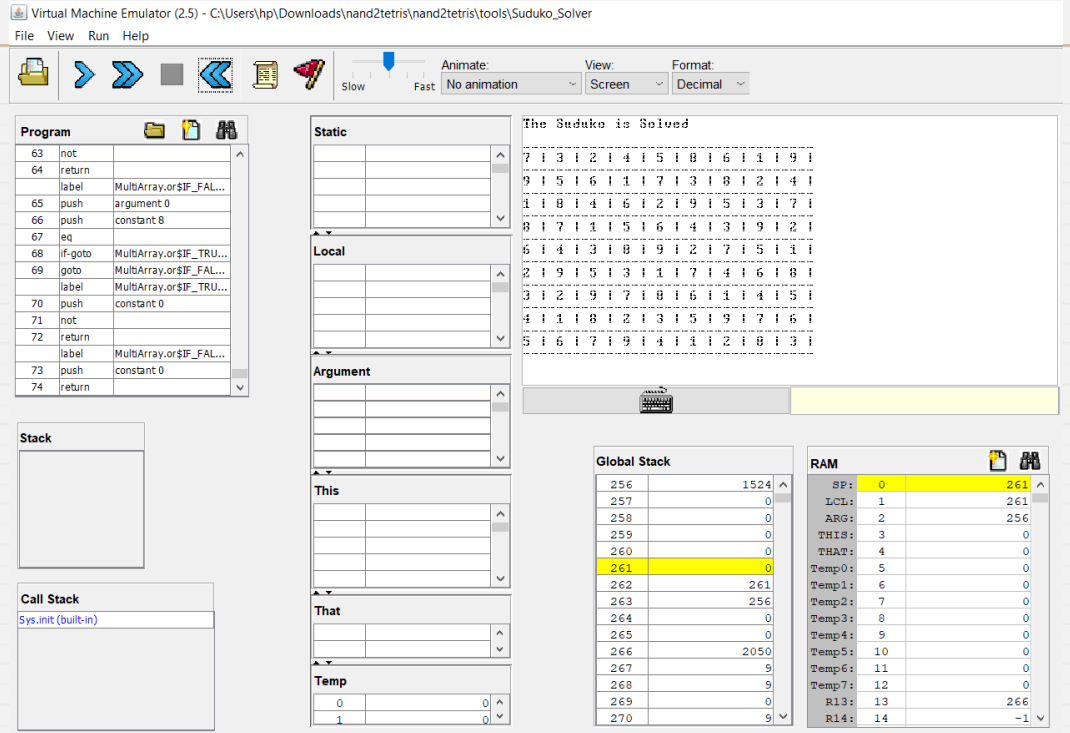
```
function boolean isvalidPlacement(MultiArray a, int n, int r, int c){
  return (~((Main.isNumberinRow(a,n,r)) | (Main.isNumberinColumn(a,n,c)) | (Main.isinBox(a,n,r,c))));
}

 function boolean solveSuduko(MultiArray a){
   var int row;
   var int col;
   var int numtotry;

   let row = 0;

   while(row < 9){
    let col = 0;
     while(col < 9){
       if((a.get(row,col)) = 0){
         let numtotry = 1;
         while(numtotry < 10){
           if(Main.isvalidPlacement(a,numtotry,row,col)){
             do a.add(row,col,numtotry);

           if(Main.solveSuduko(a)){
             return true;
           }
```

```
102                 else{
103                   do a.add(row,col,0);
104                 }
105                 }
106                 let numtotry = numtotry + 1;
107               }
108               return false;
109             }
110             let col = col + 1;
111           }
112           let row = row + 1;
113         }
114         return true;
115       }
116
117       function void main(){
118        var MultiArray b;
119        var int i, j;
120        let b = MultiArray.new(9);
```

```
125       do b.add(0,0,7);
126       do b.add(0,2,2);
127       do b.add(0,4,5);
128       do b.add(0,6,6);
129
130       do b.add(1,5,3);
131
132       do b.add(2,0,1);
133
134       do b.add(2,5,9);
135       do b.add(2,6,5);
136
137       do b.add(3,0,8);
138       do b.add(3,7,9);
139       do b.add(4,1,4);
140
141       do b.add(4,2,3);
142       do b.add(4,6,7);
143
144       do b.add(4,7,5);
145       do b.add(5,1,9);
146       do b.add(5,8,8);
147       do b.add(6,2,9);
148       do b.add(6,3,7);
```

```
158       if(Main.solveSuduko(b)){
159         do Output.printString("The Suduko is Solved");
160       }
161
162       else{
163         do Output.printString("The Suduko is Unsolved");
164       }
165
166       do Output.println();
167       do Output.println();
168
169       do Output.printString("---------------------------------");
170       do Output.println();
```

```
    do Output.println();

    do Output.printString("---------------------------------");
    do Output.println();

    let i = 0;
    while (i < 9) {
      let j = 0;
      while (j < 9) {
        do Output.printInt(b.get(i, j));
        do Output.printString(" | ");
        let j = j + 1;
      }
      do Output.println();
      do Output.printString("---------------------------------");
      do Output.println();
      let i = i + 1;
    }

  return;
}
```


Virtual Machine Emulator (2.5) - C:\Users\hp\Downloads\nand2tetris\nand2tetris\tools\Suduko_Solver
File View Run Help
Slow
Fast
Animate:
No animation
View:
Screen
Format:
Decimal
Program
63 not
64 return
label MultiArray.or$IF_FAL...
65 push argument 0
66 push constant 8
67 eq
68 if-goto MultiArray.or$IF_TRU...
69 goto MultiArray.or$IF_FAL...
label MultiArray.or$IF_TRU...
70 push constant 0
71 not
72 return
label MultiArray.or$IF_FAL...
73 push constant 0
74 return
Static
Local
Argument
This
That
Temp
0 0
1 0
The Suduko is Solved
7 | 3 | 2 | 4 | 5 | 8 | 6 | 1 | 9 |
9 | 5 | 6 | 1 | 7 | 3 | 8 | 2 | 4 |
1 | 8 | 4 | 6 | 2 | 9 | 5 | 3 | 7 |
8 | 7 | 1 | 5 | 6 | 4 | 3 | 9 | 2 |
6 | 4 | 3 | 8 | 9 | 2 | 7 | 5 | 1 |
2 | 9 | 5 | 3 | 1 | 7 | 4 | 6 | 8 |
3 | 2 | 9 | 7 | 8 | 6 | 1 | 4 | 5 |
4 | 1 | 8 | 2 | 3 | 5 | 9 | 7 | 6 |
5 | 6 | 7 | 9 | 4 | 1 | 2 | 8 | 3 |
Stack
Call Stack
Sys.init (built-in)
Global Stack
256 1524
257 0
258 0
259 0
260 0
261 0
262 261
263 256
264 0
265 0
266 2050
267 9
268 9
269 0
270 9
RAM
SP: 0 261
LCL: 1 261
ARG: 2 256
THIS: 3 0
THAT: 4 0
Temp0: 5 0
Temp1: 6 0
Temp2: 7 0
Temp3: 8 0
Temp4: 9 0
Temp5: 10 0
Temp6: 11 0
Temp7: 12 0
R13: 13 266
R14: 14 -1

THANK YOU