ELEMENTS OF COMPUTING

PRESENTED BY:

1.Shreya Arun-CB.SC.U4AIE23253

2.Varshitha Thilak Kumar-CB.SC.U4AIE23258

3.Siri Sanjana Singareddy-CB.SC.U4AIE23249

CEN, Amrita School of Artificial Intelligence,

Amrita Vishwa Vidyapeetham, Amritanagar P.O, Coimbatore-641112

OVERVIEW

INTRODUCTION

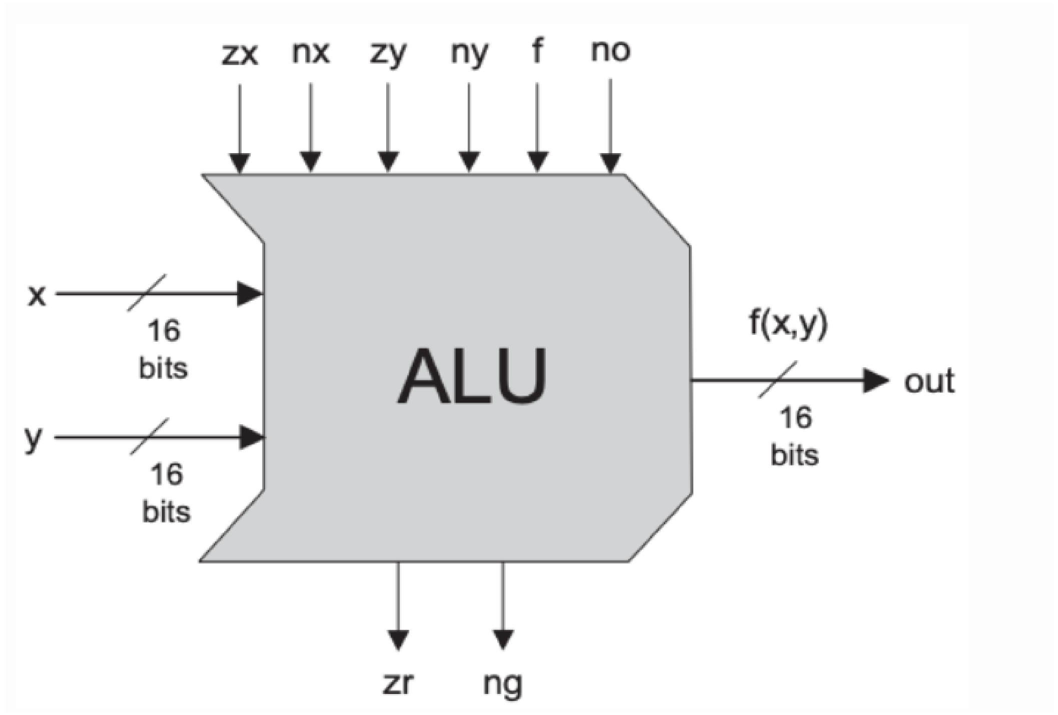
- The Hack platform is a **16-bit von Neumann machine**, designed to execute programs written in the Hack machine language.

To implement the Hack CPU,

- ALU chip capable of computing arithmetic/logical functions
- Set of registers
- Program counter, some additional gates designed to help decode, execute, and fetch instructions

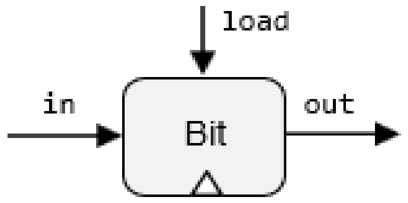
ALU

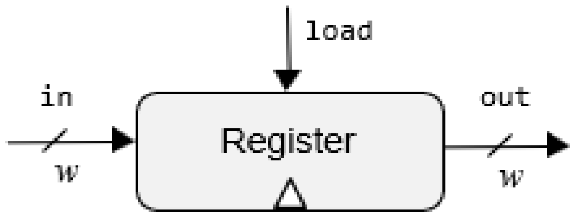
- We need a 16-bit Adder chip and a couple of logic gates including 16-bit Multiplexor, 16-bit NOT, 16-bit AND, 8-way OR, OR, and NOT



REGISTERS

- 1. Data Register (D) : This register is used for storing data values.
- 2. Address Register (A): The A register serves three different purposes, depending on the context in which it is used: storing a data value, pointing at an address in the instruction memory, or pointing at an address in the data memory.
- 3. Program Counter (PC): This register holds the address of the next instruction to be executed.





MEMORY

- The architecture which include a 16-bit address space .It uses   separate memory space for data (RAM) and instructions (ROM)

PC COUNTER

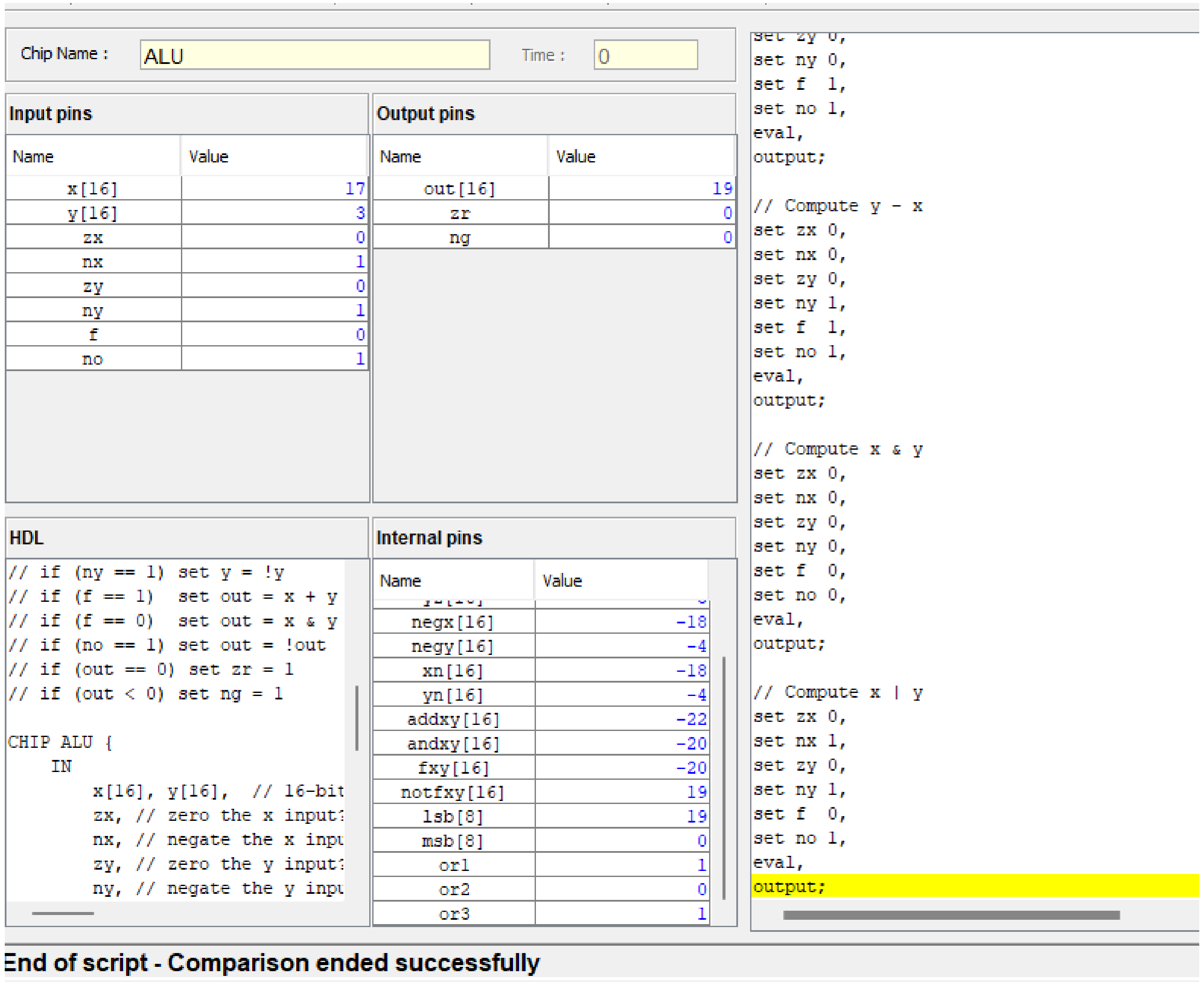
- Program Counter holds the memory address of the next instruction that would be executed.

ALU

HDL CODE

```
CHIP ALU {
    IN
        x[16], y[16],  // 16-bit inputs
        zx, // zero the x input?
        nx, // negate the x input?
        zy, // zero the y input?
        ny, // negate the y input?
        f,  // compute out = x + y (if 1) or x & y (if 0)
        no; // negate the out output?

    OUT
        out[16], // 16-bit output
        zr, // 1 if (out == 0), 0 otherwise
        ng; // 1 if (out < 0),  0 otherwise

PARTS:
Mux16(a=x,b=false,sel=zx,out=xz);
Mux16(a=y,b=false,sel=zy,out=yz);
Not16(in=xz,out=negx);
Not16(in=yz,out=negy);
Mux16(a=xz,b=negx,sel=nx,out=xn);
Mux16(a=yz,b=negy,sel=ny,out=yn);
Add16(a=xn,b=yn,out=addxy);
And16(a=xn,b=yn,out=andxy);
Mux16(a=andxy,b=addxy,sel=f,out=fxy);
Not16(in=fxy,out=notfxy);
Mux16(a=fxy,b=notfxy,sel=no,out=out,out[15]=ng,out[0..7]=lsb,out[8..15]=msb);
Or8Way(in=lsb,out=or1);
Or8Way(in=msb,out=or2);
Or(a=or1,b=or2,out=or3);
Not(in=or3,out=zr);
}
```

IMPLEMENTATION



Chip Name : ALU   Time : 0

Input pins

Name	Value
x[16]	17
y[16]	3
zx	0
nx	1
zy	0
ny	1
f	0
no	1

Output pins

Name	Value
out[16]	19
zr	0
ng	0

HDL

```
// if (ny == 1) set y = !y
// if (f == 1)  set out = x + y
// if (f == 0)  set out = x & y
// if (no == 1) set out = !out
// if (out == 0) set zr = 1
// if (out < 0) set ng = 1

CHIP ALU {
    IN
        x[16], y[16],  // 16-bit
        zx, // zero the x input
        nx, // negate the x inpu
        zy, // zero the y input
        ny, // negate the y inpu
```

Internal pins

Name	Value
negx[16]	-18
negy[16]	-4
xn[16]	-18
yn[16]	-4
addxy[16]	-22
andxy[16]	-20
fxy[16]	-20
notfxy[16]	19
lsb[8]	19
msb[8]	0
or1	1
or2	0
or3	1

```
set ny 0,
set f  1,
set no 1,
eval,
output;

// Compute y - x
set zx 0,
set nx 0,
set zy 0,
set ny 1,
set f  1,
set no 1,
eval,
output;

// Compute x & y
set zx 0,
set nx 0,
set zy 0,
set ny 0,
set f  0,
set no 0,
eval,
output;

// Compute x | y
set zx 0,
set nx 1,
set zy 0,
set ny 1,
set f  0,
set no 1,
eval,
output;
```

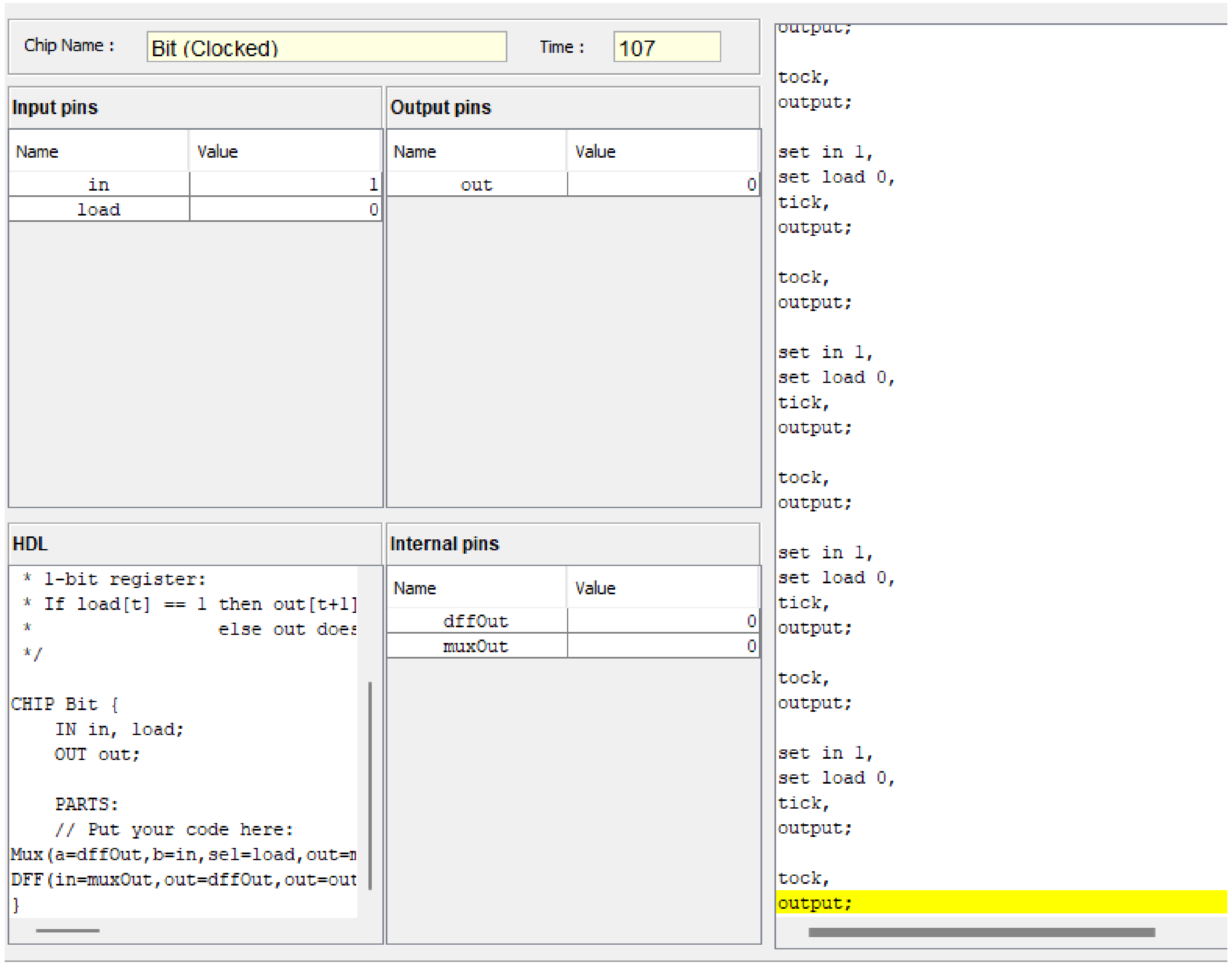
End of script - Comparison ended successfully

BIT

IMPLEMENTATION

HDL CODE

```
CHIP Bit {
    IN in, load;
    OUT out;

    PARTS:
    // Put your code here:
Mux(a=gayout,b=in,sel=load,out=a);
DFF(in=a,out=out,out=gayout);

}
```



Chip Name : Bit (Clocked)   Time : 107

Input pins

Name	Value
in	1
load	0

Output pins

Name	Value
out	0

HDL

```
 * 1-bit register:
 * If load[t] == 1 then out[t+1]
 *                 else out does
 */

CHIP Bit {
    IN in, load;
    OUT out;

    PARTS:
    // Put your code here:
Mux(a=dffOut,b=in,sel=load,out=m
DFF(in=muxOut,out=dffOut,out=out
}
```

Internal pins

Name	Value
dffOut	0
muxOut	0

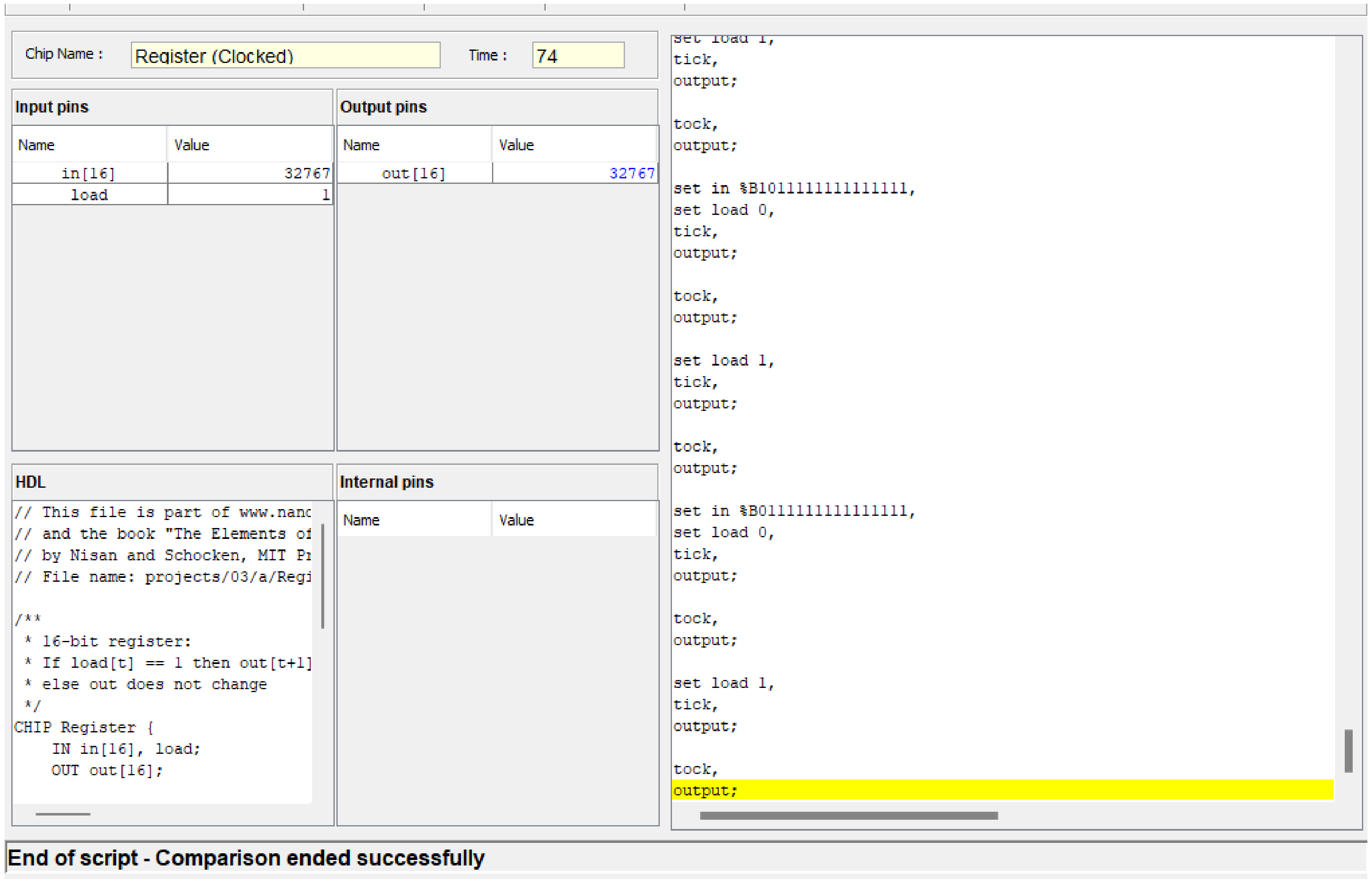
```
output;

tock,
output;

set in 1,
set load 0,
tick,
output;

tock,
output;

set in 1,
set load 0,
tick,
output;

tock,
output;

set in 1,
set load 0,
tick,
output;

tock,
output;

set in 1,
set load 0,
tick,
output;

tock,
output;
```

RESGISTERS

HDL CODE

```
CHIP Register {
    IN in[16], load;
    OUT out[16];

    PARTS:
    // Put your code here:
Bit(in=in[0],load=load,out=out[0]);
Bit(in=in[1],load=load,out=out[1]);
Bit(in=in[2],load=load,out=out[2]);
Bit(in=in[3],load=load,out=out[3]);
Bit(in=in[4],load=load,out=out[4]);
Bit(in=in[5],load=load,out=out[5]);
Bit(in=in[6],load=load,out=out[6]);
Bit(in=in[7],load=load,out=out[7]);
Bit(in=in[8],load=load,out=out[8]);
Bit(in=in[9],load=load,out=out[9]);
Bit(in=in[10],load=load,out=out[10]);
Bit(in=in[11],load=load,out=out[11]);
Bit(in=in[12],load=load,out=out[12]);
Bit(in=in[13],load=load,out=out[13]);
Bit(in=in[14],load=load,out=out[14]);
Bit(in=in[15],load=load,out=out[15]);

}
```

IMPLEMENTATION



Chip Name : Register (Clocked)   Time : 74

Input pins

Name	Value
in[16]	32767
load	1

Output pins

Name	Value
out[16]	32767

HDL

```
// This file is part of www.nanc
// and the book "The Elements of
// by Nisan and Schocken, MIT Pr
// File name: projects/03/a/Regi

/**
 * 16-bit register:
 * If load[t] == 1 then out[t+1]
 * else out does not change
 */
CHIP Register {
    IN in[16], load;
    OUT out[16];
```

Internal pins

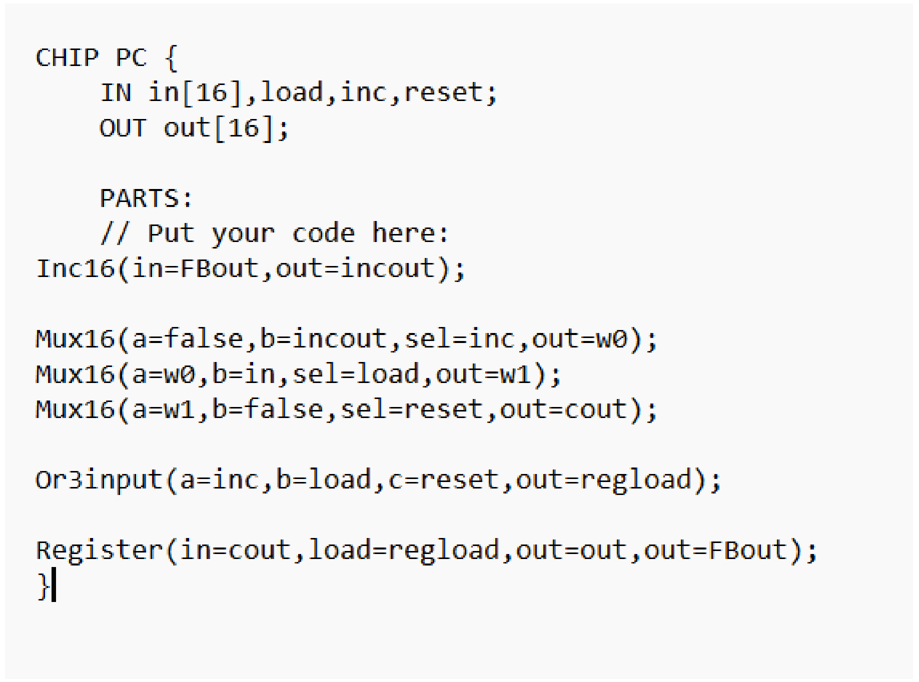
Name	Value

```
set load 1,
tick,
output;

tock,
output;

set in %B1011111111111111,
set load 0,
tick,
output;

tock,
output;

set load 1,
tick,
output;

tock,
output;

set in %B0111111111111111,
set load 0,
tick,
output;

tock,
output;

set load 1,
tick,
output;

tock,
output;
```

End of script - Comparison ended successfully

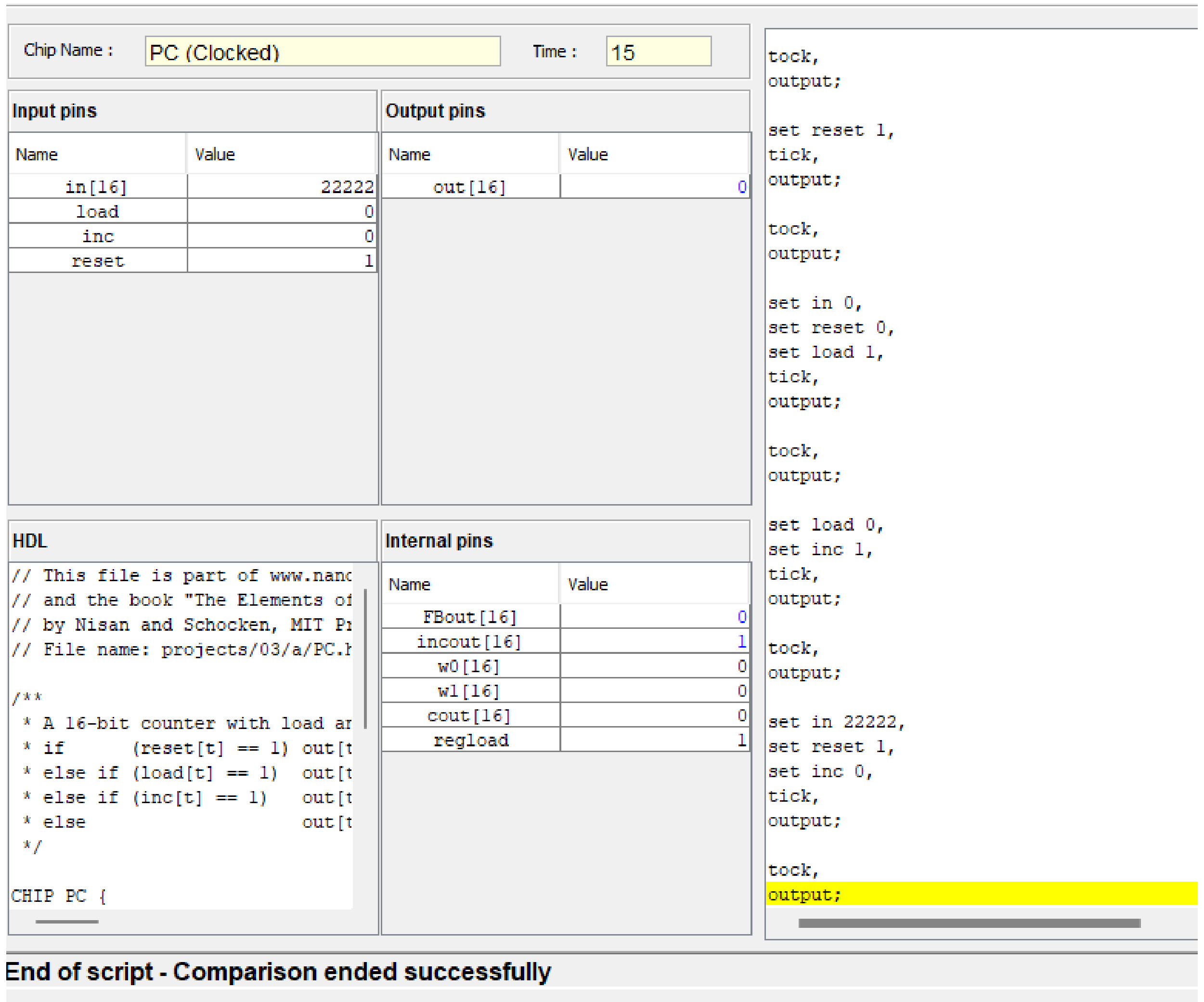
PC(PROGRAM COUNTER)

HDL CODE



```
CHIP PC {
    IN in[16],load,inc,reset;
    OUT out[16];

    PARTS:
    // Put your code here:
Inc16(in=FBout,out=incout);

Mux16(a=false,b=incout,sel=inc,out=w0);
Mux16(a=w0,b=in,sel=load,out=w1);
Mux16(a=w1,b=false,sel=reset,out=cout);

Or3input(a=inc,b=load,c=reset,out=regload);

Register(in=cout,load=regload,out=out,out=FBout);
}
```

IMPLEMENTATION



Chip Name : PC (Clocked)   Time : 15

Input pins

Name	Value
in[16]	22222
load	0
inc	0
reset	1

Output pins

Name	Value
out[16]	0

HDL

```
// This file is part of www.nanc
// and the book "The Elements of
// by Nisan and Schocken, MIT Pr
// File name: projects/03/a/PC.h

/**
 * A 16-bit counter with load ar
 * if      (reset[t] == 1) out[t
 * else if (load[t] == 1)  out[t
 * else if (inc[t] == 1)   out[t
 * else                    out[t
 */

CHIP PC {
```

Internal pins

Name	Value
FBout[16]	0
incout[16]	1
w0[16]	0
w1[16]	0
cout[16]	0
regload	1

```
tock,
output;

set reset 1,
tick,
output;

tock,
output;

set in 0,
set reset 0,
set load 1,
tick,
output;

tock,
output;

set load 0,
set inc 1,
tick,
output;

tock,
output;

set in 22222,
set reset 1,
set inc 0,
tick,
output;

tock,
output;
```

End of script - Comparison ended successfully

INBUILT CHIPS FOR SCREEN AND KEY:

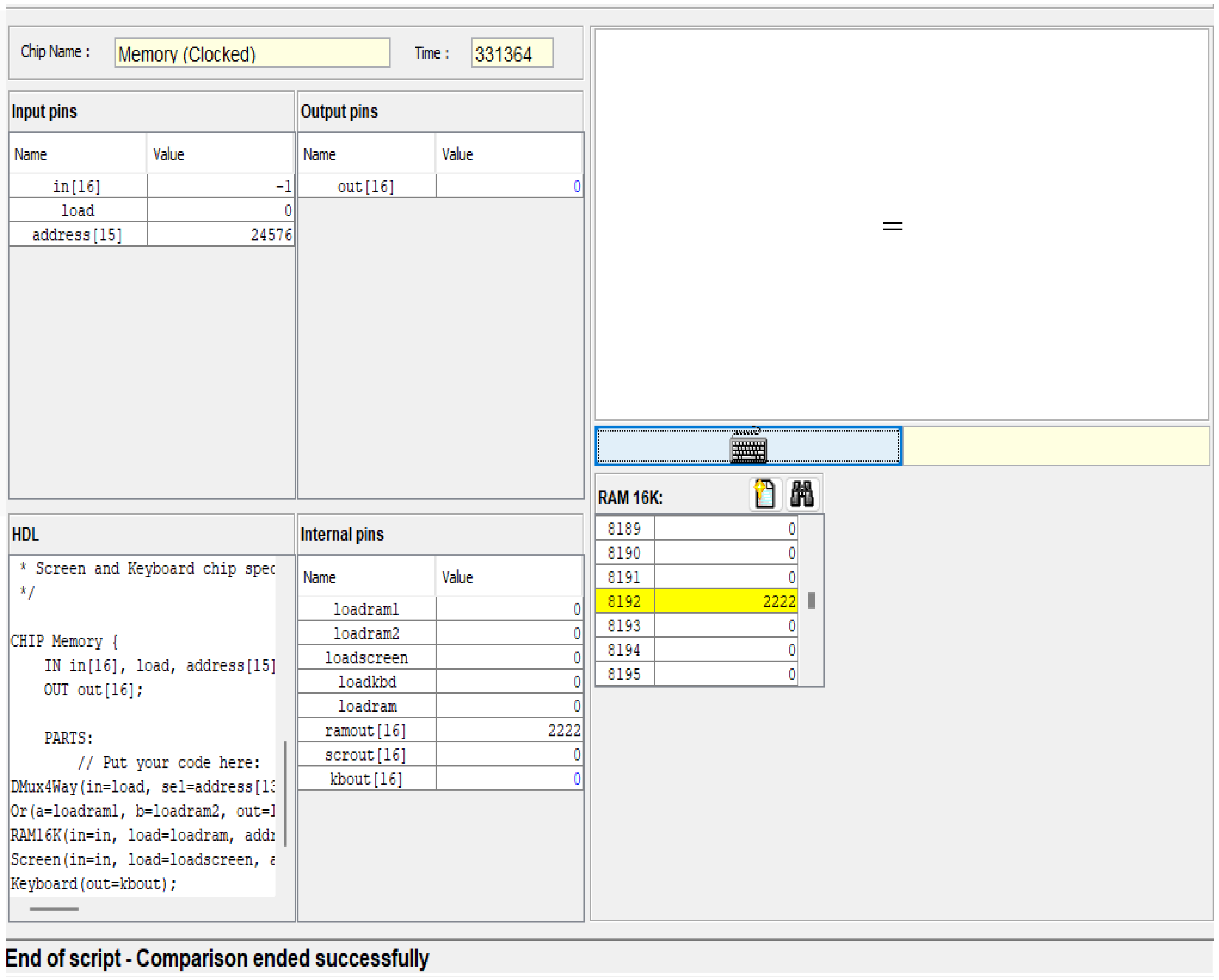
```
CHIP Keyboard {
 OUT out[16]; // The ASCII code of the pressed key,
 // or 0 if no key is currently pressed,
 // or one the special codes listed in Figure 5.5.
BUILTIN Keyboard;
}
CHIP Screen {
 IN in[16], // what to write
 load, // write-enable bit
 address[13]; // where to read/write
 OUT out[16]; // Screen value at the given address
 BUILTIN Screen;
 CLOCKED in, load;
}
```

MEMORY

HDL CODE

```
CHIP Memory {
    IN in[16], load, address[15];
    OUT out[16];

    PARTS:
        // Put your code here:
DMux4Way(in=load, sel=address[13..14], a=loadram1, b=loadram2, c=loadscreen, d=loadkbd);
Or(a=loadram1, b=loadram2, out=loadram);
RAM16K(in=in, load=loadram, address=address[0..13], out=ramout);
Screen(in=in, load=loadscreen, address=address[0..12], out=scrout);
Keyboard(out=kbout);
Mux4Way16(a=ramout, b=ramout, c=scrout, d=kbout, sel=address[13..14], out=out);

}
```

IMPLEMENTATION



Chip Name : Memory (Clocked)   Time : 331364

Input pins

Name	Value
in[16]	-1
load	0
address[15]	24576

Output pins

Name	Value
out[16]	0

HDL

```
 * Screen and Keyboard chip spec
 */

CHIP Memory {
    IN in[16], load, address[15]
    OUT out[16];

    PARTS:
        // Put your code here:
DMux4Way(in=load, sel=address[13
Or(a=loadram1, b=loadram2, out=l
RAM16K(in=in, load=loadram, addr
Screen(in=in, load=loadscreen, a
Keyboard(out=kbout);
```

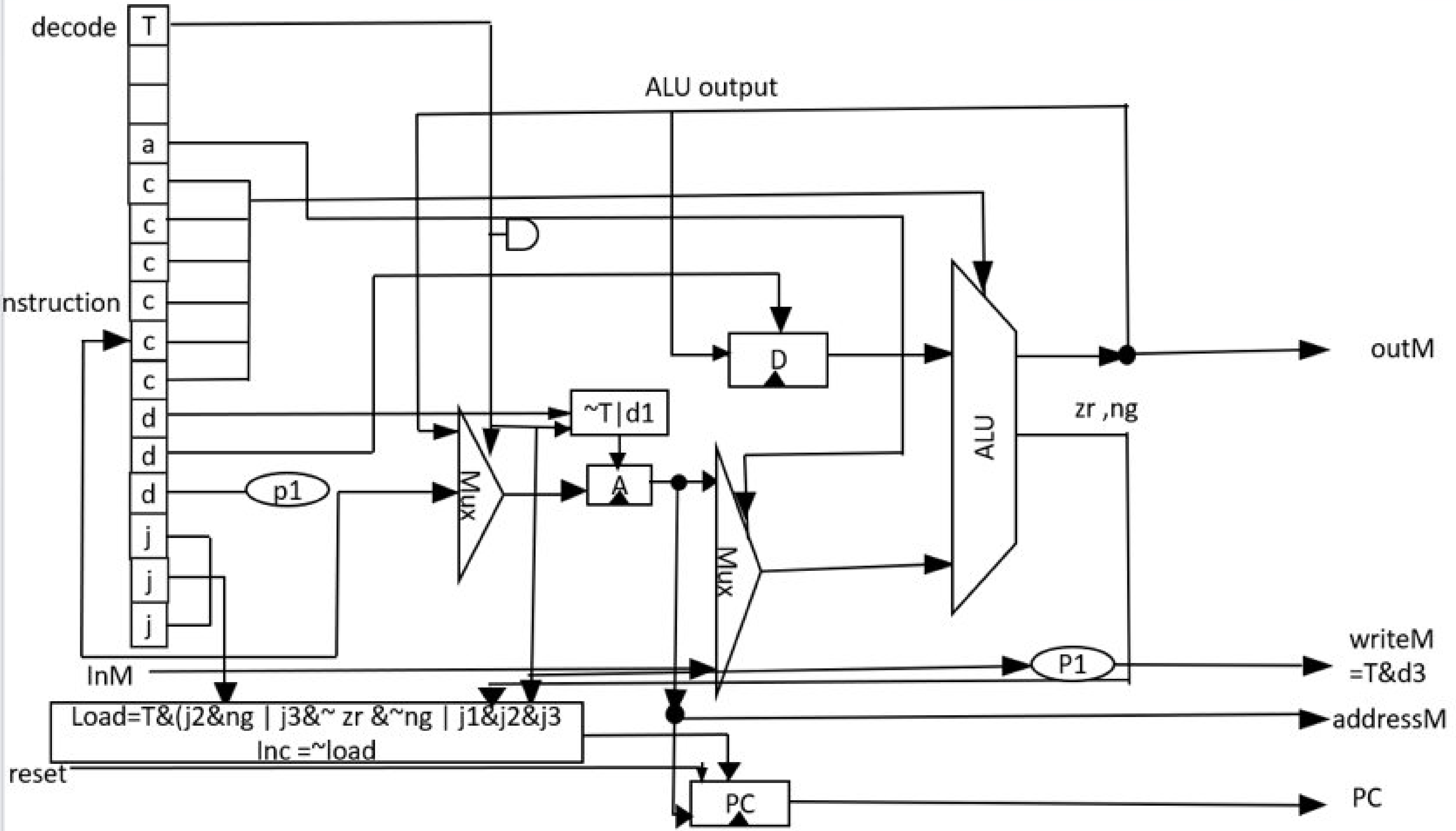
Internal pins

Name	Value
loadram1	0
loadram2	0
loadscreen	0
loadkbd	0
loadram	0
ramout[16]	2222
scrout[16]	0
kbout[16]	0

RAM 16K:

8189	0
8190	0
8191	0
8192	2222
8193	0
8194	0
8195	0

End of script - Comparison ended successfully


decode
T
a
c
c
c
c
c
c
d
d
d
j
j
j
nstruction
ALU output
D
~T|d1
p1
Mux
A
Mux
ALU
outM
zr ,ng
writeM
=T&d3
P1
InM
Load=T&(j2&ng | j3&~ zr &~ng | j1&j2&j3
Inc =~load
addressM
reset
PC
PC

CPU abstraction

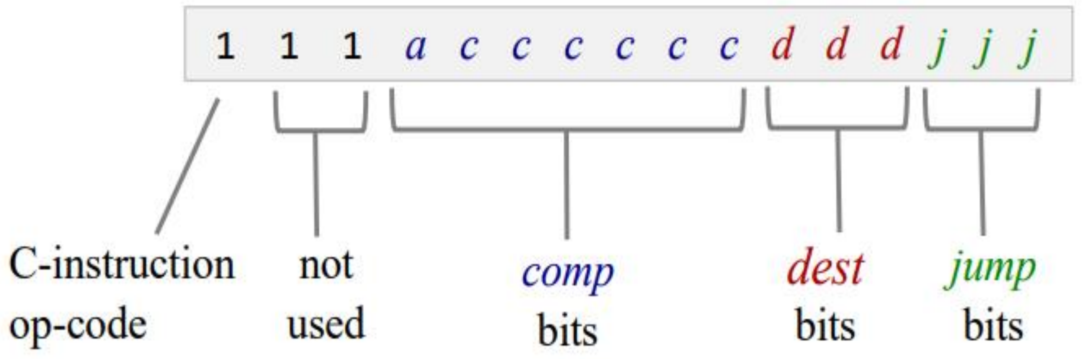
A chip that implements the Hack IS:

A instruction	Symbolic: *@xxx*	(*xxx* is a decimal value ranging from 0 to 32767, or a symbol bound to such a decimal value)
	Binary: 0 *vvvvvvvvvvvvvvv*	(*vv* ... *v* = 15-bit value of *xxx*)
C instruction	Symbolic: *dest* = *comp* ; *jump*	(*comp* is mandatory. If *dest* is empty, the = is omitted; If *jump* is empty, the ; is omitted)
	Binary: 111*accccccdddjjj*	

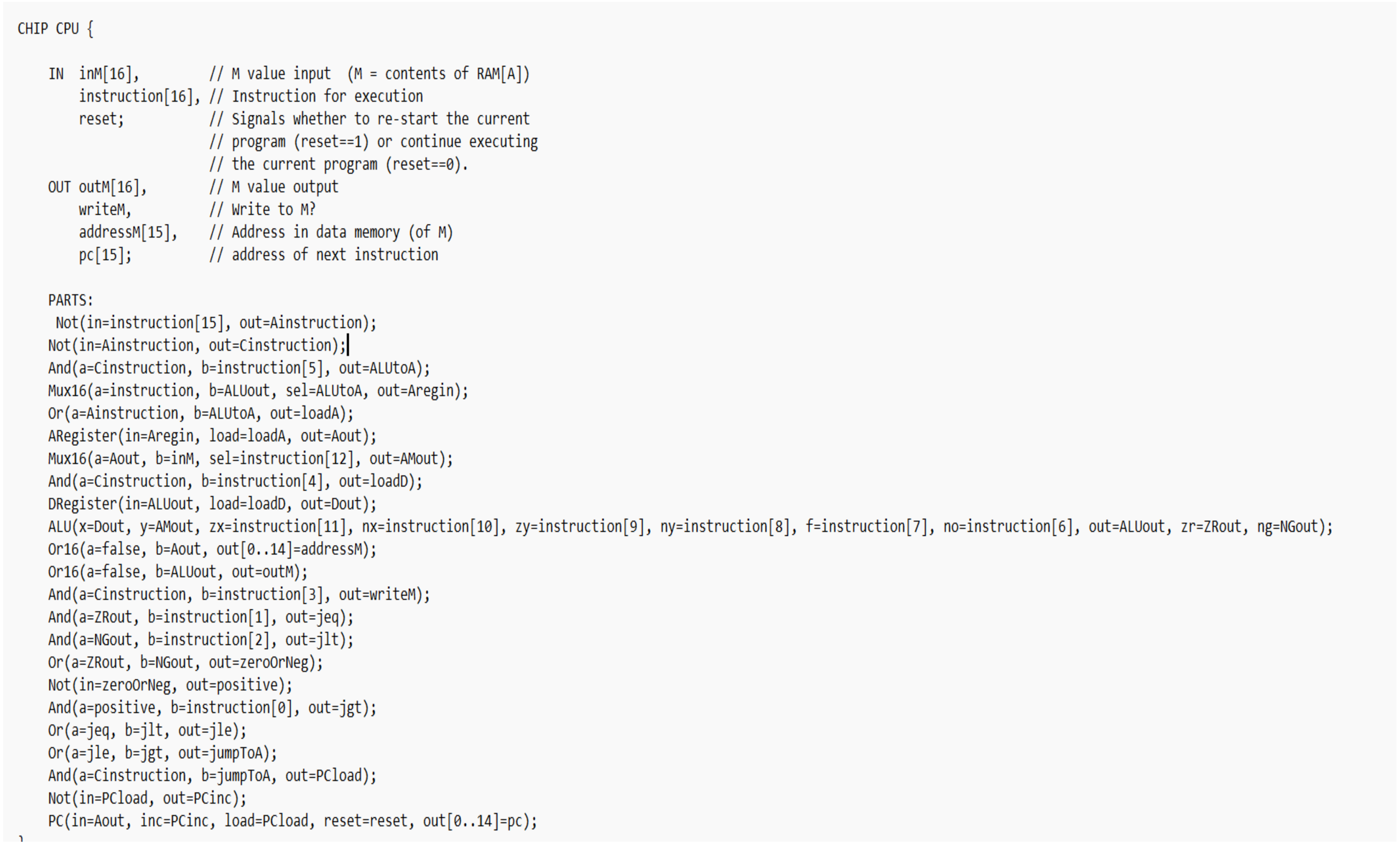
comp		*c*	*c*	*c*	*c*	*c*	*c*
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	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	*d*	*d*	*d*	Effect: store *comp* in:
null	0	0	0	the value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register (reg)
DM	0	1	1	RAM[A] and D reg
A	1	0	0	A reg
AM	1	0	1	A reg and RAM[A]
AD	1	1	0	A reg and D reg
ADM	1	1	1	A reg, D reg, and RAM[A]

jump	*j*	*j*	*j*	Effect:
null	0	0	0	no jump
JGT	0	0	1	if *comp* > 0 jump
JEQ	0	1	0	if *comp* = 0 jump
JGE	0	1	1	if *comp* ≥ 0 jump
JLT	1	0	0	if *comp* < 0 jump
JNE	1	0	1	if *comp* ≠ 0 jump
JLE	1	1	0	if *comp* ≤ 0 jump
JMP	1	1	1	unconditional jump



HDL CODE



```
CHIP CPU {

    IN  inM[16],         // M value input  (M = contents of RAM[A])
        instruction[16], // Instruction for execution
        reset;           // Signals whether to re-start the current
                         // program (reset==1) or continue executing
                         // the current program (reset==0).

    OUT outM[16],        // M value output
        writeM,          // Write to M?
        addressM[15],    // Address in data memory (of M)
        pc[15];          // address of next instruction

    PARTS:
     Not(in=instruction[15], out=Ainstruction);
    Not(in=Ainstruction, out=Cinstruction);
    And(a=Cinstruction, b=instruction[5], out=ALUtoA);
    Mux16(a=instruction, b=ALUout, sel=ALUtoA, out=Aregin);
    Or(a=Ainstruction, b=ALUtoA, out=loadA);
    ARegister(in=Aregin, load=loadA, out=Aout);
    Mux16(a=Aout, b=inM, sel=instruction[12], out=AMout);
    And(a=Cinstruction, b=instruction[4], out=loadD);
    DRegister(in=ALUout, load=loadD, out=Dout);
    ALU(x=Dout, y=AMout, zx=instruction[11], nx=instruction[10], zy=instruction[9], ny=instruction[8], f=instruction[7], no=instruction[6], out=ALUout, zr=ZRout, ng=NGout);
    Or16(a=false, b=Aout, out[0..14]=addressM);
    Or16(a=false, b=ALUout, out=outM);
    And(a=Cinstruction, b=instruction[3], out=writeM);
    And(a=ZRout, b=instruction[1], out=jeq);
    And(a=NGout, b=instruction[2], out=jlt);
    Or(a=ZRout, b=NGout, out=zeroOrNeg);
    Not(in=zeroOrNeg, out=positive);
    And(a=positive, b=instruction[0], out=jgt);
    Or(a=jeq, b=jlt, out=jle);
    Or(a=jle, b=jgt, out=jumpToA);
    And(a=Cinstruction, b=jumpToA, out=PCload);
    Not(in=PCload, out=PCinc);
    PC(in=Aout, inc=PCinc, load=PCload, reset=reset, out[0..14]=pc);
```

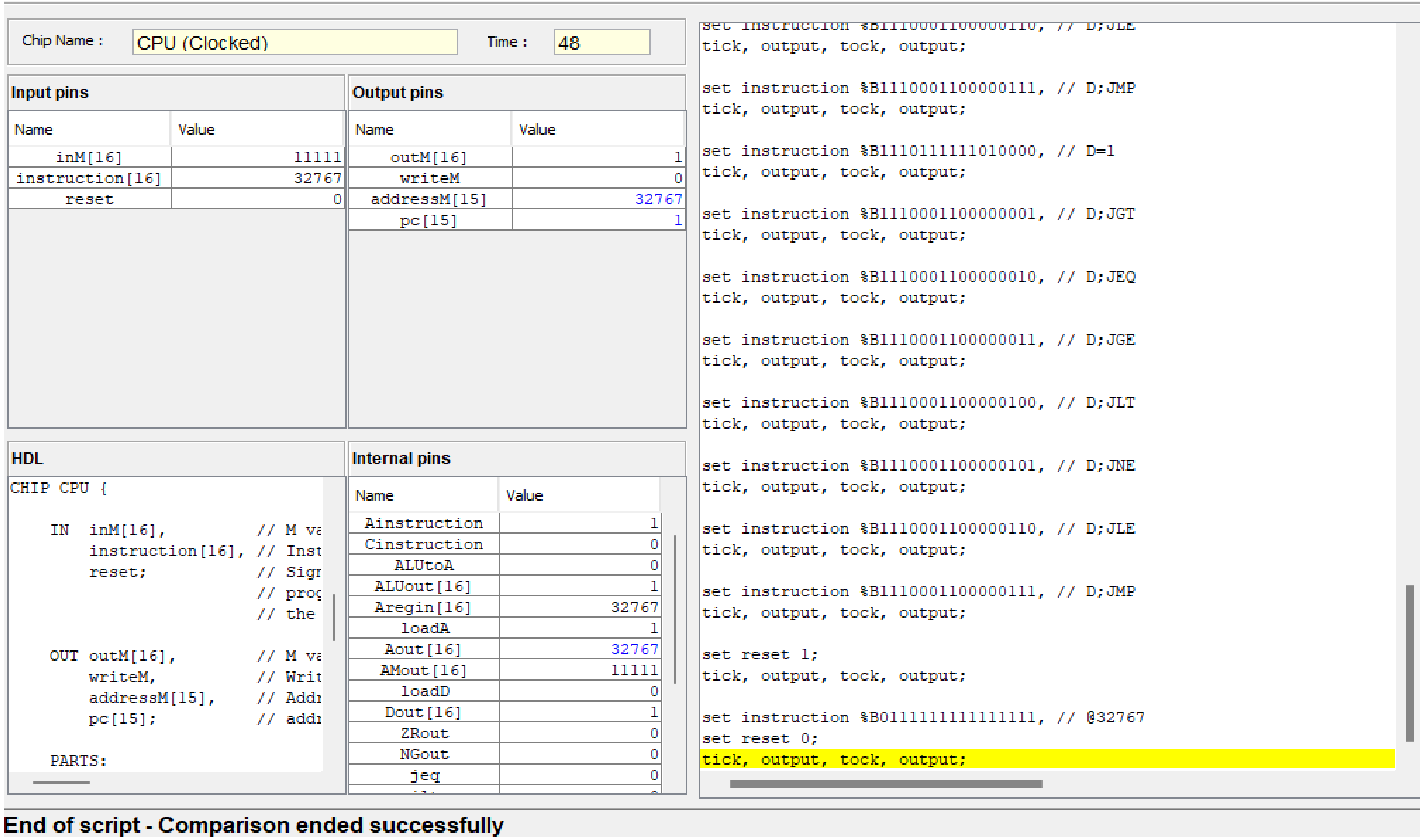
```
CHIP CPU {

    IN  inM[16],         // M value input  (M = contents of RAM[A])
    instruction[16], // Instruction for execution
    reset;           // Signals whether to re-start the current
                      // program (reset==1) or continue executing
                      // the current program (reset==0).
    OUT outM[16],        // M value output
     writeM,          // Write to M?
     addressM[15],    // Address in data memory (of M)
      pc[15];         // address of next instruction
```

```
PARTS:
//to check if the instruction in A-instruction
     Not(in=instruction[15], out=Ainstruction);//A-instruction
    Not(in=Ainstruction, out=Cinstruction);  //C-instruction(opposite)

//to check if the instruction is C-instruction
    And(a=Cinstruction, b=instruction[5], out=ALUtoA);// C-instruction
    Mux16(a=instruction, b=ALUout, sel=ALUtoA, out=Aregin);  // strores the value in A register

//if A-instruction or ALU output to the A register,loadA
    Or(a=Ainstruction, b=ALUtoA, out=loadA);
    ARegister(in=Aregin, load=loadA, out=Aout); //generates value into A register
```

```
  //selects A or M depending on the 12-bit instruction value
Mux16(a=Aout, b=inM, sel=instruction[12], out=AMout);
And(a=Cinstruction, b=instruction[4], out=loadD);// if ALU output is loaded to the D-register
DRegister(in=ALUout, load=loadD, out=Dout); // loads the value of D-register from ALU

//calculates the computations
ALU(x=Dout, y=AMout, zx=instruction[11], nx=instruction[10], zy=instruction[9], ny=instruction[8],
f=instruction[7], no=instruction[6], out=ALUout, zr=ZRout, ng=NGout);

//decides if cpu can write on data
Or16(a=false, b=Aout, out[0..14]=addressM);//address to write
Or16(a=false, b=ALUout, out=outM);//content in the data memory
And(a=Cinstruction, b=instruction[3], out=writeM);//write the data memory
```

```
//determines the jump conditions
And(a=ZRout, b=instruction[1], out=jeq);   //zero?jump if zero
    And(a=NGout, b=instruction[2], out=jlt);   //negative ?jump if negative
    Or(a=ZRout, b=NGout, out=zeroOrNeg);//negative or zero?
    Not(in=zeroOrNeg, out=positive);   //  positive(not zero nor negative)

    And(a=positive, b=instruction[0], out=jgt);
    Or(a=jeq, b=jlt, out=jle);
    Or(a=jle, b=jgt, out=jumpToA);
    And(a=Cinstruction, b=jumpToA, out=PCload); //load pc if c-instruction and conditions
   met
    Not(in=PCload, out=PCinc); //increment only if the pc is not loaded
    PC(in=Aout, inc=PCinc, load=PCload, reset=reset, out[0..14]=pc);
}
```

IMPLEMENTATION



Chip Name : CPU (Clocked)   Time : 48

Input pins

Name	Value
inM[16]	11111
instruction[16]	32767
reset	0

Output pins

Name	Value
outM[16]	1
writeM	0
addressM[15]	32767
pc[15]	1

HDL

```
CHIP CPU {

    IN  inM[16],         // M va
        instruction[16], // Inst
        reset;           // Sigr
                         // proc
                         // the

    OUT outM[16],        // M va
        writeM,          // Writ
        addressM[15],    // Addr
        pc[15];          // addr

    PARTS:
```

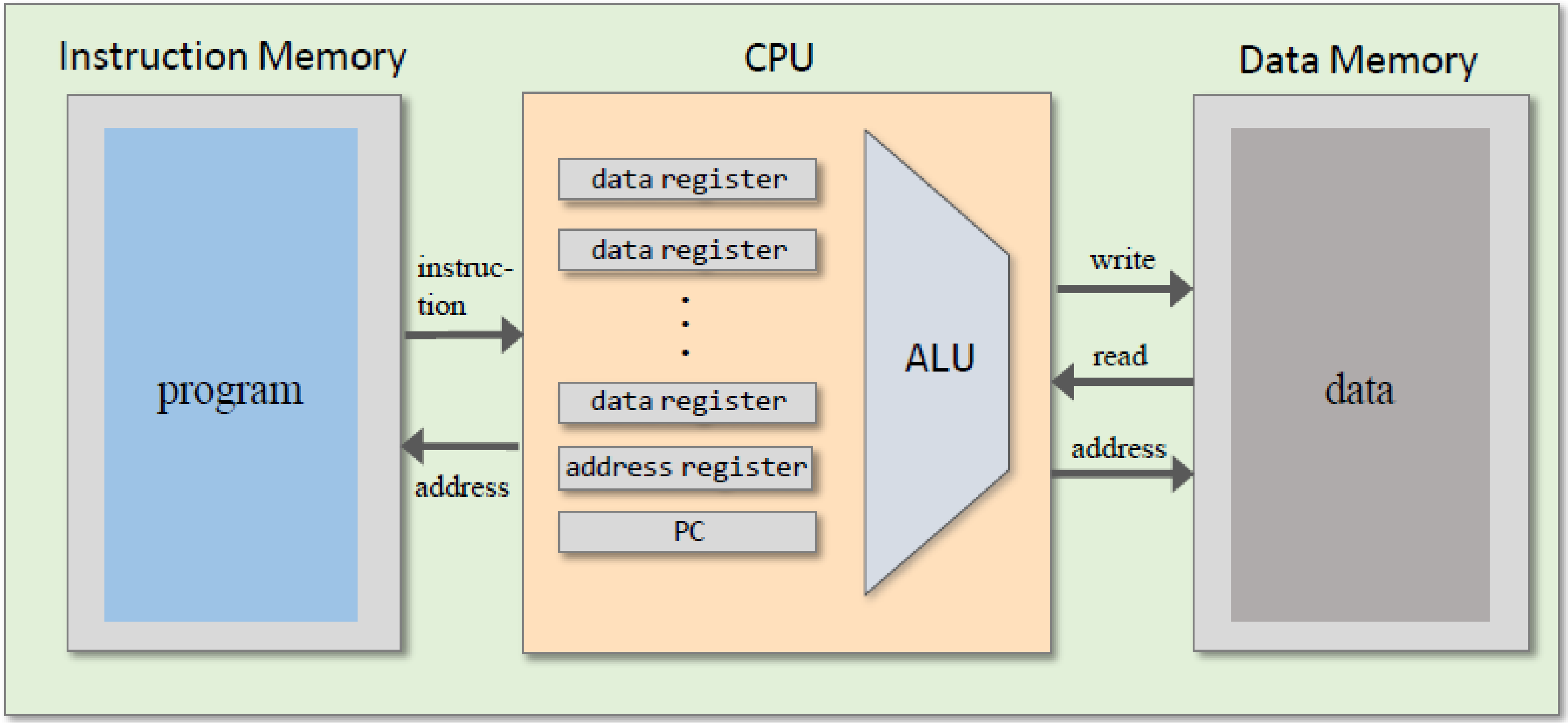
Internal pins

Name	Value
Ainstruction	1
Cinstruction	0
ALUtoA	0
ALUout[16]	1
Aregin[16]	32767
loadA	1
Aout[16]	32767
AMout[16]	11111
loadD	0
Dout[16]	1
ZRout	0
NGout	0
jeq	0

```
set instruction %B1110001100000110, // D;JLE
tick, output, tock, output;

set instruction %B1110001100000111, // D;JMP
tick, output, tock, output;

set instruction %B1110111111010000, // D=1
tick, output, tock, output;

set instruction %B1110001100000001, // D;JGT
tick, output, tock, output;

set instruction %B1110001100000010, // D;JEQ
tick, output, tock, output;

set instruction %B1110001100000011, // D;JGE
tick, output, tock, output;

set instruction %B1110001100000100, // D;JLT
tick, output, tock, output;

set instruction %B1110001100000101, // D;JNE
tick, output, tock, output;

set instruction %B1110001100000110, // D;JLE
tick, output, tock, output;

set instruction %B1110001100000111, // D;JMP
tick, output, tock, output;

set reset 1;
tick, output, tock, output;

set instruction %B0111111111111111, // @32767
set reset 0;
tick, output, tock, output;
```

End of script - Comparison ended successfully

HACK COMPUTER

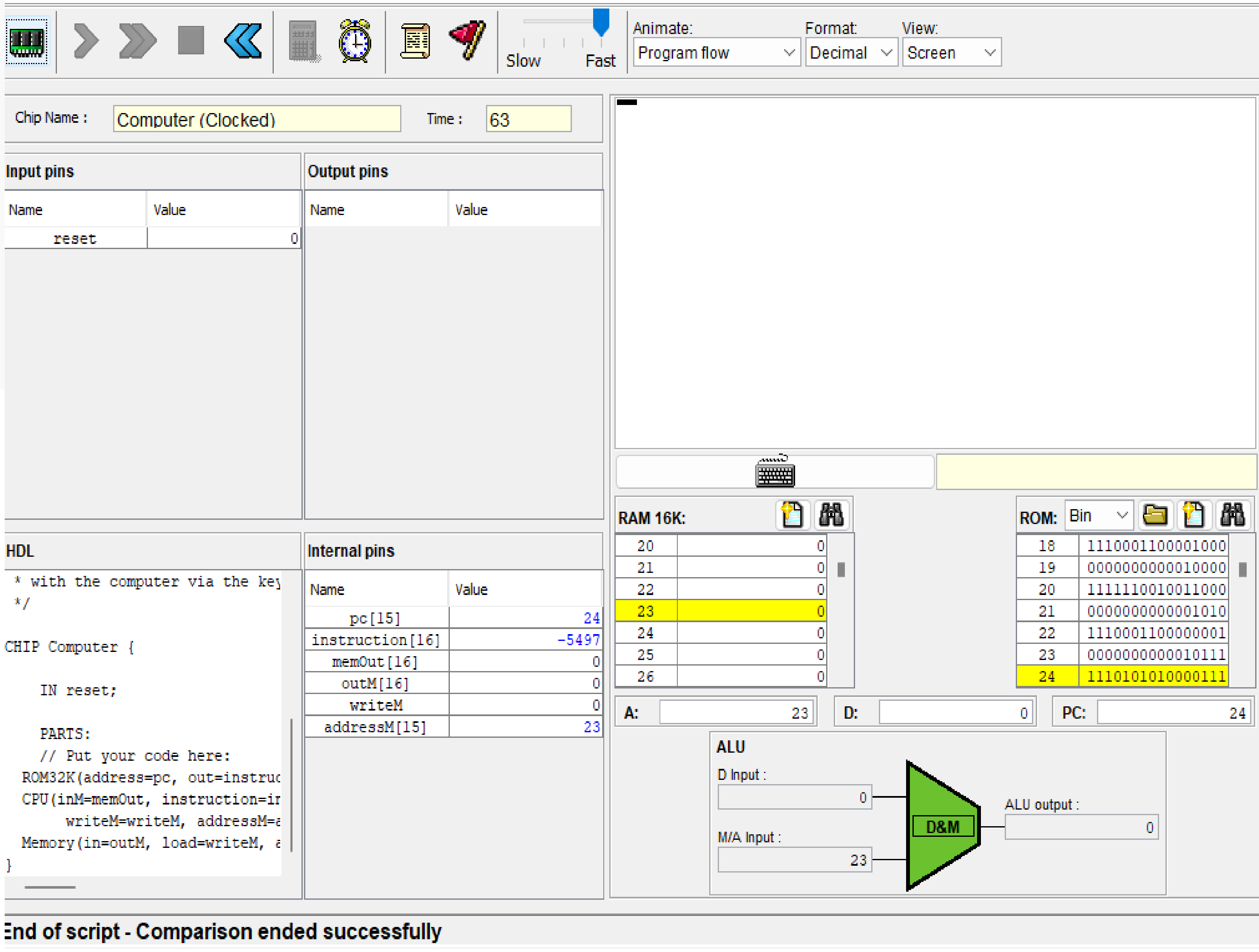


NAND2TETRIS

HDL CODE

```
CHIP Computer {

    IN reset;

    PARTS:
    // Put your code here:
  ROM32K(address=pc, out=instruction);
  CPU(inM=memOut, instruction=instruction, reset=reset, outM=outM,
       writeM=writeM, addressM=addressM, pc=pc);
  Memory(in=outM, load=writeM, address=addressM, out=memOut);
}
```

IMPLEMENTATION



Animate: Program flow
Format: Decimal
View: Screen

Chip Name : Computer (Clocked)
Time : 63

Input pins

Name	Value
reset	0

Output pins

Name	Value

HDL

```
 * with the computer via the key
 */

CHIP Computer {

    IN reset;

    PARTS:
    // Put your code here:
  ROM32K(address=pc, out=instruc
  CPU(inM=memOut, instruction=in
       writeM=writeM, addressM=a
  Memory(in=outM, load=writeM, a
}
```

Internal pins

Name	Value
pc[15]	24
instruction[16]	-5497
memOut[16]	0
outM[16]	0
writeM	0
addressM[15]	23

RAM 16K:

20	0
21	0
22	0
23	0
24	0
25	0
26	0

ROM: Bin

18	1110001100001000
19	0000000000010000
20	1111110010011000
21	0000000000001010
22	1110001100000001
23	0000000000010111
24	1110101010000111

A: 23
D: 0
PC: 24

D Input : 0
M/A Input : 23
ALU output : 0

End of script - Comparison ended successfully

PART B:

Synchronous down counter from 9 onwards to 0

INTRODUCTION

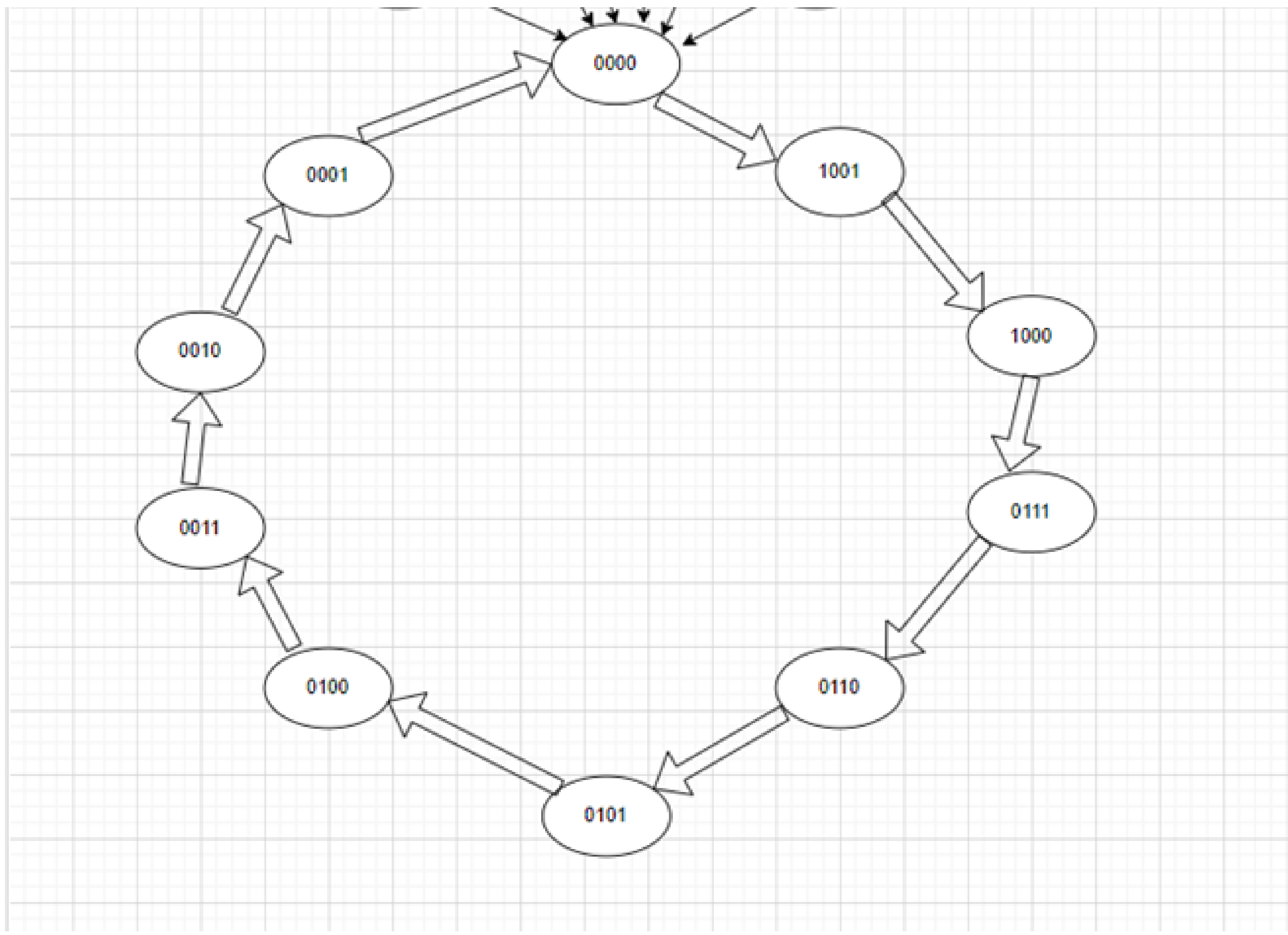
- A synchronous counter is a type of digital counter circuit in which all flip-flops or stages change state simultaneously in response to a clock signal.

- The most common types of synchronous counters are binary counters, where each flip-flop represents a binary digit in the counting sequence.

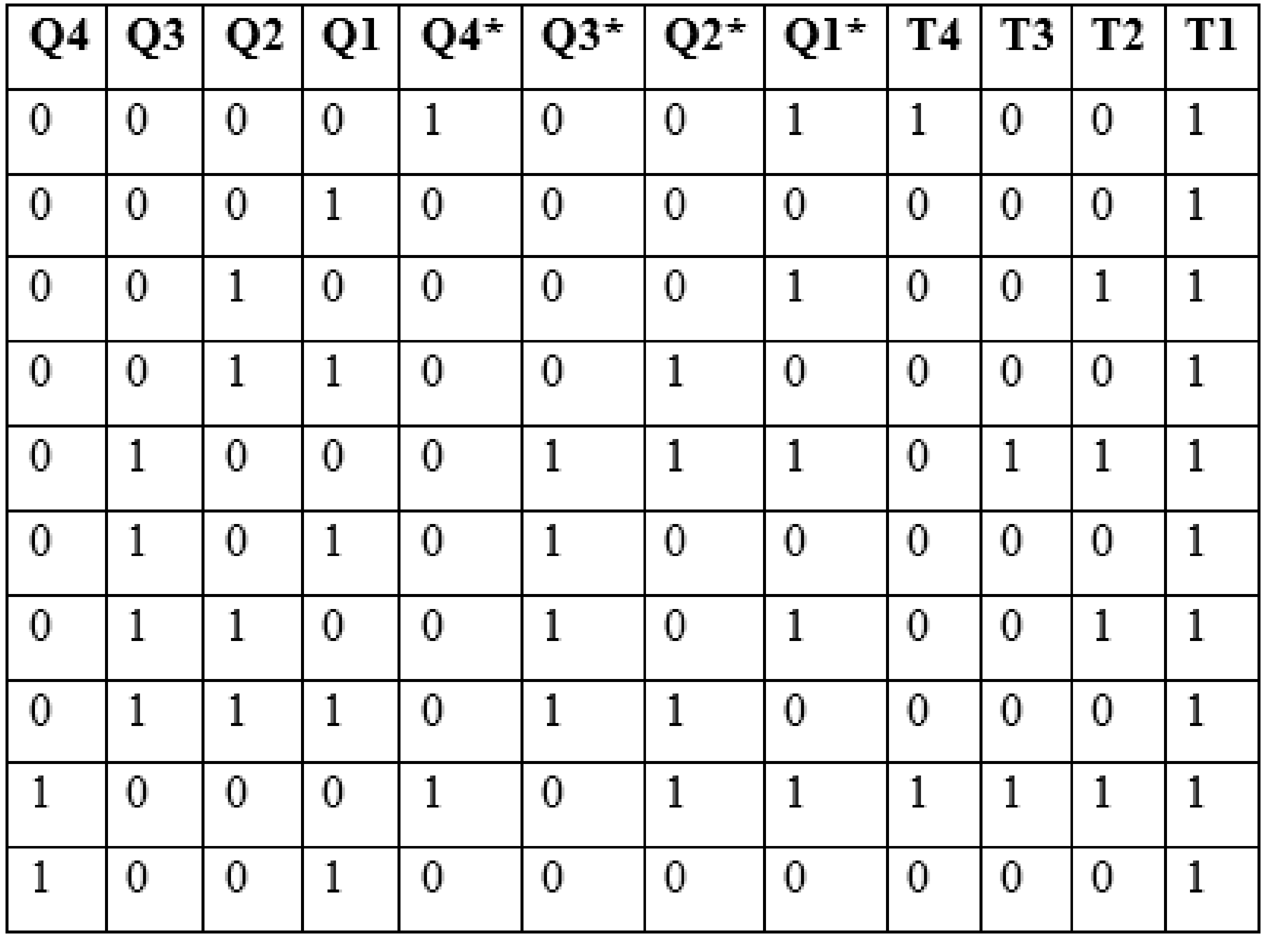
- A down counter is a type of digital counter circuit that counts down from a predetermined value to zero

- Down counters are commonly implemented using flip-flops and combinational logic circuits.

State diagram



TRUTH TABLE



Q4	Q3	Q2	Q1	Q4*	Q3*	Q2*	Q1*	T4	T3	T2	T1
0	0	0	0	1	0	0	1	1	0	0	1
0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	1	0	0	1	1
0	0	1	1	0	0	1	0	0	0	0	1
0	1	0	0	0	1	1	1	0	1	1	1
0	1	0	1	0	1	0	0	0	0	0	1
0	1	1	0	0	1	0	1	0	0	1	1
0	1	1	1	0	1	1	0	0	0	0	1
1	0	0	0	1	0	1	1	1	1	1	1
1	0	0	1	0	0	0	0	0	0	0	1

K-MAP

For T4:

Q3Q4 Q1Q2	00	01	11	10
00	1			
01				
11	x	x	x	x
10	1		x	x

= Q1’Q2’Q3’

For T3:

Q3Q4 Q1Q2	00	01	11	10
00				
01	1			
11	x	x	x	x
10	1		x	x

=Q3Q2’Q1’+Q4Q2’Q1’

For T2:

Q3Q4 Q1Q2	00	01	11	10
00				1
01	1			1
11	x	x	x	x
10	1		x	x

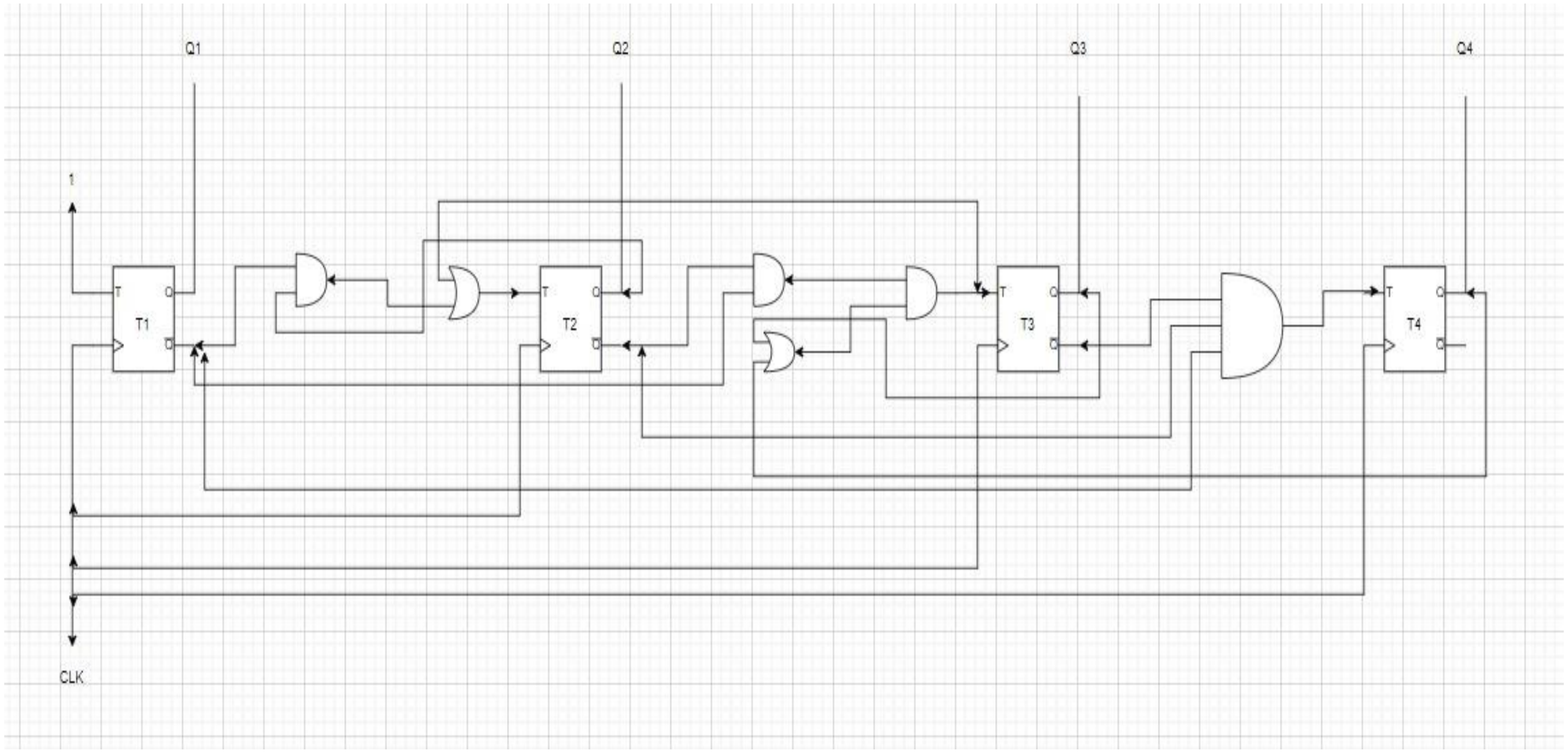
=Q2Q1’+Q3Q2’Q1’+Q4Q2’Q1’

For T1:

Q3Q4 Q1Q2	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	X	X	X	X
10	1	X	X	X

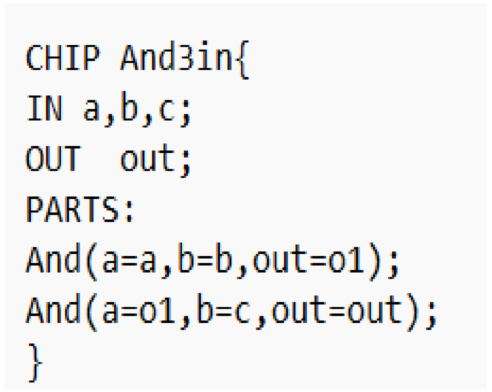
=1

BLOCK DIAGRAM



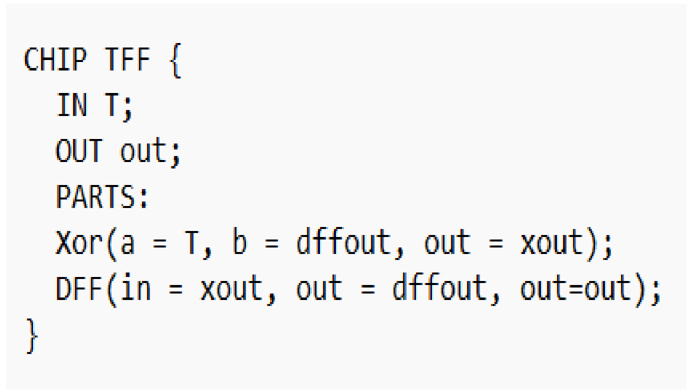
IMPLEMENTATION

And 3 input



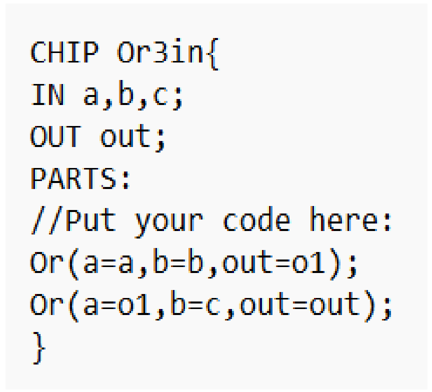
```
CHIP And3in{
IN a,b,c;
OUT  out;
PARTS:
And(a=a,b=b,out=o1);
And(a=o1,b=c,out=out);
}
```

T Flip Flop



```
CHIP TFF {
  IN T;
  OUT out;
  PARTS:
  Xor(a = T, b = dffout, out = xout);
  DFF(in = xout, out = dffout, out=out);
}
```

Or 3 input

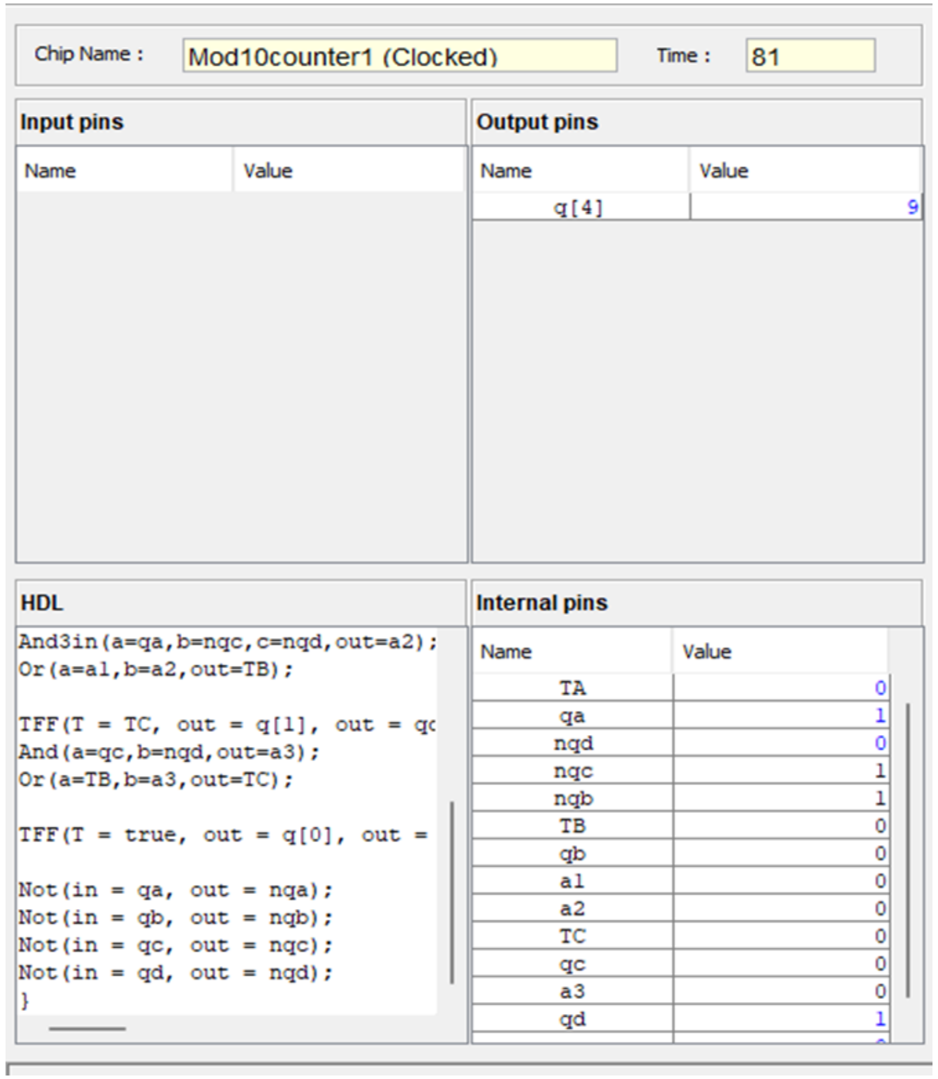


```
CHIP Or3in{
IN a,b,c;
OUT out;
PARTS:
//Put your code here:
Or(a=a,b=b,out=o1);
Or(a=o1,b=c,out=out);
}
```

HDL CODE

```
CHIP Mod10counter1
{
    OUT q[4];

    PARTS:

        TFF(T = TA, out = q[3], out = qa);
        And3in(a = nqd, b = nqc, c = nqb, out = TA);

        TFF(T = TB, out = q[2], out = qb);
        And3in(a=qb,b=nqc,c=nqd,out=a1);
And3in(a=qa,b=nqc,c=nqd,out=a2);
Or(a=a1,b=a2,out=TB);

        TFF(T = TC, out = q[1], out = qc);
        And(a=qc,b=nqd,out=a3);
Or(a=TB,b=a3,out=TC);

        TFF(T = true, out = q[0], out = qd);

        Not(in = qa, out = nqa);
        Not(in = qb, out = nqb);
        Not(in = qc, out = nqc);
        Not(in = qd, out = nqd);
}
```

IMPLEMENTATION



Chip Name : Mod10counter1 (Clocked)   Time : 81

Input pins

Name	Value

Output pins

Name	Value
q[4]	9

HDL

```
And3in(a=qa,b=nqc,c=nqd,out=a2);
Or(a=a1,b=a2,out=TB);

TFF(T = TC, out = q[1], out = qc
And(a=qc,b=nqd,out=a3);
Or(a=TB,b=a3,out=TC);

TFF(T = true, out = q[0], out =

Not(in = qa, out = nqa);
Not(in = qb, out = nqb);
Not(in = qc, out = nqc);
Not(in = qd, out = nqd);
}
```

Internal pins

Name	Value
TA	0
qa	1
nqd	0
nqc	1
nqb	1
TB	0
qb	0
a1	0
a2	0
TC	0
qc	0
a3	0
qd	1

CONCLUSION

- In summary, our investigation has led to the development and examination of a 4-bit synchronous down counter engineered to count in a decrementing sequence from 9 to 0. Employing four flip-flops to represent individual digits, the synchronous design guarantees simultaneous state changes across all flipflops with each clock pulse. The counter initiates at 1001(9) and successively transitions through binary states, concluding at 0000(0)