Digital Design and Computer Organization Laboratory

UE19CS206

3rd Semester, Academic Year 2020-21

Date: 24/10/2022

Name : YAMAN	SRN : PES2UG21CS619	Section : J	

Experiment Number: 6 Week # : 6

Title of the Program:

16 Bit Program Counter

Aim of the Program:

To Design and Implementation of a 16 bit Program Counter

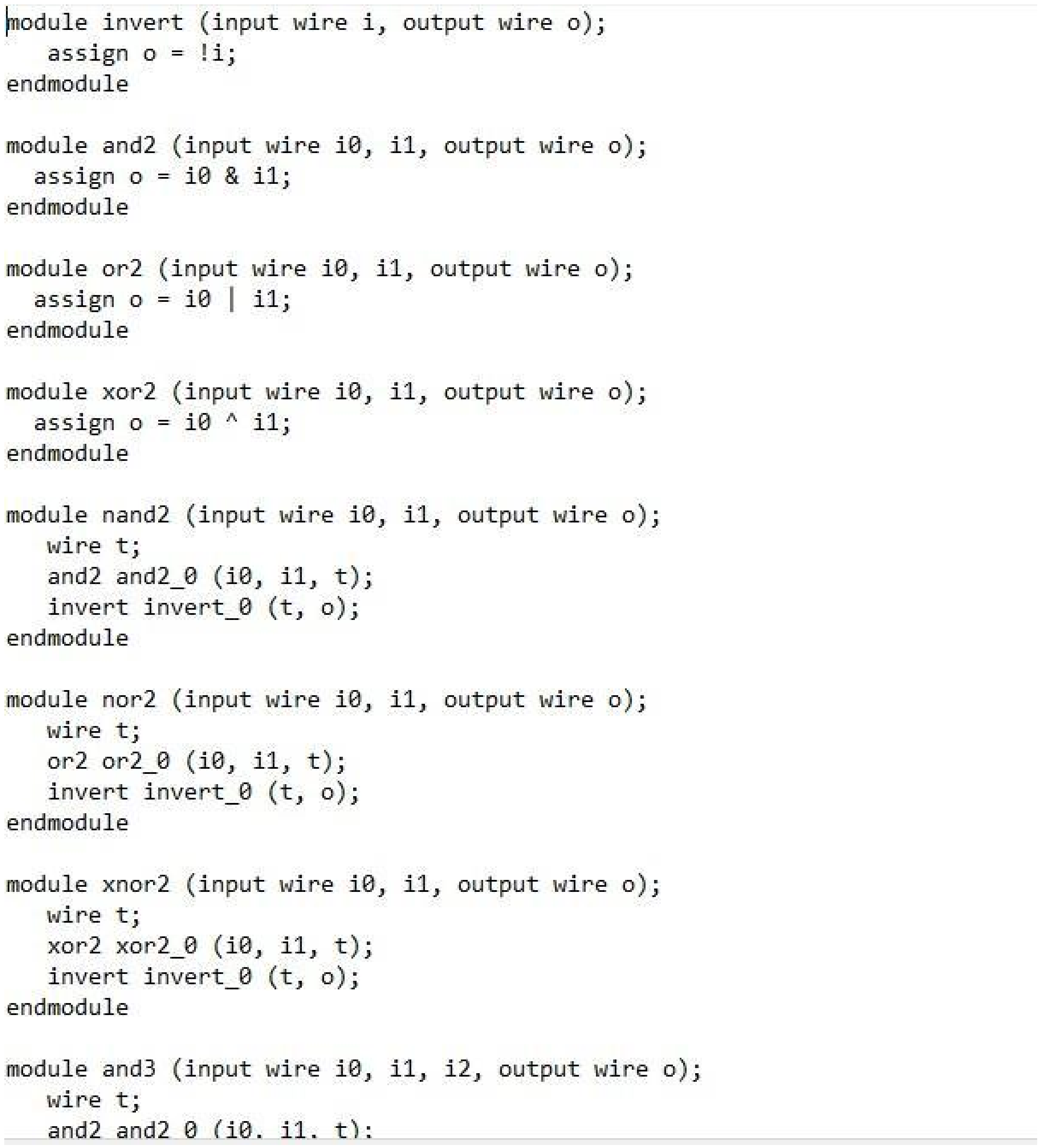
1.tb_c.v

```
//   Test bench for PC:

`timescale 1 ns / 100 ps
`define TESTVECS 5

module tb;
  reg clk, reset, inc, add, sub;
  reg [15:0] offset;
  wire [15:0] pc;
  reg [18:0] test_vecs [0:(`TESTVECS-1)];
  integer i;
  initial begin $dumpfile("tb_pc.vcd"); $dumpvars(0,tb); end
  initial begin reset = 1'b1; #12.5 reset = 1'b0; end
  initial clk = 1'b0; always #5 clk =~ clk;
  initial begin
    test_vecs[0][18] = 1'b1; test_vecs[0][17] = 1'b0; test_vecs[0][16] = 1'b0;
    test_vecs[0][15:0] = 15'hxx;
    test_vecs[1][18] = 1'b0; test_vecs[1][17] = 1'b1; test_vecs[1][16] = 1'b0;
    test_vecs[1][15:0] = 15'ha5;
    test_vecs[2][18] = 1'b0; test_vecs[2][17] = 1'b0; test_vecs[2][16] = 1'b0;
    test_vecs[2][15:0] = 15'hxx;
    test_vecs[3][18] = 1'b1; test_vecs[3][17] = 1'b0; test_vecs[3][16] = 1'b0;
    test_vecs[3][15:0] = 15'hxx;
    test_vecs[4][18] = 1'b0; test_vecs[4][17] = 1'b0; test_vecs[4][16] = 1'b1;
    test_vecs[4][15:0] = 15'h14;
  end
  initial {inc, add, sub, offset} = 0;
  pc pc_0 (clk, reset, inc, add, sub, offset, pc);
  initial begin
    #6 for(i=0;i<`TESTVECS;i=i+1)
      begin #10 {inc, add, sub, offset}=test_vecs[i]; end
    #100 $finish;
  end
always@(reset or inc or add or sub )
$monitor("At time = %t, Reset= %b,inc=%b, add=%b,sub = %b,pc =%h ", $time,reset,inc,add,sub,pc);

endmodule
```

2.Pc.v

```
//   Test bench for PC:

`timescale 1 ns / 100 ps
`define TESTVECS 5

module tb;
  reg clk, reset, inc, add, sub;
  reg [15:0] offset;
  wire [15:0] pc;
  reg [18:0] test_vecs [0:(`TESTVECS-1)];
  integer i;
  initial begin $dumpfile("tb_pc.vcd"); $dumpvars(0,tb); end
  initial begin reset = 1'b1; #12.5 reset = 1'b0; end
  initial clk = 1'b0; always #5 clk =~ clk;
  initial begin
    test_vecs[0][18] = 1'b1; test_vecs[0][17] = 1'b0; test_vecs[0][16] = 1'b0;
    test_vecs[0][15:0] = 15'hxx;
    test_vecs[1][18] = 1'b0; test_vecs[1][17] = 1'b1; test_vecs[1][16] = 1'b0;
    test_vecs[1][15:0] = 15'ha5;
    test_vecs[2][18] = 1'b0; test_vecs[2][17] = 1'b0; test_vecs[2][16] = 1'b0;
    test_vecs[2][15:0] = 15'hxx;
    test_vecs[3][18] = 1'b1; test_vecs[3][17] = 1'b0; test_vecs[3][16] = 1'b0;
    test_vecs[3][15:0] = 15'hxx;
    test_vecs[4][18] = 1'b0; test_vecs[4][17] = 1'b0; test_vecs[4][16] = 1'b1;
    test_vecs[4][15:0] = 15'h14;
  end
  initial {inc, add, sub, offset} = 0;
  pc pc_0 (clk, reset, inc, add, sub, offset, pc);
  initial begin
    #6 for(i=0;i<`TESTVECS;i=i+1)
      begin #10 {inc, add, sub, offset}=test_vecs[i]; end
    #100 $finish;
  end
always@(reset or inc or add or sub )
$monitor("At time = %t, Reset= %b,inc=%b, add=%b,sub = %b,pc =%h ", $time,reset,inc,add,sub,pc);

endmodule
module pc (input wire clk, reset, inc, add, sub, input wire [15:0] offset, output wire [15:0] pc);
  input wire load;
  input wire [15:0] c;
  or3 or3_0 (inc, add, sub, load);
  pc_slice0 pc_slice_0 (clk, reset, sub, load, inc, sub, offset[0], c[0], pc[0]);
  pc_slice pc_slice_1 (clk, reset, c[0], load, inc, sub, offset[1], c[1], pc[1]);
  pc_slice pc_slice_2 (clk, reset, c[1], load, inc, sub, offset[2], c[2], pc[2]);
  pc_slice pc_slice_3 (clk, reset, c[2], load, inc, sub, offset[3], c[3], pc[3]);
  pc_slice pc_slice_4 (clk, reset, c[3], load, inc, sub, offset[4], c[4], pc[4]);
  pc_slice pc_slice_5 (clk, reset, c[4], load, inc, sub, offset[5], c[5], pc[5]);
  pc_slice pc_slice_6 (clk, reset, c[5], load, inc, sub, offset[6], c[6], pc[6]);
  pc_slice pc_slice_7 (clk, reset, c[6], load, inc, sub, offset[7], c[7], pc[7]);
  pc_slice pc_slice_8 (clk, reset, c[7], load, inc, sub, offset[8], c[8], pc[8]);
  pc_slice pc_slice_9 (clk, reset, c[8], load, inc, sub, offset[9], c[9], pc[9]);
  pc_slice pc_slice_10 (clk, reset, c[9], load, inc, sub, offset[10], c[10], pc[10]);
  pc_slice pc_slice_11 (clk, reset, c[10], load, inc, sub, offset[11], c[11], pc[11]);
  pc_slice pc_slice_12 (clk, reset, c[11], load, inc, sub, offset[12], c[12], pc[12]);
  pc_slice pc_slice_13 (clk, reset, c[12], load, inc, sub, offset[13], c[13], pc[13]);
  pc_slice pc_slice_14 (clk, reset, c[13], load, inc, sub, offset[14], c[14], pc[14]);
  pc_slice pc_slice_15 (clk, reset, c[14], load, inc, sub, offset[15], c[15], pc[15]);
endmodule
```

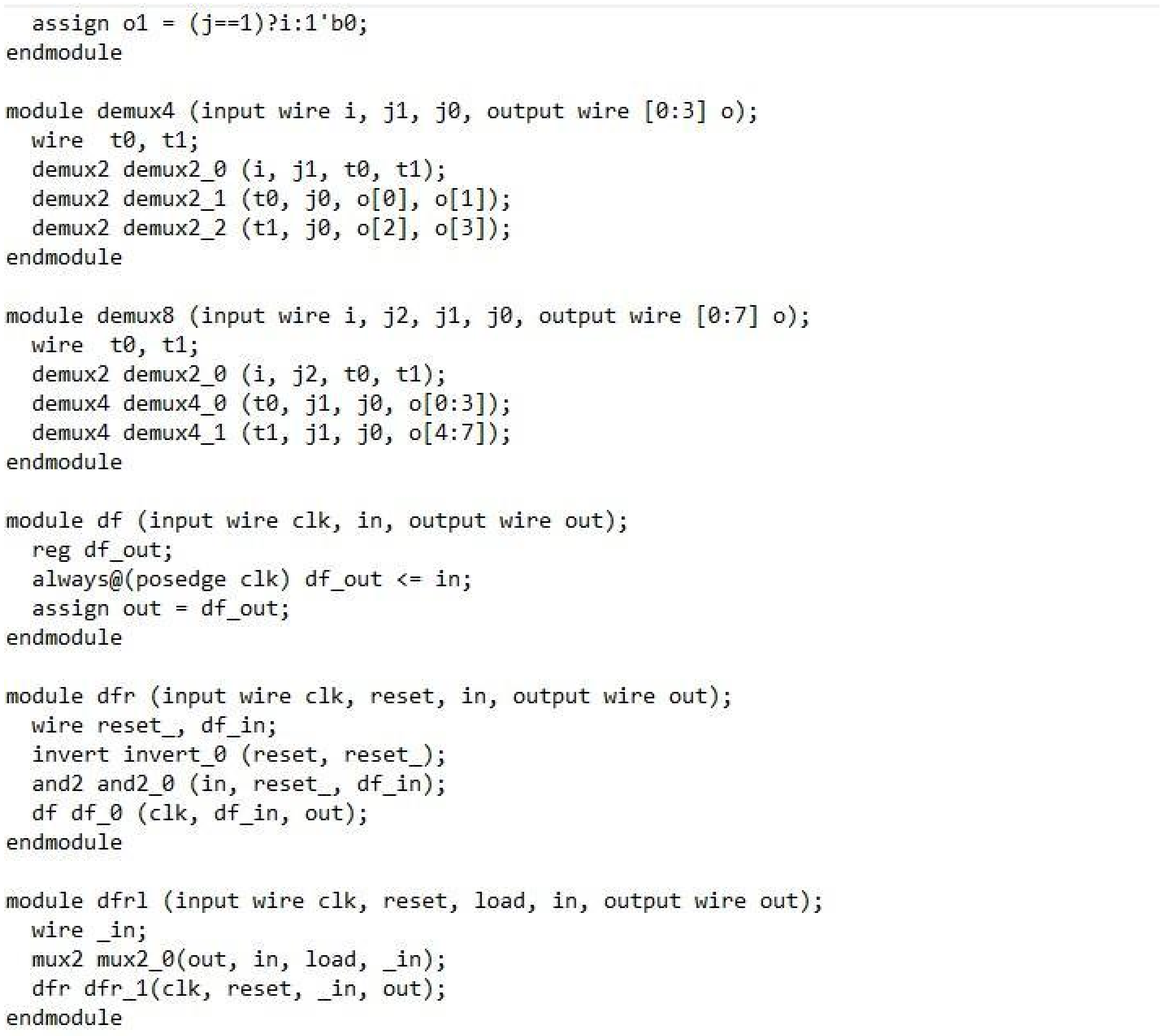
3.Lib.v



```
module invert (input wire i, output wire o);
   assign o = !i;
endmodule

module and2 (input wire i0, i1, output wire o);
  assign o = i0 & i1;
endmodule

module or2 (input wire i0, i1, output wire o);
  assign o = i0 | i1;
endmodule

module xor2 (input wire i0, i1, output wire o);
  assign o = i0 ^ i1;
endmodule

module nand2 (input wire i0, i1, output wire o);
   wire t;
   and2 and2_0 (i0, i1, t);
   invert invert_0 (t, o);
endmodule

module nor2 (input wire i0, i1, output wire o);
   wire t;
   or2 or2_0 (i0, i1, t);
   invert invert_0 (t, o);
endmodule

module xnor2 (input wire i0, i1, output wire o);
   wire t;
   xor2 xor2_0 (i0, i1, t);
   invert invert_0 (t, o);
endmodule

module and3 (input wire i0, i1, i2, output wire o);
   wire t;
   and2 and2_0 (i0, i1, t);
```

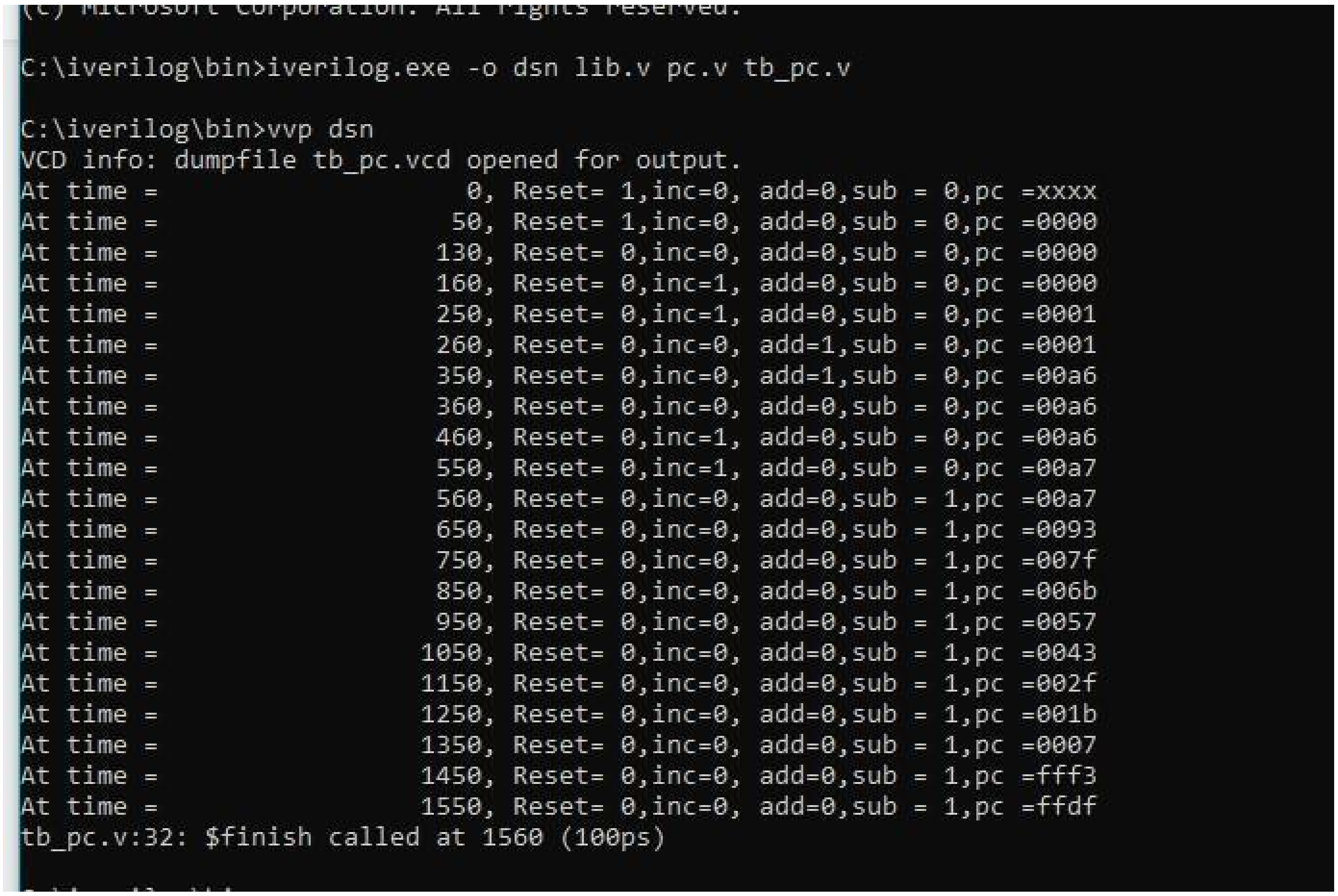
```
module and3 (input wire i0, i1, i2, output wire o);
   wire t;
   and2 and2_0 (i0, i1, t);
   and2 and2_1 (i2, t, o);
endmodule

module or3 (input wire i0, i1, i2, output wire o);
   wire t;
   or2 or2_0 (i0, i1, t);
   or2 or2_1 (i2, t, o);
endmodule

module nor3 (input wire i0, i1, i2, output wire o);
   wire t;
   or2 or2_0 (i0, i1, t);
   nor2 nor2_0 (i2, t, o);
endmodule

module nand3 (input wire i0, i1, i2, output wire o);
   wire t;
   and2 and2_0 (i0, i1, t);
   nand2 nand2_1 (i2, t, o);
endmodule

module xor3 (input wire i0, i1, i2, output wire o);
   wire t;
   xor2 xor2_0 (i0, i1, t);
   xor2 xor2_1 (i2, t, o);
endmodule

module xnor3 (input wire i0, i1, i2, output wire o);
   wire t;
   xor2 xor2_0 (i0, i1, t);
   xnor2 xnor2_0 (i2, t, o);
endmodule
```

```
module xnor3 (input wire i0, i1, i2, output wire o);
   wire t;
   xor2 xor2_0 (i0, i1, t);
   xnor2 xnor2_0 (i2, t, o);
endmodule

module mux2 (input wire i0, i1, j, output wire o);
  assign o = (j==0)?i0:i1;
endmodule

module mux4 (input wire [0:3] i, input wire j1, j0, output wire o);
  wire  t0, t1;
  mux2 mux2_0 (i[0], i[1], j1, t0);
  mux2 mux2_1 (i[2], i[3], j1, t1);
  mux2 mux2_2 (t0, t1, j0, o);
endmodule

module mux8 (input wire [0:7] i, input wire j2, j1, j0, output wire o);
  wire  t0, t1;
  mux4 mux4_0 (i[0:3], j2, j1, t0);
  mux4 mux4_1 (i[4:7], j2, j1, t1);
  mux2 mux2_0 (t0, t1, j0, o);
endmodule

module demux2 (input wire i, j, output wire o0, o1);
  assign o0 = (j==0)?i:1'b0;
  assign o1 = (j==1)?i:1'b0;
endmodule

module demux4 (input wire i, j1, j0, output wire [0:3] o);
  wire  t0, t1;
  demux2 demux2_0 (i, j1, t0, t1);
  demux2 demux2_1 (t0, j0, o[0], o[1]);
  demux2 demux2_2 (t1, j0, o[2], o[3]);
endmodule
```



```
  assign o1 = (j==1)?i:1'b0;
endmodule

module demux4 (input wire i, j1, j0, output wire [0:3] o);
  wire  t0, t1;
  demux2 demux2_0 (i, j1, t0, t1);
  demux2 demux2_1 (t0, j0, o[0], o[1]);
  demux2 demux2_2 (t1, j0, o[2], o[3]);
endmodule

module demux8 (input wire i, j2, j1, j0, output wire [0:7] o);
  wire  t0, t1;
  demux2 demux2_0 (i, j2, t0, t1);
  demux4 demux4_0 (t0, j1, j0, o[0:3]);
  demux4 demux4_1 (t1, j1, j0, o[4:7]);
endmodule

module df (input wire clk, in, output wire out);
  reg df_out;
  always@(posedge clk) df_out <= in;
  assign out = df_out;
endmodule

module dfr (input wire clk, reset, in, output wire out);
  wire reset_, df_in;
  invert invert_0 (reset, reset_);
  and2 and2_0 (in, reset_, df_in);
  df df_0 (clk, df_in, out);
endmodule

module dfrl (input wire clk, reset, load, in, output wire out);
  wire _in;
  mux2 mux2_0(out, in, load, _in);
  dfr dfr_1(clk, reset, _in, out);
endmodule
```

4.Vvp output



```
(c) Microsoft Corporation. All rights reserved.

C:\iverilog\bin>iverilog.exe -o dsn lib.v pc.v tb_pc.v

C:\iverilog\bin>vvp dsn
VCD info: dumpfile tb_pc.vcd opened for output.
At time =                    0, Reset= 1,inc=0, add=0,sub = 0,pc =xxxx
At time =                   50, Reset= 1,inc=0, add=0,sub = 0,pc =0000
At time =                  130, Reset= 0,inc=0, add=0,sub = 0,pc =0000
At time =                  160, Reset= 0,inc=1, add=0,sub = 0,pc =0000
At time =                  250, Reset= 0,inc=1, add=0,sub = 0,pc =0001
At time =                  260, Reset= 0,inc=0, add=1,sub = 0,pc =0001
At time =                  350, Reset= 0,inc=0, add=1,sub = 0,pc =00a6
At time =                  360, Reset= 0,inc=0, add=0,sub = 0,pc =00a6
At time =                  460, Reset= 0,inc=1, add=0,sub = 0,pc =00a6
At time =                  550, Reset= 0,inc=1, add=0,sub = 0,pc =00a7
At time =                  560, Reset= 0,inc=0, add=0,sub = 1,pc =00a7
At time =                  650, Reset= 0,inc=0, add=0,sub = 1,pc =0093
At time =                  750, Reset= 0,inc=0, add=0,sub = 1,pc =007f
At time =                  850, Reset= 0,inc=0, add=0,sub = 1,pc =006b
At time =                  950, Reset= 0,inc=0, add=0,sub = 1,pc =0057
At time =                 1050, Reset= 0,inc=0, add=0,sub = 1,pc =0043
At time =                 1150, Reset= 0,inc=0, add=0,sub = 1,pc =002f
At time =                 1250, Reset= 0,inc=0, add=0,sub = 1,pc =001b
At time =                 1350, Reset= 0,inc=0, add=0,sub = 1,pc =0007
At time =                 1450, Reset= 0,inc=0, add=0,sub = 1,pc =fff3
At time =                 1550, Reset= 0,inc=0, add=0,sub = 1,pc =ffdf
tb_pc.v:32: $finish called at 1560 (100ps)
```

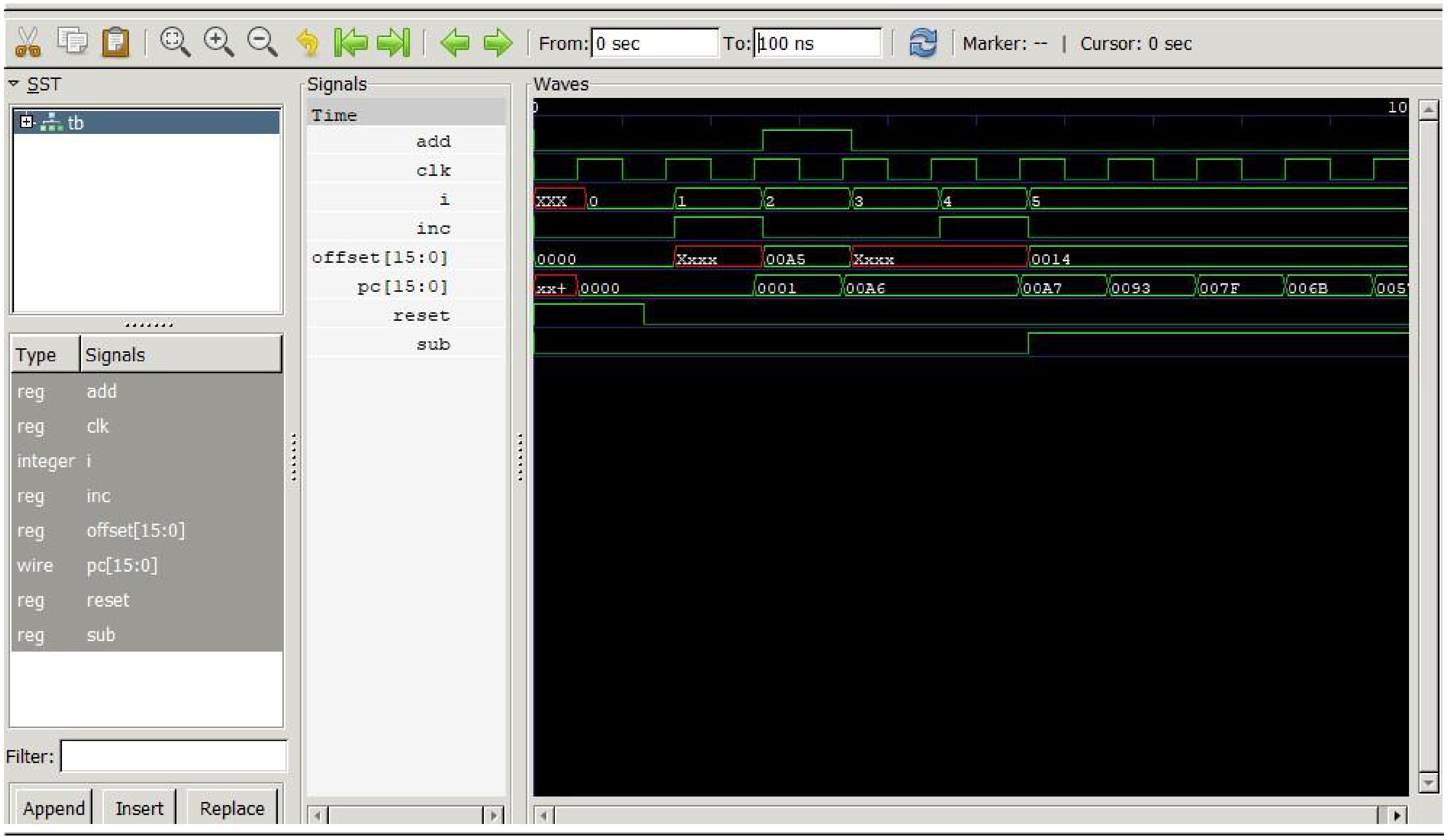
TABLE

inc	add	sub	offset [15:0]	output
Bit 18	Bit 17	Bit 16	Bit 15 to Bit0	pc[15:0]

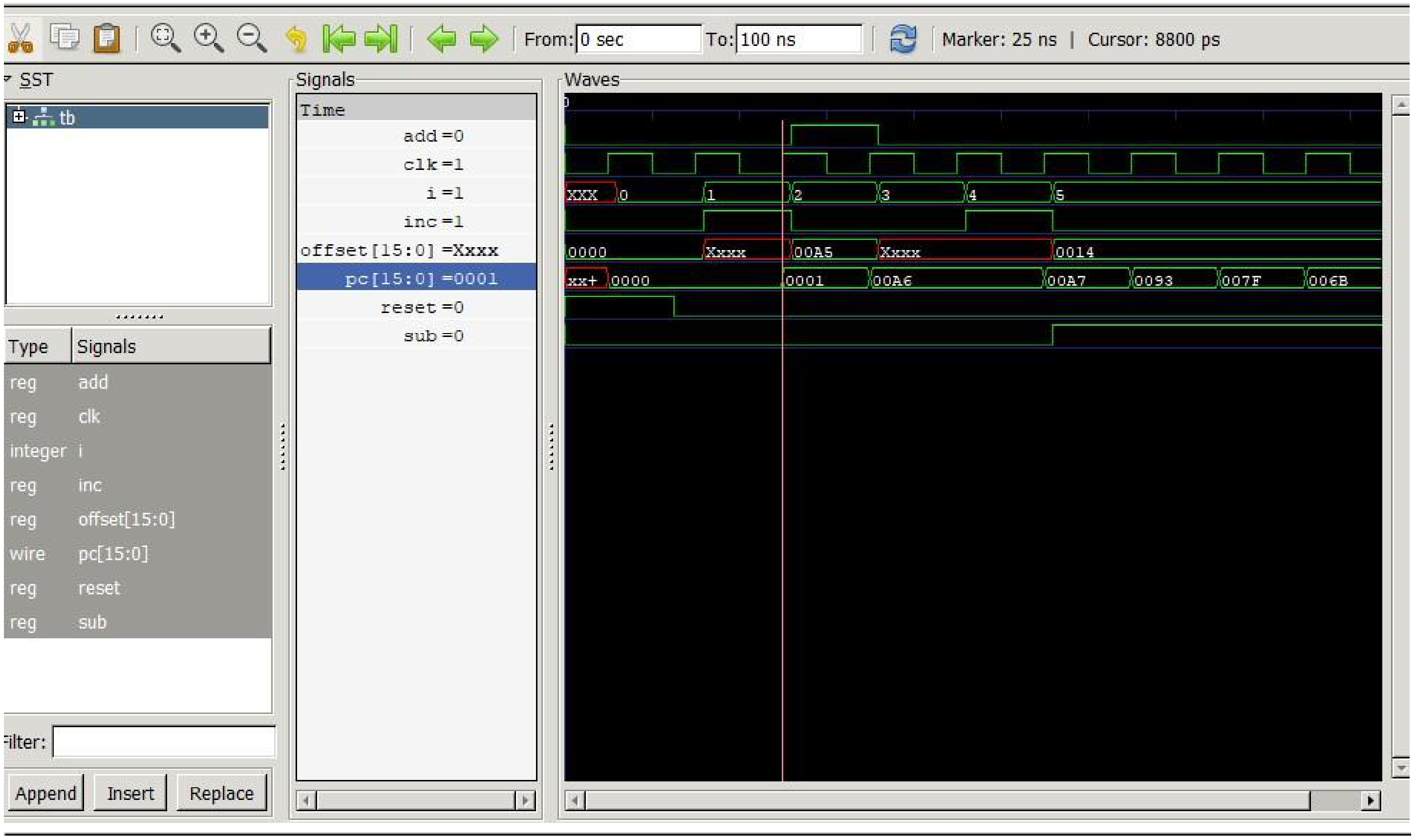
CASE 1	**1**	**0**	**0**	**XXXX**	**0001**

CASE 2	**0**	**1**	**0**	**00A5**	**00A6**
CASE 3	**0**	**0**	**0**	**XXXX**	**00A6**
CASE 4	**1**	**0**	**0**	**XXXX**	**00A7**
CASE 5	**0**	**0**	**1**	**0014**	**0093**

<u>Output waveform</u>



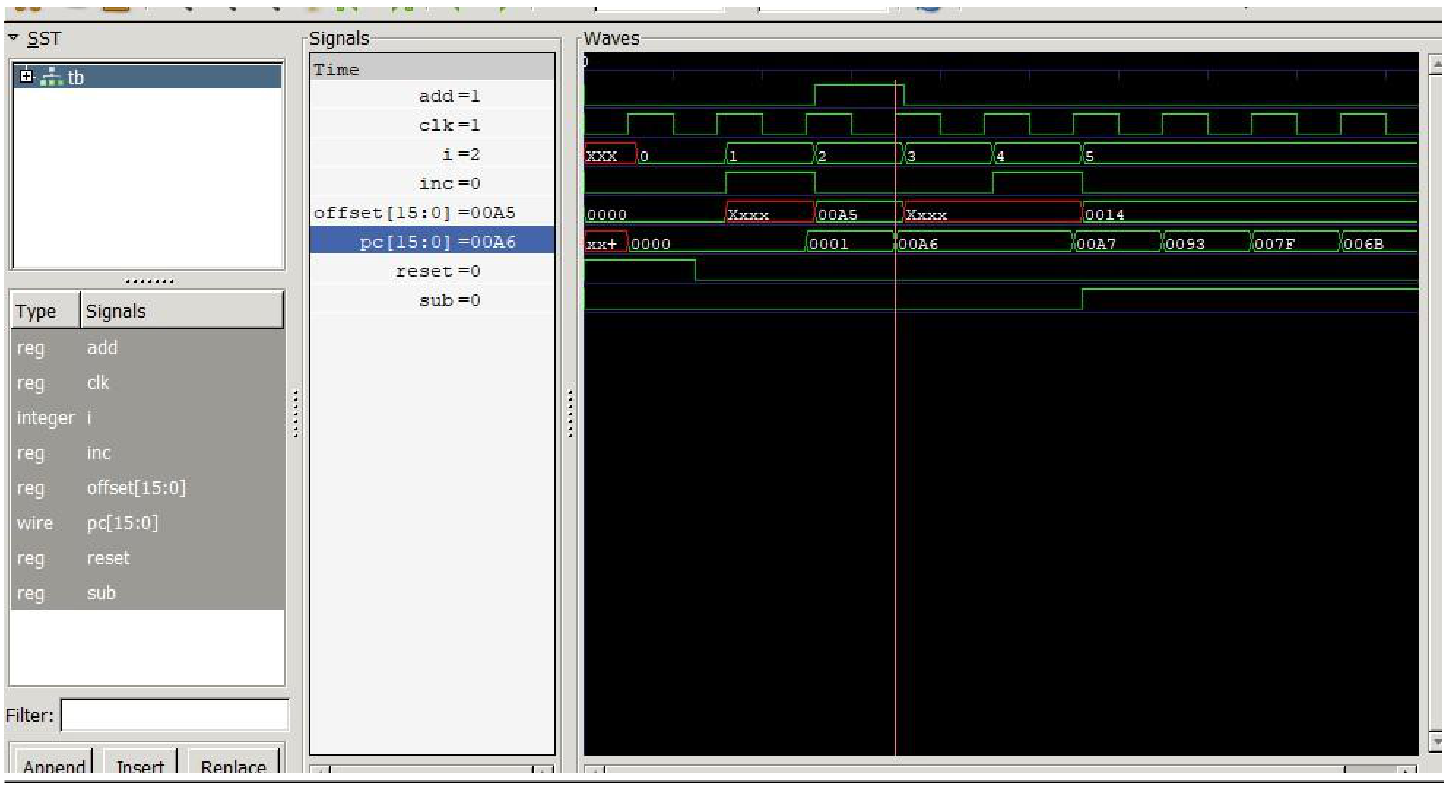
CASE1 :PC Increment Operation with no offset



```
Time
     add =0
     clk =1
       i =1
     inc =1
offset[15:0] =Xxxx
  pc[15:0] =0001
   reset =0
     sub =0
```

```
i:           XXX | 0 | 1 | 2 | 3 | 4 | 5
offset[15:0]: 0000 | Xxxx | 00A5 | Xxxx | 0014
pc[15:0]:    xx+ | 0000 | 0001 | 00A6 | 00A7 | 0093 | 007F | 006B
```

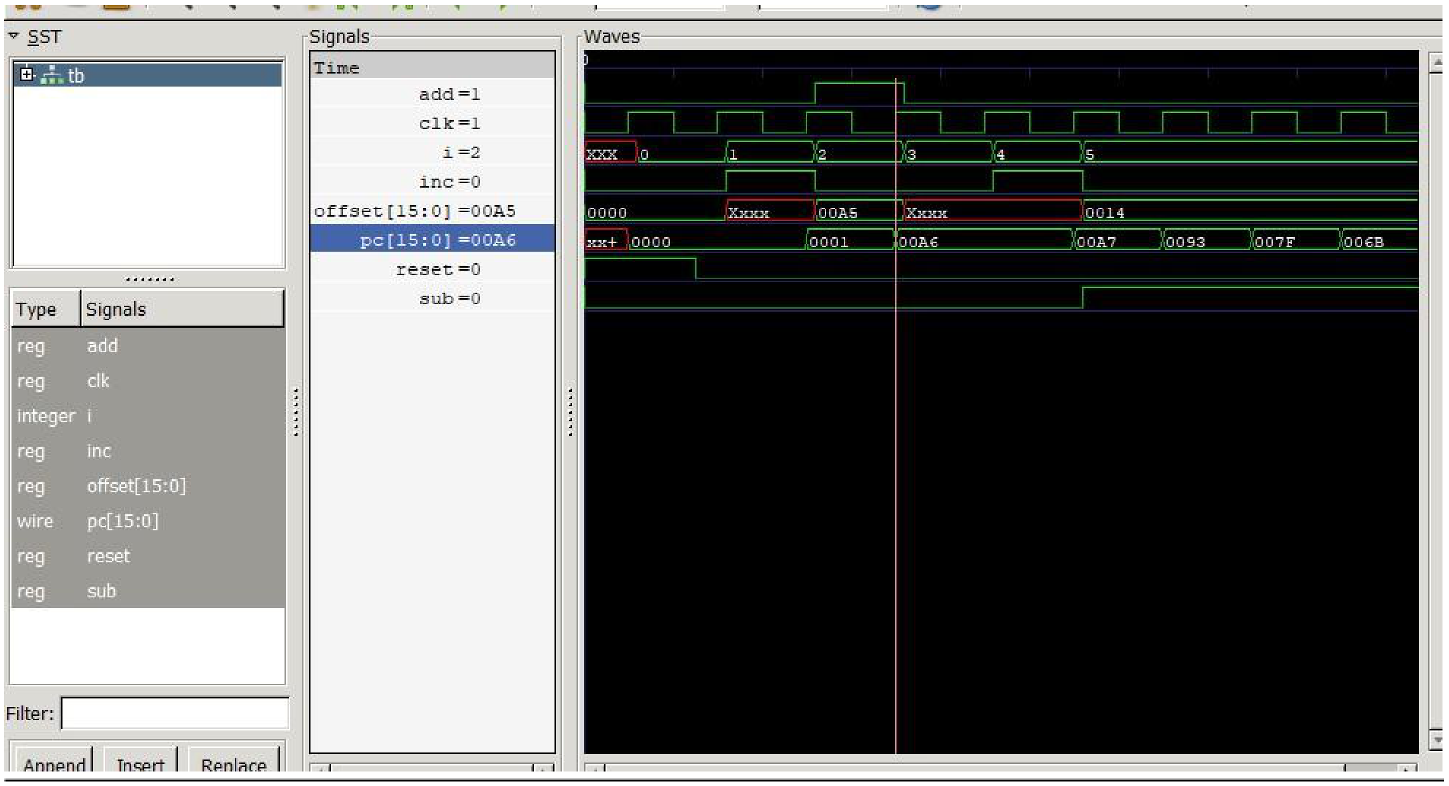
CASE 2 :Add Offset to PC



```
Time
     add =1
     clk =1
       i =2
     inc =0
offset[15:0] =00A5
  pc[15:0] =00A6
   reset =0
     sub =0
```

```
i:           XXX | 0 | 1 | 2 | 3 | 4 | 5
offset[15:0]: 0000 | Xxxx | 00A5 | Xxxx | 0014
pc[15:0]:    xx+ | 0000 | 0001 | 00A6 | 00A7 | 0093 | 007F | 006B
```

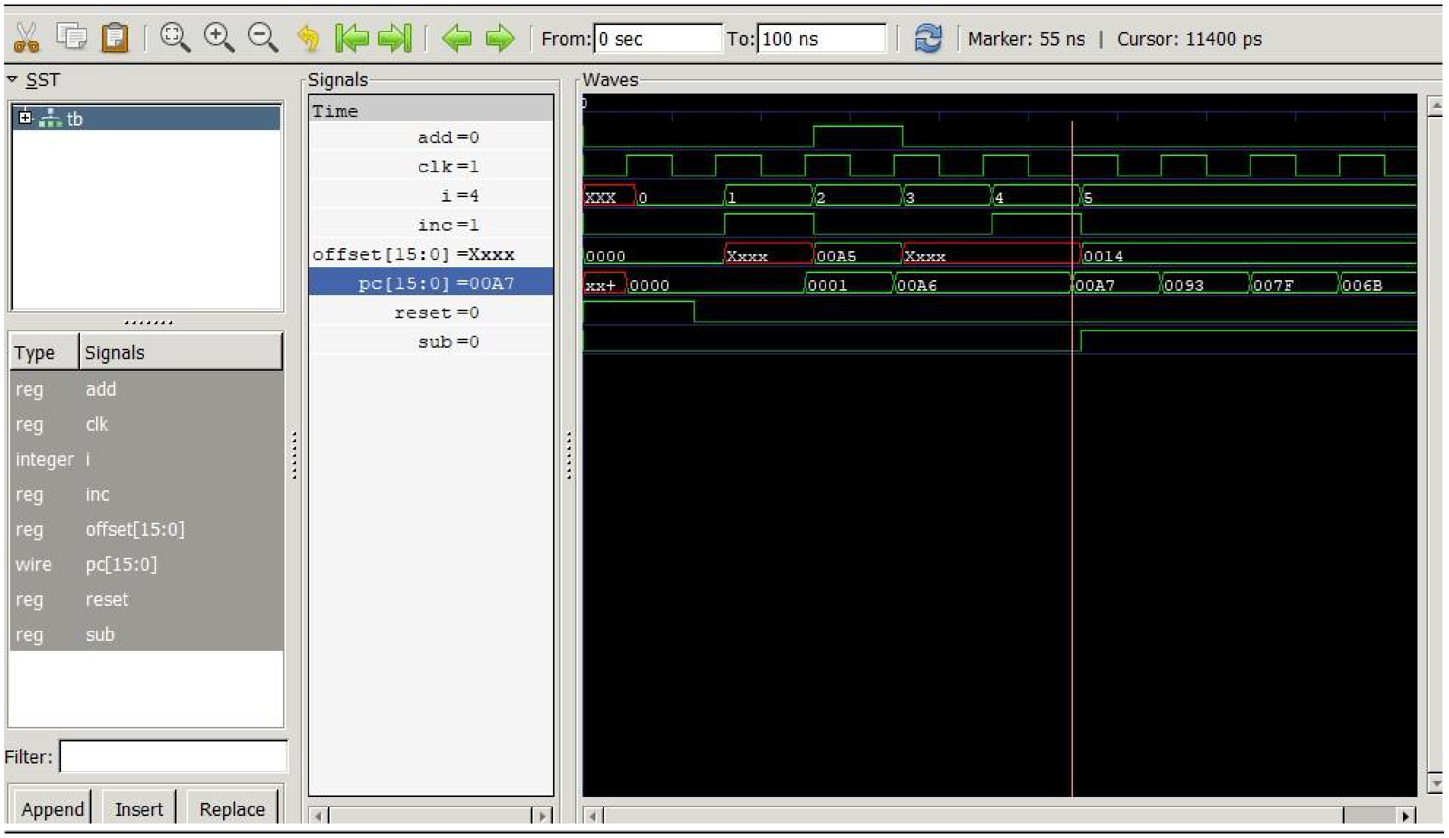
CASE 3 :No change in PC



Type	Signals
reg	add
reg	clk
integer	i
reg	inc
reg	offset[15:0]
wire	pc[15:0]
reg	reset
reg	sub

```
Time
          add =1
          clk =1
            i =2
          inc =0
offset[15:0] =00A5
  pc[15:0] =00A6
        reset =0
          sub =0
```

CASE 4 :Auto increment current value of PC



From: 0 sec To: 100 ns | Marker: 55 ns | Cursor: 11400 ps

Type	Signals
reg	add
reg	clk
integer	i
reg	inc
reg	offset[15:0]
wire	pc[15:0]
reg	reset
reg	sub

```
Time
          add =0
          clk =1
            i =4
          inc =1
offset[15:0] =Xxxx
  pc[15:0] =00A7
        reset =0
          sub =0
```

CASE 5 :Subtract offset contents from PC