# Digital Design and Computer Organization Laboratory

## UE23CS251A

## 3rd Semester, Academic Year 2024-25

Date:04/11/2024

| | | |
|---|---|---|
| Name : Keerthan P.V | SRN : PES2UG23CS272 | Section : E |

Experiment Number : 8 Week : 8

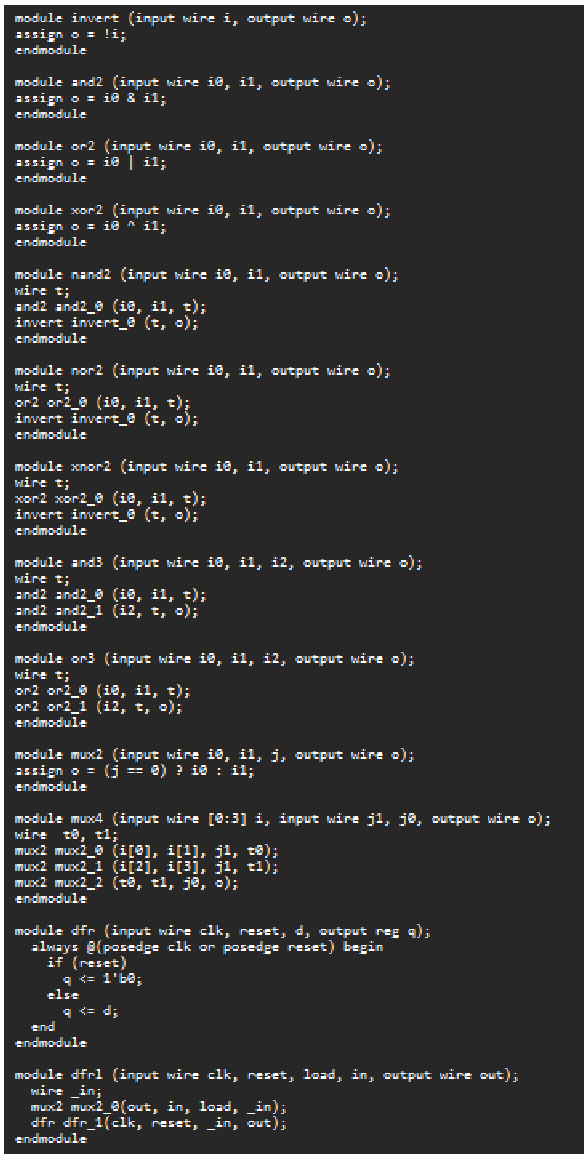
**Title of the Program:**

16 Bit Program Counter

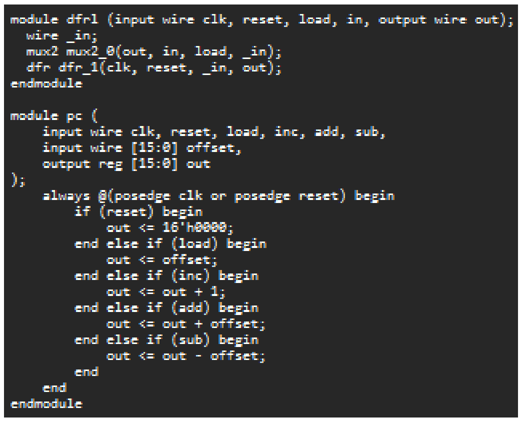
**Aim of the Program:**

To Design and Implementation of a 16 bit Program Counter

## Code (pc.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);
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 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 dfr (input wire clk, reset, d, output reg q);
  always @(posedge clk or posedge reset) begin
    if (reset)
      q <= 1'b0;
    else
      q <= d;
  end
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
```



```
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
```

```
module pc (
    input wire clk, reset, load, inc, add, sub,
    input wire [15:0] offset,
    output reg [15:0] out
);
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            out <= 16'h0000;
        end else if (load) begin
            out <= offset;
        end else if (inc) begin
            out <= out + 1;
        end else if (add) begin
            out <= out + offset;
        end else if (sub) begin
            out <= out - offset;
        end
    end
endmodule
```

## Testbench (pc_tb.v):

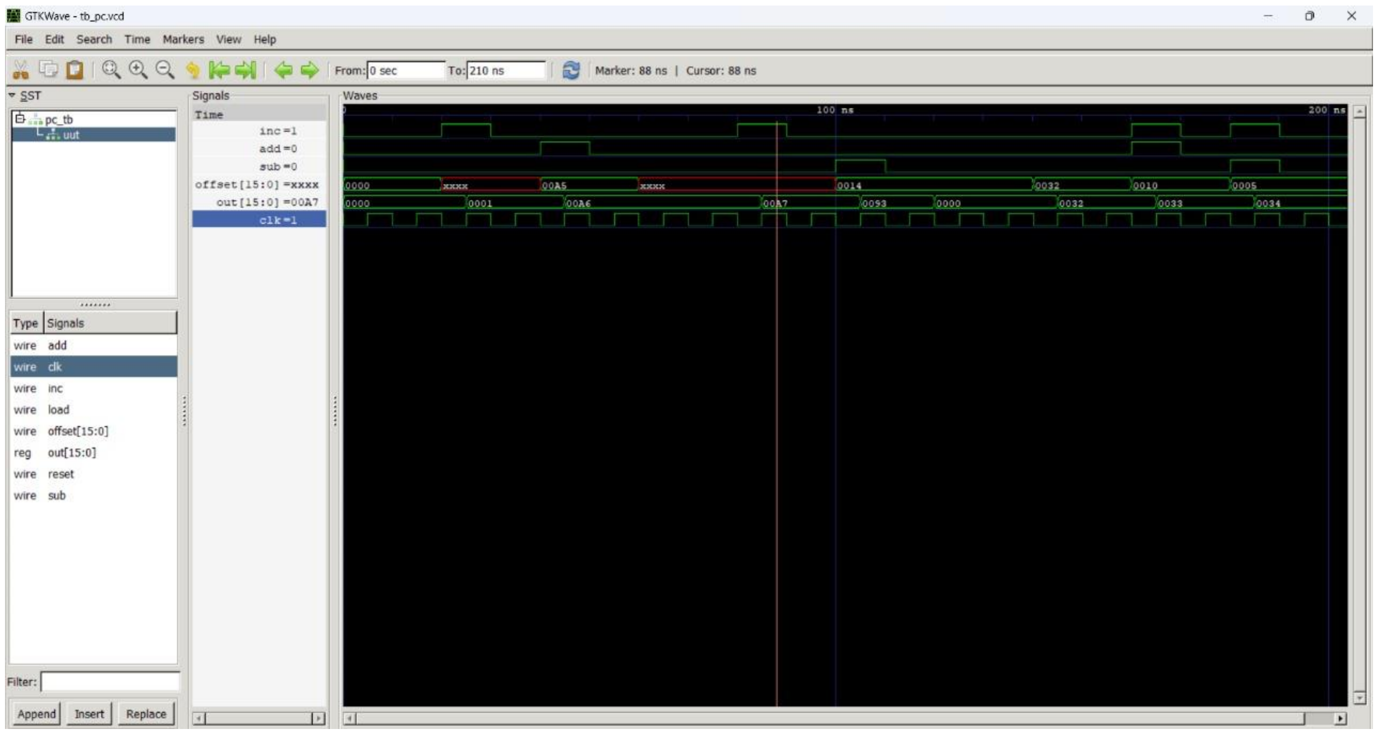
```verilog
`timescale 1ns/1ps

module pc_tb;

  reg clk, reset, load, inc, add, sub;
  reg [15:0] offset;
  wire [15:0] pc_out;

  pc uut (
      .clk(clk),
      .reset(reset),
      .load(load),
      .inc(inc),
      .add(add),
      .sub(sub),
      .offset(offset),
      .out(pc_out)
  );

  initial begin
      clk = 0;
      forever #5 clk = ~clk;
  end

  initial begin
      $dumpfile("tb_pc.vcd");
      $dumpvars(0, pc_tb);

      reset = 1; load = 0; inc = 0; add = 0; sub = 0; offset = 16'h0000;
      #10 reset = 0;

      #10 inc = 1; add = 0; sub = 0; offset = 16'hXXXX;
      #10 inc = 0;

      #10 inc = 0; add = 1; sub = 0; offset = 16'h00A5;
      #10 add = 0;

      #10 inc = 0; add = 0; sub = 0; offset = 16'hXXXX;
      #10;

      #10 inc = 1; add = 0; sub = 0; offset = 16'hXXXX;
      #10 inc = 0;

      #10 inc = 0; add = 0; sub = 1; offset = 16'h0014;
      #10 sub = 0;

      #10 reset = 1;
      #10 reset = 0;

      #10 load = 1; offset = 16'h0032;
      #10 load = 0;

      #10 inc = 1; add = 1; sub = 0; offset = 16'h0010;
      #10 inc = 0; add = 0;

      #10 inc = 1; add = 0; sub = 1; offset = 16'h0005;
      #10 inc = 0; sub = 0;

      #20 $finish;
  end

  initial begin
    $monitor("Time = %t | PC = %h | inc = %b, add = %b, sub = %b, load = %b, reset = %b, offset = %h",
             $time, pc_out, inc, add, sub, load, reset, offset);
  end

endmodule
```

**TABLE:**

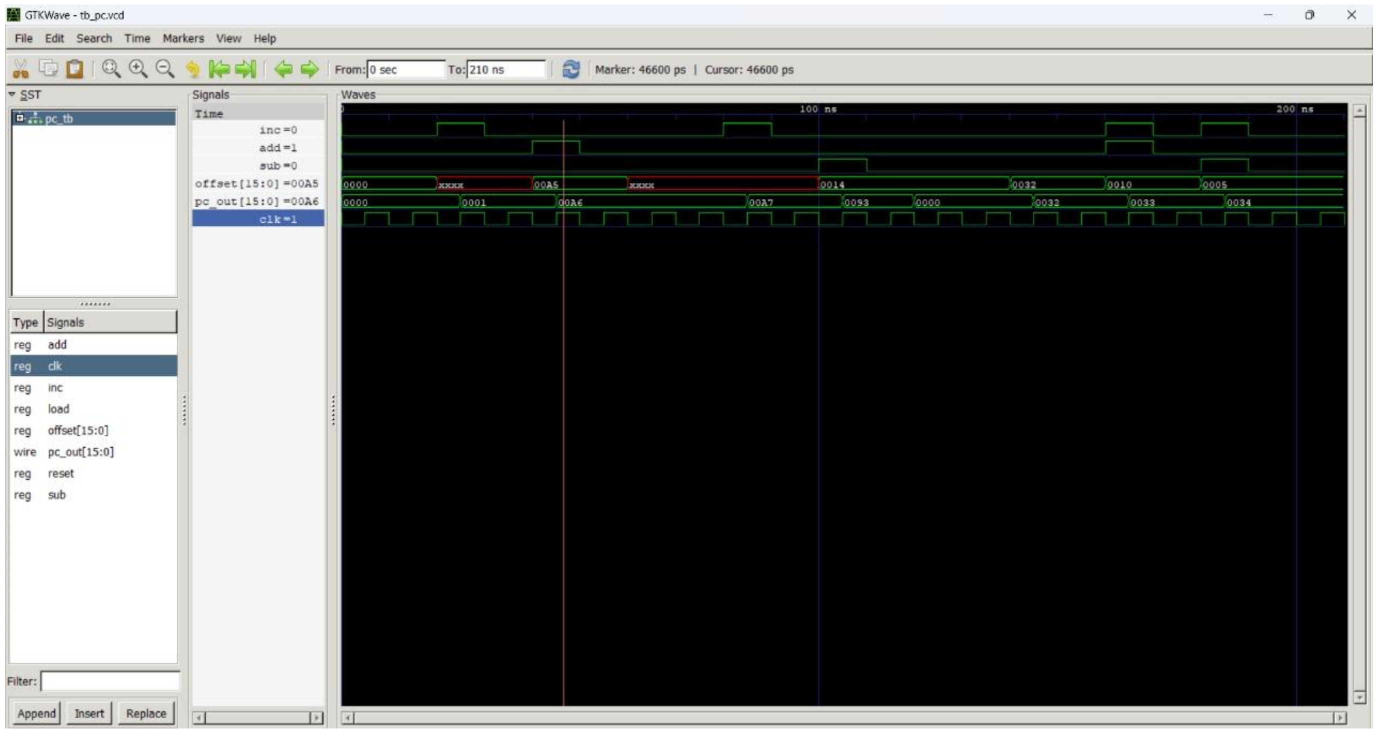
| | Inc | add | sub | offset [15:0] | output |
|---|---|---|---|---|---|
| | Bit 18 | Bit 17 | Bit 16 | Bit 15 to Bit0 | pc[15:0] |
| CASE 1 | 1 | 0 | 0 | XXXX | 0001 |
| CASE 2 | 0 | 1 | 0 | 00A5 | 00A6 |
| CASE 3 | 0 | 0 | 0 | XXXX | 00A6 |
| CASE 4 | 1 | 0 | 0 | XXXX | 00A7 |
| CASE 5 | 0 | 0 | 1 | 0014 | 0093 |

## Output waveform

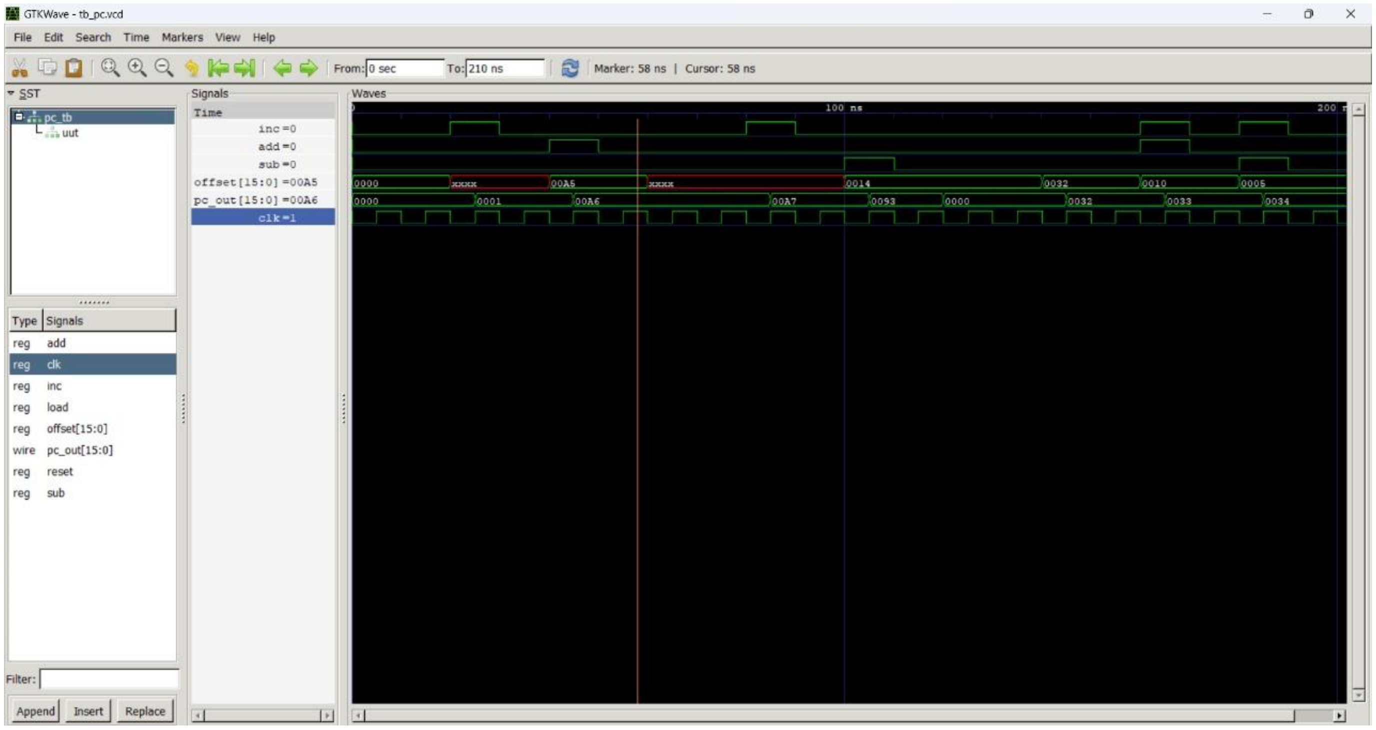
## CASE1 :PC Increment Operation with no offset



## CASE 2 :Add Offset to PC



## CASE 3 :No change in PC

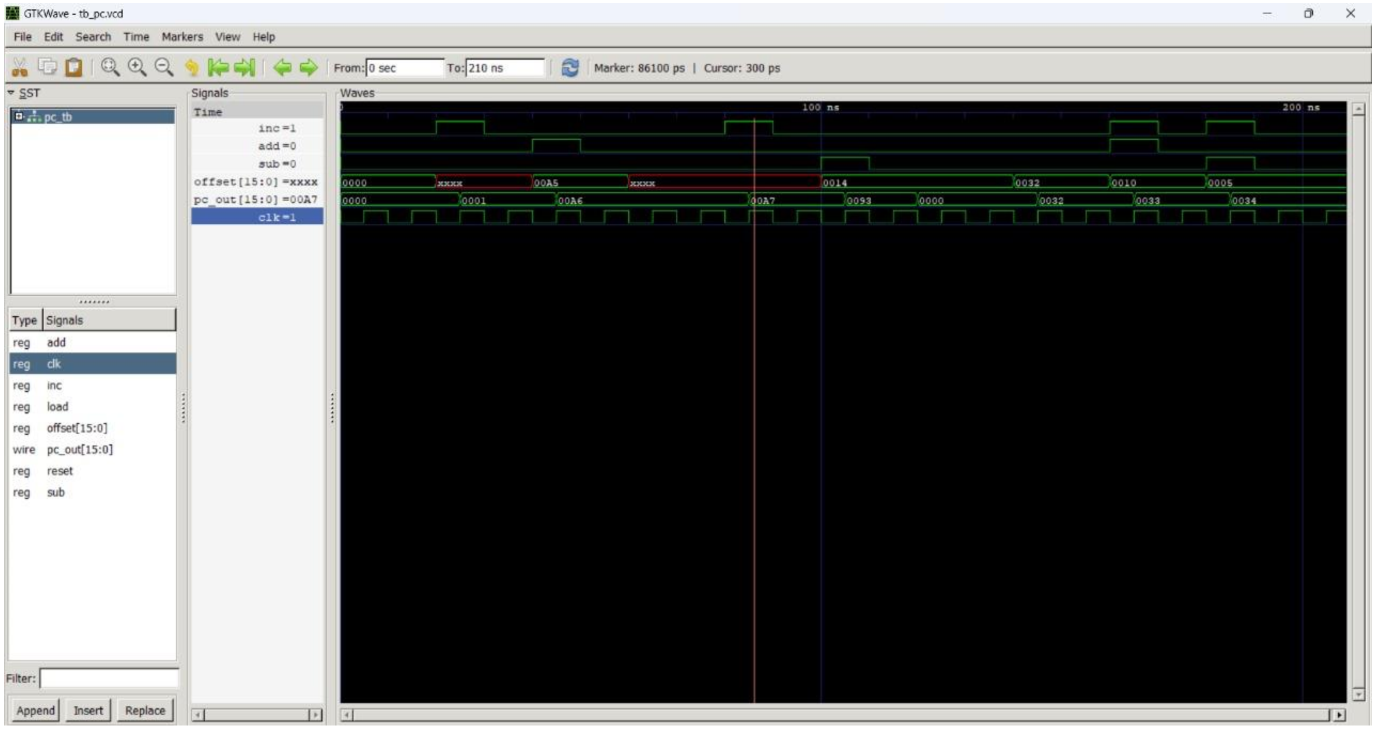


```
Time
inc=0
add=0
sub=0
offset[15:0]=00A5
pc_out[15:0]=00A6
clk=1
```

```
offset: 0000 xxxx 00A5 xxxx 0014 0032 0010 0005
pc_out: 0000 0001 00A6 00A7 0093 0000 0032 0033 0034
```

| Type | Signals |
|---|---|
| reg | add |
| reg | clk |
| reg | inc |
| reg | load |
| reg | offset[15:0] |
| wire | pc_out[15:0] |
| reg | reset |
| reg | sub |

## CASE 4 :Auto increment current value of PC

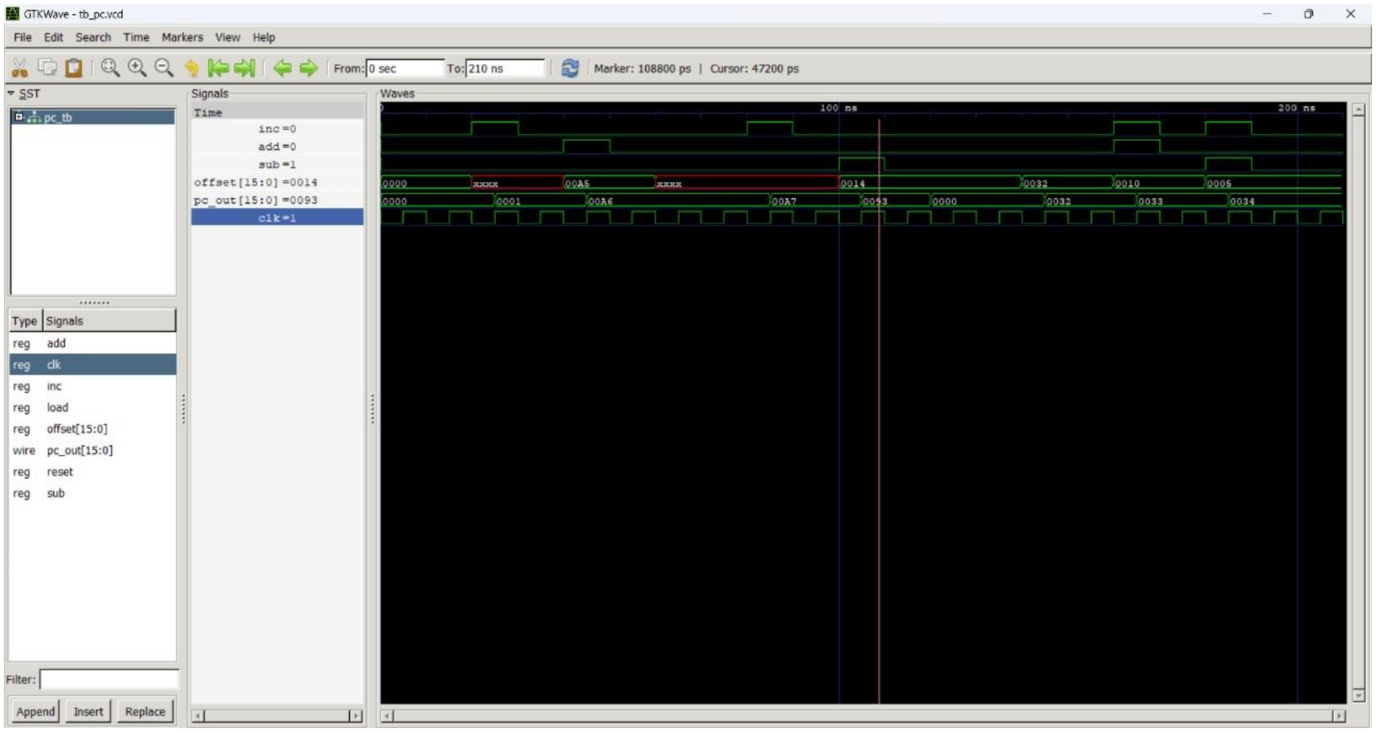


```
Time
inc=1
add=0
sub=0
offset[15:0]=xxxx
pc_out[15:0]=00A7
clk=1
```

```
offset: 0000 xxxx 00A5 xxxx 0014 0032 0010 0005
pc_out: 0000 0001 00A6 00A7 0093 0000 0032 0033 0034
```

| Type | Signals |
|---|---|
| reg | add |
| reg | clk |
| reg | inc |
| reg | load |
| reg | offset[15:0] |
| wire | pc_out[15:0] |
| reg | reset |
| reg | sub |

## CASE 5 :Subtract offset contents from PC



From: 0 sec To: 210 ns Marker: 108800 ps | Cursor: 47200 ps

| Type | Signals |
|---|---|
| reg | add |
| reg | clk |
| reg | inc |
| reg | load |
| reg | offset[15:0] |
| wire | pc_out[15:0] |
| reg | reset |
| reg | sub |

```
Time
inc =0
add =0
sub =1
offset[15:0] =0014    0000  xxxx  00A5  xxxx  0014  0032  0010  0005
pc_out[15:0] =0093    0000  0001  00A6  00A7  0093  0000  0032  0033  0034
clk =1
```