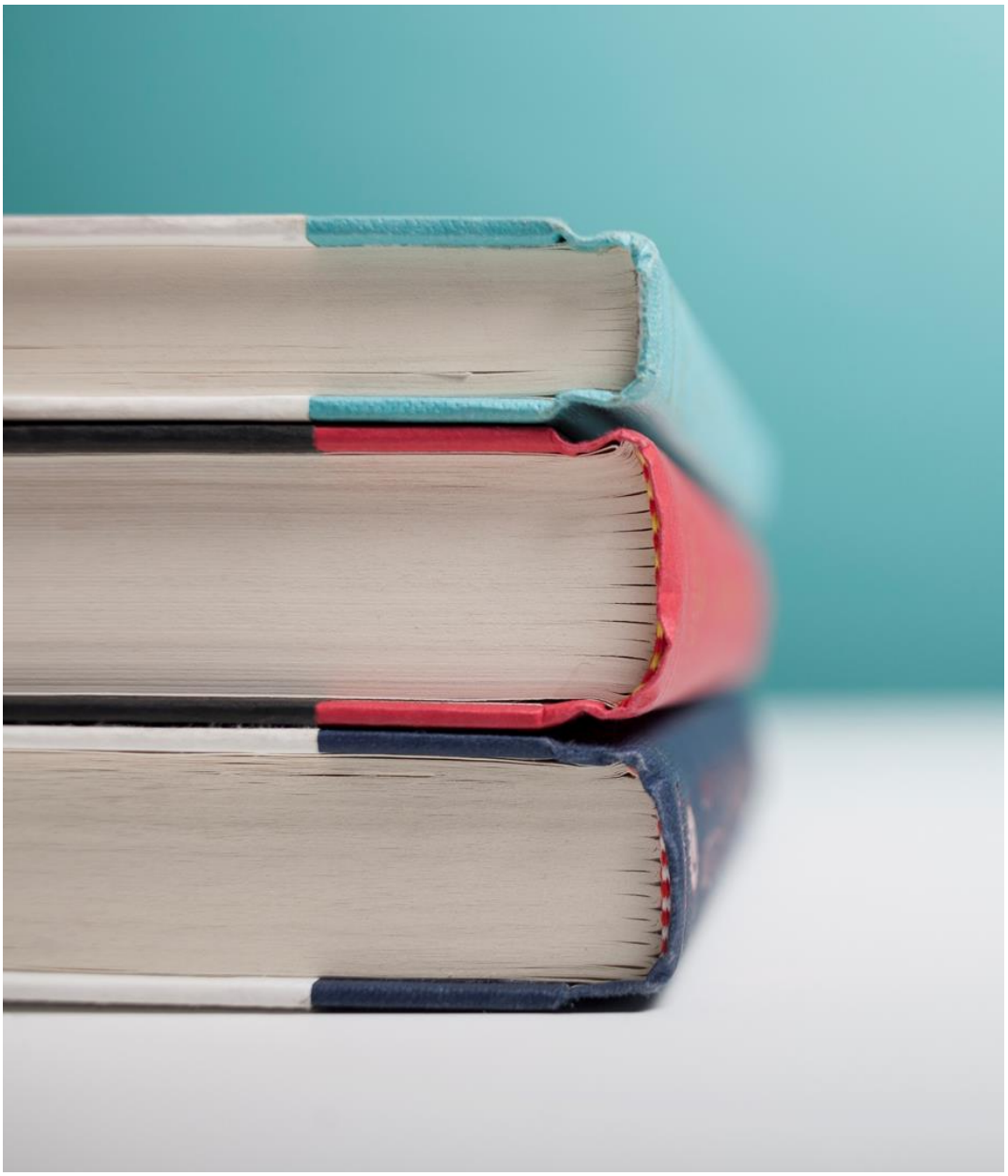


Verilog Project

Yaqout Hmaid

Dr. Abdalkarim Awad

Section 2

- **what does the attached Verilog file do ?**

Its a simple computer code has 1 input (clock) and 7 ouput (pc ,ir ,mbr , ac ,mar,outd0,outd1) , then define some instructions and give them a special opcodes for the instructions.

The atttached code call the load operation which is load the value of address 10 (hex = a) then load the address 11 (hex=b) and add it to the ac value then store the result in output device (outp0) . In other way the code add two numbers together and give you the output in hexadecimal.

In this example : 4+8 =c .

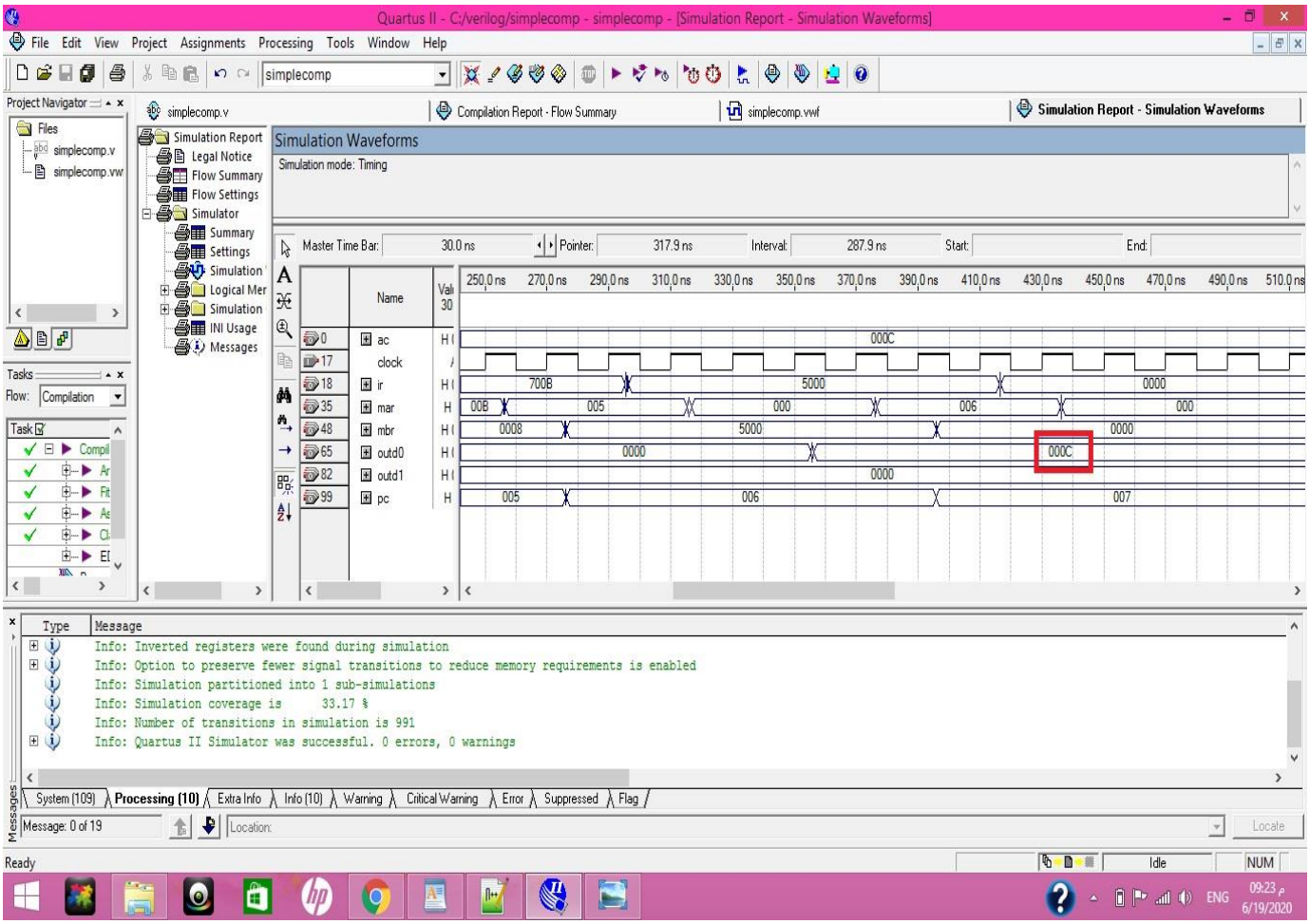
The code :

```
module simplecomp(clock,pc,ir,mbr,ac,mar,outd0, outd1);//define the input input clock;//define the output, PC:program counter, //IR:instruction register, MBR:memory buffer register, AC:accumulator, //MAR:memory address register, outdx: output devicesoutput pc,ir,mbr,ac,mar,outd0,outd1; //define the registersreg [15:0] ir,mbr,ac,obr,outd0, outd1;reg [11:0] pc,mar;reg [15:0] memory [0:63];//memory of 64 words, each word is 2 bytesreg [2:0] state;//provide opcodes for the instructionsparameter load = 4'b0011,store=4'b1011,add=4'b0111,jump=4'b0001,sub=4'b00100,outp0=4'b0101, outp1=4'b1001;initial begin //instruction memorymemory[3]=16'h300a; //load location 10 from memory to ACmemory[4]=16'h700b; //add AC with location 11 from memory and store the result in ACmemory[5]=16'h5000; //store the result on an output device//data memorymemory[10]=16'd4; //store number 4 in location 9 in memorymemory[11]=16'd8; //store number 8 in location 10 in memorypc=3; //start with PC = 3, so first instruction will be the one that in location 7 (16'h300a)state=0;//state in indicate to each state : instruction fetch, instruction decode, operand fetch, excution end
```

always @ (posedge clock) begin // loop one time for each clock rising edgecase (state)0:begin //initialization of MAR to get the instruction from memory mar <= pc; //store PC to MAR because when we want to access the memory, we should store the address in MAR state=1; //go to next step next clock end1:begin//instruction fetch mbr <= memory[mar]; //store the instuction in IR pc <= pc+1; //go one instruction a head state=2; end2:begin//instruction fetch ir <= mbr; //store the instuction in IR state=3; end3:begin //instruction decode mar <= ir[11:0];//store the address in MAR to access the memory in the next step in case the instruction is Load or Store state=4; end4:begin //operand fetch state=5; case (ir[15:12]) // fill the value of MBR with the correct pattern in each operation type load : mbr <= memory[mar]; add : mbr <= memory[mar]; sub : mbr <= memory[mar]; store: mbr <= ac; jump : mbr <= mar; outp0: obr <=ac; outp1: obr <=ac; endcase end 5:begin //excution if(ir[15:12]==4'h7)begin //addition ac <= ac+mbr; state=0; end else if(ir[15:12]==4'h4)begin //sub ac <= ac-mbr; state=0; end else if(ir[15:12]==4'h3)begin //load ac <= mbr; state=0; end else if(ir[15:12]==4'hb)begin //store memory[mar] <= mbr; state=0; end else if(ir[15:12]==4'h1)begin //jump pc <= mbr; state=0; end else if(ir[15:12]==4'h5)begin //outp0 outd0 <= obr; state=0; end else if(ir[15:12]==4'h9)begin //outp1 outd1 <= obr; state=0; end endendcaseendendmodule

- **The simulation of the attached file :**

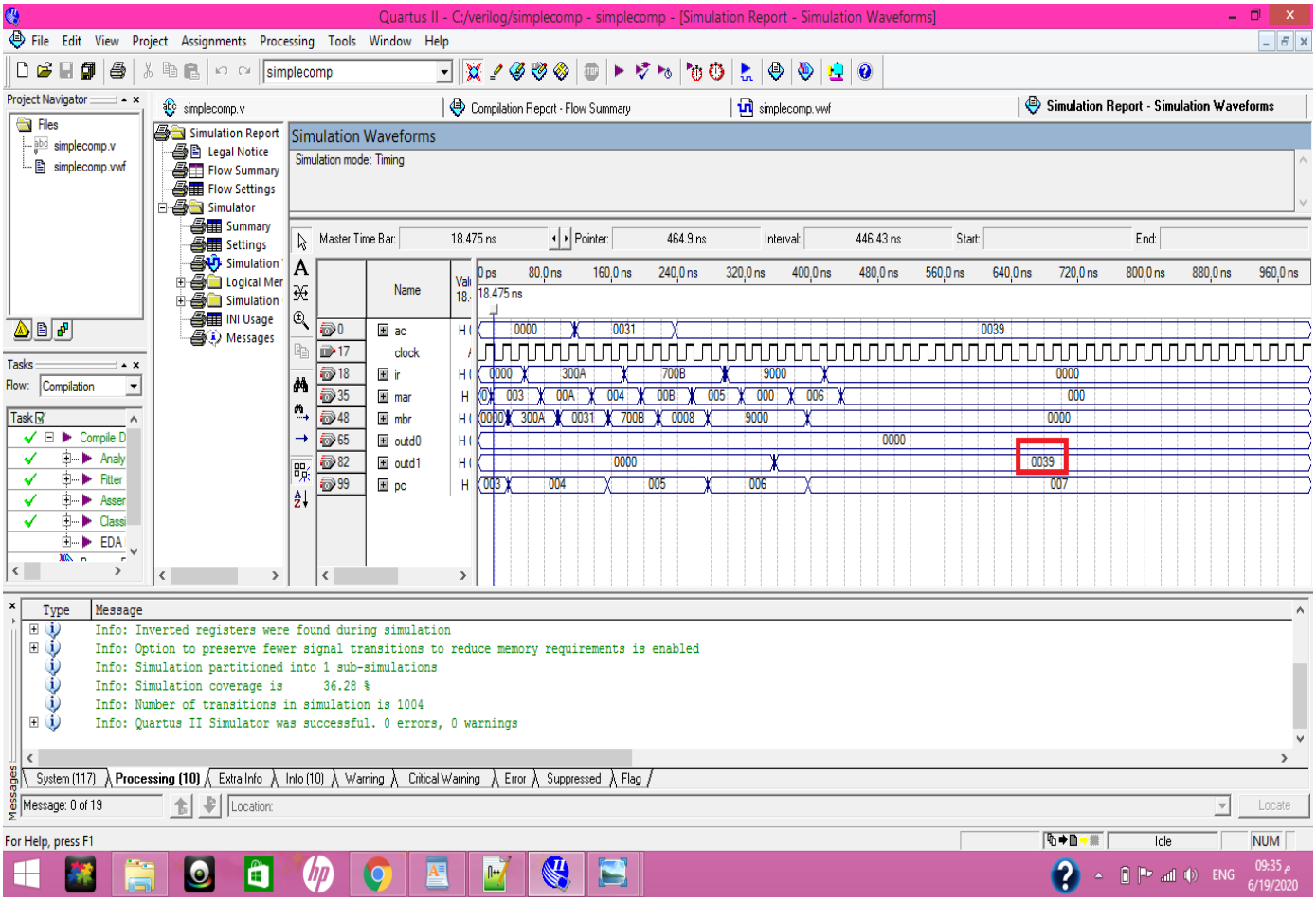
Every rising edge theres a clock first load the a address value to ac then add b address value put the result in outp0



- **Add two numbers (from ID) and show the result on the OUTD1**

My ID number is 117031 so the first two digits from right are 31 and the second two digits from right are 08 which is 8 and i put the result in outD1

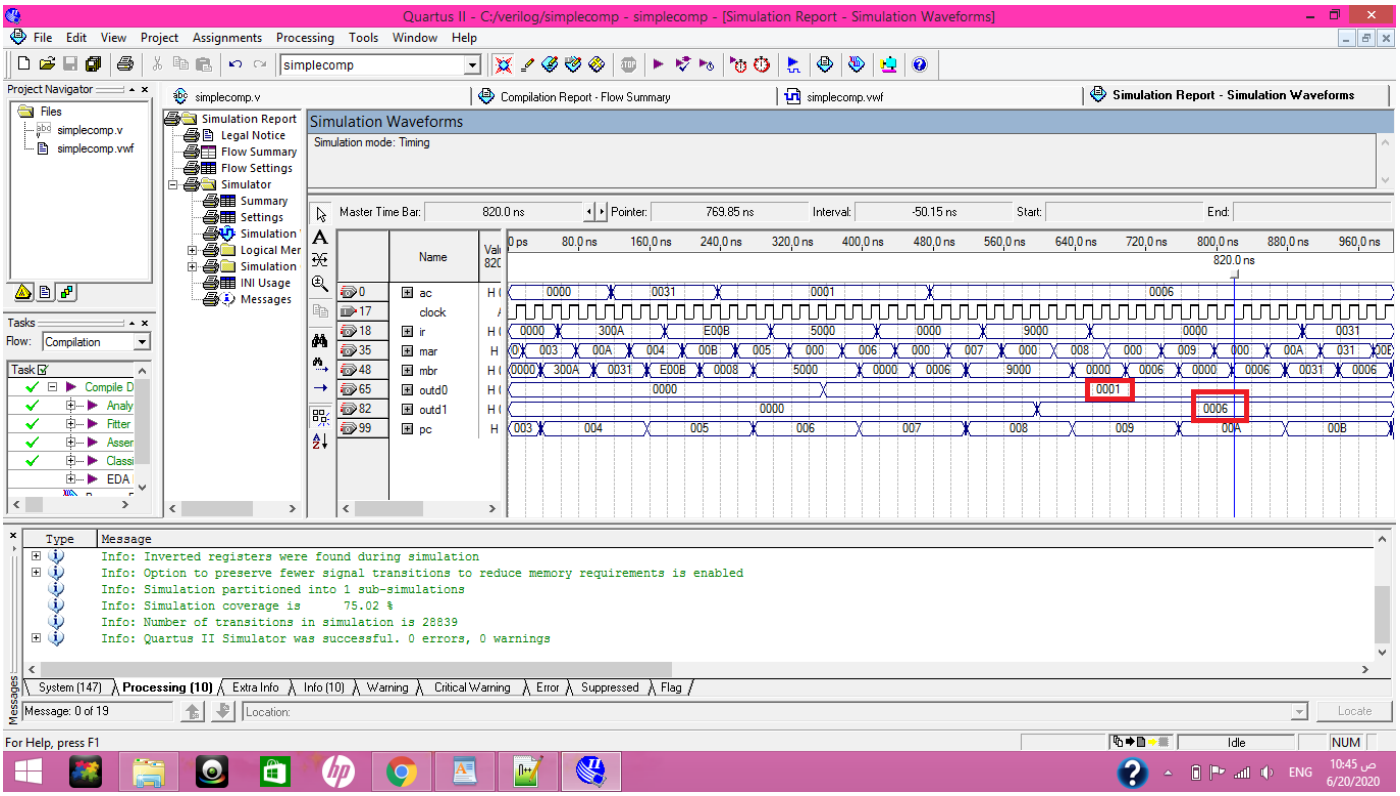
8 = 39 in hexa + 31 So the addition of them is



- **The division operation (ID numbers):**

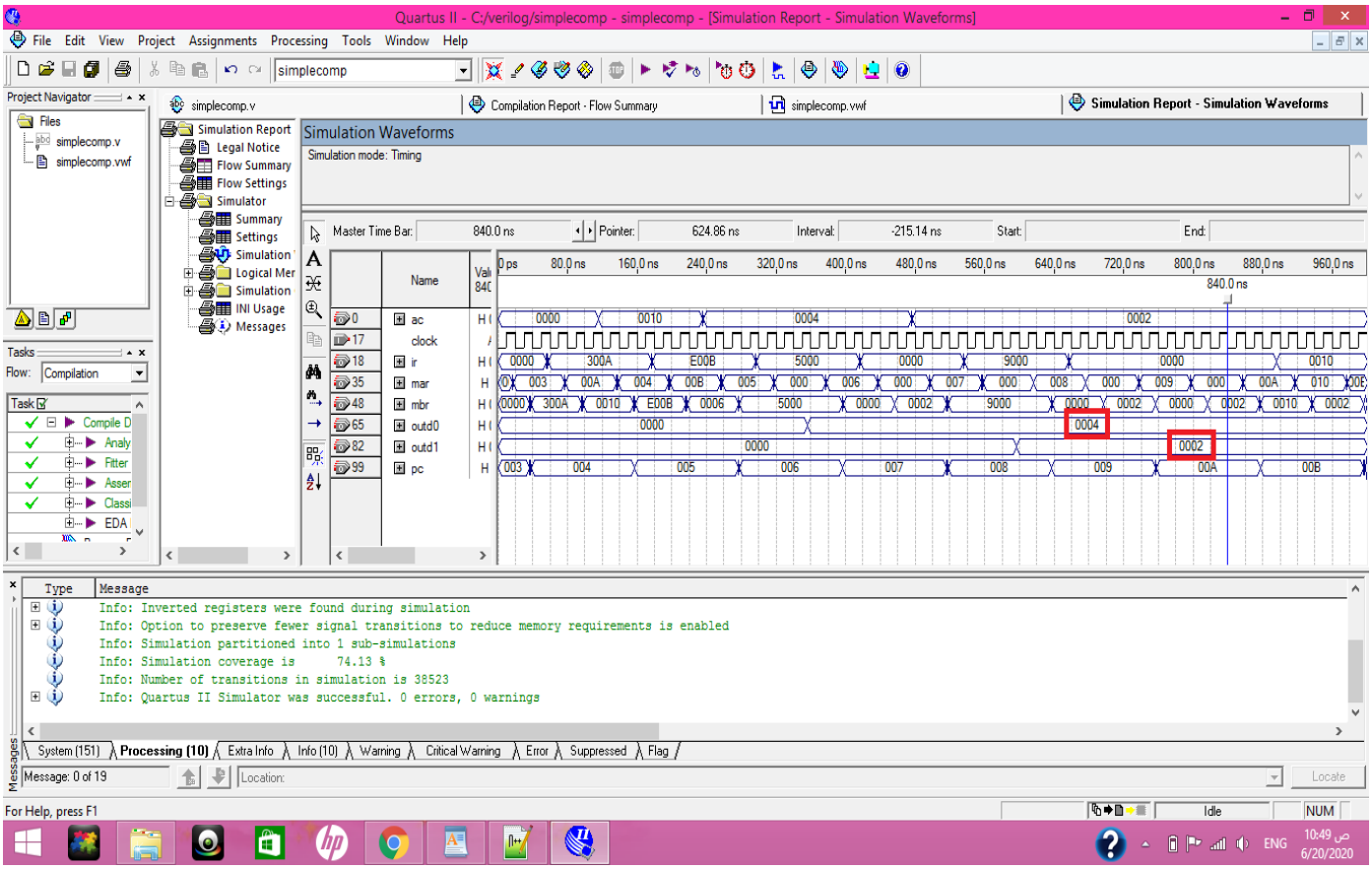
First num = 31 , the second num = 8 i put the quotient in outd0 = 6 and the remainder in outd1 = 1

First it load the value of address a then put it in ac then dived it by the value of address b .



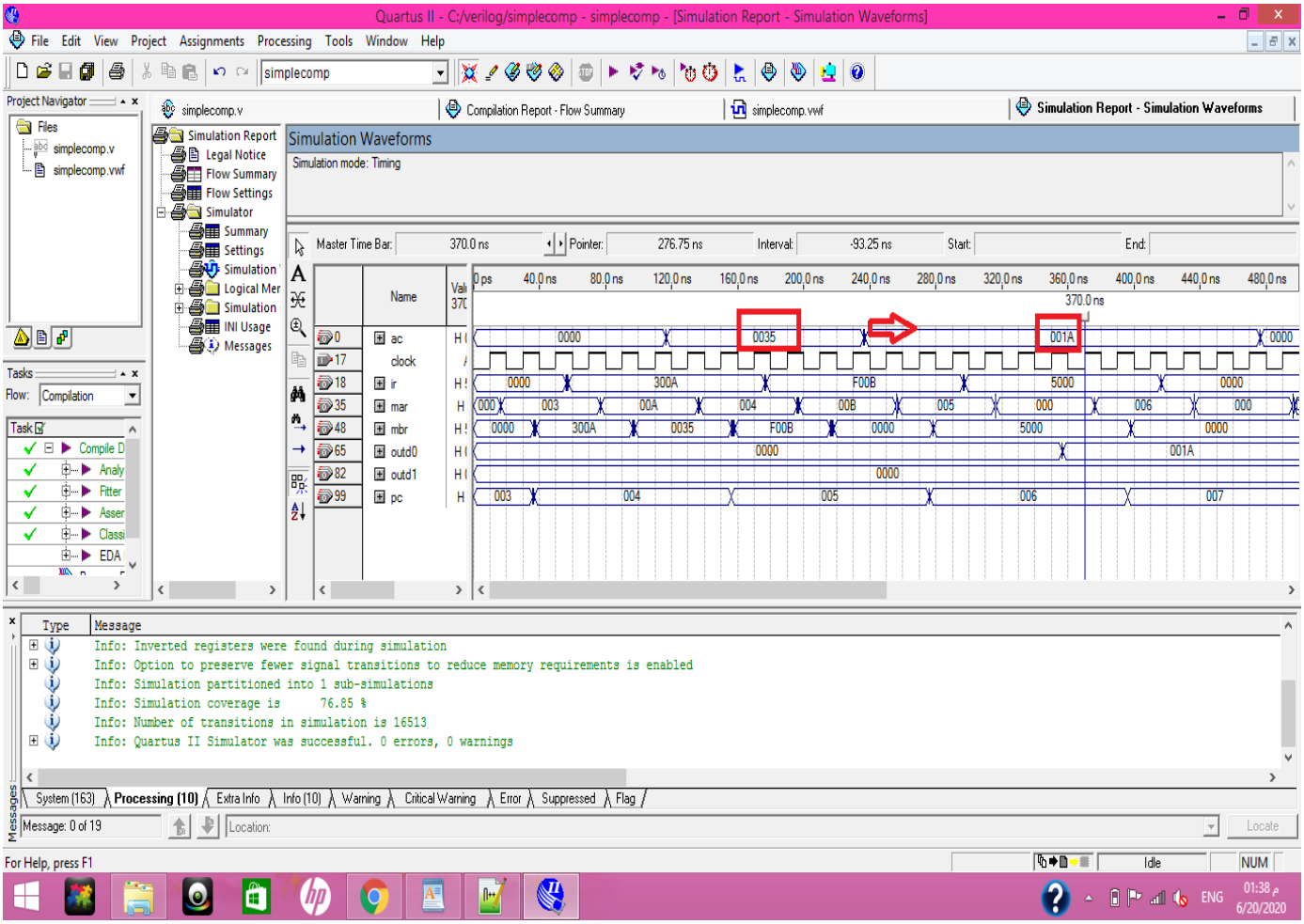
- **The division operation (any data):**

The first # = 10 , the second # = 6 put the result in outd1=2 , and the riminder in outd0= 4



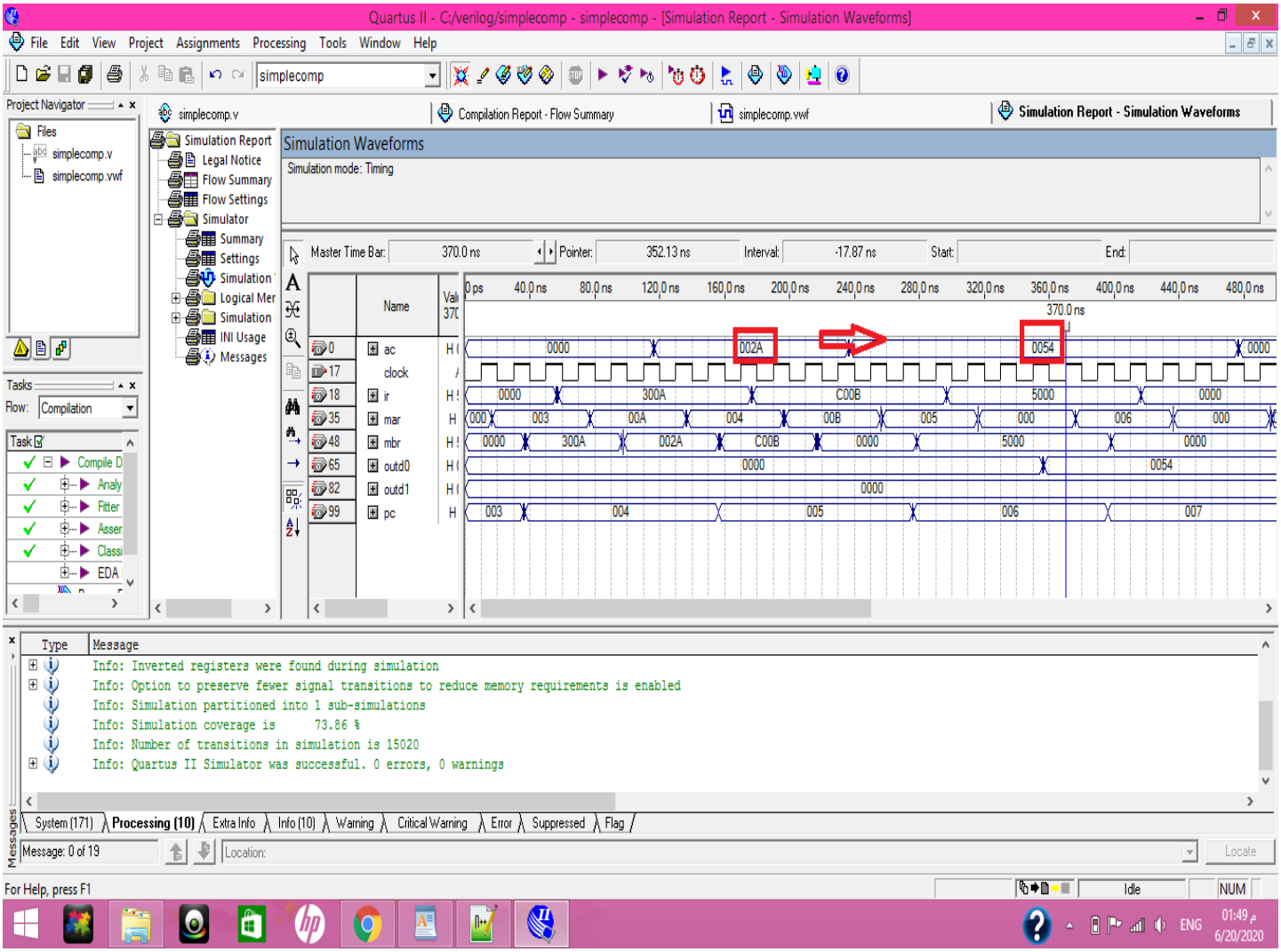
- **Right shift :**

shift the content of AC by one bit to the right and replace it by 0 , first we load the vaule of address a in the ac and then we use the shift right operation >> the number 35 in binary = 110101 so it become 011010 which is 1A



- **Left shift**

shift the content of AC by one bit to the left and replace it by 0 , first we load the vaule of address a in the ac and then we use the shift left operation << the number 2A in binary = 101010 so it become 1010100 which is 54



THE FINAL CODE :

```
module simplecomp(clock,pc,ir,mbr,ac,mar,outd0, outd1);
//define the input
input clock;
//define the output, PC:program counter,
//IR:instruction register, MBR:memory buffer register, AC:accumulator,
//MAR:memory address register, outdx: output devices
output pc,ir,mbr,ac,mar,outd0,outd1;
//define the registers
reg [15:0] ir,mbr,ac,obr,outd0, outd1;   // size 16 bit
reg [11:0] pc,mar; // size 12 bit
reg [15:0] memory [0:63];//memory of 64 words, each word is 2 bytes
reg [2:0] state;
reg [8:0] R0; // r0 reg

//provide opcodes for the instructions
parameter load = 4'b0011,store=4'b1011,add=4'b0111,jump=4'b0001,sub=4'b00100,outp0=4'b0101, outp1=4'b1001 ,
div=4'b1110,loadr0=4'b0000, shr=4'b1111, shl=4'b1100;
```

```
initial begin
//instruction memory
memory[3]=16'h300a; //load location 10 from memory to AC
memory[4]=16'hE00b; //div AC with location 11 from memory and store the riminder in AC
memory[5]=16'h5000; //outp0
memory[6]=16'h0000; //load r0
memory[7]=16'h9000; // outp1


//data memory
memory[10]=16'h2a; //store number 4 in location 9 in memory
memory[11]=16'h6; //store number 8 in location 10 in memory

pc=3; //start with PC = 3, so first instruction will be the one that in location 7 (16'h300a)
state=0;//state in indicate to each state : instruction fetch, instruction decode, operand fetch, excution
end

always @ (posedge clock) begin // loop one time for each clock rising edge
case (state)
```

```
0:begin //initialization of MAR to get the instruction from memory
  mar <= pc; //store PC to MAR because when we want to access the memory, we should store the address in MAR
  state=1; //go to next step next clock
  end
1:begin//instruction fetch
  mbr <= memory[mar]; //store the instuction in IR
  pc <= pc+1; //go one instruction a head
  state=2;
  end
2:begin//instruction fetch
  ir <= mbr; //store the instuction in IR
  state=3;
  end
3:begin //instruction decode
  mar <= ir[11:0];//store the address in MAR to access the memory in the next step in case the instruction is Load or Store
  state=4;
  end
4:begin //operand fetch
  state=5;
  case (ir[15:12]) // fill the value of MBR with the correct pattern in each operation type
```

```
   load : mbr <= memory[mar];
   add  : mbr <= memory[mar];
   sub  : mbr <= memory[mar];
   store: mbr <= ac;
   jump : mbr <= mar;
   outp0: obr <=ac;
   outp1: obr <=ac;
   div  : mbr <= memory[mar]; // the value of mar in the mbr
   loadr0  : mbr <= R0;      // put the r0 in mbr
   shr  : mbr <= memory[mar]; // the value of mar in the mbr
   shl  : mbr <= memory[mar]; // the value of mar in the mbr


  endcase
  end
5:begin //excution
  if(ir[15:12]==4'h7)begin //addition
    ac <= ac+mbr;
    state=0;
  end
  else if(ir[15:12]==4'h4)begin //sub
    ac <= ac-mbr;
    state=0;
  end
   else if(ir[15:12]==4'hE)begin //div
```

```
   ac <= ac%mbr;              // divide the value of ac by mbr value
and put the riminder in ac
   R0 <= ac/mbr;              // divide the value of ac by mbr value
and put the result in R0

   state=0;
  end
  else if(ir[15:12]==4'h3)begin //load
   ac <= mbr;
   state=0;
  end
  else if(ir[15:12]==4'hb)begin //store
   memory[mar] <= mbr;
   state=0;
  end
  else if(ir[15:12]==4'h1)begin //jump
   pc <= mbr;
   state=0;
  end
   else if(ir[15:12]==4'h5)begin //outp0
   outd0 <= obr;
   state=0;
  end
   else if(ir[15:12]==4'h9)begin //outp1
   outd1 <= obr;
```

```
   state=0;

  end
   else if(ir[15:12]==4'h0)begin //LOAD R0
   ac <= mbr;
   state=0;
  end
   else if(ir[15:12]==4'hf)begin //shr
   ac <= ac>>1;              // shift the ac value one bit to the left
   state=0;
  end
  else if(ir[15:12]==4'hc)begin //shl
   ac <= ac<<1;              // shift the ac value one bit to the left

      state=0;
  end
end
endcase
end
endmodule
```