**Summer Intership Project 2:**

**SAYEH PROCESSOR**

Name of the students with ID:

1. N190376 - L. Uma
2. N190753 - M. John David

**Introduction:**

The CPU is simple architecture, yet enough hardware (SAYEH) that has been designed for educational and benchmarking purposes. For a better understanding of the material presented here, the reader is expected to have a general understanding of computer architectures.

**Details of processor functionality:**

The simple CPU example discussed here has a register file that is used for data processing instructions. The CPU has a 16-bit data bus and a 16-bit address bus. The processor has 8 and 16-bit instructions.

A diagram of a data flow

Description automatically generated

**Fig\_1.**  SAYEH Interface Signals

**CPU components:**

SAYEH uses its register file for most of its data instructions. Addressing modes of this processor also take advantage of this structure. Because of this, the addressing hardware of SAYEH is a simple one and the register file output is used in address calculations.

SAYEH components that are used by its instructions include the standard registers such as the Program Counter, Instruction Register, the Arithmetic Logic Unit, and Status Register. In addition, this processor has a register file forming registers R0, R1, R2, and R3 as well as a Window Pointer that defines R0, R1, R2, and R3 within the register file. CPU components and a brief description of each are shown below.

**PC.** Program Counter, 16 bits

**R0, R1, R2,** **and** **R3.** General purpose registers part of the register file, 16 bits

**Reg file.** The general-purpose registers form a window of 4 in a register file of 8 registers.

**WP.** Window Pointer points to the register file to define R0, R1, R2, and R3, 3 bits.

**IR.** Instruction Register that is loaded with a 16-bit, an 8-bit, or two 8-bit instructions, 16 bits.

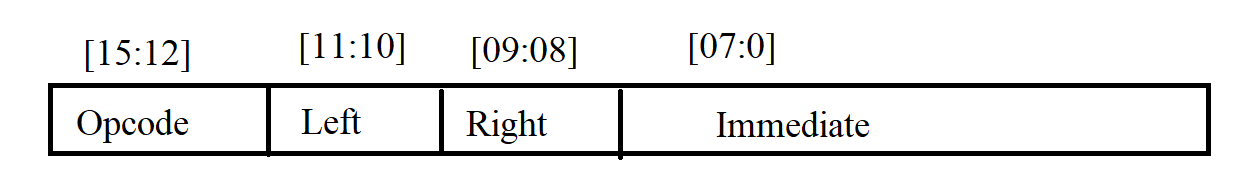
**ALU.** The ALU that can AND, OR, NOT, Shift, Compare, Add, Subtract, and Multiply its inputs, 16-bit operands.

**Z flag.** Becomes 1 when the ALU output is 0.

**C flag.** Becomes 1 when the ALU has a carry output.

**SAYEH instructions:**

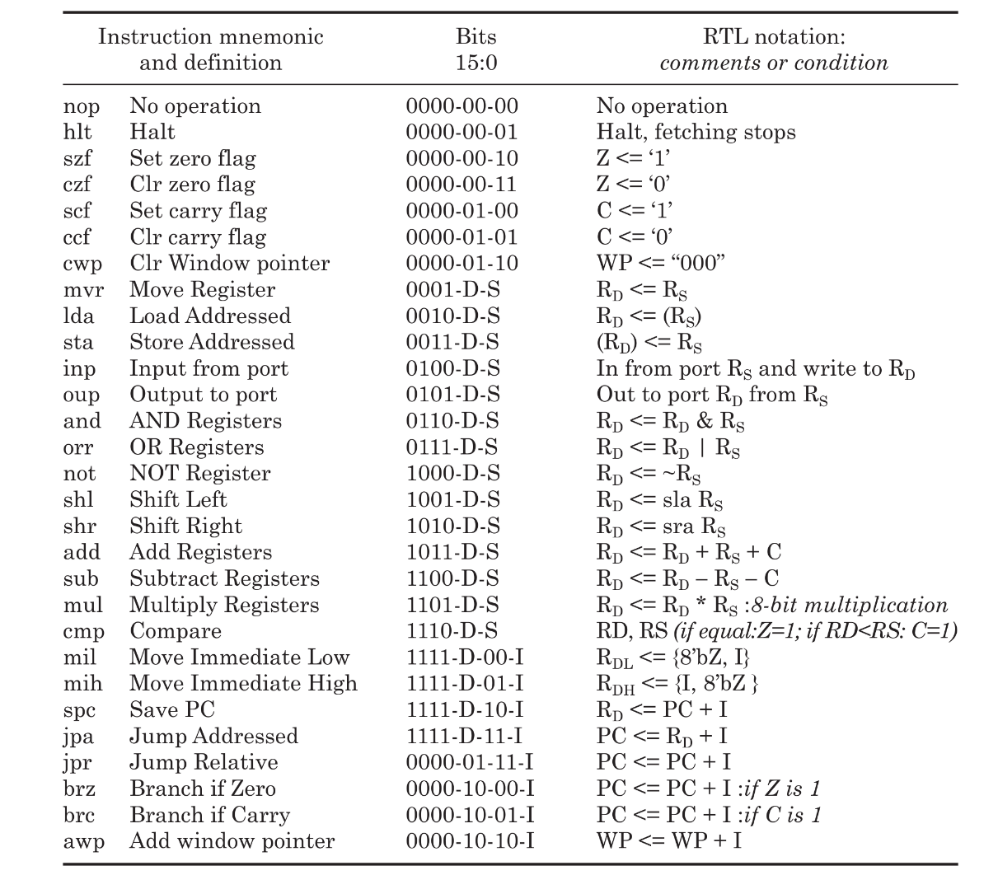
The general format of 8-bit and 16-bit SAYEH instructions is shown below:

****

**Fig\_2. SAYEH Instruction Format**

The 16-bit instructions have the Immediate field and the 8-bit instructions do not. The OPCODE filed is a 4-bit code that specifies the type of instruction. The Left and Right fields are two-bit codes selecting R0 through R3 for source and/or destination of an instruction.

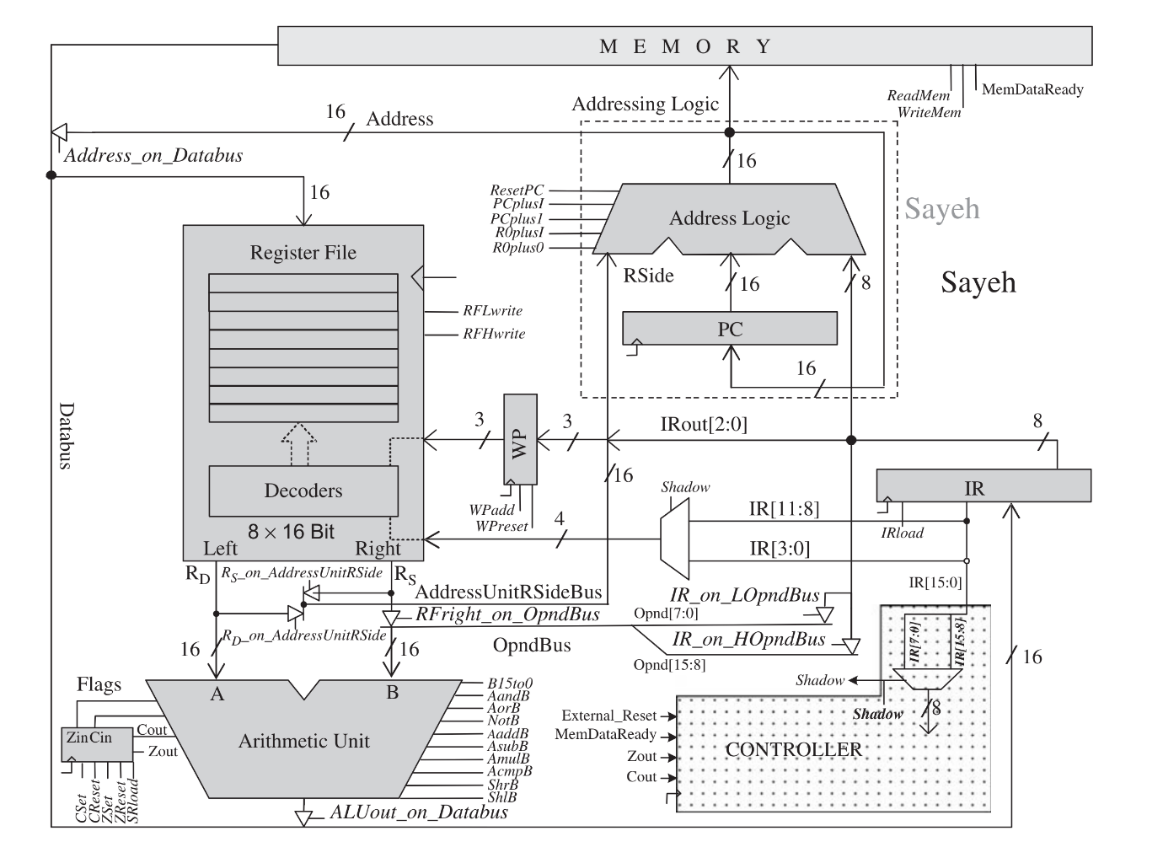
Our processor has a total of 29 instructions as shown in Figure.

****

**Fig\_3 Instruction Set of SAYEH**

**3.**  **SAYEH Data Path:**

The Datapath of SAYEH is:



**Fig\_4. Data Path**

Datapath components are Addressing Unit, Instruction Register, Window Pointer, Register File, Arithmetic Logic Unit, and the Flags register. The Addressing Unit is further partitioned into the Program Counter and Address Logic.

**3.1**  **Data Path Components:**

1. Addressing Unit
   1. Program counter (PC)
   2. Address Logic
2. Instruction register (IR)
3. Window pointer (WP)
4. Register File
5. Arithmetic logic unit (ALU)
6. Flags

**3.2**  **Data Path Components Verilog Code:**

**Addressing Unit:**

//Overall Addressing unit...

module AddressingUnit(input [15:0]Rside, input [7:0] Iside, [15:0]Address,input clk, ResetPC, PCplusI, PCplus1, RplusI, Rplus0, PCenable);

wire [15:0] PCout;

programcounter PC (Address, PCenable, clk, PCout);

addresslogic AL (PCout, Rside, Iside, ResetPC, PCplusI, PCplus1, RplusI, Rplus0,Address);

endmodule

**Program Counter:**

//Program Counter

module programcounter(

input [15:0] in, input enable, clk, output reg [15:0] out);

always @ (negedge clk)

if (enable) out <= in;

endmodule

**Address Logic:**

//Address LOgic

module addresslogic ( input [15:0] PCside, Rside,

input [7:0] Iside,

input ResetPC, PCplusI, PCplus1, RplusI, Rplus0,

output reg [15:0] ALout );

always @ (\*)

case ({ResetPC, PCplusI, PCplus1, RplusI, Rplus0})

5'b10000: ALout = 0;

5'b01000: ALout = PCside + Iside;

5'b00100: ALout = PCside + 1;

5'b00010: ALout = Rside + Iside;

5'b00001: ALout = Rside;

default: ALout = PCside;

endcase

endmodule

**Register File:**

// Register File

module RegisterFile ( input [15:0] in,input clk,input [1:0] Laddr, Raddr, input [2:0] Base,input RFLwrite, RFHwrite,output [15:0] Lout, Rout );

reg [15:0] MemoryFile [0:7];

wire [2:0] Laddress = Base + Laddr;

wire [2:0] Raddress = Base + Raddr;

assign Lout = MemoryFile [Laddress];

assign Rout = MemoryFile [Raddress];

reg [15:0] TempReg;

always @(negedge clk) begin

TempReg = MemoryFile [Laddress];

if (RFLwrite) TempReg [7:0] = in [7:0];

if (RFHwrite) TempReg [15:8] = in [15:8];

MemoryFile [Laddress] = TempReg;

end

endmodule

**Arthmetic Unit:**

//Arthmetic Unit ALU..

`define B15to0H 10'b1000000000

`define AandBH 10'b0100000000

`define AorBH 10'b0010000000

`define notBH 10'b0001000000

`define shlBH 10'b0000100000

`define shrBH 10'b0000010000

`define AaddBH 10'b0000001000

`define AsubBH 10'b0000000100

`define AmulBH 10'b0000000010

`define AcmpBH 10'b0000000001

module ArithmeticUnit ( A, B, B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB, aluout, cin, zout, cout);

input [15:0] A, B;

input B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB;

input cin;

output [15:0] aluout;

output zout, cout;

reg [15:0] aluout;

reg zout, cout;

always @(\*)

begin

zout = 0; cout = 0; aluout = 0;

case ({B15to0, AandB, AorB, notB, shlB,shrB, AaddB, AsubB, AmulB, AcmpB})

`B15to0H:aluout = B;

`AandBH: aluout = A & B;

`AorBH: aluout = A | B;

`notBH: aluout = ~B;

`shlBH: aluout = {B[15:0], B[0]};

`shrBH: aluout = {B[15], B[15:1]};

`AaddBH: {cout, aluout} = A + B + cin;

`AsubBH: {cout, aluout} = A - B - cin;

`AmulBH: aluout = A[7:0] \* B[7:0];

`AcmpBH: begin

aluout = A;

if (A> B) cout = 1;

else cout = 0;

if (A==B) zout = 1;

else zout = 0;

end

default: aluout = 0;

endcase

if (aluout == 0)

zout = 1'b1;

end

endmodule

**Instruction Register:**

//Instruction Register

InstrunctionRegister (in, IRload, clk, out);

input [15:0] in;

input IRload, clk;

output [15:0] out;

reg [15:0] out;

always @(posedge clk)

if (IRload == 1)

out <= in;

endmodule

**Window Pointer:**

/// Window Pointer....

module WindowPointer(IRout[2:0],clk,WPreset,WPadd,WPout);

input [2:0]IRout;

input WPreset,WPadd,clk;

output reg [2:0]WPout;

always@(posedge clk)

begin

if(WPreset)

WPout<=3'b000;

else if(WPadd)

WPout<= WPout+IRout;

end

endmodule

**Status Register:**

// Status Register

module StatusRegister(SRCin,SRZin,SRload,clk,Cset,Creset,Zset,Zreset,SRCout,

SRZout);

input SRCin,SRZin,clk,Cset,Creset,Zset,Zreset,SRload; output SRCout,SRZout;

reg SRCout;

reg SRZout;

always@ (posedge clk)begin

if(SRload) begin

SRCout<= (SRCin|Cset)&(~Creset);

SRZout<= (SRZin|Zset)&(~Zreset);

end

end

endmodule

**Complete Data Path:**

// Complete Datapath

module datapath(clk, Databus, Addressbus,ResetPC, PCplusI, PCplus1, RplusI, Rplus0,Rs\_on\_AddressUnitRSide, Rd\_on\_AddressUnitRSide, EnablePC,B15to0, AandB, AorB, notB,

shlB, shrB,AaddB, AsubB, AmulB, AcmpB,RFLwrite, RFHwrite,WPreset, WPadd, IRload, SRload, Address\_on\_Databus,ALU\_on\_Databus,IR\_on\_LOpndBus,

IR\_on\_HOpndBus, RFright\_on\_OpndBus,Cset, Creset, Zset, Zreset, Shadow, Instruction, Cout, Zout );

input clk;

inout [15:0] Databus;

output [15:0] Addressbus, Instruction;

output Cout, Zout;

input ResetPC, PCplusI, PCplus1, RplusI, Rplus0,Rs\_on\_AddressUnitRSide, Rd\_on\_AddressUnitRSide, EnablePC,B15to0, AandB, AorB, notB, shlB, shrB,AaddB, AsubB,

AmulB, AcmpB,RFLwrite, RFHwrite, WPreset, WPadd, IRload, SRload,Address\_on\_Databus, ALU\_on\_Databus, IR\_on\_LOpndBus,IR\_on\_HOpndBus, RFright\_on\_OpndBus,

Cset, Creset, Zset, Zreset, Shadow;

wire [15:0] Right, Left, OpndBus, ALUout, IRout, Address,AddressUnitRSideBus;

wire SRCin, SRZin, SRZout, SRCout;

wire [2:0] WPout;

wire [1:0] Laddr, Raddr;

AddressingUnit AU (AddressUnitRSideBus, IRout[7:0], Address,clk, ResetPC, PCplusI, PCplus1, RplusI,Rplus0, EnablePC);

ArithmeticUnit AL (Left, OpndBus, B15to0, AandB, AorB, notB,shlB, shrB, AaddB, AsubB, AmulB, AcmpB,ALUout, SRCout, SRZin, SRCin);

RegisterFile RF (Databus, clk, Laddr, Raddr, WPout, RFLwrite,RFHwrite, Left, Right);

InstrunctionRegister IR (Databus, IRload, clk, IRout);

StatusRegister SR (SRCin, SRZin, SRload, clk, Cset, Creset,Zset, Zreset, SRCout, SRZout);

WindowPointer WP (IRout[2:0], clk, WPreset, WPadd, WPout);

assign AddressUnitRSideBus = (Rs\_on\_AddressUnitRSide) ? Right:(Rd\_on\_AddressUnitRSide) ?Left :16'bZZZZZZZZZZZZZZZZ;

assign Addressbus = Address;

assign Databus = (Address\_on\_Databus) ? Address :(ALU\_on\_Databus) ? ALUout :16'bZZZZZZZZZZZZZZZZ;

assign OpndBus[07:0] = IR\_on\_LOpndBus == 1 ? IRout[7:0] :8'bZZZZZZZZ;

assign OpndBus[15:8] = IR\_on\_HOpndBus == 1 ? IRout[7:0] :8'bZZZZZZZZ;

assign OpndBus = RFright\_on\_OpndBus == 1 ? Right :16'bZZZZZZZZZZZZZZZZ;

assign Zout = SRZout;

assign Cout = SRCout;

assign Instruction = IRout[15:0];

assign Laddr = (~Shadow) ? IRout[11:10] : IRout[3:2];

assign Raddr = (~Shadow) ? IRout[09:08] : IRout[1:0];

endmodule

**SAYEH Controller:**

The controller of SAYEH is a state machine with eight states that generates appropriate control signals to the Data Path. The Instruction register output and ALU flags and external control signals like ExternalReset are the inputs of Contorller. The outputs of Controller are 32 control signals going to Datapath.

The State Daigram of Controller:

**A diagram of a diagram

Description automatically generated**

State **reset** is the initial state of the machine in **fetch** state the processor fetches the 16-bit instruction that can include 8-bit shadow instruction in to the Databus and in **memread** state the instruction in the Databus is loaded into the Instruction register. Execution of instructions is performed in the **exec1** state.The instructions like lda is not completed by the **exec1** state and requires the additional state of **exec1lda** to complete its memory read.

States **exec2** and **exec2lda** are like **exec1** and **exce1lda** except that they handle the shadow part of an instruction. Certain instructions that use the address bus for their execution cannot increment PC while they are being executed. For these instructions, the incpc state increments the program counter.

**Modelling Controller Unit Verilog HDL:**

//Controller Verilog code

module controller( Instruction,ExternalReset,MemDataReady, clk,Cflag, Zflag,ResetPC, PCplusI, PCplus1, RplusI, Rplus0,Rs\_on\_AddressUnitRSide, Rd\_on\_AddressUnitRSide, EnablePC,

B15to0, AandB, AorB, notB, shlB, shrB,AaddB, AsubB, AmulB, AcmpB,RFLwrite, RFHwrite,WPreset, WPadd, IRload, SRload, Address\_on\_Databus,ALU\_on\_Databus,

IR\_on\_LOpndBus, IR\_on\_HOpndBus, RFright\_on\_OpndBus,

Cset, Creset, Zset, Zreset,ReadMem,WriteMem,Shadow );

input ExternalReset,clk,Cflag, Zflag,MemDataReady;

input [15:0] Instruction;

output ResetPC, PCplusI, PCplus1, RplusI, Rplus0, Rs\_on\_AddressUnitRSide, Rd\_on\_AddressUnitRSide, EnablePC,

B15to0, AandB, AorB, notB, shlB, shrB,AaddB, AsubB, AmulB, AcmpB,

RFLwrite, RFHwrite,WPreset, WPadd, IRload, SRload,

Address\_on\_Databus,ALU\_on\_Databus,IR\_on\_LOpndBus, IR\_on\_HOpndBus, RFright\_on\_OpndBus,

Cset, Creset, Zset, Zreset,ReadMem,WriteMem,Shadow;

reg ResetPC, PCplusI, PCplus1, RplusI, Rplus0, Rs\_on\_AddressUnitRSide, Rd\_on\_AddressUnitRSide, EnablePC,

B15to0, AandB, AorB, notB, shlB, shrB,AaddB, AsubB, AmulB, AcmpB,

RFLwrite, RFHwrite,WPreset, WPadd, IRload, SRload,

Address\_on\_Databus,ALU\_on\_Databus,IR\_on\_LOpndBus, IR\_on\_HOpndBus, RFright\_on\_OpndBus,

Cset, Creset, Zset, Zreset, Shadow,ReadMem,WriteMem;

parameter [3:0] reset = 0, halt = 1, fetch = 2, memread = 3, exec1 = 4, exec2 = 5, exec1lda = 6, exec2lda = 7, incpc = 8;

reg [3:0] Pstate, Nstate;

//output ShadowEn ;

wire ShadowEn = ~(Instruction[7:0] == 8'b00001111);

parameter hlt = 4'b0001;

parameter nop = 4'b0000;

parameter szf = 4'b0010;

parameter czf = 4'b0011;

parameter scf = 4'b0100;

parameter ccf = 4'b0101;

parameter cwp = 4'b0110;

parameter jpr = 4'b0111;

parameter brz = 4'b1000;

parameter brc = 4'b1001;

parameter awp = 4'b1010;

parameter mvr = 4'b0001;

parameter lda = 4'b0010;

parameter sta = 4'b0011;

parameter inp = 4'b0100;

parameter oup = 4'b0101;

parameter andd = 4'b0110;

parameter orr = 4'b0111;

parameter nott = 4'b1000;

parameter shl = 4'b1001;

parameter shr = 4'b1010;

parameter add = 4'b1011;

parameter sub = 4'b1100;

parameter mul = 4'b1101;

parameter cmp = 4'b1110;

parameter mil = 2'b00;

parameter mih = 2'b01;

parameter spc = 2'b10;

parameter jpa = 2'b11;

always @( negedge clk ) begin

if( ExternalReset )

Pstate <= reset;

else

Pstate <= Nstate;

end

always @(\*)

begin

ResetPC = 1'b0; PCplusI=1'b0; PCplus1=1'b0; RplusI= 1'b0; Rplus0= 1'b0; Rs\_on\_AddressUnitRSide= 1'b0; Rd\_on\_AddressUnitRSide= 1'b0; EnablePC= 1'b0;

B15to0= 1'b0; AandB= 1'b0; AorB= 1'b0; notB= 1'b0; shlB= 1'b0; shrB= 1'b0; AaddB= 1'b0; AsubB= 1'b0; AmulB= 1'b0; AcmpB= 1'b0;

RFLwrite= 1'b0; RFHwrite= 1'b0; WPreset= 1'b0; WPadd= 1'b0; IRload= 1'b0; SRload= 1'b0;

Address\_on\_Databus= 1'b0; ALU\_on\_Databus= 1'b0;IR\_on\_LOpndBus= 1'b0; IR\_on\_HOpndBus= 1'b0; RFright\_on\_OpndBus= 1'b0;

Cset= 1'b0; Creset= 1'b0; Zset= 1'b0; Zreset= 1'b0; Shadow= 1'b0; ReadMem = 1'b0; WriteMem = 1'b0;

case(Pstate)

reset: begin

ResetPC = 1'b1;

EnablePC = 1'b1;

SRload = 1'b1;

Creset = 1'b1;

Zreset = 1'b1;

WPreset = 1'b1;

Nstate=fetch;

end

//End Reset

halt:begin

Nstate = halt;

end

fetch: begin

ReadMem = 1'b1;

Nstate=memread;

end

//End fetch

memread: begin

if(MemDataReady == 1'b1) begin

IRload = 1'b1;

Nstate=exec1;

end

else begin

ReadMem = 1'b1;

Nstate = memread;

end

end

//End readmem

exec1:

if(Instruction[15:12] == 4'b0000) begin

case(Instruction[11:8])

nop : begin

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

hlt: begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = halt;

end

scf : begin

Cset = 1'b1;

SRload = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

ccf : begin

Creset = 1'b1;

SRload = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

szf : begin

Zset = 1'b1;

SRload = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

czf : begin

Zreset = 1'b1;

SRload = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

cwp : begin

WPreset = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

// 16 bit instruction with immediate data

jpr : begin

PCplusI=1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

brz : begin

if(Zflag == 1) begin

PCplusI=1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

else begin

PCplus1=1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

brc : begin

if(Cflag == 1) begin

PCplusI=1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

else begin

PCplus1=1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

awp : begin

WPadd = 1'b1;

PCplus1=1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

endcase

end// if END

else if (Instruction[15:12] == 4'b1111) begin

case(Instruction[9:8])

mil : begin

IR\_on\_LOpndBus= 1'b1;

B15to0= 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

mih : begin

IR\_on\_HOpndBus= 1'b1;

B15to0= 1'b1;

ALU\_on\_Databus= 1'b1;

RFHwrite = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

spc : begin

PCplusI = 1'b1;

Address\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

Nstate = incpc;

end

jpa : begin

Rd\_on\_AddressUnitRSide= 1'b1;

RplusI= 1'b1;

EnablePC=1'b1;

Nstate=fetch;

end

endcase

end //elseif END

else begin

case(Instruction[15:12])

lda : begin

Rs\_on\_AddressUnitRSide = 1'b1;

Rplus0 = 1'b1;

//PCplusI=1'b0;

ReadMem = 1'b1;

Nstate = exec1lda;

end

sta : begin

RFright\_on\_OpndBus = 1'b1;

B15to0 = 1'b1;

ALU\_on\_Databus = 1'b1;

Rplus0 = 1'b1;

Rd\_on\_AddressUnitRSide = 1'b1;

WriteMem = 1'b1;

Nstate = incpc;

end

mvr : begin

RFright\_on\_OpndBus = 1'b1;

B15to0 = 1'b1;

ALU\_on\_Databus = 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

andd : begin

RFright\_on\_OpndBus= 1'b1;

AandB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

orr : begin

RFright\_on\_OpndBus= 1'b1;

AorB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

nott : begin

RFright\_on\_OpndBus= 1'b1;

notB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

shl : begin

RFright\_on\_OpndBus= 1'b1;

shlB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

shr : begin

RFright\_on\_OpndBus= 1'b1;

shrB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1= 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

add : begin

RFright\_on\_OpndBus = 1'b1;

AaddB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

SRload = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

sub : begin

RFright\_on\_OpndBus= 1'b1;

AsubB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

SRload = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

mul : begin

RFright\_on\_OpndBus= 1'b1;

AmulB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

SRload = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

cmp : begin

RFright\_on\_OpndBus= 1'b1;

AcmpB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

SRload = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

endcase

end //else END

exec1lda : begin

if(MemDataReady == 1'b1) begin

RFLwrite = 1'b1;

RFHwrite = 1'b1;

if (ShadowEn==1'b1)

Nstate = exec2;

else begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end

else begin

ReadMem = 1'b1;

Rplus0 = 1'b1;

Rs\_on\_AddressUnitRSide = 1'b1;

Nstate = exec1lda;

end

end

exec2 : begin

Shadow = 1'b1;

if(Instruction[7:4] == 4'b0000) begin

case(Instruction[3:0])

nop : begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

hlt: begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = halt;

end

scf : begin

Cset = 1'b1;

SRload = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

ccf : begin

Creset = 1'b1;

SRload = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

szf : begin

Zset = 1'b1;

SRload = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

czf : begin

Zreset = 1'b1;

SRload = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

cwp : begin

WPreset = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

endcase

end// if END

else begin

case(Instruction[7:4])

lda : begin

Rs\_on\_AddressUnitRSide = 1'b1;

Rplus0 = 1'b1;

ReadMem = 1'b1;

Nstate = exec2lda;

end

sta : begin

RFright\_on\_OpndBus = 1'b1;

B15to0 = 1'b1;

ALU\_on\_Databus = 1'b1;

Rplus0 = 1'b1;

Rd\_on\_AddressUnitRSide = 1'b1;

WriteMem = 1'b1;

Nstate = fetch;

end

mvr : begin

RFright\_on\_OpndBus = 1'b1;

B15to0 = 1'b1;

ALU\_on\_Databus = 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

andd : begin

RFright\_on\_OpndBus= 1'b1;

AandB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

orr : begin

RFright\_on\_OpndBus= 1'b1;

AorB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate <= fetch;

end

nott : begin

RFright\_on\_OpndBus= 1'b1;

notB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

shl : begin

RFright\_on\_OpndBus= 1'b1;

shlB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

shr : begin

RFright\_on\_OpndBus= 1'b1;

shrB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

add : begin

RFright\_on\_OpndBus = 1'b1;

AaddB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

SRload = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

sub : begin

RFright\_on\_OpndBus= 1'b1;

AsubB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

SRload = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

mul : begin

RFright\_on\_OpndBus= 1'b1;

AmulB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

SRload = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

cmp : begin

RFright\_on\_OpndBus= 1'b1;

AcmpB = 1'b1;

ALU\_on\_Databus= 1'b1;

RFLwrite = 1'b1;

RFHwrite = 1'b1;

SRload = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

endcase

end //else END

end //exec2

exec2lda:

begin

if(MemDataReady == 1'b1) begin

RFLwrite = 1'b1;

RFHwrite = 1'b1;

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

end//END of Exedc2lda

incpc : begin

PCplus1 = 1'b1;

EnablePC=1'b1;

Nstate = fetch;

end

default : Nstate=reset;

endcase

end

endmodule

**Complete Sayeh Processor:**

Modelling the Complete Sayeh by instantiating the both Datapath module and Controller module using Verilog. In the Overall module control signal outputs of controller are wired to the similarly named signals of DataPath.

Top level Verilog code of Sayeh Processor:

//COMPLETE SAYEH PROCESSOR

module Sayeh ( clk, ReadMem, WriteMem,Databus, Addressbus, ExternalReset);

input clk;

output ReadMem, WriteMem;

inout [15: 0] Databus;

output [15: 0] Addressbus;

input ExternalReset;

wire [15:0]Instruction;

wire ResetPC, PCplusI, PCplus1, RplusI, Rplus0, Rs\_on\_AddressUnitRSide, Rd\_on\_AddressUnitRSide, EnablePC,

B15to0, AandB, AorB, notB, shlB, shrB,AaddB, AsubB, AmulB, AcmpB,

RFLwrite, RFHwrite,WPreset, WPadd, IRload, SRload,

Address\_on\_Databus,ALU\_on\_Databus,IR\_on\_LOpndBus, IR\_on\_HOpndBus, RFright\_on\_OpndBus,

Cset, Creset, Zset, Zreset, Shadow,ReadMem,WriteMem,Cout,Zout;

//wire Cout,Zout;

datapath dp(clk, Databus, Addressbus,ResetPC, PCplusI, PCplus1, RplusI, Rplus0,Rs\_on\_AddressUnitRSide, Rd\_on\_AddressUnitRSide, EnablePC,B15to0, AandB, AorB, notB,

shlB, shrB,AaddB, AsubB, AmulB, AcmpB,RFLwrite, RFHwrite,WPreset, WPadd, IRload, SRload, Address\_on\_Databus,ALU\_on\_Databus,IR\_on\_LOpndBus,

IR\_on\_HOpndBus, RFright\_on\_OpndBus,Cset, Creset, Zset, Zreset, Shadow, Instruction, Cout, Zout);

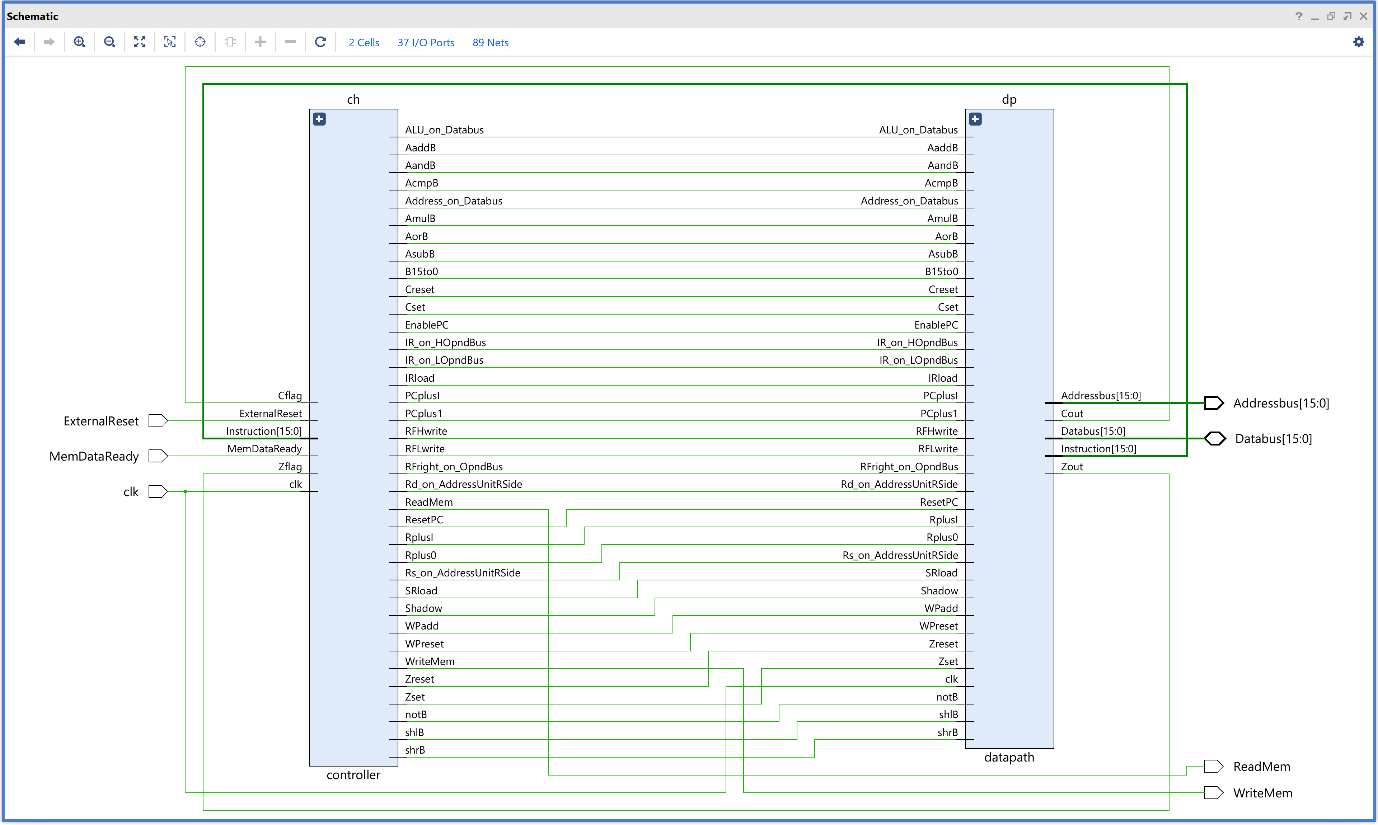
controller ch( Instruction,ExternalReset, clk,Cout, Zout,ResetPC, PCplusI, PCplus1, RplusI, Rplus0,Rs\_on\_AddressUnitRSide, Rd\_on\_AddressUnitRSide, EnablePC,

B15to0, AandB, AorB, notB, shlB, shrB,AaddB, AsubB, AmulB, AcmpB,RFLwrite, RFHwrite,WPreset, WPadd, IRload, SRload, Address\_on\_Databus,

ALU\_on\_Databus,IR\_on\_LOpndBus, IR\_on\_HOpndBus, RFright\_on\_OpndBus,Cset, Creset, Zset, Zreset,ReadMem,WriteMem,Shadow);

endmodule

**Schematic Daigram of Sayeh Processor:**

****

**Instruction Examples and Results:**

For Testing the Sayeh Processor the instructions we created are as follows:

@0 f003

@1 f100

@2 f404

@3 f500

@4 b10f

Lets decode and understand the above instructions. “@0” implies at 0th location of the memory. These instruction are in hexadecimal format will convert into binary so that we can understand them easily. (F003)16 🡺 (1111-0000-0000-0011)2 From the Opcode bits we can decode and identify it as the **mil** (move immediate Low) instruction. That is in this instruction we are actually storing the immediate data (03) in to the **R0** register in the Register File. Likewise we can decode and understand the other instructions also.

**Test Bench for SAYEH Processor :**

For testing the SAYEH Processor we need a memory file to store instructions. So we created memory file mamed as “ instructions.mem ”. Each line in a **.mem** file represents a memory address and its corresponding data. The data can be in any format like binary, hexadecimal or decimal. **$readmemh**("instructions.mem", memory); This system task reads the hexadecimal data from **instructions.mem** and initializes the **memory** with the values.

//TESTBENCH

module testbench;

reg ExternalReset=1, clk=1;

reg MemDataReady;

wire ReadMem, WriteMem;

wire [15:0] Addressbus;

wire [15:0] Databus;

reg [15:0] memory[0:63]; // Declare memory array

reg [15:0]mem\_data;

initial begin

MemDataReady=0;

mem\_data=16'bz;

// Read the memory file into the array

$readmemh("instructions.mem", memory);

#50 ExternalReset=1'b0;

end

Sayeh UUT(clk, ReadMem, WriteMem,Databus, Addressbus, ExternalReset,MemDataReady);

always #10 clk = ~clk;

//reg control=0;

always@(posedge clk) begin : Memory\_Read\_Write

if(ReadMem) begin

MemDataReady <= 1;

mem\_data<=memory[Addressbus];

end

else begin

MemDataReady <= 0;

mem\_data <= 16'bz;

end

if(WriteMem) begin

#1 memory[Addressbus]=Databus;

end

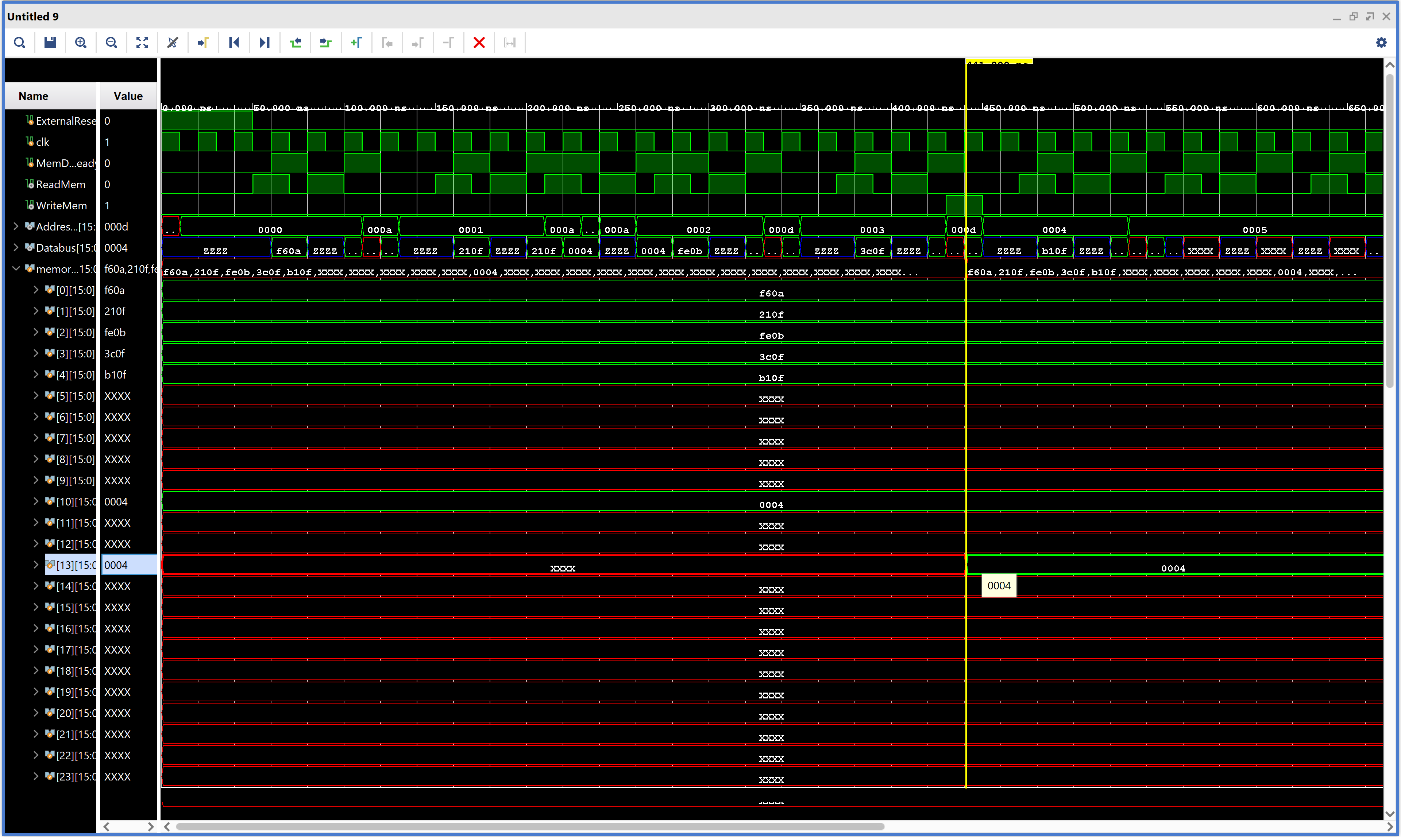
end

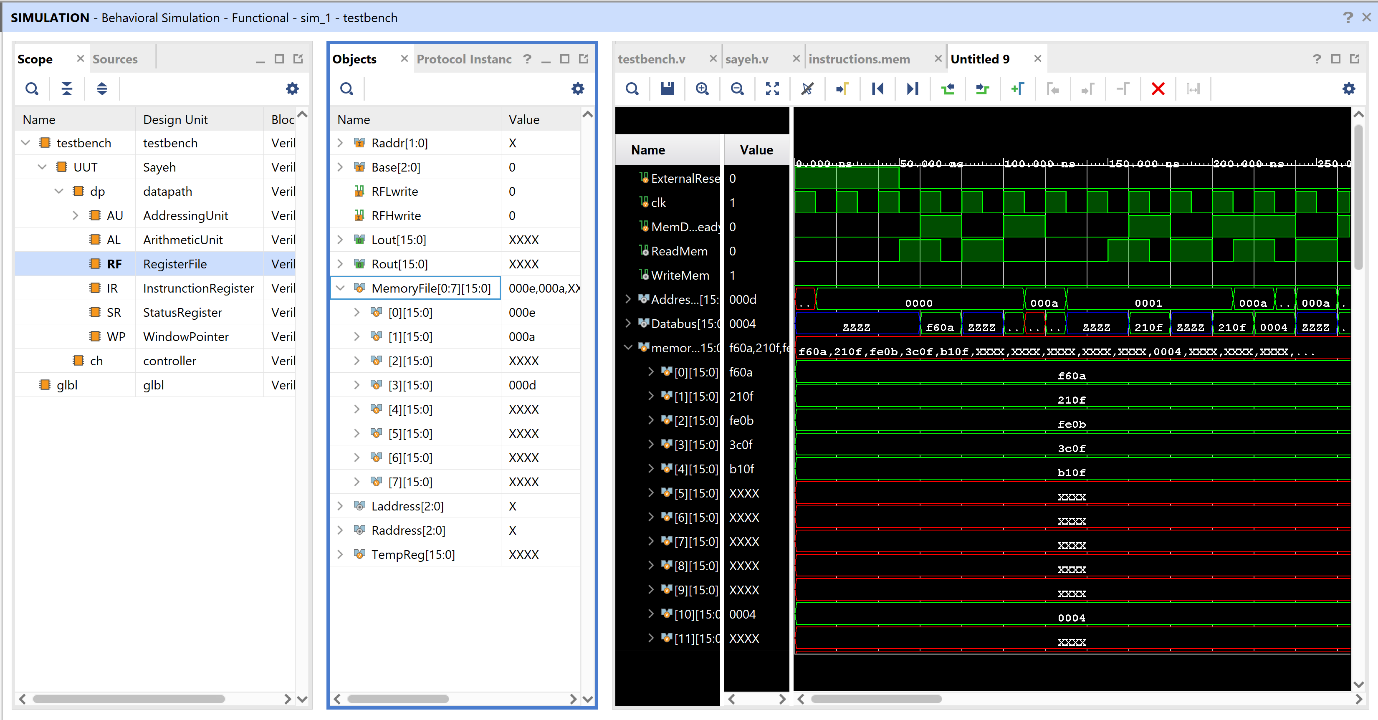
// Assigning( Copying) the mem\_data to the data\_bus

assign Databus = mem\_data;

endmodule

**Result:**

****

****