Term Project

Fall 2017

Rachel Peckham

Bjorn Goriatcheff

Contribution: 50/50

Dr. Behnam S Arad

CSC/CpE 142

Section 02

Date Due: 10/5/17

Date Submitted: 10/5/17

Table of Contents

Project Status Report

CSc/CPE 142
Term Project Status Report

Complete this form by typing the requested information and include the completed form in your report after TOC. Gray cells will be filled by the instructor.

Name	*% Contribution*	*Grade*
Rachel Peckham	*50*	
Bjorn Goriatcheff	*50*	

Please do not write in the first table

Project Report/Presentation 20%	/200
Functionality of the individual components 40%	/400
Functionality of the overall design 25%	/250
Design Approach 5%	/50
Total points	/900

A: List all the instructions that were implemented correctly and verified by the assembly program on your system:

Instructions	State any issue regarding the instruction.
Signed addition	Unsigned operation
Signed subtraction	Unsigned operation
Signed multiplication	Unsigned operation Upper half not sent to R15.
Signed division	Unsigned operation
Move	No issue.
SWAP	No issue.
AND immediate	No issue.

OR immediate	No issue.
Load byte unsigned	No issue.
Store byte	No issue.
Load	No issue.
Store	No issue.
Branch on less than	No issue.a
Branch on greater than	No issue.
Branch on equal	No issue.
jump	No issue.
Halt	No issue.

B: Fill out the next table:

Individual Components	Does your system have this component?	List the student who designed and verified the block	Does it work ?	List problems with the component, if any.
ALU	Yes.	Rachel.	Yes.	No issue.
ALU control unit	Yes.	Bjorn.	Yes.	No issue.
Memory Unit	Yes.	Rachel.	Yes.	No issue.
Register File	Yes.	Bjorn.	Yes.	No issue.
PC	Yes.	Bjorn.	Yes.	No issue.
IR	Yes.	Bjorn.	Yes.	No issue.
	Yes.	Rachel.	Yes.	No issue.

Other registers				
BUFFERS	Yes.	Bjorn.	Yes.	Forwarding issue
exception handler 1. Unknown opcode 2. Arith. Overflow …..	1. No. 2. Yes.	1. 2. Rachel.	1. No. 2. Yes.	1. 2. No issue.
Control Units 1. main 2. forwarding 3. lw hazard detection	1. Yes. 2. Yes. 3. Yes.	1. Bjorn. 2. Rachel. 3. Rachel.	1. Yes. 2. No. 3. No.	1. No issue. 2. 3.

How many stages do you have in your pipeline? ……..
5 Stages : IF/ID ID/EX EX/MEM MEM/WB

C: **State any issue regarding the overall operation of the datapath? Be Specific.**

The Hazard Detection Unit and Forwarding modules have been created, but have not been entirely installed into the datapath. Arithmetic Overflows are caught, but not handled yet.

We have submitted a one-cycle implementation to verify that we are able to perform all instructions in the folder called “CPU_buff”.

The pipeline is working but forwarding issues are problematic to compute the first two instruction correctly. ADD R1 R2, SUB R1 R2 use the same input registers and the content of R1 is not updated at the correct time for SUB operation which causes a wrong result “ed0” instead of “f00”. The forwarding unit could solve that problem by forwarding the result of the first operation to the ALU directly.

The overflow is detected inside the ALU by performing the following operations:
ADD operation: if(FFFF - op1 < op2 -1) => overflow
MUL operation: if(FFFF / op1 < op2 -1) => overflow
SUB operation: if (op2 > op1) => overflow (unsigned operation)
The conditions were found but not handled yet as an exception.

The promising results of the pipelined datapath were not correct because of the lack of hazard detection and forwarding units. The Single cycle system was performing every operation correctly because the architecture was much more simple.

Almost every controls are concatenated inside the buffer to prevent errors of computation (every stage got the right control at the right time) but some are still missing. The fixed size of the buffer made it very difficult to add new controls inside it.

Truth Tables

Control unit:

Input	ALU src1 – 2bits	ALU src2 – 2bits	SRC 15 - 1 bit	OFF set - 1 bit	Imm - 1 bit	Down - 1 bit	Mbyte - 1 bit	MV1 src - 1 bit	Halt	Branch 2 bits	Zero 2 bits
Signed add	00	00	0	0	0	0	0	1	0	11	01
Signed sub	00	00	0	0	0	0	0	1	0	11	01
Signed mult	00	00	0	0	0	0	0	1	0	11	01
Signed div	00	00	0	0	0	0	0	1	0	11	01
Move	00	00	0	0	0	0	0	0	0	11	01
SWAP	00	00	0	0	0	0	0	0	0	11	01
ANDi	00	00	0	1	1	0	0	1	0	11	01
ORi	00	00	0	0	1	0	0	1	0	11	01
Load byte	00	00	0	1	0	1	1	1	0	11	01
Store byte	00	00	0	0	0	0	0	1	0	11	01
Load	00	00	0	1	0	1	0	1	0	11	01
Store	00	00	0	0	0	0	0	1	0	11	01
Branch less than	00	00	1	0	0	X	0	1	0	01	00
Branch greater than	00	00	1	0	0	X	0	1	0	10	00
Branch equal	00	00	1	1	0	X	0	1	0	00	00
Jump	X	X	0	1	X	X	0	1	0	XX	00
Halt	X	X	0	X	X	X	0	1	1	01	01

ALU Control:

Input	Writedst - 2 bits	MemW - 2 bit	ALUop - 4 bits
Signed add	00	00	0000
Signed sub	00	00	0001
Signed mult	10	00	0100
Signed div	10	00	0101
Move	00	00	0111
SWAP	01	00	1000
AND	00	00	1001
OR	00	00	1011
Load byte	00	00	0000
Store byte	11	01	0000
Load	00	00	0000
Store	11	10	0000
Branch less than	11	00	X
Branch greater than	11	00	X
Branch equal	11	00	X
Jump	11	00	X
Halt	11	00	X

Source Code

```
module Adder2#(parameter N=15)
    (output reg[N:0] add,
    input [N:0] pc
    );
always@(*)
begin
    add=pc+2;
end
endmodule

module AdderBr#(parameter N=15)
(    output reg[N:0] Next,
    input [N:0] Addr,
    input [N:0] Offset
);

always@(*)
begin
    Next = Addr + Offset;
end

endmodule

module ALU(
    output reg[15:0] Upper,
    output reg[15:0] Lower,
    output reg[1:0] Zero,
    output reg Overflow,
    input [15:0] Data1,
    input [15:0] Data2,
    input [3:0] ALUOp
```

```
);

always@(*)
begin
   if(ALUOp == 4'b0000)  //Signed Add
   begin
         if((16'hFFFF - Data1) < Data2)
               Overflow = 1'b1;
         else
         begin
               Overflow = 1'b0;
               Lower = Data1 + Data2;
         begin
   end
   else if(ALUOp == 4'b0001)  //Signed Sub
   begin
         if((16'hFFFF + Data1) < Data2)
               Overflow = 1'b1;
         else
         begin
               Overflow = 1'b0;
               Lower = Data1 - Data2;
         begin
   end
   else if(ALUOp == 4'b0100)  //Signed Mul
         if((32'hFFFFFFFF / Data1) < Data2)
               Overflow = 1'b1;
         else
         begin
               Overflow = 1'b0;
               Lower = Data1 * Data2;
         begin
   else if(ALUOp == 4'b0101)  //Signed Div
```

```
    begin
        Overflow = 1'b0;
        Lower = Data1 / Data2;
        Upper = Data1 % Data2;
    end
    else if(ALUOp == 4'b0111)  //Move
    begin
        Overflow = 1'b0;
        Lower = Data2;
    end
    else if(ALUOp == 4'b1000)  //Swap
    begin
        Overflow = 1'b0;
        Lower = Data2;
        Upper = Data1;
    end
    else if(ALUOp == 4'b1001)  //AND
    begin
        Overflow = 1'b0;
        Lower = Data1 & Data2;
    end
    else if(ALUOp == 4'b1011) //OR
    begin
        Overflow = 1'b0;
        Lower = Data1 | Data2;
    end
end

endmodule

module ALU_CONTROL(
    output reg[3:0]ALUop,
    input [3:0]ctrl,
```

```
    input [3:0]func
);

always@(*)
begin
    case(ctrl)
        //ADD SUB MUL DIV MOVE SWAP
        4'b1111: ALUop = func;
        //ANDi
        4'b1000: ALUop = 4'b1001;
        //ORi
        4'b1001: ALUop = 4'b1011;
        //LBi
        4'b1010: ALUop = 4'b0000;
        //SBi
        4'b1011: ALUop = 4'b0000;
        //Lw
        4'b1100: ALUop = 4'b0000;
        //Sw
        4'b1101: ALUop = 4'b0000;
    endcase

end
endmodule

module BufferEXMEM#(parameter S=15, N=4, C=1)
(
    output reg[S:0] OutUpper,
    output reg[S:0] OutLower,
    output reg[S:0] OutWord,
    output reg[7:0] OutByte,
    output reg[3:0] Forward,
    output reg[3:0] OutCtrlW,
```

```
    output reg[3:0] OutCtrlM,
    //forward out
    input [S:0] InUpper,
    input [S:0] InLower,
    input [S:0] InWord,
    input [7:0] InByte,
    input [3:0] For,
    //forward in
    input [3:0]InCtrlW,
    input [3:0]InCtrlM,
    input clk,
    input rst
);

reg[S:0] Upper;
reg[S:0] Lower;
reg[S:0] Word;
reg[7:0] Byte;
reg[3:0] Forw;
reg [3:0]CtrlM;
reg [3:0]CtrlW;



//Combinatorial READ
always@(*)
begin

    OutUpper=Upper;
    OutLower=Lower;
    OutWord=Word;
    OutByte=Byte;
    OutCtrlW=CtrlW;
```

```
    OutCtrlM=CtrlM;
    Forward=Forw;

end
always@(posedge clk, negedge rst)
begin
    if(!rst)
    begin
            Upper<=16'h0000;
                    Lower<=16'h0000;
                    Word<=16'h0000;
                    Byte<=8'h00;
                    CtrlM<=4'h0;
            CtrlW<=4'h0;
            Forw<=4'h0;
    end
    if(InCtrlM==4'h0)
           begin
                    Upper<=16'h0000;
                    Lower<=16'h0000;
                    Word<=16'h0000;
                    Byte<=8'h00;
                    CtrlW<=4'h0;
            CtrlM<=4'h0;
    end
    else
    begin
            Upper<=InUpper;
            Lower<=InLower;
            Word<=InWord;
            Byte<=InByte;
            CtrlM<=InCtrlM;
            CtrlW<=InCtrlW;
```

```
            Forw<=For;
    end
end
endmodule

module BufferIDEX#(parameter S=15, C=15)
(
    output reg[S:0] OutData1,
    output reg[S:0] OutData2,
    output reg[S:0] OutData15,
    output reg[3:0] OutDataRS,
    output reg[3:0] OutDataRT1,
    output reg[3:0] OutDataRT2,
    output reg[3:0] OutDataRD,
    output reg[C:0] OutCtrl,
    input [S:0] InData1,
    input [S:0] InData2,
    input [S:0] InData15,
    input [3:0] InRS,
    input [3:0] InRT1,
    input [3:0] InRT2,
    input [3:0] InRD,
    input [C:0] InCtrl,
    input clk,
    input rst
);

reg[S:0] Data1;
reg[S:0] Data2;
reg[S:0] Data15;
reg[3:0] DataRS;
reg[3:0] DataRT1;
reg[3:0] DataRT2;
```

```
reg[3:0] DataRD;
reg[C:0] Ctrl;


//Combinatorial READ
always@(*)
begin
   OutData1=Data1;
   OutData2=Data2;
   OutData15=Data15;
   OutDataRS=DataRS;
   OutDataRT1=DataRT1;
   OutDataRT2=DataRT2;
   OutDataRD=DataRD;
   OutCtrl=Ctrl;
end
always@(posedge clk, negedge rst)
begin
   if(!rst)
         begin
                  Data1<=15'h0000;
                  Data2<=15'h0000;
                  Data15<=15'h0000;
                  DataRS<=4'h0;
                  DataRT1<=4'h0;
                  DataRT2<=4'h0;
                  DataRD<=4'h0;
                  Ctrl<=15'h0000;

         end

   if(InCtrl==15'h000)
   begin
```

```
            Data1<=15'h0000;
                    Data2<=15'h0000;
                    Data15<=15'h0000;
                    DataRS<=4'h0;
                    DataRT1<=4'h0;
                    DataRT2<=4'h0;
                    DataRD<=4'h0;
            Ctrl<=15'h000;
    end
    else
    begin
            Data1<=InData1;
            Data2<=InData2;
            Data15<=InData15;
            DataRS<=InRS;
            DataRT1<=InRT1;
            DataRT2<=InRT2;
            DataRD<=InRD;
            Ctrl<=InCtrl;
    end
end

endmodule

module BufferIFID#(parameter S=15)
(
    output reg[S:0] OutInstr,
    input [S:0] InInstr,
    input ctrl,
    //flush/hazard
    input clk,
    input rst
);
```

```
reg [S:0]mem;

always@(*)
begin
    OutInstr = mem;
end
always@(posedge clk, negedge rst)
begin
   if(!rst)
          mem=15'h0000;
   if(!ctrl)
   begin
          mem=15'h0000;
          OutInstr=15'h0000;
   end
   else
   begin
          mem  = InInstr;
   end
end
endmodule

module BufferMEMWB#(parameter S=15, N=3)
(
   output reg[S:0] OutWord,
   output reg[7:0] OutByte,
   output reg[3:0] ForwardOut,
   output reg[3:0] WB,
   input [S:0] InWord,
   input [7:0] InByte,
   input [3:0]InCtrl,
   input [3:0] ForwardIn,
   input clk,
```

```
    input rst
);

reg[S:0] Word;
reg[7:0] Byte;
reg[3:0] Forw;
reg[3:0] BWB;
always@(*)
begin
    OutWord=Word;
    OutByte=Byte;
    ForwardOut=Forw;
    WB=BWB;
end
always@(*)
begin
    if(!rst)
            begin
                    Word<=16'h0000;
                    Byte<=8'h00;
                    Forw<=4'h0;
             BWB<=4'h0;
            end
    if(!InCtrl)
    begin
            Word<=16'h0000;
                     Byte<=8'h00;
                     Forw<=4'h0;
            BWB<=4'h0;
    end
    else
    begin
            Word<=InWord;
```

```
            Byte<=InByte;
            Forw<=ForwardIn;
            BWB<=InCtrl;
    end
end
endmodule

module CMP#(parameter N=15)
(
    output reg[1:0] res,
    input [N:0] op1,
    input [N:0] op2
);
always@(*)
begin
    if(op1 < op2)
            res = 2'b01;
    else if(op1 > op2)
            res = 2'b10;
    else if(op1 == op2)
            res = 2'b00;

end
endmodule

module CONTROL(
//    output reg[1:0]ALUsrc1,
//    output reg[1:0]ALUsrc2,
//    output reg SRC15,
    output reg OFFset,
    output reg Imm,
    output reg Down,
    output reg Mbyte,
```

```
    output reg MV1src,
    output reg Halt,
    output reg Fdst,
    output reg[1:0] Bra,
    output reg[1:0] Wdst,
    output reg[1:0] MemW,
    output reg wb,
    output reg MEMflush,
    output reg IFflush,
    output reg[1:0] IDflush,
    output reg EXflush,
    input [3:0] opcode,
    input [3:0] func
);

always@(*)
begin
//    ALUsrc1 = 2'b00;
//    ALUsrc2 = 2'b00;
//    SRC15 = 1'b0;
    OFFset = 1'b0;
    Imm = 1'b0;
    Down = 1'b0;
    Mbyte = 1'b0;
    MV1src = 1'b1;
    Halt = 1'b0;
    Wdst = 2'b00;
    MemW = 2'b00;
    //TODO modif zith cases
    wb=1'b1;
    Bra = 2'b11;
    // NO FLUSH == 1
    MEMflush=1'b1;
```

```
IFflush = 1'b1;
IDflush=2'b11;
EXflush = 1'b1;
//RT or RS
Fdst=1'b0;
case(opcode)
	//TYPE A
	4'b1111:
		case(func)
			//MUL
			4'b0100:
				Wdst = 2'b10;
			//DIV
			4'b0101:

				Wdst = 2'b10;
			//MOVE
			4'b0111:
				MV1src = 1'b0;
			//SWAP
			4'b1000:
			begin
				MV1src = 1'b0;
				Wdst = 2'b01;
			end
		endcase
	//TYPE C
	4'b1000:
	begin
		Imm = 1'b1;
	end
	4'b1001:
		Imm = 1'b1;
```

```
//TYPE B
//Load Byte
4'b1010:
begin
        OFFset = 1'b1;
        Mbyte = 1'b1;
        Down = 1'b1;
end
//Store Byte
4'b1011:
begin
        OFFset = 1'b1;
        MemW = 2'b01;
end
//Load Word
4'b1100:
begin
        OFFset = 1'b1;
        Down = 1'b1;
end
//Store Word
4'b1101:
begin
        OFFset = 1'b1;
        MemW = 2'b10;
end
//TYPE C
// BLT
4'b0101: Bra = 2'b01;
//BGT
4'b0100: Bra = 2'b10;
//BEQ
4'b0110: Bra = 2'b00;
```

```
    endcase


end
endmodule




module Forward(
    output reg[1:0] HighMUX,
    output reg[1:0] LowMUX,
    input [3:0] RT,
    input [3:0] RS,
    input [3:0] EMRD,
    input [3:0] MWRD,
    input [3:0]CTRLMEM,
    input [3:0]CTRLWB
);

//Combinatorial READ
always@(*)
begin

    HighMUX=2'b00;
    LowMUX=2'b00;
    if(CTRLMEM==4'h0)
    begin
            if(EMRD==RS)
            begin
                    HighMUX=2'b10;
            end
```

```
            if(EMRD==RT)
            begin
                LowMUX=2'b10;
            end
    end
    if(CTRLWB==4'h0)
    begin
            if(MWRD==RS)
            begin
                HighMUX=2'b01;
            end
            if(MWRD==RT)
            begin
                LowMUX=2'b01;
            end
    end
end
endmodule

module Gather(
    output reg[15:0] OUT,
    input WB,
    input [1:0]DEST,
    input MBY,
    input MEMF,
    input DOWN,
    input [1:0]MEMW,
    input FDST,
    input MV1,
    input OFFSET,
    input IMM,
    input [3:0] AOP
);
```

```
always@(*)
begin
   OUT[0]=WB;
   OUT[2:1]=DEST;
   OUT[3]=MBY;
   OUT[4]=MEMF;
   OUT[5]=DOWN;
   OUT[7:6]=MEMW;
   OUT[8]=FDST;
   OUT[9]=MV1;
   OUT[10]=OFFSET;
   OUT[11]=IMM;
   OUT[15:12]=AOP;

end
endmodule

module INS#(parameter SIZE=64, NS=15)
   (output reg[15:0] out,
   input [NS:0] in,
   input clk,
   input rst
   );
integer i;
reg[15:0] data [SIZE-1:0];

always@(posedge clk, negedge rst)
begin
   if(!rst)
   begin
         //ADD R1 R2
         data[0]=16'b1111000100100000; //data[0]=16'hF120;
         //SUB R1 R2
```

```
data[1]=16'b1111000100100001; // data[1] 16'hF121
//ORi R3 FF
data[2]=16'h93FF;
//ANDi R3 4C
data[3]=16'h834C;
//MUL R5 R6
data[4]=16'hF564;
//DIV R1 R5
data[5]=16'hF155;
//SUB R15 R15
data[6]=16'hFFF1;
//MOV R4, R8
data[7]=16'hF487;
//SWP R4 R6
data[8]=16'hF468;
//ORI R4 2
data[9]=16'h9402;
//LBU R6 4(R9)
data[10]=16'hA694;
//SB R6, 6(R9)
data[11]=16'hB696;
//LW R6 6(R9)
data[12]=16'hC696;
//BEQ R7 4
data[13]=16'h6704;
//ADD R11 R1
data[14]=16'hFB10;
//BLT R7 5
data[15]=16'h5705;
//ADD R11 R2
data[16]=16'hFB20;
//BGT R7 2
data[17]=16'h4702;
```

```
        //ADD R1 R1
        data[18]=16'hF110;
        //ADD R1 R1
        data[19]=16'hF110;
        //LW R8 0(R9)
        data[20]=16'hC890;
        //ADD R8 R8
        data[21]=16'hF880;
        //SW R8 2(R9)
        data[22]=16'hD892;
        //LW R10 2(R9)
        data[23]=16'hCA92;
        //ADD R12 R12
        data[24]=16'hFCC0;
        //SUB R13 R13
        data[25]=16'hFDD1;
        //ADD R12 R13
        data[26]=16'hFCD0;
        //EFFF
        data[27]=16'hEFFF;



end
else if(clk)
        //case(in)
        //    8'h00: out = data[0];
        //    8'h02: out = data[1];
        //    8'h04: out = data[2];
        //    8'h06: out = data[3];
        //    8'h08: out = data[4];
        //    8'h0A: out = data[5];
        //endcase
```

```
        out = data[in/2];
end
endmodule

module Memory#(parameter N=999)
(
   output reg[7:0] Byte,
   output reg[15:0] Word,
   input [15:0] Addr,
   input [15:0] WriteW,
   input [7:0] WriteB,
   input [1:0] MemW,
   input clk,
   input rst
);
integer inc;
reg[7:0] mem [99:0]; //reg[15:0] ?

//Combinatorial READ
always@(*)
begin
   Byte=mem[Addr+1];
   Word[15:8]=mem[Addr];
   Word[7:0]=mem[Addr+1];
end

//Sequential WRITE
always@(posedge clk, negedge rst)
begin
   if(!rst)
   begin
        for(inc=0;inc<N;inc=inc+1)
                mem[inc]<=8'h00;
```

```
            mem[16'h0000]<=8'h2B;
            mem[16'h0001]<=8'hCD;
            mem[16'h0002]<=8'h00;
            mem[16'h0003]<=8'h00;
            mem[16'h0004]<=8'h12;
            mem[16'h0005]<=8'h34;
            mem[16'h0006]<=8'hDE;
            mem[16'h0007]<=8'hAD;
            mem[16'h0008]<=8'hBE;
            mem[16'h0009]<=8'hEF;
    end
    if(MemW==2'b10)
    begin
            //store word at addr
            mem[Addr]<=WriteW[15:8];
            mem[Addr+1]<=WriteW[7:0];
    end
    else if(MemW==2'b01)
    begin
            //store byte at addr
            mem[Addr+1]<=WriteB;
    end

end
endmodule

module MU3x1#(parameter SIZE=15)
(
    output reg[SIZE:0] out,
    input [SIZE:0] a,
    input [SIZE:0] b,
    input ctrl
```

```
);
always@(*)
begin
   if(ctrl)
          out=a;
   else
          out=b;
end
endmodule

module MU4x1#(parameter SIZE=15)
(
   output reg[SIZE:0] out,
   input [SIZE:0] a,
   input [SIZE:0] b,
   input [SIZE:0] c,
   input[1:0] ctrl

);
always@(*)
begin
   if(ctrl==2'b00)
          out=a;
   else if(ctrl==2'b01)
          out=b;
   else if(ctrl==2'b10)
          out=c;
end
endmodule

module PC#(parameter N=15)
(
   output reg[N:0] out,
```

```
    input [N:0]add,
    input clk,
    input rst
);
always@(posedge clk, negedge rst)
begin
    if(!rst)
    begin
            out=16'h0000;
    end
    else
    begin
            out=add;
    end
end
endmodule

module Registers(
    output reg[15:0] Data1,
    output reg[15:0] Data2,
    output reg[15:0] Data15,
    input [3:0] ReadAdd1,
    input [3:0] ReadAdd2,
    input [15:0] WriteReg1,
    input [15:0] WriteReg2,
    input [15:0] WriteReg15,
    input [1:0] WriteDst,
    input clk,
    input rst
);
//15 Registers
reg[15:0]Reg [15:0];
integer count1,count2, inc;
```

```
//Combinatorial READ
always@(*)
begin
   Data1=Reg[ReadAdd1];
   //Read Register from Address 2
   Data2=Reg[ReadAdd2];
   //Read Register 15
   Data15=Reg[15];

end

//Sequential Write
always@(posedge clk, negedge rst)
begin
   if(!rst)
   begin
         for(inc=0;inc<16;inc=inc+1)
               Reg[inc]=16'h0000;
         Reg[1]=16'h0F00;
         Reg[2]=16'h0050;
         Reg[3]=16'hFF0F;
         Reg[4]=16'hF0FF;
         Reg[5]=16'h0040;
         Reg[6]=16'h6666;
         Reg[7]=16'h00FF;
         Reg[8]=16'hFF88;
         Reg[9]=16'h0000;
         Reg[10]=16'h0000;
         Reg[11]=16'h0000;
         Reg[12]=16'hCCCC;
         Reg[13]=16'h0002;

   end
```

```
    else if(WriteDst==2'b00)
            Reg[ReadAdd1]=WriteReg1;
    else if(WriteDst==2'b01)
    begin
            Reg[ReadAdd1]=WriteReg1;
            Reg[ReadAdd2]=WriteReg2;
    end
    else if(WriteDst==2'b10)
    begin
            Reg[ReadAdd1]=WriteReg1;
            Reg[15]=WriteReg15;
    end
end
endmodule

module Sign(
    output reg[15:0] out,
    input [3:0] in
);
always@(*)
begin
    out[3:0]=in;
    if(out[3]==1'b1) out[15:4]=12'hFFF;
    else if(out[3]==1'b0) out[15:4]=12'h000;
end

endmodule

module SignBr(
    output reg[15:0] out,
    input [7:0] in
);
always@(*)
```

```
begin
   out[7:0]=in;
   if(out[7]==1'b1) out[15:8]=8'hFF;
   else if(out[7]==1'b0) out[15:8]=8'h00;
end

endmodule

module SSL(
   output reg[15:0] Shifted,
   input [15:0] Unshifted
);

always@(*)
begin
   Shifted = Unshifted << 2;
end

endmodule

module XOR#(parameter N=1)
(
   output reg res,
   input [N:0] op1,
   input [N:0] op2
);
always@(*)
begin
   case(op1^op2)
         2'b00: res = 1'b1;
         default: res = 1'b0;
   endcase
```

```
end
endmodule

module Zero(
    output reg[15:0] out,
    input [7:0] in
);
always@(*)

begin
    out[7:0]=in;
    out[15:8]=12'h00;
end
endmodule

module HazardDetectionUnit(
    output reg PCWrite,
    output reg DWrite,
    output reg StallIE,
    input MemRead,
    input [7:0] IERT,
    input [7:0] IIRT,
    input [7:0] IIRS
);

//Combinatorial READ
always@(*)
begin
    if(MemRead && ((IERT==IIRS) || (IERT==IIRT)))
    begin
            PCWrite=1'b1;
            DWrite=1'b1;
```

```
        StallIE=1'b1;
    end
    else
    begin
        PCWrite=1'b0;
        DWrite=1'b0;
        StallIE=1'b0;
    end
end
Endmodule
```

Test Files

```
`include "ALU.v"

module ALU_fixture;

reg [15:0] dat1,dat2;
reg [3:0] op;
wire [15:0] up, low;
wire [1:0]ze;

initial
        $vcdpluson;
initial
        $monitor($time, "Data1 = %h, Data2 = %h, Upper = %b, Lower = %b, Zero =
%b",dat1,dat2, up, low, ze);

ALU a(up, low, ze, dat1, dat2, op);

initial
begin

   dat1=16'h0000;
   dat2=16'h0000;
   op=4'h0;
   // ADD 0F,01 = 10
   #5 dat1 = 16'h000F;
   dat2 = 16'h0001;
   op = 4'h0;
   //SUB 100, 0F = 910
   #10 dat1 = 16'h0A00;
   dat2 = 16'h00F0;
   op = 4'h1;
```

```
    #20 dat1 = 16'h00FF;
    dat2 = 16'hF0F0;
    op = 4'b1001;
    #40 dat1 = 16'hxxx;
    dat2 = 16'hxxx;
    op = 4'hx;

end
initial
begin

    #400 $finish;
end
endmodule


`include "BufferIDEX.v"

module BufferIDEX_Fixture;

wire [15:0] OD1;
wire [15:0] OD2;
wire [15:0] OD15;
wire [3:0] ORS;
wire [3:0] ORT1;
wire [3:0] ORT2;
wire [3:0] ORD;
wire [15:0] OC;
reg [15:0] ID1;
reg [15:0] ID2;
reg [15:0] ID15;
reg [3:0] IRS;
reg [3:0] IRT1;
```

```
reg [3:0] IRT2;
reg [3:0] IRD;
reg [15:0] IC;
reg C;
reg R;

initial
  $vcdpluson;

initial
  $monitor($time," InData1 = %h, InData2 = %h, InData15 = %h, InRS = %h, InRT = %h, InRT2
= %h, InRD = %h, InCtrl = %h\n\t\t OutData1 = %h, OutData2 = %h, OutData15 = %h, OutCtrl =
%h, OutRS = %h, OutRT = %h, OutRT2 = %h, OutRD = %h", ID1, ID2, ID15, IRS, IRT1, IRT2,
IRD,IC, OD1, OD2, OD15, OC, ORS, ORT1, ORT2, ORD);

BufferIDEX
a1(OD1,OD2,OD15,ORS,ORT1,ORT2,ORD,OC,ID1,ID2,ID15,IRS,IRT1,IRT2,IRD,IC,C,R);

initial
begin
   R=0;
   #10;
   R=1;

   //receive ctrl
   #10;
   //receive data 15
   IC=12'hFFF;
   ID15=16'hFFF0;
   #10;
   //receive data 1
   ID1=16'h0A01;
   #10;
```

```
    //receive data 2
    ID2=16'h00B3;
    #10;
    //receive D15
    ID15=16'h000F;
    #10;
    //receive RS
    IRS=4'hE;
    #10;
    //receive RT1
    IRT1=4'h5;
    IC=12'h000;
    #10;
    //receive RT2
    IRT2=4'h3;
    IC[0]=1'b0;
    #10;
    //receive RD
    IRD=4'h2;
    #10;

end

initial
begin
  C = 1'b0;
  forever #5 C = ~C;
end

initial
begin
  #200 $finish;
end
```

```
endmodule


`include "BufferEXMEM.v"

module BufferEXMEM_Fixture;

wire [15:0] OU;
wire [15:0] OL;
wire [15:0] OW;
wire [7:0] OB;
wire [3:0] FOR;
wire [3:0]OCW;
wire [3:0]OCM;
//
reg [15:0]IU;
reg [15:0]IL;
reg [15:0]IW;
reg [3:0] IFOR;
reg [7:0]IB;
reg [3:0]ICW;
reg [3:0]ICM;
//
reg C;
reg R;

initial
  $vcdpluson;

initial
  $monitor($time," IU %h IL %h IW %h IB %h ICW %h ICM %h IFOR %h OU %h OL %h OW
%h FOR %h OB %h OCW %h OCM %h", IU, IL, IW, IB, ICW, ICM, IFOR, OU, OL, OW, FOR,
OB, OCW, OCM);
```

```
BufferEXMEM a1(OU, OL, OW, OB,FOR, OCW, OCM, IU, IL, IW, IB, IFOR, ICW, ICM,C, R);

initial
begin
    R=0;
    #10;
    R=1;
    ICW=4'h1;
    ICM=4'h1;
    //receive instr
    IU=16'hF230;
    #10;
    //receive instr
    IL=16'hF400;
    IFOR=4'h5;
    #10;
    //receive instr
    IW=16'hF500;
    #10;
    //receive instr
    IB=8'hF6;
    IFOR=4'hF;
    #10;
    //receive instr
    ICM=4'h0;
          IB=8'hF6;
    #10;
    IL=16'hF400;
    IW=16'hF500;
    #10;
    ICM=4'h1;
    IL=16'hF400;
```

```
    #10;

end

initial
begin
  C = 1'b0;
  forever #5 C = ~C;
end

initial
begin
  #100 $finish;
end
endmodule


`include "BufferIFID.v"

module BufferIFID_Fixture;

wire [15:0] OI;
reg [15:0]II;
reg FLUSH;
reg C;
reg R;

initial
  $vcdpluson;

initial
  $monitor($time," InInstr = %h, OutInstr = %h, FLUSH =%b", II, OI, FLUSH);
```

```
BufferIFID a1(OI,II,FLUSH,C,R);

initial
begin
    R=0;
    FLUSH=1'b0;
    #10;
    R=1;

    //receive instr
    II=16'hF230;
    #10;
    //receive instr
    II=16'hF400;
    #10;
    //receive instr
    II=16'hF500;
    #10;
    //receive instr
    II=16'hF600;
    FLUSH=1'b1;
    #10;
    //receive instr
    II=16'hF700;
    #10;
end

initial
begin
  C = 1'b0;
  forever #5 C = ~C;
end
```

```
initial
begin
  #100 $finish;
end
endmodule


`include "BufferMEMWB.v"

module BufferMEMWB_Fixture;

wire [15:0] OW;
wire [7:0] OB;
wire [3:0] OF;
reg [15:0]IW;
reg [7:0]IB;
reg [3:0]IF;
reg [3:0]IC;
wire [3:0]OC;
reg C;
reg R;

initial
  $vcdpluson;

initial
  $monitor($time," InW = %h, InB = %h, InF = %h, IC=%b, OutW = %h, OutB = %h, OutF = %h,
OUT_WB = %h", IW, IB, IF,IC, OW, OB, OF, OC);

BufferMEMWB a1(OW,OB,OF,OC,IW,IB,IC,IF,C,R);

initial
begin
```

```
    R=0;
    #10;
    R=1;
    IC=1'b0;
    //receive word
    IW=16'hA237;
    #10;
    //receive byte
    IB=8'hF0;
    #10;
    //receive forward
    IF=4'hF;
    #10;
    //receive ctrl
    IC=1'b1;
    #10;
    //receive word
    IW=16'h8400;
    #10;
    IW=16'h8400;
    #10;
end

initial
begin
  C = 1'b0;
  forever #5 C = ~C;
end

initial
begin
  #100 $finish;
end
```

```
endmodule


`include "Forward.v"

module Forward_Fixture;

wire [1:0] HM;
wire [1:0] LM;
reg [3:0]RT;
reg [3:0]RS;
reg [3:0]EMRD;
reg [3:0]MWRD;
reg [3:0]CTRL1;
reg [3:0]CTRL2;


initial
  $vcdpluson;

initial
  $monitor($time," HM = %b, LM = %b, RT = %h, RS = %h, EMRD = %h, MWRD = %h, CTRL1
= %h, CTRL2 = %h", HM, LM, RT, RS, EMRD, MWRD, CTRL1, CTRL2);

Forward a1(HM,LM,RT,RS,EMRD,MWRD,CTRL1, CTRL2);

initial
begin
    CTRL1=4'h0;
    //receive locations
    RT=4'hA;
    RS=4'hB;
    EMRD=4'hC;
```

```
    MWRD=4'hD;
    #10;
    //receive locations
    RT=4'h8;
    RS=4'h9;
    EMRD=4'h3;
    MWRD=4'h8;
    #10;
    //receive locations
    CTRL2=4'h1;
    RT=4'hF;
    RS=4'hE;
    EMRD=4'hF;
    MWRD=4'hE;
    #10;
    //receive locations
    CTRL1=4'h2;
    RT=4'hF;
    RS=4'hE;
    EMRD=4'hE;
    MWRD=4'hF;
    #10;
    //receive locations
    CTRL2=4'h5;
    RT=4'hA;
    RS=4'hB;
    EMRD=4'hC;
    MWRD=4'hD;
    #10;
end

initial
begin
```

```
  #100 $finish;
end
endmodule


`include "PC.v"
`include "Adder2.v"
`include "MU3x1.v"
`include "INS.v"

module Count_fixture;

wire[15:0] inst;
wire[15:0] count, addr, n_add;
reg CLOCK, CLEAR, CTRL;

initial
        $vcdpluson;
initial
        $monitor($time, "PC = %b      add = %b   ctrl=%b     inst=%h", count, addr, CTRL, inst);
MU3x1 #(15)mux(n_add, count, addr, CTRL);
Adder2 up(addr, count);
PC pc(count, n_add, CLOCK, CLEAR);
INS ins(inst, count, CLOCK, CLEAR);
initial
begin
        CLEAR=1'b0;
    CTRL=1'b1;
        #20 CLEAR=1'b1;
    CTRL=1'b0;
    #50 CTRL=1'b1;
    #200 CTRL=1'b0;
end
```

```
initial
begin
        CLOCK=1'b0;
        forever #5 CLOCK=~CLOCK;
end
initial
begin
        #400 $finish;
end




endmodule


`include "Memory.v"

module Module_Fixture;

wire [7:0] B;
wire [15:0] W;
reg [15:0]A;
reg [15:0]WW;
reg [7:0]WB;
reg [1:0]MW;
reg C;
reg R;

initial
  $vcdpluson;
```

```
initial
  $monitor($time," Input W = %h, Input B = %h, Address = %h, Output W = %h, Output B = %h,
ctrl = %b", WW, WB, A, W, B, MW);

Memory a1(B,W,A,WW,WB,MW,C,R);

initial
begin
  R=0;
  #10;
  R=1;
//lw
  MW=2'b00;
  A=16'h0006;
  #10;
//lw
  A=16'h0008;
  #10;
//sb
  MW=2'b01;
  A=16'h0000;
  WB=8'h0F;
  #10;
//lb
  MW=2'b00;
  A=16'h0000;
  #10;
//sw
  MW=2'b10;
  A=16'h0006;
  WW=16'h000A;
  #10;
```

```
//lw
  MW=2'b00;
  A=16'h0006;
  #10;
//lb
  MW=2'b00;
  A=16'h0000;
end

initial
begin
  C = 1'b0;
  forever #10 C = ~C;
end

initial
begin
  #100 $finish;
end
endmodule


`include "Registers.v"
module Registers_fixture;

reg [3:0] add1, add2;
reg [1:0] dst;
wire [15:0] dat1, dat2, dat15, w1, w2, w15;
reg CLOCK, CLEAR;
initial
        $vcdpluson;
initial
        $monitor($time, "data1 = %h, data2 = %h, data15 = %h ", dat1, dat2, dat15);
```

```
Registers re(dat1, dat2, dat15, add1, add2, w1, w2, w15, dst, CLOCK, CLEAR);

initial
begin
   CLEAR=2'b0;
   dst=2'b11;
   #10 CLEAR=1'b1;
   add1 = 4'h1;
   add2 = 4'h2;
   #30 add1 = 4'h3;
   add2 = 4'h4;
end

initial
begin
        CLOCK=1'b0;
        forever #5 CLOCK=~CLOCK;
end
initial
begin
        #400 $finish;
end



endmodule


`include "PC.v"

module PC_fixture;
```

```
reg CLOCK, CLEAR;
reg [15:0]add;
wire [15:0]counter;

initial
   $vcdpluson;
initial
   $monitor($time, "PC = %b  add = %b ", counter, add);

PC pc(counter, add, CLOCK, CLEAR);
initial
begin
   CLEAR=1'b0;
   #30 CLEAR=1'b1;
   add=16'h0002;
   #50 add=16'h0004;
end
initial
begin
   CLOCK=1'b0;
   forever #5 CLOCK=~CLOCK;
end
initial
begin
   #100 $finish;
end
endmodule


`include "Sign.v"
module Sign_fixture;

reg [3:0] in;
```

```
wire [15:0] out;
initial
        $vcdpluson;
initial
        $monitor($time, " in = %b out = %b", in, out);

Sign s(out, in);

initial
begin
   in = 4'h0;
   #10 in = 8'h1;
   #20 in = 8'hF;
end

initial
begin
        #100 $finish;
end


endmodule

`include "XOR.v"

module XOR_fixture;

reg[1:0] in1, in2;
wire out;

initial
        $vcdpluson;
initial
```

```
        $monitor($time, " in1 = %b, in2 = %b , out = %b ", in1, in2, out);

XOR x(out, in1, in2);

initial
begin
   in1=2'b00;
   in2=2'b00;
   #5 in1=2'b01;
        in2=2'b00;
   #10 in1=2'b10;
        in2=2'b00;
   #15 in1=2'b11;
        in2=2'b00;
   #20 in1=2'b00;
        in2=2'b01;
   #25 in1=2'b01;
        in2=2'b01;
   #30 in1=2'b10;
        in2=2'b01;
        #40 in1=2'b11;
        in2=2'b01;
        #50 in1=2'b00;
        in2=2'b10;
        #60 in1=2'b01;
        in2=2'b10;
        #70 in1=2'b10;
        in2=2'b10;
        #80 in1=2'b11;
        in2=2'b10;
        #90 in1=2'b00;
        in2=2'b11;
        #100 in1=2'b01;
```

```
        in2=2'b11;
        #110 in1=2'b10;
        in2=2'b11;
        #120 in1=2'b11;
        in2=2'b11;
end
endmodule

`include "Zero.v"
module Sign_fixture;

reg [7:0] in;
wire [15:0] out;
initial
        $vcdpluson;
initial
        $monitor($time, " in = %b out = %b", in, out);

Zero s(out, in);

initial
begin
   in = 8'h00;
   #10 in = 8'h01;
   #20 in = 8'hFF;
end

initial
begin
        #100 $finish;
end
```

```
endmodule


`include "HazardDetectionUnit.v"

module HazardDetectionUnit_Fixture;

wire PCW;
wire DW;
wire SIE;
reg  MR;
reg [7:0] IERT;
reg [7:0] IIRT;
reg [7:0] IIRS;

initial
  $vcdpluson;

initial
  $monitor($time," IERT = %h IIRT = %h IIRS = %h PCW = %h DW = %h SIE = %h MR = %h",
IERT, IIRT, IIRS, PCW, DW, SIE, MR);

HazardDetectionUnit a1(PCW,DW,SIE,MR,IERT,IIRT,IIRS);

initial
begin
    //receive bad signal
    MR=1'b1;
    IERT=8'h0F;
    IIRS=8'h0F;
    #10;
    //receive good signal
    MR=1'b0;
```

```
    IERT=8'h0C;
    IIRS=8'h0C;
    #10;
    //receive bad signal
    MR=1'b1;
    IERT=8'h03;
    IIRT=8'h03;
    #10;
    //receive good signal
    MR=1'b0;
    IERT=8'h1C;
    IIRS=8'h09;
    #10;
    //receive good signal
    MR=1'b1;
    #10;
end

initial
begin
  #100 $finish;
end
endmodule


`include "AdderBr.v"

module AdderBr_Fixture;

wire [7:0] N;
reg [7:0] A;
reg [7:0] O;
```

```
initial
  $vcdpluson;

initial
  $monitor($time," Addr = %h, Offset = %h, Out = %h", A, O, N);

AdderBr a1(N,A,O);

initial
begin
    A = 8'h01;
    O = 8'h0F;
    #10;
    A = 8'h20;
    O = 8'h18;
    #10;
end

initial
begin
  #100 $finish;
end
endmodule
```

Test Assembly Code

Register	Content
======	======
R1	0F00
R2	0050
R3	FF0F
R4	F0FF
R5	0040
R6	6666
R7	00FF
R8	FF88
R9	0000
R10	0000
R11	0000
R12	CCCC
R13	0002

Others 0000

Instruction Memory

Address	Content
======	======
00	ADD R1, R2
02	SUB R1, R2
04	ORi R3, FF
06	ANDi R3, 4C
08	MUL R5, R6
0A	DIV R1, R5
0C	SUB R15, R15
0E	MOV R4, R8
10	SWP R4, R6

12 ORi R4, 2
14 LBU R6, 4(R9)
16 SB R6, 6(R9)
18 LW R6, 6(R9)
1A BEQ R7, 4
1C ADD R11, R1
1E BLT R7, 5
20 ADD R11, R2
22 BGT R7, 2
24 ADD R1, R1
26 ADD R1, R1
28 LW R8, 0(R9)
2A ADD R8, R8
2C SW R8, 2 (R9)
2E LW R10, 2 (R9)
30 ADD R12, R12
32 SUB R13, R13
34 ADD R12, R13
36 EFFF

Other memory locations = 0000

Data Memory

======

00 2BCD
02 0000
04 1234
06 DEAD

08 BEEF

Other memory locations = 0000

Stimulus Module

```
`include "cpu.v"
module cpu_fixture;

wire[15:0] inst_b, inst, data1, data2, data15, data1_b, data2_b, data15_b, low, up, low_b, up_b,
z_ex, s_ex, src1, src2, src1_sel, src2_sel;
wire[15:0] w1, word, word_b, w15, mm, mmb, dw, w_word;
wire[15:0] count, addr, n_add, b_ex, b_ssl, n_br,n_pc;
wire[7:0] byte, byte_b, w_byte;
wire [3:0] op, rt_b, rt2_b, rs_b, rd_b, f2_out,f3_out, f4_out;
wire[1:0] ze, cm;
reg CLOCK, CLEAR;
wire HALT, OFFSET, IMM, MV1, MBYTE, DOWN, BR, FORIN, IFFLUSH,EXFLUSH, DEST;
wire OFFSET_B, IMM_B, MV1_B, MBYTE_B, MBYTE_IN, DOWN_B, FDST_B;
reg HAL;
wire [1:0]WRITEDST, WRITEDST_B, MEMW, MEMW_B,MEMW_IN, BRANCH, IDCTRL,
HMUX, LMUX;
wire [1:0]IDFLUSH;
wire[15:0]IN, OUT;
wire[3:0] WB_OUT, MEM_OUT,OP_B, WB, WBC;

initial
        $vcdpluson;
initial
$monitor($time, "  PROGRAM COUNTER:  PC = %h, New_PC = %h Branch = %h\n\t\t
INSTRUCTION: %h, Read.op1 = %h, Read.op2 = %h\n\t\t ALU: Lower = %h, op1 = %h, op2 =
%h, opcode = %b, func = %b,\n\t\t REGISTERS: WriteReg1 = %h, WriteReg2 = %h,
WriteReg15 = %h, Wdst = %b \n\t\t MEMORY: Byte = %h, Word = %h\n\n\t\t INPUT IF/ID : INST
= %h, IFFLUSH = %b OUTPUT INST = %h \n\t\tINPUT ID/EX_BUFFER : data1 = %h, data2 =
%h, data15 = %h \n\t\t OUTPUT ID/EX : data1 = %h, data2 =%h, data15 = %h, func = %b\n\t\t
INPUT EXMEM : Lower = %h, Upper = %h, Word %h, Byte = %h \n\t\t OUTPUT EXMEM:
Lower = %h, Upper = %h, Word = %h, Byte = %h\n\t\t INPUT MEMWB: Word = %h, Byte = %h,
```

```
OUTPUT: Word = %h, Byte = %h\n\t\t ID/EX.FOR = %h, EX/MEM.FOR = %h, MEM/WB.FOR =
%h\n\t\t HMUX = %h, LMUX = %h\n", count, n_add, n_br, inst_b, inst[11:8], inst[7:4], low, src1,
src2, OUT[15:12], op, w1, data1, w15,  WRITEDST, byte, word, inst_b, IFFLUSH, inst ,data1_b,
data2_b, data15_b, data1, data2, data15, OUT[15:12], low_b, up_b, data1, data1[7:0], low, w15,
w_word, w_byte, w_word, w_byte, word, byte, f2_out, f3_out, f4_out, HMUX, LMUX);



MU3x1 #(15)mux(n_add, count, addr, HAL);
MU3x1 #(15)off(src1_sel, s_ex, data1, OUT[10]);
MU3x1 #(15)imm(src2_sel, z_ex, data2, OUT[11]);
MU3x1 #(15)mby(mm, mmb, word, WB[3]);
MU3x1 #(15)dwn(dw, mm, low, MEM_OUT[1]);
MU3x1 #(15)mv1(w1, dw, data2, OUT[9]);
MU3x1 #(15)bran(n_pc, n_br, n_add, BR);
MU3x1 #(3)forw(f2_out,rd_b, rt_b, FORIN);
MU4x1 sel(src1,src1_sel,w15,dw, HMUX);
MU4x1 sel2(src2,src2_sel, w15, dw, LMUX);

CMP cmp(cm,data1, data15);
XOR x(BR, cm, BRANCH);

BufferIFID buf_ifid(inst, inst_b, IFFLUSH, CLOCK, CLEAR);
BufferIDEX buf_idex(data1, data2, data15, rs_b, rt_b, rt2_b, rd_b, OUT, data1_b, data2_b,
data15_b, inst[11:8], inst[7:4], inst[3:0], inst[3:0], IN, CLOCK, CLEAR);
BufferEXMEM buf_exmem(w15, low, w_word, w_byte, f3_out, WB_OUT, MEM_OUT, up_b,
low_b, data1, data1[7:0], f2_out, OUT[3:0], OUT[7:4], CLOCK, CLEAR);
BufferMEMWB buf_memwb(word, byte, f4_out, WB, word_b, byte_b, WB_OUT, f3_out,
CLOCK, CLEAR);

Gather g(IN, WB_B, WRITEDST_B, MBYTE_B, MEM_B, DOWN_B, MEMW_B, FDST_B,
MV1_B, OFFSET_B, IMM_B, OP_B);
Forward frw(HMUX, LMUX, rt_b, rs_b, f3_out, f4_out, WB_OUT,WB );
```

```
Zero z(z_ex, inst[7:0]);
Zero m_z(mmb, byte);
Sign s(s_ex, inst[3:0]);
SignBr br_s(b_ex, inst[7:0]);
SSL br_ssl(b_ssl, b_ex);
Adder2 ad(addr, count);
AdderBr br(n_br,count,b_ssl);


PC pc(count, n_pc, CLOCK, CLEAR);
INS ins(inst_b, count, CLOCK, CLEAR);
Registers re(data1_b, data2_b, data15_b, inst[11:8], inst[7:4], w1, data1, w15, WB[2:1],
CLOCK, CLEAR);
ALU a(up_b, low_b, ze, src1, src2, OUT[15:12]);
ALU_CONTROL ctrl_a(OP_B, inst[15:12], inst[3:0]);
CONTROL ctrl(OFFSET_B, IMM_B, DOWN_B ,MBYTE_B, MV1_B, HALT, FDST_B, BRANCH,
WRITEDST, MEMW_B,MEM_B, WB_B, IFFLUSH, IDFLUSH, EXFLUSH,  inst[15:12], inst[3:0]);
Memory mem(byte_b, word_b, low, w_word, w_byte, MEM_OUT[3:2], CLOCK, CLEAR);

initial
begin
        CLEAR=1'b0;
   HAL=1'b1;
   #5 CLEAR=1'b1;
   HAL=1'b0;
end
initial
begin
        CLOCK=1'b0;
        forever #5 CLOCK=~CLOCK;
end
```

```
initial
begin
        #300 $finish;

end

endmodule
```

Expected Results

Chronologic VCS simulator copyright 1991-2014

Contains Synopsys proprietary information.

Compiler version J-2014.12-SP3_Full64; Runtime version J-2014.12-SP3_Full64; Dec 5 21:25 2017

VCD+ Writer J-2014.12-SP3_Full64 Copyright (c) 1991-2014 by Synopsys Inc.

0 PROGRAM COUNTER: PC = 0000, New_PC = 0000 Branch = xxxx

INSTRUCTION: xxxx, Read.op1 = x, Read.op2 = x

ALU: Lower = xxxx, op1 = xxxx, op2 = xxxx, opcode = xxxx, func = zzzz,

REGISTERS: WriteReg1 = xxxx, WriteReg2 = xxxx, WriteReg15 = xxxx, Wdst = xx

MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = xxxx, IFFLUSH = x OUTPUT INST = xxxx

INPUT ID/EX_BUFFER : data1 = xxxx, data2 = xxxx, data15 = 0000

OUTPUT ID/EX : data1 = xxxx, data2 =xxxx, data15 = xxxx, func = xxxx

INPUT EXMEM : Lower = xxxx, Upper = xxxx, Word xxxx, Byte = xx

OUTPUT EXMEM: Lower = xxxx, Upper = xxxx, Word = xxxx, Byte = xx

INPUT MEMWB: Word = xxxx, Byte = xx, OUTPUT: Word = xxxx, Byte = xx

ID/EX.FOR = x, EX/MEM.FOR = x, MEM/WB.FOR = x

HMUX = x, LMUX = x

5 PROGRAM COUNTER: PC = 0000, New_PC = 0002 Branch = xxxx

INSTRUCTION: f120, Read.op1 = x, Read.op2 = x

ALU: Lower = xxxx, op1 = xxxx, op2 = xxxx, opcode = xxxx, func = zzzz,

REGISTERS: WriteReg1 = xxxx, WriteReg2 = xxxx, WriteReg15 = xxxx, Wdst = xx

MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f120, IFFLUSH = x OUTPUT INST = xxxx

INPUT ID/EX_BUFFER : data1 = xxxx, data2 = xxxx, data15 = 0000

OUTPUT ID/EX : data1 = xxxx, data2 =xxxx, data15 = 0000, func = xxxx

INPUT EXMEM : Lower = xxxx, Upper = xxxx, Word xxxx, Byte = xx

OUTPUT EXMEM: Lower = xxxx, Upper = xxxx, Word = xxxx, Byte = xx

INPUT MEMWB: Word = xxxx, Byte = xx, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = x, EX/MEM.FOR = x, MEM/WB.FOR = x
HMUX = x, LMUX = x

15 PROGRAM COUNTER: PC = 0002, New_PC = 0004 Branch = 0082
INSTRUCTION: f121, Read.op1 = 1, Read.op2 = 2
ALU: Lower = xxxx, op1 = xxxx, op2 = xxxx, opcode = xxxx, func = zzzz,
REGISTERS: WriteReg1 = xxxx, WriteReg2 = xxxx, WriteReg15 = xxxx, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f121, IFFLUSH = 1 OUTPUT INST = f120
INPUT ID/EX_BUFFER : data1 = 0f00, data2 = 0050, data15 = 0000
OUTPUT ID/EX : data1 = xxxx, data2 =xxxx, data15 = 0000, func = xxxx
INPUT EXMEM : Lower = xxxx, Upper = xxxx, Word xxxx, Byte = xx
OUTPUT EXMEM: Lower = xxxx, Upper = xxxx, Word = xxxx, Byte = xx
INPUT MEMWB: Word = xxxx, Byte = xx, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = x, EX/MEM.FOR = x, MEM/WB.FOR = x
HMUX = x, LMUX = x

25 PROGRAM COUNTER: PC = 0004, New_PC = 0006 Branch = 0088
INSTRUCTION: 93ff, Read.op1 = 1, Read.op2 = 2
ALU: Lower = xxxx, op1 = 0f00, op2 = 0050, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = xxxx, WriteReg2 = 0f00, WriteReg15 = xxxx, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = 93ff, IFFLUSH = 1 OUTPUT INST = f121
INPUT ID/EX_BUFFER : data1 = 0f00, data2 = 0050, data15 = 0000
OUTPUT ID/EX : data1 = 0f00, data2 =0050, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0f50, Upper = xxxx, Word 0f00, Byte = 00
OUTPUT EXMEM: Lower = xxxx, Upper = xxxx, Word = xxxx, Byte = xx
INPUT MEMWB: Word = xxxx, Byte = xx, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 2, EX/MEM.FOR = x, MEM/WB.FOR = x
HMUX = 0, LMUX = 0

35 PROGRAM COUNTER: PC = 0006, New_PC = 0008 Branch = 0002
INSTRUCTION: 834c, Read.op1 = 3, Read.op2 = f
ALU: Lower = 0f50, op1 = 0f00, op2 = 0050, opcode = 0001, func = zzzz,
REGISTERS: WriteReg1 = 0f50, WriteReg2 = 0f00, WriteReg15 = xxxx, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = 834c, IFFLUSH = 1 OUTPUT INST = 93ff
INPUT ID/EX_BUFFER : data1 = ff0f, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 0f00, data2 =0050, data15 = 0000, func = 0001
INPUT EXMEM : Lower = 0eb0, Upper = xxxx, Word 0f00, Byte = 00
OUTPUT EXMEM: Lower = 0f50, Upper = xxxx, Word = 0f00, Byte = 00
INPUT MEMWB: Word = 0f00, Byte = 00, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 2, EX/MEM.FOR = 2, MEM/WB.FOR = 2
HMUX = 0, LMUX = 0

45 PROGRAM COUNTER: PC = 0008, New_PC = 000a Branch = 0138
INSTRUCTION: f564, Read.op1 = 3, Read.op2 = 4
ALU: Lower = 0eb0, op1 = ff0f, op2 = 004c, opcode = 1011, func = zzzz,
REGISTERS: WriteReg1 = 0eb0, WriteReg2 = ff0f, WriteReg15 = xxxx, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f564, IFFLUSH = 1 OUTPUT INST = 834c
INPUT ID/EX_BUFFER : data1 = ff0f, data2 = f0ff, data15 = 0000
OUTPUT ID/EX : data1 = ff0f, data2 =0000, data15 = 0000, func = 1011
INPUT EXMEM : Lower = ff4f, Upper = xxxx, Word ff0f, Byte = 0f
OUTPUT EXMEM: Lower = 0eb0, Upper = xxxx, Word = 0f00, Byte = 00
INPUT MEMWB: Word = 0f00, Byte = 00, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = f, EX/MEM.FOR = 2, MEM/WB.FOR = 2
HMUX = 0, LMUX = 0

55 PROGRAM COUNTER: PC = 000a, New_PC = 000c Branch = 019a
INSTRUCTION: f155, Read.op1 = 5, Read.op2 = 6

ALU: Lower = ff4f, op1 = ff0f, op2 = 0064, opcode = 1001, func = zzzz,
REGISTERS: WriteReg1 = ff4f, WriteReg2 = ff0f, WriteReg15 = xxxx, Wdst = 10
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f155, IFFLUSH = 1 OUTPUT INST = f564
INPUT ID/EX_BUFFER : data1 = 0040, data2 = 6666, data15 = 0000
OUTPUT ID/EX : data1 = ff0f, data2 =f0ff, data15 = 0000, func = 1001
INPUT EXMEM : Lower = 0004, Upper = xxxx, Word ff0f, Byte = 0f
OUTPUT EXMEM: Lower = ff4f, Upper = xxxx, Word = ff0f, Byte = 0f
INPUT MEMWB: Word = ff0f, Byte = 0f, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 4, EX/MEM.FOR = f, MEM/WB.FOR = f
HMUX = 0, LMUX = 0

65 PROGRAM COUNTER: PC = 000c, New_PC = 000e Branch = 0160
INSTRUCTION: fff1, Read.op1 = 1, Read.op2 = 5
ALU: Lower = 0004, op1 = 0040, op2 = 6666, opcode = 0100, func = zzzz,
REGISTERS: WriteReg1 = 0004, WriteReg2 = 0040, WriteReg15 = xxxx, Wdst = 10
MEMORY: Byte = 34, Word = 1234

INPUT IF/ID : INST = fff1, IFFLUSH = 1 OUTPUT INST = f155
INPUT ID/EX_BUFFER : data1 = 0f00, data2 = 0040, data15 = 0000
OUTPUT ID/EX : data1 = 0040, data2 =6666, data15 = 0000, func = 0100
INPUT EXMEM : Lower = 9980, Upper = xxxx, Word 0040, Byte = 40
OUTPUT EXMEM: Lower = 0004, Upper = xxxx, Word = ff0f, Byte = 0f
INPUT MEMWB: Word = ff0f, Byte = 0f, OUTPUT: Word = 1234, Byte = 34
ID/EX.FOR = 6, EX/MEM.FOR = 4, MEM/WB.FOR = 4
HMUX = 0, LMUX = 0

75 PROGRAM COUNTER: PC = 000e, New_PC = 0010 Branch = ffd2
INSTRUCTION: f487, Read.op1 = f, Read.op2 = f
ALU: Lower = 9980, op1 = 0f00, op2 = 0040, opcode = 0101, func = zzzz,
REGISTERS: WriteReg1 = 9980, WriteReg2 = 0f00, WriteReg15 = xxxx, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f487, IFFLUSH = 1 OUTPUT INST = fff1
INPUT ID/EX_BUFFER : data1 = 0000, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 0f00, data2 =0040, data15 = 0000, func = 0101
INPUT EXMEM : Lower = 003c, Upper = 0000, Word 0f00, Byte = 00
OUTPUT EXMEM: Lower = 9980, Upper = xxxx, Word = 0040, Byte = 40
INPUT MEMWB: Word = 0040, Byte = 40, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 5, EX/MEM.FOR = 6, MEM/WB.FOR = 6
HMUX = 0, LMUX = 0

85 PROGRAM COUNTER: PC = 0010, New_PC = 0012 Branch = fe2c
INSTRUCTION: f468, Read.op1 = 4, Read.op2 = 8
ALU: Lower = 003c, op1 = 0000, op2 = 0000, opcode = 0001, func = zzzz,
REGISTERS: WriteReg1 = 003c, WriteReg2 = 0000, WriteReg15 = 0000, Wdst = 00
MEMORY: Byte = 00, Word = 0000

INPUT IF/ID : INST = f468, IFFLUSH = 1 OUTPUT INST = f487
INPUT ID/EX_BUFFER : data1 = f0ff, data2 = ff88, data15 = 0000
OUTPUT ID/EX : data1 = 0000, data2 =0000, data15 = 0000, func = 0001
INPUT EXMEM : Lower = 0000, Upper = 0000, Word 0000, Byte = 00
OUTPUT EXMEM: Lower = 003c, Upper = 0000, Word = 0f00, Byte = 00
INPUT MEMWB: Word = 0f00, Byte = 00, OUTPUT: Word = 0000, Byte = 00
ID/EX.FOR = f, EX/MEM.FOR = 5, MEM/WB.FOR = 5
HMUX = 0, LMUX = 0

95 PROGRAM COUNTER: PC = 0012, New_PC = 0014 Branch = 01b2
INSTRUCTION: 9402, Read.op1 = 4, Read.op2 = 6
ALU: Lower = 0000, op1 = f0ff, op2 = ff88, opcode = 0111, func = zzzz,
REGISTERS: WriteReg1 = ff88, WriteReg2 = f0ff, WriteReg15 = 0000, Wdst = 01
MEMORY: Byte = cd, Word = 2bcd

INPUT IF/ID : INST = 9402, IFFLUSH = 1 OUTPUT INST = f468
INPUT ID/EX_BUFFER : data1 = f0ff, data2 = 6666, data15 = 0000

OUTPUT ID/EX : data1 = f0ff, data2 =ff88, data15 = 0000, func = 0111
INPUT EXMEM : Lower = ff88, Upper = 0000, Word f0ff, Byte = ff
OUTPUT EXMEM: Lower = 0000, Upper = 0000, Word = 0000, Byte = 00
INPUT MEMWB: Word = 0000, Byte = 00, OUTPUT: Word = 2bcd, Byte = cd
ID/EX.FOR = 8, EX/MEM.FOR = f, MEM/WB.FOR = f
HMUX = 0, LMUX = 0

105 PROGRAM COUNTER: PC = 0014, New_PC = 0016 Branch = 001c
INSTRUCTION: a694, Read.op1 = 4, Read.op2 = 0
ALU: Lower = ff88, op1 = f0ff, op2 = 6666, opcode = 1000, func = zzzz,
REGISTERS: WriteReg1 = 6666, WriteReg2 = f0ff, WriteReg15 = 0000, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = a694, IFFLUSH = 1 OUTPUT INST = 9402
INPUT ID/EX_BUFFER : data1 = f0ff, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = f0ff, data2 =6666, data15 = 0000, func = 1000
INPUT EXMEM : Lower = 6666, Upper = f0ff, Word f0ff, Byte = ff
OUTPUT EXMEM: Lower = ff88, Upper = 0000, Word = f0ff, Byte = ff
INPUT MEMWB: Word = f0ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 6, EX/MEM.FOR = 8, MEM/WB.FOR = 8
HMUX = 0, LMUX = 0

115 PROGRAM COUNTER: PC = 0016, New_PC = 0018 Branch = fe66
INSTRUCTION: b696, Read.op1 = 6, Read.op2 = 9
ALU: Lower = 6666, op1 = f0ff, op2 = 0094, opcode = 1011, func = zzzz,
REGISTERS: WriteReg1 = 6666, WriteReg2 = f0ff, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = b696, IFFLUSH = 1 OUTPUT INST = a694
INPUT ID/EX_BUFFER : data1 = 6666, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = f0ff, data2 =0000, data15 = 0000, func = 1011
INPUT EXMEM : Lower = f0ff, Upper = f0ff, Word f0ff, Byte = ff
OUTPUT EXMEM: Lower = 6666, Upper = f0ff, Word = f0ff, Byte = ff

INPUT MEMWB: Word = f0ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 0, EX/MEM.FOR = 6, MEM/WB.FOR = 6
HMUX = 0, LMUX = 0

125 PROGRAM COUNTER: PC = 0018, New_PC = 001a Branch = fe70
INSTRUCTION: c696, Read.op1 = 6, Read.op2 = 9
ALU: Lower = f0ff, op1 = 0006, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = f0ff, WriteReg2 = 6666, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = c696, IFFLUSH = 1 OUTPUT INST = b696
INPUT ID/EX_BUFFER : data1 = 6666, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 6666, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0006, Upper = f0ff, Word 6666, Byte = 66
OUTPUT EXMEM: Lower = f0ff, Upper = f0ff, Word = f0ff, Byte = ff
INPUT MEMWB: Word = f0ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 9, EX/MEM.FOR = 0, MEM/WB.FOR = 0
HMUX = 0, LMUX = 0

135 PROGRAM COUNTER: PC = 001a, New_PC = 001c Branch = fe72
INSTRUCTION: 6704, Read.op1 = 6, Read.op2 = 9
ALU: Lower = 0006, op1 = 0006, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 00ad, WriteReg2 = 6666, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = ad, Word = dead

INPUT IF/ID : INST = 6704, IFFLUSH = 1 OUTPUT INST = c696
INPUT ID/EX_BUFFER : data1 = 6666, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 6666, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0006, Upper = f0ff, Word 6666, Byte = 66
OUTPUT EXMEM: Lower = 0006, Upper = f0ff, Word = 6666, Byte = 66
INPUT MEMWB: Word = 6666, Byte = 66, OUTPUT: Word = dead, Byte = ad
ID/EX.FOR = 9, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

145 PROGRAM COUNTER: PC = 001c, New_PC = 001e Branch = 002c
INSTRUCTION: fb10, Read.op1 = 7, Read.op2 = 0
ALU: Lower = 0006, op1 = 0004, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 0006, WriteReg2 = 6666, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = ad, Word = dead

INPUT IF/ID : INST = fb10, IFFLUSH = 1 OUTPUT INST = 6704
INPUT ID/EX_BUFFER : data1 = 00ff, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 6666, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0004, Upper = f0ff, Word 6666, Byte = 66
OUTPUT EXMEM: Lower = 0006, Upper = f0ff, Word = 6666, Byte = 66
INPUT MEMWB: Word = 6666, Byte = 66, OUTPUT: Word = dead, Byte = ad
ID/EX.FOR = 9, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

155 PROGRAM COUNTER: PC = 001e, New_PC = 0020 Branch = 005e
INSTRUCTION: 5705, Read.op1 = b, Read.op2 = 1
ALU: Lower = 0004, op1 = 00ff, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 1234, WriteReg2 = 00ff, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = 34, Word = 1234

INPUT IF/ID : INST = 5705, IFFLUSH = 1 OUTPUT INST = fb10
INPUT ID/EX_BUFFER : data1 = 0000, data2 = 0f00, data15 = 0000
OUTPUT ID/EX : data1 = 00ff, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 00ff, Upper = f0ff, Word 00ff, Byte = ff
OUTPUT EXMEM: Lower = 0004, Upper = f0ff, Word = 6666, Byte = 66
INPUT MEMWB: Word = 6666, Byte = 66, OUTPUT: Word = 1234, Byte = 34
ID/EX.FOR = 0, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

165 PROGRAM COUNTER: PC = 0020, New_PC = 0022 Branch = 0034
INSTRUCTION: fb20, Read.op1 = 7, Read.op2 = 0

ALU: Lower = 00ff, op1 = 0000, op2 = 0f00, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 00ff, WriteReg2 = 0000, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = fb20, IFFLUSH = 1 OUTPUT INST = 5705
INPUT ID/EX_BUFFER : data1 = 00ff, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 0000, data2 =0f00, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0f00, Upper = f0ff, Word 0000, Byte = 00
OUTPUT EXMEM: Lower = 00ff, Upper = f0ff, Word = 00ff, Byte = ff
INPUT MEMWB: Word = 00ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 1, EX/MEM.FOR = 0, MEM/WB.FOR = 0
HMUX = 0, LMUX = 0

175 PROGRAM COUNTER: PC = 0022, New_PC = 0024 Branch = 00a2
INSTRUCTION: 4702, Read.op1 = b, Read.op2 = 2
ALU: Lower = 0f00, op1 = 00ff, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 0f00, WriteReg2 = 00ff, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = 4702, IFFLUSH = 1 OUTPUT INST = fb20
INPUT ID/EX_BUFFER : data1 = 0000, data2 = 0050, data15 = 0000
OUTPUT ID/EX : data1 = 00ff, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 00ff, Upper = f0ff, Word 00ff, Byte = ff
OUTPUT EXMEM: Lower = 0f00, Upper = f0ff, Word = 0000, Byte = 00
INPUT MEMWB: Word = 0000, Byte = 00, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 0, EX/MEM.FOR = 1, MEM/WB.FOR = 1
HMUX = 0, LMUX = 0

185 PROGRAM COUNTER: PC = 0024, New_PC = 0026 Branch = 002c
INSTRUCTION: f110, Read.op1 = 7, Read.op2 = 0
ALU: Lower = 00ff, op1 = 0000, op2 = 0050, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 00ff, WriteReg2 = 0000, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f110, IFFLUSH = 1 OUTPUT INST = 4702
INPUT ID/EX_BUFFER : data1 = 00ff, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 0000, data2 =0050, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0050, Upper = f0ff, Word 0000, Byte = 00
OUTPUT EXMEM: Lower = 00ff, Upper = f0ff, Word = 00ff, Byte = ff
INPUT MEMWB: Word = 00ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 2, EX/MEM.FOR = 0, MEM/WB.FOR = 0
HMUX = 0, LMUX = 0

195 PROGRAM COUNTER: PC = 0026, New_PC = 0028 Branch = 0066
INSTRUCTION: f110, Read.op1 = 1, Read.op2 = 1
ALU: Lower = 0050, op1 = 00ff, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 0050, WriteReg2 = 00ff, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = 00, Word = 0000

INPUT IF/ID : INST = f110, IFFLUSH = 1 OUTPUT INST = f110
INPUT ID/EX_BUFFER : data1 = 0f00, data2 = 0f00, data15 = 0000
OUTPUT ID/EX : data1 = 00ff, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 00ff, Upper = f0ff, Word 00ff, Byte = ff
OUTPUT EXMEM: Lower = 0050, Upper = f0ff, Word = 0000, Byte = 00
INPUT MEMWB: Word = 0000, Byte = 00, OUTPUT: Word = 0000, Byte = 00
ID/EX.FOR = 0, EX/MEM.FOR = 2, MEM/WB.FOR = 2
HMUX = 0, LMUX = 0

205 PROGRAM COUNTER: PC = 0028, New_PC = 002a Branch = 0068
INSTRUCTION: c890, Read.op1 = 1, Read.op2 = 1
ALU: Lower = 00ff, op1 = 0f00, op2 = 0f00, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 00ff, WriteReg2 = 0f00, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = c890, IFFLUSH = 1 OUTPUT INST = f110
INPUT ID/EX_BUFFER : data1 = 0f00, data2 = 0f00, data15 = 0000

OUTPUT ID/EX : data1 = 0f00, data2 =0f00, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 1e00, Upper = f0ff, Word 0f00, Byte = 00
OUTPUT EXMEM: Lower = 00ff, Upper = f0ff, Word = 00ff, Byte = ff
INPUT MEMWB: Word = 00ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 1, EX/MEM.FOR = 0, MEM/WB.FOR = 0
HMUX = 0, LMUX = 0

215 PROGRAM COUNTER: PC = 002a, New_PC = 002c Branch = fe6a
INSTRUCTION: f880, Read.op1 = 8, Read.op2 = 9
ALU: Lower = 1e00, op1 = 0f00, op2 = 0f00, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 1e00, WriteReg2 = 0f00, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f880, IFFLUSH = 1 OUTPUT INST = c890
INPUT ID/EX_BUFFER : data1 = ff88, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 0f00, data2 =0f00, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 1e00, Upper = f0ff, Word 0f00, Byte = 00
OUTPUT EXMEM: Lower = 1e00, Upper = f0ff, Word = 0f00, Byte = 00
INPUT MEMWB: Word = 0f00, Byte = 00, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 1, EX/MEM.FOR = 1, MEM/WB.FOR = 1
HMUX = 0, LMUX = 0

225 PROGRAM COUNTER: PC = 002c, New_PC = 002e Branch = fe2c
INSTRUCTION: d892, Read.op1 = 8, Read.op2 = 8
ALU: Lower = 1e00, op1 = 0000, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 1e00, WriteReg2 = ff88, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = d892, IFFLUSH = 1 OUTPUT INST = f880
INPUT ID/EX_BUFFER : data1 = ff88, data2 = ff88, data15 = 0000
OUTPUT ID/EX : data1 = ff88, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0000, Upper = f0ff, Word ff88, Byte = 88
OUTPUT EXMEM: Lower = 1e00, Upper = f0ff, Word = 0f00, Byte = 00

INPUT MEMWB: Word = 0f00, Byte = 00, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 9, EX/MEM.FOR = 1, MEM/WB.FOR = 1
HMUX = 0, LMUX = 0

235 PROGRAM COUNTER: PC = 002e, New_PC = 0030 Branch = fe76
INSTRUCTION: ca92, Read.op1 = 8, Read.op2 = 9
ALU: Lower = 0000, op1 = ff88, op2 = ff88, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 2bcd, WriteReg2 = ff88, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = cd, Word = 2bcd

INPUT IF/ID : INST = ca92, IFFLUSH = 1 OUTPUT INST = d892
INPUT ID/EX_BUFFER : data1 = ff88, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = ff88, data2 =ff88, data15 = 0000, func = 0000
INPUT EXMEM : Lower = ff10, Upper = f0ff, Word ff88, Byte = 88
OUTPUT EXMEM: Lower = 0000, Upper = f0ff, Word = ff88, Byte = 88
INPUT MEMWB: Word = ff88, Byte = 88, OUTPUT: Word = 2bcd, Byte = cd
ID/EX.FOR = 8, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

245 PROGRAM COUNTER: PC = 0030, New_PC = 0032 Branch = fe78
INSTRUCTION: fcc0, Read.op1 = a, Read.op2 = 9
ALU: Lower = ff10, op1 = 0002, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = ff10, WriteReg2 = ff88, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = fcc0, IFFLUSH = 1 OUTPUT INST = ca92
INPUT ID/EX_BUFFER : data1 = 0000, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = ff88, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0002, Upper = f0ff, Word ff88, Byte = 88
OUTPUT EXMEM: Lower = ff10, Upper = f0ff, Word = ff88, Byte = 88
INPUT MEMWB: Word = ff88, Byte = 88, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 9, EX/MEM.FOR = 8, MEM/WB.FOR = 8
HMUX = 0, LMUX = 0

255 PROGRAM COUNTER: PC = 0032, New_PC = 0034 Branch = ff32
INSTRUCTION: fdd1, Read.op1 = c, Read.op2 = c
ALU: Lower = 0002, op1 = 0000, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 0002, WriteReg2 = 0000, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = 00, Word = 0000

INPUT IF/ID : INST = fdd1, IFFLUSH = 1 OUTPUT INST = fcc0
INPUT ID/EX_BUFFER : data1 = cccc, data2 = cccc, data15 = 0000
OUTPUT ID/EX : data1 = 0000, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0000, Upper = f0ff, Word 0000, Byte = 00
OUTPUT EXMEM: Lower = 0002, Upper = f0ff, Word = ff88, Byte = 88
INPUT MEMWB: Word = ff88, Byte = 88, OUTPUT: Word = 0000, Byte = 00
ID/EX.FOR = 9, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

265 PROGRAM COUNTER: PC = 0034, New_PC = 0036 Branch = ff78
INSTRUCTION: fcd0, Read.op1 = d, Read.op2 = d
ALU: Lower = 0000, op1 = cccc, op2 = cccc, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 2bcd, WriteReg2 = cccc, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = cd, Word = 2bcd

INPUT IF/ID : INST = fcd0, IFFLUSH = 1 OUTPUT INST = fdd1
INPUT ID/EX_BUFFER : data1 = 0002, data2 = 0002, data15 = 0000
OUTPUT ID/EX : data1 = cccc, data2 =cccc, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 9998, Upper = f0ff, Word cccc, Byte = cc
OUTPUT EXMEM: Lower = 0000, Upper = f0ff, Word = 0000, Byte = 00
INPUT MEMWB: Word = 0000, Byte = 00, OUTPUT: Word = 2bcd, Byte = cd
ID/EX.FOR = c, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

275 PROGRAM COUNTER: PC = 0036, New_PC = 0038 Branch = ff76
INSTRUCTION: efff, Read.op1 = c, Read.op2 = d

ALU: Lower = 9998, op1 = 0002, op2 = 0002, opcode = 0001, func = zzzz,
REGISTERS: WriteReg1 = 9998, WriteReg2 = 0002, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = efff, IFFLUSH = 1 OUTPUT INST = fcd0
INPUT ID/EX_BUFFER : data1 = cccc, data2 = 0002, data15 = 0000
OUTPUT ID/EX : data1 = 0002, data2 =0002, data15 = 0000, func = 0001
INPUT EXMEM : Lower = 0000, Upper = f0ff, Word 0002, Byte = 02
OUTPUT EXMEM: Lower = 9998, Upper = f0ff, Word = cccc, Byte = cc
INPUT MEMWB: Word = cccc, Byte = cc, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = d, EX/MEM.FOR = c, MEM/WB.FOR = c
HMUX = 0, LMUX = 0

285 PROGRAM COUNTER: PC = 0038, New_PC = 003a Branch = 0034
INSTRUCTION: xxxx, Read.op1 = f, Read.op2 = f
ALU: Lower = 0000, op1 = cccc, op2 = 0002, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 0000, WriteReg2 = cccc, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = cd, Word = 2bcd

INPUT IF/ID : INST = xxxx, IFFLUSH = 1 OUTPUT INST = efff
INPUT ID/EX_BUFFER : data1 = 0000, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = cccc, data2 =0002, data15 = 0000, func = 0000
INPUT EXMEM : Lower = ccce, Upper = f0ff, Word cccc, Byte = cc
OUTPUT EXMEM: Lower = 0000, Upper = f0ff, Word = 0002, Byte = 02
INPUT MEMWB: Word = 0002, Byte = 02, OUTPUT: Word = 2bcd, Byte = cd
ID/EX.FOR = d, EX/MEM.FOR = d, MEM/WB.FOR = d
HMUX = 0, LMUX = 0

295 PROGRAM COUNTER: PC = 003a, New_PC = 003c Branch = xxxx
INSTRUCTION: xxxx, Read.op1 = x, Read.op2 = x
ALU: Lower = ccce, op1 = 0000, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = ccce, WriteReg2 = 0000, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

```
          INPUT IF/ID : INST = xxxx, IFFLUSH = 1 OUTPUT INST = xxxx
         INPUT ID/EX_BUFFER : data1 = xxxx, data2 = xxxx, data15 = 0000
          OUTPUT ID/EX : data1 = 0000, data2 =0000, data15 = 0000, func = 0000
          INPUT EXMEM : Lower = 0000, Upper = f0ff, Word 0000, Byte = 00
          OUTPUT EXMEM: Lower = ccce, Upper = f0ff, Word = cccc, Byte = cc
          INPUT MEMWB: Word = cccc, Byte = cc, OUTPUT: Word = xxxx, Byte = xx
          ID/EX.FOR = f, EX/MEM.FOR = d, MEM/WB.FOR = d
          HMUX = 0, LMUX = 0

$finish called from file "cpu_fixture.v", line 81.
$finish at simulation time                300
         V C S   S i m u l a t i o n   R e p o r t
Time: 300
CPU Time:      0.350 seconds;       Data structure size:   0.0Mb
Tue Dec  5 21:25:19 2017

Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3_Full64; Runtime version J-2014.12-SP3_Full64;  Dec  5 21:25
2017
VCD+ Writer J-2014.12-SP3_Full64 Copyright (c) 1991-2014 by Synopsys Inc.
                   0  PROGRAM COUNTER:  PC = 0000, New_PC = 0000 Branch = xxxx
          INSTRUCTION: xxxx, Read.op1 = x, Read.op2 = x
          ALU: Lower = xxxx, op1 = xxxx, op2 = xxxx, opcode = xxxx, func = zzzz,
          REGISTERS: WriteReg1 = xxxx, WriteReg2 = xxxx, WriteReg15 = xxxx, Wdst = xx
          MEMORY: Byte = xx, Word = xxxx

          INPUT IF/ID : INST = xxxx, IFFLUSH = x OUTPUT INST = xxxx
         INPUT ID/EX_BUFFER : data1 = xxxx, data2 = xxxx, data15 = 0000
          OUTPUT ID/EX : data1 = xxxx, data2 =xxxx, data15 = xxxx, func = xxxx
          INPUT EXMEM : Lower = xxxx, Upper = xxxx, Word xxxx, Byte = xx
          OUTPUT EXMEM: Lower = xxxx, Upper = xxxx, Word = xxxx, Byte = xx
```

INPUT MEMWB: Word = xxxx, Byte = xx, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = x, EX/MEM.FOR = x, MEM/WB.FOR = x
HMUX = x, LMUX = x

5 PROGRAM COUNTER: PC = 0000, New_PC = 0002 Branch = xxxx
INSTRUCTION: f120, Read.op1 = x, Read.op2 = x
ALU: Lower = xxxx, op1 = xxxx, op2 = xxxx, opcode = xxxx, func = zzzz,
REGISTERS: WriteReg1 = xxxx, WriteReg2 = xxxx, WriteReg15 = xxxx, Wdst = xx
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f120, IFFLUSH = x OUTPUT INST = xxxx
INPUT ID/EX_BUFFER : data1 = xxxx, data2 = xxxx, data15 = 0000
OUTPUT ID/EX : data1 = xxxx, data2 =xxxx, data15 = 0000, func = xxxx
INPUT EXMEM : Lower = xxxx, Upper = xxxx, Word xxxx, Byte = xx
OUTPUT EXMEM: Lower = xxxx, Upper = xxxx, Word = xxxx, Byte = xx
INPUT MEMWB: Word = xxxx, Byte = xx, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = x, EX/MEM.FOR = x, MEM/WB.FOR = x
HMUX = x, LMUX = x

15 PROGRAM COUNTER: PC = 0002, New_PC = 0004 Branch = 0082
INSTRUCTION: f121, Read.op1 = 1, Read.op2 = 2
ALU: Lower = xxxx, op1 = xxxx, op2 = xxxx, opcode = xxxx, func = zzzz,
REGISTERS: WriteReg1 = xxxx, WriteReg2 = xxxx, WriteReg15 = xxxx, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f121, IFFLUSH = 1 OUTPUT INST = f120
INPUT ID/EX_BUFFER : data1 = 0f00, data2 = 0050, data15 = 0000
OUTPUT ID/EX : data1 = xxxx, data2 =xxxx, data15 = 0000, func = xxxx
INPUT EXMEM : Lower = xxxx, Upper = xxxx, Word xxxx, Byte = xx
OUTPUT EXMEM: Lower = xxxx, Upper = xxxx, Word = xxxx, Byte = xx
INPUT MEMWB: Word = xxxx, Byte = xx, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = x, EX/MEM.FOR = x, MEM/WB.FOR = x
HMUX = x, LMUX = x

25 PROGRAM COUNTER: PC = 0004, New_PC = 0006 Branch = 0088
INSTRUCTION: 93ff, Read.op1 = 1, Read.op2 = 2
ALU: Lower = xxxx, op1 = 0f00, op2 = 0050, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = xxxx, WriteReg2 = 0f00, WriteReg15 = xxxx, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = 93ff, IFFLUSH = 1 OUTPUT INST = f121
INPUT ID/EX_BUFFER : data1 = 0f00, data2 = 0050, data15 = 0000
OUTPUT ID/EX : data1 = 0f00, data2 =0050, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0f50, Upper = xxxx, Word 0f00, Byte = 00
OUTPUT EXMEM: Lower = xxxx, Upper = xxxx, Word = xxxx, Byte = xx
INPUT MEMWB: Word = xxxx, Byte = xx, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 2, EX/MEM.FOR = x, MEM/WB.FOR = x
HMUX = 0, LMUX = 0

35 PROGRAM COUNTER: PC = 0006, New_PC = 0008 Branch = 0002
INSTRUCTION: 834c, Read.op1 = 3, Read.op2 = f
ALU: Lower = 0f50, op1 = 0f00, op2 = 0050, opcode = 0001, func = zzzz,
REGISTERS: WriteReg1 = 0f50, WriteReg2 = 0f00, WriteReg15 = xxxx, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = 834c, IFFLUSH = 1 OUTPUT INST = 93ff
INPUT ID/EX_BUFFER : data1 = ff0f, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 0f00, data2 =0050, data15 = 0000, func = 0001
INPUT EXMEM : Lower = 0eb0, Upper = xxxx, Word 0f00, Byte = 00
OUTPUT EXMEM: Lower = 0f50, Upper = xxxx, Word = 0f00, Byte = 00
INPUT MEMWB: Word = 0f00, Byte = 00, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 2, EX/MEM.FOR = 2, MEM/WB.FOR = 2
HMUX = 0, LMUX = 0

45 PROGRAM COUNTER: PC = 0008, New_PC = 000a Branch = 0138
INSTRUCTION: f564, Read.op1 = 3, Read.op2 = 4

ALU: Lower = 0eb0, op1 = ff0f, op2 = 004c, opcode = 1011, func = zzzz,
REGISTERS: WriteReg1 = 0eb0, WriteReg2 = ff0f, WriteReg15 = xxxx, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f564, IFFLUSH = 1 OUTPUT INST = 834c
INPUT ID/EX_BUFFER : data1 = ff0f, data2 = f0ff, data15 = 0000
OUTPUT ID/EX : data1 = ff0f, data2 =0000, data15 = 0000, func = 1011
INPUT EXMEM : Lower = ff4f, Upper = xxxx, Word ff0f, Byte = 0f
OUTPUT EXMEM: Lower = 0eb0, Upper = xxxx, Word = 0f00, Byte = 00
INPUT MEMWB: Word = 0f00, Byte = 00, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = f, EX/MEM.FOR = 2, MEM/WB.FOR = 2
HMUX = 0, LMUX = 0

55 PROGRAM COUNTER: PC = 000a, New_PC = 000c Branch = 019a
INSTRUCTION: f155, Read.op1 = 5, Read.op2 = 6
ALU: Lower = ff4f, op1 = ff0f, op2 = 0064, opcode = 1001, func = zzzz,
REGISTERS: WriteReg1 = ff4f, WriteReg2 = ff0f, WriteReg15 = xxxx, Wdst = 10
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f155, IFFLUSH = 1 OUTPUT INST = f564
INPUT ID/EX_BUFFER : data1 = 0040, data2 = 6666, data15 = 0000
OUTPUT ID/EX : data1 = ff0f, data2 =f0ff, data15 = 0000, func = 1001
INPUT EXMEM : Lower = 0004, Upper = xxxx, Word ff0f, Byte = 0f
OUTPUT EXMEM: Lower = ff4f, Upper = xxxx, Word = ff0f, Byte = 0f
INPUT MEMWB: Word = ff0f, Byte = 0f, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 4, EX/MEM.FOR = f, MEM/WB.FOR = f
HMUX = 0, LMUX = 0

65 PROGRAM COUNTER: PC = 000c, New_PC = 000e Branch = 0160
INSTRUCTION: fff1, Read.op1 = 1, Read.op2 = 5
ALU: Lower = 0004, op1 = 0040, op2 = 6666, opcode = 0100, func = zzzz,
REGISTERS: WriteReg1 = 0004, WriteReg2 = 0040, WriteReg15 = xxxx, Wdst = 10
MEMORY: Byte = 34, Word = 1234

INPUT IF/ID : INST = fff1, IFFLUSH = 1 OUTPUT INST = f155
INPUT ID/EX_BUFFER : data1 = 0f00, data2 = 0040, data15 = 0000
OUTPUT ID/EX : data1 = 0040, data2 =6666, data15 = 0000, func = 0100
INPUT EXMEM : Lower = 9980, Upper = xxxx, Word 0040, Byte = 40
OUTPUT EXMEM: Lower = 0004, Upper = xxxx, Word = ff0f, Byte = 0f
INPUT MEMWB: Word = ff0f, Byte = 0f, OUTPUT: Word = 1234, Byte = 34
ID/EX.FOR = 6, EX/MEM.FOR = 4, MEM/WB.FOR = 4
HMUX = 0, LMUX = 0

75 PROGRAM COUNTER: PC = 000e, New_PC = 0010 Branch = ffd2
INSTRUCTION: f487, Read.op1 = f, Read.op2 = f
ALU: Lower = 9980, op1 = 0f00, op2 = 0040, opcode = 0101, func = zzzz,
REGISTERS: WriteReg1 = 9980, WriteReg2 = 0f00, WriteReg15 = xxxx, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f487, IFFLUSH = 1 OUTPUT INST = fff1
INPUT ID/EX_BUFFER : data1 = 0000, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 0f00, data2 =0040, data15 = 0000, func = 0101
INPUT EXMEM : Lower = 003c, Upper = 0000, Word 0f00, Byte = 00
OUTPUT EXMEM: Lower = 9980, Upper = xxxx, Word = 0040, Byte = 40
INPUT MEMWB: Word = 0040, Byte = 40, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 5, EX/MEM.FOR = 6, MEM/WB.FOR = 6
HMUX = 0, LMUX = 0

85 PROGRAM COUNTER: PC = 0010, New_PC = 0012 Branch = fe2c
INSTRUCTION: f468, Read.op1 = 4, Read.op2 = 8
ALU: Lower = 003c, op1 = 0000, op2 = 0000, opcode = 0001, func = zzzz,
REGISTERS: WriteReg1 = 003c, WriteReg2 = 0000, WriteReg15 = 0000, Wdst = 00
MEMORY: Byte = 00, Word = 0000

INPUT IF/ID : INST = f468, IFFLUSH = 1 OUTPUT INST = f487
INPUT ID/EX_BUFFER : data1 = f0ff, data2 = ff88, data15 = 0000

OUTPUT ID/EX : data1 = 0000, data2 =0000, data15 = 0000, func = 0001
INPUT EXMEM : Lower = 0000, Upper = 0000, Word 0000, Byte = 00
OUTPUT EXMEM: Lower = 003c, Upper = 0000, Word = 0f00, Byte = 00
INPUT MEMWB: Word = 0f00, Byte = 00, OUTPUT: Word = 0000, Byte = 00
ID/EX.FOR = f, EX/MEM.FOR = 5, MEM/WB.FOR = 5
HMUX = 0, LMUX = 0

95 PROGRAM COUNTER: PC = 0012, New_PC = 0014 Branch = 01b2
INSTRUCTION: 9402, Read.op1 = 4, Read.op2 = 6
ALU: Lower = 0000, op1 = f0ff, op2 = ff88, opcode = 0111, func = zzzz,
REGISTERS: WriteReg1 = ff88, WriteReg2 = f0ff, WriteReg15 = 0000, Wdst = 01
MEMORY: Byte = cd, Word = 2bcd

INPUT IF/ID : INST = 9402, IFFLUSH = 1 OUTPUT INST = f468
INPUT ID/EX_BUFFER : data1 = f0ff, data2 = 6666, data15 = 0000
OUTPUT ID/EX : data1 = f0ff, data2 =ff88, data15 = 0000, func = 0111
INPUT EXMEM : Lower = ff88, Upper = 0000, Word f0ff, Byte = ff
OUTPUT EXMEM: Lower = 0000, Upper = 0000, Word = 0000, Byte = 00
INPUT MEMWB: Word = 0000, Byte = 00, OUTPUT: Word = 2bcd, Byte = cd
ID/EX.FOR = 8, EX/MEM.FOR = f, MEM/WB.FOR = f
HMUX = 0, LMUX = 0

105 PROGRAM COUNTER: PC = 0014, New_PC = 0016 Branch = 001c
INSTRUCTION: a694, Read.op1 = 4, Read.op2 = 0
ALU: Lower = ff88, op1 = f0ff, op2 = 6666, opcode = 1000, func = zzzz,
REGISTERS: WriteReg1 = 6666, WriteReg2 = f0ff, WriteReg15 = 0000, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = a694, IFFLUSH = 1 OUTPUT INST = 9402
INPUT ID/EX_BUFFER : data1 = f0ff, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = f0ff, data2 =6666, data15 = 0000, func = 1000
INPUT EXMEM : Lower = 6666, Upper = f0ff, Word f0ff, Byte = ff
OUTPUT EXMEM: Lower = ff88, Upper = 0000, Word = f0ff, Byte = ff

INPUT MEMWB: Word = f0ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 6, EX/MEM.FOR = 8, MEM/WB.FOR = 8
HMUX = 0, LMUX = 0

115 PROGRAM COUNTER: PC = 0016, New_PC = 0018 Branch = fe66
INSTRUCTION: b696, Read.op1 = 6, Read.op2 = 9
ALU: Lower = 6666, op1 = f0ff, op2 = 0094, opcode = 1011, func = zzzz,
REGISTERS: WriteReg1 = 6666, WriteReg2 = f0ff, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = b696, IFFLUSH = 1 OUTPUT INST = a694
INPUT ID/EX_BUFFER : data1 = 6666, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = f0ff, data2 =0000, data15 = 0000, func = 1011
INPUT EXMEM : Lower = f0ff, Upper = f0ff, Word f0ff, Byte = ff
OUTPUT EXMEM: Lower = 6666, Upper = f0ff, Word = f0ff, Byte = ff
INPUT MEMWB: Word = f0ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 0, EX/MEM.FOR = 6, MEM/WB.FOR = 6
HMUX = 0, LMUX = 0

125 PROGRAM COUNTER: PC = 0018, New_PC = 001a Branch = fe70
INSTRUCTION: c696, Read.op1 = 6, Read.op2 = 9
ALU: Lower = f0ff, op1 = 0006, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = f0ff, WriteReg2 = 6666, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = c696, IFFLUSH = 1 OUTPUT INST = b696
INPUT ID/EX_BUFFER : data1 = 6666, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 6666, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0006, Upper = f0ff, Word 6666, Byte = 66
OUTPUT EXMEM: Lower = f0ff, Upper = f0ff, Word = f0ff, Byte = ff
INPUT MEMWB: Word = f0ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 9, EX/MEM.FOR = 0, MEM/WB.FOR = 0
HMUX = 0, LMUX = 0

135 PROGRAM COUNTER: PC = 001a, New_PC = 001c Branch = fe72
INSTRUCTION: 6704, Read.op1 = 6, Read.op2 = 9
ALU: Lower = 0006, op1 = 0006, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 00ad, WriteReg2 = 6666, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = ad, Word = dead

INPUT IF/ID : INST = 6704, IFFLUSH = 1 OUTPUT INST = c696
INPUT ID/EX_BUFFER : data1 = 6666, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 6666, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0006, Upper = f0ff, Word 6666, Byte = 66
OUTPUT EXMEM: Lower = 0006, Upper = f0ff, Word = 6666, Byte = 66
INPUT MEMWB: Word = 6666, Byte = 66, OUTPUT: Word = dead, Byte = ad
ID/EX.FOR = 9, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

145 PROGRAM COUNTER: PC = 001c, New_PC = 001e Branch = 002c
INSTRUCTION: fb10, Read.op1 = 7, Read.op2 = 0
ALU: Lower = 0006, op1 = 0004, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 0006, WriteReg2 = 6666, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = ad, Word = dead

INPUT IF/ID : INST = fb10, IFFLUSH = 1 OUTPUT INST = 6704
INPUT ID/EX_BUFFER : data1 = 00ff, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 6666, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0004, Upper = f0ff, Word 6666, Byte = 66
OUTPUT EXMEM: Lower = 0006, Upper = f0ff, Word = 6666, Byte = 66
INPUT MEMWB: Word = 6666, Byte = 66, OUTPUT: Word = dead, Byte = ad
ID/EX.FOR = 9, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

155 PROGRAM COUNTER: PC = 001e, New_PC = 0020 Branch = 005e
INSTRUCTION: 5705, Read.op1 = b, Read.op2 = 1

ALU: Lower = 0004, op1 = 00ff, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 1234, WriteReg2 = 00ff, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = 34, Word = 1234

INPUT IF/ID : INST = 5705, IFFLUSH = 1 OUTPUT INST = fb10
INPUT ID/EX_BUFFER : data1 = 0000, data2 = 0f00, data15 = 0000
OUTPUT ID/EX : data1 = 00ff, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 00ff, Upper = f0ff, Word 00ff, Byte = ff
OUTPUT EXMEM: Lower = 0004, Upper = f0ff, Word = 6666, Byte = 66
INPUT MEMWB: Word = 6666, Byte = 66, OUTPUT: Word = 1234, Byte = 34
ID/EX.FOR = 0, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

165 PROGRAM COUNTER: PC = 0020, New_PC = 0022 Branch = 0034
INSTRUCTION: fb20, Read.op1 = 7, Read.op2 = 0
ALU: Lower = 00ff, op1 = 0000, op2 = 0f00, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 00ff, WriteReg2 = 0000, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = fb20, IFFLUSH = 1 OUTPUT INST = 5705
INPUT ID/EX_BUFFER : data1 = 00ff, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 0000, data2 =0f00, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0f00, Upper = f0ff, Word 0000, Byte = 00
OUTPUT EXMEM: Lower = 00ff, Upper = f0ff, Word = 00ff, Byte = ff
INPUT MEMWB: Word = 00ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 1, EX/MEM.FOR = 0, MEM/WB.FOR = 0
HMUX = 0, LMUX = 0

175 PROGRAM COUNTER: PC = 0022, New_PC = 0024 Branch = 00a2
INSTRUCTION: 4702, Read.op1 = b, Read.op2 = 2
ALU: Lower = 0f00, op1 = 00ff, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 0f00, WriteReg2 = 00ff, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = 4702, IFFLUSH = 1 OUTPUT INST = fb20
INPUT ID/EX_BUFFER : data1 = 0000, data2 = 0050, data15 = 0000
OUTPUT ID/EX : data1 = 00ff, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 00ff, Upper = f0ff, Word 00ff, Byte = ff
OUTPUT EXMEM: Lower = 0f00, Upper = f0ff, Word = 0000, Byte = 00
INPUT MEMWB: Word = 0000, Byte = 00, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 0, EX/MEM.FOR = 1, MEM/WB.FOR = 1
HMUX = 0, LMUX = 0

185 PROGRAM COUNTER: PC = 0024, New_PC = 0026 Branch = 002c
INSTRUCTION: f110, Read.op1 = 7, Read.op2 = 0
ALU: Lower = 00ff, op1 = 0000, op2 = 0050, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 00ff, WriteReg2 = 0000, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f110, IFFLUSH = 1 OUTPUT INST = 4702
INPUT ID/EX_BUFFER : data1 = 00ff, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 0000, data2 =0050, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0050, Upper = f0ff, Word 0000, Byte = 00
OUTPUT EXMEM: Lower = 00ff, Upper = f0ff, Word = 00ff, Byte = ff
INPUT MEMWB: Word = 00ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 2, EX/MEM.FOR = 0, MEM/WB.FOR = 0
HMUX = 0, LMUX = 0

195 PROGRAM COUNTER: PC = 0026, New_PC = 0028 Branch = 0066
INSTRUCTION: f110, Read.op1 = 1, Read.op2 = 1
ALU: Lower = 0050, op1 = 00ff, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 0050, WriteReg2 = 00ff, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = 00, Word = 0000

INPUT IF/ID : INST = f110, IFFLUSH = 1 OUTPUT INST = f110
INPUT ID/EX_BUFFER : data1 = 0f00, data2 = 0f00, data15 = 0000

OUTPUT ID/EX : data1 = 00ff, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 00ff, Upper = f0ff, Word 00ff, Byte = ff
OUTPUT EXMEM: Lower = 0050, Upper = f0ff, Word = 0000, Byte = 00
INPUT MEMWB: Word = 0000, Byte = 00, OUTPUT: Word = 0000, Byte = 00
ID/EX.FOR = 0, EX/MEM.FOR = 2, MEM/WB.FOR = 2
HMUX = 0, LMUX = 0

205 PROGRAM COUNTER: PC = 0028, New_PC = 002a Branch = 0068
INSTRUCTION: c890, Read.op1 = 1, Read.op2 = 1
ALU: Lower = 00ff, op1 = 0f00, op2 = 0f00, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 00ff, WriteReg2 = 0f00, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = c890, IFFLUSH = 1 OUTPUT INST = f110
INPUT ID/EX_BUFFER : data1 = 0f00, data2 = 0f00, data15 = 0000
OUTPUT ID/EX : data1 = 0f00, data2 =0f00, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 1e00, Upper = f0ff, Word 0f00, Byte = 00
OUTPUT EXMEM: Lower = 00ff, Upper = f0ff, Word = 00ff, Byte = ff
INPUT MEMWB: Word = 00ff, Byte = ff, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 1, EX/MEM.FOR = 0, MEM/WB.FOR = 0
HMUX = 0, LMUX = 0

215 PROGRAM COUNTER: PC = 002a, New_PC = 002c Branch = fe6a
INSTRUCTION: f880, Read.op1 = 8, Read.op2 = 9
ALU: Lower = 1e00, op1 = 0f00, op2 = 0f00, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 1e00, WriteReg2 = 0f00, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = f880, IFFLUSH = 1 OUTPUT INST = c890
INPUT ID/EX_BUFFER : data1 = ff88, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = 0f00, data2 =0f00, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 1e00, Upper = f0ff, Word 0f00, Byte = 00
OUTPUT EXMEM: Lower = 1e00, Upper = f0ff, Word = 0f00, Byte = 00

INPUT MEMWB: Word = 0f00, Byte = 00, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 1, EX/MEM.FOR = 1, MEM/WB.FOR = 1
HMUX = 0, LMUX = 0

225 PROGRAM COUNTER: PC = 002c, New_PC = 002e Branch = fe2c
INSTRUCTION: d892, Read.op1 = 8, Read.op2 = 8
ALU: Lower = 1e00, op1 = 0000, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 1e00, WriteReg2 = ff88, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = d892, IFFLUSH = 1 OUTPUT INST = f880
INPUT ID/EX_BUFFER : data1 = ff88, data2 = ff88, data15 = 0000
OUTPUT ID/EX : data1 = ff88, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0000, Upper = f0ff, Word ff88, Byte = 88
OUTPUT EXMEM: Lower = 1e00, Upper = f0ff, Word = 0f00, Byte = 00
INPUT MEMWB: Word = 0f00, Byte = 00, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 9, EX/MEM.FOR = 1, MEM/WB.FOR = 1
HMUX = 0, LMUX = 0

235 PROGRAM COUNTER: PC = 002e, New_PC = 0030 Branch = fe76
INSTRUCTION: ca92, Read.op1 = 8, Read.op2 = 9
ALU: Lower = 0000, op1 = ff88, op2 = ff88, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 2bcd, WriteReg2 = ff88, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = cd, Word = 2bcd

INPUT IF/ID : INST = ca92, IFFLUSH = 1 OUTPUT INST = d892
INPUT ID/EX_BUFFER : data1 = ff88, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = ff88, data2 =ff88, data15 = 0000, func = 0000
INPUT EXMEM : Lower = ff10, Upper = f0ff, Word ff88, Byte = 88
OUTPUT EXMEM: Lower = 0000, Upper = f0ff, Word = ff88, Byte = 88
INPUT MEMWB: Word = ff88, Byte = 88, OUTPUT: Word = 2bcd, Byte = cd
ID/EX.FOR = 8, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

245 PROGRAM COUNTER: PC = 0030, New_PC = 0032 Branch = fe78
INSTRUCTION: fcc0, Read.op1 = a, Read.op2 = 9
ALU: Lower = ff10, op1 = 0002, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = ff10, WriteReg2 = ff88, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = fcc0, IFFLUSH = 1 OUTPUT INST = ca92
INPUT ID/EX_BUFFER : data1 = 0000, data2 = 0000, data15 = 0000
OUTPUT ID/EX : data1 = ff88, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0002, Upper = f0ff, Word ff88, Byte = 88
OUTPUT EXMEM: Lower = ff10, Upper = f0ff, Word = ff88, Byte = 88
INPUT MEMWB: Word = ff88, Byte = 88, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = 9, EX/MEM.FOR = 8, MEM/WB.FOR = 8
HMUX = 0, LMUX = 0

255 PROGRAM COUNTER: PC = 0032, New_PC = 0034 Branch = ff32
INSTRUCTION: fdd1, Read.op1 = c, Read.op2 = c
ALU: Lower = 0002, op1 = 0000, op2 = 0000, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 0002, WriteReg2 = 0000, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = 00, Word = 0000

INPUT IF/ID : INST = fdd1, IFFLUSH = 1 OUTPUT INST = fcc0
INPUT ID/EX_BUFFER : data1 = cccc, data2 = cccc, data15 = 0000
OUTPUT ID/EX : data1 = 0000, data2 =0000, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 0000, Upper = f0ff, Word 0000, Byte = 00
OUTPUT EXMEM: Lower = 0002, Upper = f0ff, Word = ff88, Byte = 88
INPUT MEMWB: Word = ff88, Byte = 88, OUTPUT: Word = 0000, Byte = 00
ID/EX.FOR = 9, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

265 PROGRAM COUNTER: PC = 0034, New_PC = 0036 Branch = ff78
INSTRUCTION: fcd0, Read.op1 = d, Read.op2 = d

ALU: Lower = 0000, op1 = cccc, op2 = cccc, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 2bcd, WriteReg2 = cccc, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = cd, Word = 2bcd

INPUT IF/ID : INST = fcd0, IFFLUSH = 1 OUTPUT INST = fdd1
INPUT ID/EX_BUFFER : data1 = 0002, data2 = 0002, data15 = 0000
OUTPUT ID/EX : data1 = cccc, data2 =cccc, data15 = 0000, func = 0000
INPUT EXMEM : Lower = 9998, Upper = f0ff, Word cccc, Byte = cc
OUTPUT EXMEM: Lower = 0000, Upper = f0ff, Word = 0000, Byte = 00
INPUT MEMWB: Word = 0000, Byte = 00, OUTPUT: Word = 2bcd, Byte = cd
ID/EX.FOR = c, EX/MEM.FOR = 9, MEM/WB.FOR = 9
HMUX = 0, LMUX = 0

275 PROGRAM COUNTER: PC = 0036, New_PC = 0038 Branch = ff76
INSTRUCTION: efff, Read.op1 = c, Read.op2 = d
ALU: Lower = 9998, op1 = 0002, op2 = 0002, opcode = 0001, func = zzzz,
REGISTERS: WriteReg1 = 9998, WriteReg2 = 0002, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = xx, Word = xxxx

INPUT IF/ID : INST = efff, IFFLUSH = 1 OUTPUT INST = fcd0
INPUT ID/EX_BUFFER : data1 = cccc, data2 = 0002, data15 = 0000
OUTPUT ID/EX : data1 = 0002, data2 =0002, data15 = 0000, func = 0001
INPUT EXMEM : Lower = 0000, Upper = f0ff, Word 0002, Byte = 02
OUTPUT EXMEM: Lower = 9998, Upper = f0ff, Word = cccc, Byte = cc
INPUT MEMWB: Word = cccc, Byte = cc, OUTPUT: Word = xxxx, Byte = xx
ID/EX.FOR = d, EX/MEM.FOR = c, MEM/WB.FOR = c
HMUX = 0, LMUX = 0

285 PROGRAM COUNTER: PC = 0038, New_PC = 003a Branch = 0034
INSTRUCTION: xxxx, Read.op1 = f, Read.op2 = f
ALU: Lower = 0000, op1 = cccc, op2 = 0002, opcode = 0000, func = zzzz,
REGISTERS: WriteReg1 = 0000, WriteReg2 = cccc, WriteReg15 = f0ff, Wdst = 00
MEMORY: Byte = cd, Word = 2bcd

```
 INPUT IF/ID : INST = xxxx, IFFLUSH = 1 OUTPUT INST = efff
INPUT ID/EX_BUFFER : data1 = 0000, data2 = 0000, data15 = 0000
 OUTPUT ID/EX : data1 = cccc, data2 =0002, data15 = 0000, func = 0000
 INPUT EXMEM : Lower = ccce, Upper = f0ff, Word cccc, Byte = cc
 OUTPUT EXMEM: Lower = 0000, Upper = f0ff, Word = 0002, Byte = 02
 INPUT MEMWB: Word = 0002, Byte = 02, OUTPUT: Word = 2bcd, Byte = cd
 ID/EX.FOR = d, EX/MEM.FOR = d, MEM/WB.FOR = d
 HMUX = 0, LMUX = 0

        295  PROGRAM COUNTER:  PC = 003a, New_PC = 003c Branch = xxxx
 INSTRUCTION: xxxx, Read.op1 = x, Read.op2 = x
 ALU: Lower = ccce, op1 = 0000, op2 = 0000, opcode = 0000, func = zzzz,
 REGISTERS: WriteReg1 = ccce, WriteReg2 = 0000, WriteReg15 = f0ff, Wdst = 00
 MEMORY: Byte = xx, Word = xxxx

 INPUT IF/ID : INST = xxxx, IFFLUSH = 1 OUTPUT INST = xxxx
INPUT ID/EX_BUFFER : data1 = xxxx, data2 = xxxx, data15 = 0000
 OUTPUT ID/EX : data1 = 0000, data2 =0000, data15 = 0000, func = 0000
 INPUT EXMEM : Lower = 0000, Upper = f0ff, Word 0000, Byte = 00
 OUTPUT EXMEM: Lower = ccce, Upper = f0ff, Word = cccc, Byte = cc
 INPUT MEMWB: Word = cccc, Byte = cc, OUTPUT: Word = xxxx, Byte = xx
 ID/EX.FOR = f, EX/MEM.FOR = d, MEM/WB.FOR = d
 HMUX = 0, LMUX = 0

$finish called from file "cpu_fixture.v", line 81.
$finish at simulation time                300
         V C S   S i m u l a t i o n   R e p o r t
Time: 300
CPU Time:      0.350 seconds;       Data structure size:   0.0Mb
Tue Dec  5 21:25:19 2017
```