Contents

[Code 3](#_Toc37018952)

[RAM (Synchronous) 3](#_Toc37018953)

[Select and Encode Logic 4](#_Toc37018954)

[CON\_FF Logic 9](#_Toc37018955)

[R0 11](#_Toc37018956)

[Datapath 12](#_Toc37018957)

[Block Schematics 35](#_Toc37018958)

[Datapath 35](#_Toc37018959)

[Testbench Files 36](#_Toc37018960)

[Load 36](#_Toc37018961)

[Load Immediate 42](#_Toc37018962)

[Load with Register Offset 49](#_Toc37018963)

[Load Immediate with Register Offset 57](#_Toc37018964)

[Store 64](#_Toc37018965)

[Store with Register Offset 72](#_Toc37018966)

[ADD/AND/OR Immediate 82](#_Toc37018967)

[BRZR/BRNZ/BRPL/BRMI 89](#_Toc37018968)

[Jump 98](#_Toc37018969)

[Jump and Link 105](#_Toc37018970)

[MFHI/LO 112](#_Toc37018971)

[Out 119](#_Toc37018972)

[In 126](#_Toc37018973)

[Output Waveforms 134](#_Toc37018974)

[Load R1, $55 134](#_Toc37018975)

[Load R0, $35(R1) 134](#_Toc37018976)

[Loadi R1, $55 135](#_Toc37018977)

[Loadi R0, $35(R1) 135](#_Toc37018978)

[Store $90, R1 135](#_Toc37018979)

[Store $8F, R1 135](#_Toc37018980)

[Store $90, (R1)R1 136](#_Toc37018981)

[Addi R2, R1, -5 136](#_Toc37018982)

[Andi R2, R1, $26 137](#_Toc37018983)

[Ori R2, R1, $26 137](#_Toc37018984)

[Brzr R2, 35 (branch correctly taken) 137](#_Toc37018985)

[Brzr R2, 35 (branch not taken) 138](#_Toc37018986)

[Brnz R2, 35 (branch correctly taken) 138](#_Toc37018987)

[Brnz R2, 35 (branch not taken) 139](#_Toc37018988)

[Brpl R2, 35 (branch correctly taken) 139](#_Toc37018989)

[Brpl R2, 35 (branch not taken) 140](#_Toc37018990)

[Brmi R2, 35 (branch correctly taken) 140](#_Toc37018991)

[Brmi R2, 35 (branch not taken) 141](#_Toc37018992)

[Jr R1 141](#_Toc37018993)

[Jal R1 142](#_Toc37018994)

[Mfhi R2 142](#_Toc37018995)

[Mflo R2 142](#_Toc37018996)

[Out R2 143](#_Toc37018997)

[In R1 143](#_Toc37018998)

[Memory Contents 144](#_Toc37018999)

# Code

## RAM (Synchronous)

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity ram32 is

port (

clk : in std\_logic;

reset\_n : in std\_logic;

w\_en : in std\_logic;

r\_en : in std\_logic;

dataIn : in std\_logic\_vector(31 downto 0);

addr : in std\_logic\_vector(31 downto 0);

dataOut : out std\_logic\_vector(31 downto 0)

);

end entity;

architecture behaviour of ram32 is

type ram\_array is array(0 to 511) of std\_logic\_vector(31 downto 0);

signal ram : ram\_array;

begin

process(clk, reset\_n)

begin

if (reset\_n = '0') then

ram <= (others => x"00000000"); -- set all ram values to 0;

-- Here is where ram is initialized for each instruction

elsif (rising\_edge(clk)) then

if (w\_en = '1') then

ram(to\_integer(unsigned(addr(8 downto 0)))) <= dataIn;

elsif (r\_en = '1') then

dataOut <= ram(to\_integer(unsigned(addr(8 downto 0))));

end if;

end if;

end process;

end architecture;

## Select and Encode Logic

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity select\_and\_encode is

port (

IRout : in std\_logic\_vector(31 downto 0);

Gra : in std\_logic;

Grb : in std\_logic;

Grc : in std\_logic;

Rin : in std\_logic;

Rout : in std\_logic;

BAout : in std\_logic;

R0out : out std\_logic;

R1out : out std\_logic;

R2out : out std\_logic;

R3out : out std\_logic;

R4out : out std\_logic;

R5out : out std\_logic;

R6out : out std\_logic;

R7out : out std\_logic;

R8out : out std\_logic;

R9out : out std\_logic;

R10out : out std\_logic;

R11out : out std\_logic;

R12out : out std\_logic;

R13out : out std\_logic;

R14out : out std\_logic;

R15out : out std\_logic;

R0in : out std\_logic;

R1in : out std\_logic;

R2in : out std\_logic;

R3in : out std\_logic;

R4in : out std\_logic;

R5in : out std\_logic;

R6in : out std\_logic;

R7in : out std\_logic;

R8in : out std\_logic;

R9in : out std\_logic;

R10in : out std\_logic;

R11in : out std\_logic;

R12in : out std\_logic;

R13in : out std\_logic;

R14in : out std\_logic;

R15in : out std\_logic;

c\_sign\_extended : out std\_logic\_vector(31 downto 0)

);

end entity;

architecture behaviour of select\_and\_encode is

signal opcode : std\_logic\_vector(4 downto 0);

signal ra, rb, rc : std\_logic\_vector(3 downto 0);

signal s : std\_logic;

signal decoderIn : std\_logic\_vector(3 downto 0);

signal decoderOut : std\_logic\_vector(15 downto 0);

begin

opcode <= IRout(31 downto 27);

ra <= IRout(26 downto 23) when Gra = '1' else (others => '0');

rb <= IRout(22 downto 19) when Grb = '1' else (others => '0');

rc <= IRout(18 downto 15) when Grc = '1' else (others => '0');

decoderIn <= ra OR rb OR rc;

s <= Rout OR BAout;

with decoderIn select

decoderOut <= x"0001" when "0000",

x"0002" when "0001",

x"0004" when "0010",

x"0008" when "0011",

x"0010" when "0100",

x"0020" when "0101",

x"0040" when "0110",

x"0080" when "0111",

x"0100" when "1000",

x"0200" when "1001",

x"0400" when "1010",

x"0800" when "1011",

x"1000" when "1100",

x"2000" when "1101",

x"4000" when "1110",

x"8000" when others;

R0in <= decoderOut(0) AND Rin;

R0out <= decoderOut(0) AND s;

R1in <= decoderOut(1) AND Rin;

R1out <= decoderOut(1) AND s;

R2in <= decoderOut(2) AND Rin;

R2out <= decoderOut(2) AND s;

R3in <= decoderOut(3) AND Rin;

R3out <= decoderOut(3) AND s;

R4in <= decoderOut(4) AND Rin;

R4out <= decoderOut(4) AND s;

R5in <= decoderOut(5) AND Rin;

R5out <= decoderOut(5) AND s;

R6in <= decoderOut(6) AND Rin;

R6out <= decoderOut(6) AND s;

R7in <= decoderOut(7) AND Rin;

R7out <= decoderOut(7) AND s;

R8in <= decoderOut(8) AND Rin;

R8out <= decoderOut(8) AND s;

R9in <= decoderOut(9) AND Rin;

R9out <= decoderOut(9) AND s;

R10in <= decoderOut(10) AND Rin;

R10out <= decoderOut(10) AND s;

R11in <= decoderOut(11) AND Rin;

R11out <= decoderOut(11) AND s;

R12in <= decoderOut(12) AND Rin;

R12out <= decoderOut(12) AND s;

R13in <= decoderOut(13) AND Rin;

R13out <= decoderOut(13) AND s;

R14in <= decoderOut(14) AND Rin;

R14out <= decoderOut(14) AND s;

R15in <= decoderOut(15) AND Rin;

R15out <= decoderOut(15) AND s;

c\_sign\_extended <= std\_logic\_vector(resize(signed(IRout(18 downto 0)), c\_sign\_extended'length));

end architecture;

## CON\_FF Logic

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity conFF is

port (

clk : in std\_logic;

clr : in std\_logic;

IRout : in std\_logic\_vector(31 downto 0);

BusMuxOut : in std\_logic\_vector(31 downto 0);

CONin : in std\_logic;

CONout : out std\_logic

);

end entity;

architecture behaviour of conFF is

signal opcode : std\_logic\_vector(4 downto 0);

signal ra : std\_logic\_vector(3 downto 0);

signal c2 : std\_logic\_vector(3 downto 0);

signal c : std\_logic\_vector(18 downto 0);

signal decoderIn : std\_logic\_vector(1 downto 0);

signal decoderOut : std\_logic\_vector(3 downto 0);

signal NORBus, s1, s2, s3, s4, f : std\_logic;

begin

opcode <= IRout(31 downto 27);

ra <= IRout(26 downto 23);

c2 <= IRout(22 downto 19);

c <= IRout(18 downto 0);

decoderIn(0) <= c2(0); decoderIn(1) <= c2(1);

with decoderIn select

decoderOut <= "0001" when "00",

"0010" when "01",

"0100" when "10",

"1000" when others;

NORBus <= '1' when (BusMuxOut = x"00000000") else '0';

s1 <= NORBus AND decoderOut(0);

s2 <= (NOT NORBus) AND decoderOut(1);

s3 <= (NOT BusMuxOut(31)) AND decoderOut(2);

s4 <= BusMuxOut(31) AND decoderOut(3);

f <= s1 OR s2 OR s3 OR s4;

process(clk, CONin)

begin

if (clr = '0') then

CONout <= '0';

elsif (rising\_edge(clk)) then

if (CONin = '1') then

CONout <= f;

end if;

end if;

end process;

end architecture;

## R0

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

LIBRARY work;

ENTITY r0 IS

PORT

(

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

R0in : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R0 : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END r0;

ARCHITECTURE bdf\_type OF r0 IS

COMPONENT reg32

PORT(clk : IN STD\_LOGIC;

reset\_n : IN STD\_LOGIC;

en : IN STD\_LOGIC;

d : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

q : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

SIGNAL BAout\_vect : STD\_LOGIC\_VECTOR(31 downto 0);

SIGNAL output : STD\_LOGIC\_VECTOR(31 downto 0);

BEGIN

b2v\_r0 : reg32 PORT MAP(

clk => clk,

reset\_n => clear,

en => R0in,

d => BusMuxOut,

q => output

);

BAout\_vect <= (others => NOT BAout);

BUSMuxIn\_R0 <= output AND BAout\_vect;

END bdf\_type;

## Datapath

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

LIBRARY work;

ENTITY datapath IS

PORT

(

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END datapath;

ARCHITECTURE bdf\_type OF datapath IS

COMPONENT alu

PORT(alu\_A : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

alu\_B : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SEL : IN STD\_LOGIC\_VECTOR(3 DOWNTO 0);

C\_hi : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

C\_lo : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT alu\_addr\_encoder

PORT(ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

ALU\_sel : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0)

);

END COMPONENT;

COMPONENT bidirectional\_bus

PORT(R0out : IN STD\_LOGIC;

R1out : IN STD\_LOGIC;

R2out : IN STD\_LOGIC;

R3out : IN STD\_LOGIC;

R4out : IN STD\_LOGIC;

R5out : IN STD\_LOGIC;

R6out : IN STD\_LOGIC;

R7out : IN STD\_LOGIC;

R8out : IN STD\_LOGIC;

R9out : IN STD\_LOGIC;

R10out : IN STD\_LOGIC;

R11out : IN STD\_LOGIC;

R12out : IN STD\_LOGIC;

R13out : IN STD\_LOGIC;

R14out : IN STD\_LOGIC;

R15out : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

BusMuxIn\_HI : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_InPort : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_LO : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_MDR : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_PC : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R0in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R10in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R11in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R12in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R13in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R14in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R15in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R1in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R2in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R3in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R4in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R5in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R6in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R7in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R8in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R9in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zhigh : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zlow : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

c\_sign\_extended : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT conff

PORT(clk : IN STD\_LOGIC;

clr : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

IRout : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC

);

END COMPONENT;

COMPONENT gp

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

R0in : IN STD\_LOGIC;

R1in : IN STD\_LOGIC;

R2in : IN STD\_LOGIC;

R3in : IN STD\_LOGIC;

R4in : IN STD\_LOGIC;

R5in : IN STD\_LOGIC;

R6in : IN STD\_LOGIC;

R7in : IN STD\_LOGIC;

R8in : IN STD\_LOGIC;

R9in : IN STD\_LOGIC;

R10in : IN STD\_LOGIC;

R11in : IN STD\_LOGIC;

R12in : IN STD\_LOGIC;

R13in : IN STD\_LOGIC;

R14in : IN STD\_LOGIC;

R15in : IN STD\_LOGIC;

BAOut : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R0in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R10in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R11in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R12in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R13in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R14in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R15in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R1in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R2in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R3in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R4in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R5in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R6in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R7in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R8in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R9in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT hi

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_HI : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT reg32

PORT(clk : IN STD\_LOGIC;

reset\_n : IN STD\_LOGIC;

en : IN STD\_LOGIC;

d : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

q : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT ir

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

IROut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT lo

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_LO : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT mar

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

MemAddrIn : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MemAddrOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT mdr

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MemDataIn : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MDRDataOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT pc

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_PC : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT ram32

PORT(clk : IN STD\_LOGIC;

reset\_n : IN STD\_LOGIC;

w\_en : IN STD\_LOGIC;

r\_en : IN STD\_LOGIC;

addr : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

dataIn : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

dataOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT select\_and\_encode

PORT(Gra : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

IRout : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

R0out : OUT STD\_LOGIC;

R1out : OUT STD\_LOGIC;

R2out : OUT STD\_LOGIC;

R3out : OUT STD\_LOGIC;

R4out : OUT STD\_LOGIC;

R5out : OUT STD\_LOGIC;

R6out : OUT STD\_LOGIC;

R7out : OUT STD\_LOGIC;

R8out : OUT STD\_LOGIC;

R9out : OUT STD\_LOGIC;

R10out : OUT STD\_LOGIC;

R11out : OUT STD\_LOGIC;

R12out : OUT STD\_LOGIC;

R13out : OUT STD\_LOGIC;

R14out : OUT STD\_LOGIC;

R15out : OUT STD\_LOGIC;

R0in : OUT STD\_LOGIC;

R1in : OUT STD\_LOGIC;

R2in : OUT STD\_LOGIC;

R3in : OUT STD\_LOGIC;

R4in : OUT STD\_LOGIC;

R5in : OUT STD\_LOGIC;

R6in : OUT STD\_LOGIC;

R7in : OUT STD\_LOGIC;

R8in : OUT STD\_LOGIC;

R9in : OUT STD\_LOGIC;

R10in : OUT STD\_LOGIC;

R11in : OUT STD\_LOGIC;

R12in : OUT STD\_LOGIC;

R13in : OUT STD\_LOGIC;

R14in : OUT STD\_LOGIC;

R15in : OUT STD\_LOGIC;

c\_sign\_extended : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT y

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

ALUIn\_Y : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT z

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Zhi\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

Zlo\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zhi : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zlow : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

SIGNAL SYNTHESIZED\_WIRE\_0 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_1 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_2 : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_3 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_4 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_5 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_6 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_7 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_8 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_9 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_10 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_11 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_12 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_13 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_14 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_15 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_16 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_17 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_18 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_19 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_20 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_21 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_22 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_23 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_24 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_25 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_26 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_27 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_28 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_29 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_30 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_31 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_32 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_33 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_34 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_35 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_36 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_37 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_38 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_39 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_40 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_41 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_42 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_78 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_45 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_46 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_47 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_48 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_49 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_50 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_51 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_52 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_53 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_54 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_55 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_56 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_57 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_58 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_59 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_60 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_67 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_70 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_74 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_75 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BEGIN

BusData <= SYNTHESIZED\_WIRE\_1;

b2v\_ALU : alu

PORT MAP(alu\_A => SYNTHESIZED\_WIRE\_0,

alu\_B => SYNTHESIZED\_WIRE\_1,

SEL => SYNTHESIZED\_WIRE\_2,

C\_hi => SYNTHESIZED\_WIRE\_74,

C\_lo => SYNTHESIZED\_WIRE\_75);

b2v\_ALU\_ADDR\_ENCODER : alu\_addr\_encoder

PORT MAP(ANDin => ANDin,

ORin => ORin,

NOTin => NOTin,

NEGin => NEGin,

ADDin => ADDin,

SUBin => SUBin,

MULin => MULin,

DIVin => DIVin,

SRAin => SRAin,

SLAin => SLAin,

RORin => RORin,

ROLin => ROLin,

ALU\_sel => SYNTHESIZED\_WIRE\_2);

b2v\_BUS : bidirectional\_bus

PORT MAP(R0out => SYNTHESIZED\_WIRE\_3,

R1out => SYNTHESIZED\_WIRE\_4,

R2out => SYNTHESIZED\_WIRE\_5,

R3out => SYNTHESIZED\_WIRE\_6,

R4out => SYNTHESIZED\_WIRE\_7,

R5out => SYNTHESIZED\_WIRE\_8,

R6out => SYNTHESIZED\_WIRE\_9,

R7out => SYNTHESIZED\_WIRE\_10,

R8out => SYNTHESIZED\_WIRE\_11,

R9out => SYNTHESIZED\_WIRE\_12,

R10out => SYNTHESIZED\_WIRE\_13,

R11out => SYNTHESIZED\_WIRE\_14,

R12out => SYNTHESIZED\_WIRE\_15,

R13out => SYNTHESIZED\_WIRE\_16,

R14out => SYNTHESIZED\_WIRE\_17,

R15out => SYNTHESIZED\_WIRE\_18,

PCout => PCout,

HIout => HIout,

LOout => LOout,

Zhighout => Zhighout,

Zlowout => Zlowout,

MDRout => MDRout,

InPortout => InPortout,

Cout => Cout,

BusMuxIn\_HI => SYNTHESIZED\_WIRE\_19,

BusMuxIn\_InPort => SYNTHESIZED\_WIRE\_20,

BusMuxIn\_LO => SYNTHESIZED\_WIRE\_21,

BusMuxIn\_MDR => SYNTHESIZED\_WIRE\_22,

BusMuxIn\_PC => SYNTHESIZED\_WIRE\_23,

BusMuxIn\_R0in => SYNTHESIZED\_WIRE\_24,

BusMuxIn\_R10in => SYNTHESIZED\_WIRE\_25,

BusMuxIn\_R11in => SYNTHESIZED\_WIRE\_26,

BusMuxIn\_R12in => SYNTHESIZED\_WIRE\_27,

BusMuxIn\_R13in => SYNTHESIZED\_WIRE\_28,

BusMuxIn\_R14in => SYNTHESIZED\_WIRE\_29,

BusMuxIn\_R15in => SYNTHESIZED\_WIRE\_30,

BusMuxIn\_R1in => SYNTHESIZED\_WIRE\_31,

BusMuxIn\_R2in => SYNTHESIZED\_WIRE\_32,

BusMuxIn\_R3in => SYNTHESIZED\_WIRE\_33,

BusMuxIn\_R4in => SYNTHESIZED\_WIRE\_34,

BusMuxIn\_R5in => SYNTHESIZED\_WIRE\_35,

BusMuxIn\_R6in => SYNTHESIZED\_WIRE\_36,

BusMuxIn\_R7in => SYNTHESIZED\_WIRE\_37,

BusMuxIn\_R8in => SYNTHESIZED\_WIRE\_38,

BusMuxIn\_R9in => SYNTHESIZED\_WIRE\_39,

BusMuxIn\_Zhigh => SYNTHESIZED\_WIRE\_40,

BusMuxIn\_Zlow => SYNTHESIZED\_WIRE\_41,

c\_sign\_extended => SYNTHESIZED\_WIRE\_42,

BusMuxOut => SYNTHESIZED\_WIRE\_1);

b2v\_CONFF : conff

PORT MAP(clk => clk,

clr => clear,

CONin => CONin,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

IRout => SYNTHESIZED\_WIRE\_78,

CONout => CONout);

b2v\_GP : gp

PORT MAP(clk => clk,

clear => clear,

R0in => SYNTHESIZED\_WIRE\_45,

R1in => SYNTHESIZED\_WIRE\_46,

R2in => SYNTHESIZED\_WIRE\_47,

R3in => SYNTHESIZED\_WIRE\_48,

R4in => SYNTHESIZED\_WIRE\_49,

R5in => SYNTHESIZED\_WIRE\_50,

R6in => SYNTHESIZED\_WIRE\_51,

R7in => SYNTHESIZED\_WIRE\_52,

R8in => SYNTHESIZED\_WIRE\_53,

R9in => SYNTHESIZED\_WIRE\_54,

R10in => SYNTHESIZED\_WIRE\_55,

R11in => SYNTHESIZED\_WIRE\_56,

R12in => SYNTHESIZED\_WIRE\_57,

R13in => SYNTHESIZED\_WIRE\_58,

R14in => SYNTHESIZED\_WIRE\_59,

R15in => SYNTHESIZED\_WIRE\_60,

BAOut => BAout,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

BusMuxIn\_R0in => SYNTHESIZED\_WIRE\_24,

BusMuxIn\_R10in => SYNTHESIZED\_WIRE\_25,

BusMuxIn\_R11in => SYNTHESIZED\_WIRE\_26,

BusMuxIn\_R12in => SYNTHESIZED\_WIRE\_27,

BusMuxIn\_R13in => SYNTHESIZED\_WIRE\_28,

BusMuxIn\_R14in => SYNTHESIZED\_WIRE\_29,

BusMuxIn\_R15in => SYNTHESIZED\_WIRE\_30,

BusMuxIn\_R1in => SYNTHESIZED\_WIRE\_31,

BusMuxIn\_R2in => SYNTHESIZED\_WIRE\_32,

BusMuxIn\_R3in => SYNTHESIZED\_WIRE\_33,

BusMuxIn\_R4in => SYNTHESIZED\_WIRE\_34,

BusMuxIn\_R5in => SYNTHESIZED\_WIRE\_35,

BusMuxIn\_R6in => SYNTHESIZED\_WIRE\_36,

BusMuxIn\_R7in => SYNTHESIZED\_WIRE\_37,

BusMuxIn\_R8in => SYNTHESIZED\_WIRE\_38,

BusMuxIn\_R9in => SYNTHESIZED\_WIRE\_39);

b2v\_HI : hi

PORT MAP(clk => clk,

clear => clear,

HIin => HIin,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

BusMuxIn\_HI => SYNTHESIZED\_WIRE\_19);

b2v\_IN\_PORT : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => InPortin,

d => InputDeviceData,

q => SYNTHESIZED\_WIRE\_20);

b2v\_IR : ir

PORT MAP(clk => clk,

clear => clear,

IRin => IRin,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

IROut => SYNTHESIZED\_WIRE\_78);

b2v\_LO : lo

PORT MAP(clk => clk,

clear => clear,

LOin => LOin,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

BusMuxIn\_LO => SYNTHESIZED\_WIRE\_21);

b2v\_MAR : mar

PORT MAP(clk => clk,

clear => clear,

MARin => MARin,

MemAddrIn => SYNTHESIZED\_WIRE\_1,

MemAddrOut => SYNTHESIZED\_WIRE\_70);

b2v\_MDR : mdr

PORT MAP(clk => clk,

clear => clear,

MDRin => MDRin,

Read => Read,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

MemDataIn => SYNTHESIZED\_WIRE\_67,

MDRDataOut => SYNTHESIZED\_WIRE\_22);

b2v\_OUT\_PORT : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => OutPortin,

d => SYNTHESIZED\_WIRE\_1,

q => OutputDeviceData);

b2v\_PC : pc

PORT MAP(clk => clk,

clear => clear,

PCin => PCin,

IncPC => IncPC,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

BusMuxIn\_PC => SYNTHESIZED\_WIRE\_23);

b2v\_RAM : ram32

PORT MAP(clk => clk,

reset\_n => clear,

w\_en => Write,

r\_en => Read,

addr => SYNTHESIZED\_WIRE\_70,

dataIn => SYNTHESIZED\_WIRE\_22,

dataOut => SYNTHESIZED\_WIRE\_67);

b2v\_SELECT\_AND\_ENCODE : select\_and\_encode

PORT MAP(Gra => Gra,

Grb => Grb,

Grc => Grc,

Rin => Rin,

Rout => Rout,

BAout => BAout,

IRout => SYNTHESIZED\_WIRE\_78,

R0out => SYNTHESIZED\_WIRE\_3,

R1out => SYNTHESIZED\_WIRE\_4,

R2out => SYNTHESIZED\_WIRE\_5,

R3out => SYNTHESIZED\_WIRE\_6,

R4out => SYNTHESIZED\_WIRE\_7,

R5out => SYNTHESIZED\_WIRE\_8,

R6out => SYNTHESIZED\_WIRE\_9,

R7out => SYNTHESIZED\_WIRE\_10,

R8out => SYNTHESIZED\_WIRE\_11,

R9out => SYNTHESIZED\_WIRE\_12,

R10out => SYNTHESIZED\_WIRE\_13,

R11out => SYNTHESIZED\_WIRE\_14,

R12out => SYNTHESIZED\_WIRE\_15,

R13out => SYNTHESIZED\_WIRE\_16,

R14out => SYNTHESIZED\_WIRE\_17,

R15out => SYNTHESIZED\_WIRE\_18,

R0in => SYNTHESIZED\_WIRE\_45,

R1in => SYNTHESIZED\_WIRE\_46,

R2in => SYNTHESIZED\_WIRE\_47,

R3in => SYNTHESIZED\_WIRE\_48,

R4in => SYNTHESIZED\_WIRE\_49,

R5in => SYNTHESIZED\_WIRE\_50,

R6in => SYNTHESIZED\_WIRE\_51,

R7in => SYNTHESIZED\_WIRE\_52,

R8in => SYNTHESIZED\_WIRE\_53,

R9in => SYNTHESIZED\_WIRE\_54,

R10in => SYNTHESIZED\_WIRE\_55,

R11in => SYNTHESIZED\_WIRE\_56,

R12in => SYNTHESIZED\_WIRE\_57,

R13in => SYNTHESIZED\_WIRE\_58,

R14in => SYNTHESIZED\_WIRE\_59,

R15in => SYNTHESIZED\_WIRE\_60,

c\_sign\_extended => SYNTHESIZED\_WIRE\_42);

b2v\_Y : y

PORT MAP(clk => clk,

clear => clear,

Yin => Yin,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

ALUIn\_Y => SYNTHESIZED\_WIRE\_0);

b2v\_Z : z

PORT MAP(clk => clk,

clear => clear,

Zin => Zin,

Zhi\_in => SYNTHESIZED\_WIRE\_74,

Zlo\_in => SYNTHESIZED\_WIRE\_75,

BusMuxIn\_Zhi => SYNTHESIZED\_WIRE\_40,

BusMuxIn\_Zlow => SYNTHESIZED\_WIRE\_41);

END bdf\_type;

# Block Schematics

## Datapath

A picture containing screenshot

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

# Testbench Files

## Load

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_load IS

END ENTITY tb\_load;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_load\_arch OF tb\_load IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rin\_tb, Cout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, T0, T1, T2, T3, T4, T5, T6, T7);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => '0',

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => '0',

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"00000000",

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN T5 =>

Present\_state <= T6;

WHEN T6 =>

Present\_state <= T7;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; BAout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; ADD\_tb <= '0';

Grb\_tb <= '0'; Gra\_tb <= '0';

Clear\_tb <= '0';

WHEN T0 => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800055: 32-bit instruction for LD R1, $85(R0)

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; BAout\_tb <= '1'; Yin\_tb <= '1';

WHEN T4 =>

Yin\_tb <= '0'; Grb\_tb <= '0'; BAout\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN T5 =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; MARin\_tb <= '1';

WHEN T6 =>

Zlowout\_tb <= '0'; MARin\_tb <= '0';

Read\_tb <= '1'; MDRin\_tb <= '1';

WHEN T7 =>

MDRout\_tb <= '1'; GRA\_tb <= '1'; Rin\_tb <= '1';

WHEN OTHERS =>

END CASE;

END PROCESS;

END ARCHITECTURE tb\_load\_arch;

## Load Immediate

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_loadi IS

END ENTITY tb\_loadi;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_loadi\_arch OF tb\_loadi IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rin\_tb, Cout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, T0, T1, T2, T3, T4, T5);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => '0',

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => '0',

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"00000000",

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; BAout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; ADD\_tb <= '0';

Grb\_tb <= '0'; Gra\_tb <= '0';

Clear\_tb <= '0';

WHEN T0 => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800055: 32-bit instruction for LDI R1, $85

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Yin\_tb <= '1'; Grb\_tb <= '1'; BAout\_tb <= '1';

WHEN T4 =>

Yin\_tb <= '0'; Grb\_tb <= '0'; BAout\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN T5 =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

WHEN OTHERS =>

END CASE;

END PROCESS;

END ARCHITECTURE tb\_loadi\_arch;

## Load with Register Offset

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_load\_with\_reg IS

END ENTITY tb\_load\_with\_reg;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_load\_with\_reg\_arch OF tb\_load\_with\_reg IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rout\_tb, Rin\_tb, Cout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, preLoadRegA, preLoadRegB, preLoadRegC, preLoadRegD, preLoadRegE, preLoadRegF,

T0, T1, T2, T3, T4, T5, T6, T7);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => '0',

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => Rout\_tb,

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"00000000",

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= preLoadRegA;

WHEN preLoadRegA =>

Present\_state <= preLoadRegB;

WHEN preLoadRegB =>

Present\_state <= preLoadRegC;

WHEN preLoadRegC =>

Present\_state <= preLoadRegD;

WHEN preLoadRegD =>

Present\_state <= preLoadRegE;

WHEN preLoadRegE =>

Present\_state <= preLoadRegF;

WHEN preLoadRegF =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN T5 =>

Present\_state <= T6;

WHEN T6 =>

Present\_state <= T7;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; Rout\_tb <= '0'; BAout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; ADD\_tb <= '0';

Grb\_tb <= '0'; Gra\_tb <= '0';

Clear\_tb <= '0';

WHEN preLoadRegA => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN preLoadRegB =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800055: 32-bit instruction for LDI R1, $55

WHEN preLoadRegC =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN preLoadRegD =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; BAout\_tb <= '1'; Yin\_tb <= '1';

WHEN preLoadRegE =>

Grb\_tb <= '0'; BAout\_tb <= '0'; Yin\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN preLoadRegF =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

WHEN T0 => -- see if you need to de-assert these signals

Zlowout\_tb <= '0'; Gra\_tb <= '0'; Rin\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00080030: 32-bit instruction for LD R0, $30(R1)

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; Rout\_tb <= '1'; Yin\_tb <= '1';

WHEN T4 =>

Grb\_tb <= '0'; Rout\_tb <= '0'; Yin\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN T5 =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; MARin\_tb <= '1';

WHEN T6 =>

Zlowout\_tb <= '0'; MARin\_tb <= '0';

Read\_tb <= '1'; MDRin\_tb <= '1';

WHEN T7 =>

MDRout\_tb <= '1'; GRA\_tb <= '1'; Rin\_tb <= '1';

WHEN OTHERS =>

END CASE;

END PROCESS;

END ARCHITECTURE tb\_load\_with\_reg\_arch;

## Load Immediate with Register Offset

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_loadi\_with\_reg IS

END ENTITY tb\_loadi\_with\_reg;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_loadi\_with\_reg\_arch OF tb\_loadi\_with\_reg IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rout\_tb, Rin\_tb, Cout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, preLoadRegA, preLoadRegB, preLoadRegC, preLoadRegD, preLoadRegE, preLoadRegF,

T0, T1, T2, T3, T4, T5);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => '0',

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => Rout\_tb,

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"00000000",

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= preLoadRegA;

WHEN preLoadRegA =>

Present\_state <= preLoadRegB;

WHEN preLoadRegB =>

Present\_state <= preLoadRegC;

WHEN preLoadRegC =>

Present\_state <= preLoadRegD;

WHEN preLoadRegD =>

Present\_state <= preLoadRegE;

WHEN preLoadRegE =>

Present\_state <= preLoadRegF;

WHEN preLoadRegF =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; Rout\_tb <= '0'; BAout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; ADD\_tb <= '0';

Grb\_tb <= '0'; Gra\_tb <= '0';

Clear\_tb <= '0';

WHEN preLoadRegA => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN preLoadRegB =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800055: 32-bit instruction for LDI R1, $55

WHEN preLoadRegC =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN preLoadRegD =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; BAout\_tb <= '1'; Yin\_tb <= '1';

WHEN preLoadRegE =>

Grb\_tb <= '0'; BAout\_tb <= '0'; Yin\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN preLoadRegF =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

WHEN T0 => -- see if you need to de-assert these signals

Zlowout\_tb <= '0'; Gra\_tb <= '0'; Rin\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00080030: 32-bit instruction for LD R0, $30(R1)

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; Rout\_tb <= '1'; Yin\_tb <= '1';

WHEN T4 =>

Grb\_tb <= '0'; Rout\_tb <= '0'; Yin\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN T5 =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

WHEN OTHERS =>

END CASE;

END PROCESS;

END ARCHITECTURE tb\_loadi\_with\_reg\_arch;

## Store

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_store IS

END ENTITY tb\_store;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_store\_arch OF tb\_store IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, Write\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rout\_tb, Cout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, Rin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, T0, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => Write\_tb,

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => '0',

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => Rout\_tb,

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"00000000",

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN T5 =>

Present\_state <= T6;

WHEN T6 =>

Present\_state <= T7;

WHEN T7 =>

Present\_state <= T8;

WHEN T8 =>

Present\_state <= T9;

WHEN T9 =>

Present\_state <= T10;

WHEN T10 =>

Present\_state <= T11;

WHEN T11 =>

Present\_state <= T12;

WHEN T12 =>

Present\_state <= T13;

WHEN T13 =>

Present\_state <= T14;

WHEN T14 =>

Present\_state <= T15;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; BAout\_tb <= '0'; Rout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; Write\_tb <= '0'; ADD\_tb <= '0';

Gra\_tb <= '0'; Grb\_tb <= '0';

Clear\_tb <= '0';

WHEN T0 => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800090: 32-bit instruction for ST $90(R0), R1

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Yin\_tb <= '1'; Grb\_tb <= '1'; BAout\_tb <= '1';

WHEN T4 =>

Yin\_tb <= '0'; Grb\_tb <= '0'; BAout\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN T5 =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; MARin\_tb <= '1';

WHEN T6 =>

Zlowout\_tb <= '0'; MARin\_tb <= '0';

MDRin\_tb <= '1'; Gra\_tb <= '1'; Rout\_tb <= '1';

WHEN T7 =>

MDRin\_tb <= '0'; Gra\_tb <= '0'; Rout\_tb <= '0';

MDRout\_tb <= '1'; Write\_tb <= '1';

WHEN T8 => -- begin load

MDRout\_tb <= '0'; Write\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN T9 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800090: 32-bit instruction for LD R1, $90(R0)

WHEN T10 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T11 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; BAout\_tb <= '1'; Yin\_tb <= '1';

WHEN T12 =>

Yin\_tb <= '0'; Grb\_tb <= '0'; BAout\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN T13 =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; MARin\_tb <= '1';

WHEN T14 =>

Zlowout\_tb <= '0'; MARin\_tb <= '0';

Read\_tb <= '1'; MDRin\_tb <= '1';

WHEN T15 =>

MDRout\_tb <= '1'; GRA\_tb <= '1'; Rin\_tb <= '1';

WHEN OTHERS =>

END CASE;

END PROCESS;

END ARCHITECTURE tb\_store\_arch;

## Store with Register Offset

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_store\_with\_reg IS

END ENTITY tb\_store\_with\_reg;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_store\_with\_reg\_arch OF tb\_store\_with\_reg IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, Write\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rout\_tb, Cout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, Rin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, preLoadRegA, preLoadRegB, preLoadRegC, preLoadRegD, preLoadRegE, preLoadRegF,

T0, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => Write\_tb,

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => '0',

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => Rout\_tb,

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"00000000",

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= preLoadRegA;

WHEN preLoadRegA =>

Present\_state <= preLoadRegB;

WHEN preLoadRegB =>

Present\_state <= preLoadRegC;

WHEN preLoadRegC =>

Present\_state <= preLoadRegD;

WHEN preLoadRegD =>

Present\_state <= preLoadRegE;

WHEN preLoadRegE =>

Present\_state <= preLoadRegF;

WHEN preLoadRegF =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN T5 =>

Present\_state <= T6;

WHEN T6 =>

Present\_state <= T7;

WHEN T7 =>

Present\_state <= T8;

WHEN T8 =>

Present\_state <= T9;

WHEN T9 =>

Present\_state <= T10;

WHEN T10 =>

Present\_state <= T11;

WHEN T11 =>

Present\_state <= T12;

WHEN T12 =>

Present\_state <= T13;

WHEN T13 =>

Present\_state <= T14;

WHEN T14 =>

Present\_state <= T15;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; BAout\_tb <= '0'; Rout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; Write\_tb <= '0'; ADD\_tb <= '0';

Gra\_tb <= '0'; Grb\_tb <= '0';

Clear\_tb <= '0';

WHEN preLoadRegA => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN preLoadRegB =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800055: 32-bit instruction for LDI R1, $35

WHEN preLoadRegC =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN preLoadRegD =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; BAout\_tb <= '1'; Yin\_tb <= '1';

WHEN preLoadRegE =>

Grb\_tb <= '0'; BAout\_tb <= '0'; Yin\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN preLoadRegF =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

WHEN T0 => -- see if you need to de-assert these signals

Zlowout\_tb <= '0'; Gra\_tb <= '0'; Rin\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00880090: 32-bit instruction for ST $90(R1), R1

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Yin\_tb <= '1'; Grb\_tb <= '1'; BAout\_tb <= '1';

WHEN T4 =>

Yin\_tb <= '0'; Grb\_tb <= '0'; BAout\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN T5 =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; MARin\_tb <= '1';

WHEN T6 =>

Zlowout\_tb <= '0'; MARin\_tb <= '0';

MDRin\_tb <= '1'; Gra\_tb <= '1'; Rout\_tb <= '1';

WHEN T7 =>

MDRin\_tb <= '0'; Gra\_tb <= '0'; Rout\_tb <= '0';

MDRout\_tb <= '1'; Write\_tb <= '1';

WHEN T8 => -- begin load

MDRout\_tb <= '0'; Write\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN T9 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800090: 32-bit instruction for LD R1, $90(R0)

WHEN T10 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T11 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Yin\_tb <= '1'; Grb\_tb <= '1'; BAout\_tb <= '1';

WHEN T12 =>

Yin\_tb <= '0'; Grb\_tb <= '0'; BAout\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN T13 =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; MARin\_tb <= '1';

WHEN T14 =>

Zlowout\_tb <= '0'; MARin\_tb <= '0';

Read\_tb <= '1'; MDRin\_tb <= '1';

WHEN T15 =>

MDRout\_tb <= '1'; GRA\_tb <= '1'; Rin\_tb <= '1';

WHEN OTHERS =>

END CASE;

END PROCESS;

END ARCHITECTURE tb\_store\_with\_reg\_arch;

## ADD/AND/OR Immediate

Note that everything is the same except for the instruction control signals (ADDin, ANDin, Orin)

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_addi IS

END ENTITY tb\_addi;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_addi\_arch OF tb\_addi IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rout\_tb, Rin\_tb, Cout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, preLoadRegA, preLoadRegB, preLoadRegC, preLoadRegD, preLoadRegE, preLoadRegF,

T0, T1, T2, T3, T4, T5);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => '0',

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => Rout\_tb,

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"00000000",

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= preLoadRegA;

WHEN preLoadRegA =>

Present\_state <= preLoadRegB;

WHEN preLoadRegB =>

Present\_state <= preLoadRegC;

WHEN preLoadRegC =>

Present\_state <= preLoadRegD;

WHEN preLoadRegD =>

Present\_state <= preLoadRegE;

WHEN preLoadRegE =>

Present\_state <= preLoadRegF;

WHEN preLoadRegF =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; Rout\_tb <= '0'; BAout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; ADD\_tb <= '0';

Grb\_tb <= '0'; Gra\_tb <= '0';

Clear\_tb <= '0';

WHEN preLoadRegA => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN preLoadRegB =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800006: 32-bit instruction for LDI R1, 6

WHEN preLoadRegC =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN preLoadRegD =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; BAout\_tb <= '1'; Yin\_tb <= '1';

WHEN preLoadRegE =>

Grb\_tb <= '0'; BAout\_tb <= '0'; Yin\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN preLoadRegF =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

WHEN T0 => -- see if you need to de-assert these signals

Zlowout\_tb <= '0'; Gra\_tb <= '0'; Rin\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 010FFFFB: 32-bit instruction for ADDI R2, R1, -5

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; Rout\_tb <= '1'; Yin\_tb <= '1';

WHEN T4 =>

Grb\_tb <= '0'; Rout\_tb <= '0'; Yin\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN T5 =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

WHEN OTHERS =>

END CASE;

END PROCESS;

END ARCHITECTURE tb\_addi\_arch;

## BRZR/BRNZ/BRPL/BRMI

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_branch IS

END ENTITY tb\_branch;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_branch\_arch OF tb\_branch IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, MDRin\_tb, MDRout\_tb, CONin\_tb, CONout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rin\_tb, Rout\_tb, Cout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, preLoadRegA, preLoadRegB, preLoadRegC, preLoadRegD, preLoadRegE, preLoadRegF, preLoadPCA, preLoadPCB, preLoadPCC,

T0, T1, T2, T3, T4, T5, T6);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => '0',

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => Rout\_tb,

BAout => BAout\_tb,

CONin => CONin\_tb,

InputDeviceData => x"00000000",

CONout => CONout\_tb,

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= preLoadRegA;

WHEN preLoadRegA =>

Present\_state <= preLoadRegB;

WHEN preLoadRegB =>

Present\_state <= preLoadRegC;

WHEN preLoadRegC =>

Present\_state <= preLoadRegD;

WHEN preLoadRegD =>

Present\_state <= preLoadRegE;

WHEN preLoadRegE =>

Present\_state <= preLoadRegF;

WHEN preLoadRegF =>

Present\_state <= preLoadPCA;

WHEN preLoadPCA =>

Present\_state <= preLoadPCB;

WHEN preLoadPCB =>

Present\_state <= preLoadPCC;

WHEN preLoadPCC =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN T5 =>

Present\_state <= T6;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; Rout\_tb <= '0'; BAout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0'; CONin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; ADD\_tb <= '0';

Grb\_tb <= '0'; Gra\_tb <= '0';

Clear\_tb <= '0';

WHEN preLoadRegA => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN preLoadRegB =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800055: 32-bit instruction for LDI R1, $55

WHEN preLoadRegC =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN preLoadRegD =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; BAout\_tb <= '1'; Yin\_tb <= '1';

WHEN preLoadRegE =>

Grb\_tb <= '0'; BAout\_tb <= '0'; Yin\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN preLoadRegF =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

WHEN preLoadPCA =>

Zlowout\_tb <= '0'; Gra\_tb <= '0'; Rin\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

WHEN preLoadPCB =>

PCout\_tb <= '0'; MARin\_tb <= '0';

MDRin\_tb <= '1';

WHEN preLoadPCC =>

MDRin\_tb <= '0'; Read\_tb <= '0';

PCin\_tb <= '1'; MDRout\_tb <= '1';

-- load PC with 0x0000000A

WHEN T0 => -- see if you need to de-assert these signals

PCin\_tb <= '0'; MDRout\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 01000035: 32-bit instruction for BRZR R2, $35

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Gra\_tb <= '1'; Rout\_tb <= '1'; CONin\_tb <= '1';

WHEN T4 =>

Gra\_tb <= '0'; Rout\_tb <= '0'; CONin\_tb <= '0';

PCout\_tb <= '1'; Yin\_tb <= '1';

WHEN T5 =>

PCout\_tb <= '0'; Yin\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN T6 =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1';

if (CONout\_tb = '1') then

PCin\_tb <= '1';

end if;

WHEN OTHERS =>

END CASE;

END PROCESS;

END ARCHITECTURE tb\_branch\_arch;

## Jump

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_jump IS

END ENTITY tb\_jump;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_jump\_arch OF tb\_jump IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rin\_tb, Rout\_tb, Cout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, preLoadRegA, preLoadRegB, preLoadRegC, preLoadRegD, preLoadRegE, preLoadRegF,

T0, T1, T2, T3);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => '0',

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => Rout\_tb,

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"00000000",

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= preLoadRegA;

WHEN preLoadRegA =>

Present\_state <= preLoadRegB;

WHEN preLoadRegB =>

Present\_state <= preLoadRegC;

WHEN preLoadRegC =>

Present\_state <= preLoadRegD;

WHEN preLoadRegD =>

Present\_state <= preLoadRegE;

WHEN preLoadRegE =>

Present\_state <= preLoadRegF;

WHEN preLoadRegF =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; Rout\_tb <= '0'; BAout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; ADD\_tb <= '0';

Grb\_tb <= '0'; Gra\_tb <= '0';

Clear\_tb <= '0';

WHEN preLoadRegA => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN preLoadRegB =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800055: 32-bit instruction for LDI R1, $55

WHEN preLoadRegC =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN preLoadRegD =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; BAout\_tb <= '1'; Yin\_tb <= '1';

WHEN preLoadRegE =>

Grb\_tb <= '0'; BAout\_tb <= '0'; Yin\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN preLoadRegF =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

WHEN T0 => -- see if you need to de-assert these signals

Zlowout\_tb <= '0'; Gra\_tb <= '0'; Rin\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 0080000A: 32-bit instruction for JR R1

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Gra\_tb <= '1'; Rout\_tb <= '1'; PCin\_tb <= '1';

END CASE;

END PROCESS;

END ARCHITECTURE tb\_jump\_arch;

## Jump and Link

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_jump\_and\_link IS

END ENTITY tb\_jump\_and\_link;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_jump\_and\_link\_arch OF tb\_jump\_and\_link IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rin\_tb, Rout\_tb, Cout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, preLoadRegA, preLoadRegB, preLoadRegC, preLoadRegD, preLoadRegE, preLoadRegF,

T0, T1, T2, T3);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => '0',

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => Rout\_tb,

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"00000000",

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= preLoadRegA;

WHEN preLoadRegA =>

Present\_state <= preLoadRegB;

WHEN preLoadRegB =>

Present\_state <= preLoadRegC;

WHEN preLoadRegC =>

Present\_state <= preLoadRegD;

WHEN preLoadRegD =>

Present\_state <= preLoadRegE;

WHEN preLoadRegE =>

Present\_state <= preLoadRegF;

WHEN preLoadRegF =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; Rout\_tb <= '0'; BAout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; ADD\_tb <= '0';

Grb\_tb <= '0'; Gra\_tb <= '0';

Clear\_tb <= '0';

WHEN preLoadRegA => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN preLoadRegB =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 0F000000: 32-bit instruction for LD R15, (R0)PC+1

WHEN preLoadRegC =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN preLoadRegD =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; BAout\_tb <= '1'; Yin\_tb <= '1';

WHEN preLoadRegE =>

Grb\_tb <= '0'; BAout\_tb <= '0'; Yin\_tb <= '0';

PCout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN preLoadRegF =>

PCout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

WHEN T0 => -- see if you need to de-assert these signals

Zlowout\_tb <= '0'; Gra\_tb <= '0'; Rin\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 0080000A: 32-bit instruction for JR R1

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Gra\_tb <= '1'; Rout\_tb <= '1'; PCin\_tb <= '1';

END CASE;

END PROCESS;

END ARCHITECTURE tb\_jump\_and\_link\_arch;

## MFHI/LO

The only difference is the control signals for the registers (HIout/in vs LOout/in)

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_mfhi IS

END ENTITY tb\_mfhi;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_mfhi\_arch OF tb\_mfhi IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb, HIin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rin\_tb, Rout\_tb, Cout\_tb, HIout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, preLoadHIRegA, preLoadHIRegB, preLoadHIRegC, T0, T1, T2, T3, T4, T5);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => HIin\_tb,

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => '0',

PCout => PCout\_tb,

HIout => HIout\_tb,

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => Rout\_tb,

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"00000000",

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= preLoadHIRegA;

WHEN preLoadHIRegA =>

Present\_state <= preLoadHIRegB;

WHEN preLoadHIRegB =>

Present\_state <= preLoadHIRegC;

WHEN preLoadHIRegC =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; Rout\_tb <= '0'; BAout\_tb <= '0'; HIout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0'; HIin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; ADD\_tb <= '0';

Grb\_tb <= '0'; Gra\_tb <= '0';

Clear\_tb <= '0';

WHEN preLoadHIRegA => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN preLoadHIRegB =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

WHEN preLoadHIRegC =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; HIin\_tb <= '1';

WHEN T0 => -- see if you need to de-assert these signals

MDRout\_tb <= '0'; HIin\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; BAout\_tb <= '1'; Yin\_tb <= '1';

WHEN T4 =>

Grb\_tb <= '0'; BAout\_tb <= '0'; Yin\_tb <= '0';

PCout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN T5 =>

PCout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

HIout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

END CASE;

END PROCESS;

END ARCHITECTURE tb\_mfhi\_arch;

## Out

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_out\_port IS

END ENTITY tb\_out\_port;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_out\_port\_arch OF tb\_out\_port IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb, OutPortin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rin\_tb, Rout\_tb, Cout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb, OutputDeviceData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, preLoadRegA, preLoadRegB, preLoadRegC, preLoadRegD, preLoadRegE, preLoadRegF,

T0, T1, T2, T3);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => OutPortin\_tb,

InPortin => '0',

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => '0',

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => Rout\_tb,

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"00000000",

BusData => BusData\_tb,

OutputDeviceData => OutputDeviceData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= preLoadRegA;

WHEN preLoadRegA =>

Present\_state <= preLoadRegB;

WHEN preLoadRegB =>

Present\_state <= preLoadRegC;

WHEN preLoadRegC =>

Present\_state <= preLoadRegD;

WHEN preLoadRegD =>

Present\_state <= preLoadRegE;

WHEN preLoadRegE =>

Present\_state <= preLoadRegF;

WHEN preLoadRegF =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; Rout\_tb <= '0'; BAout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0'; OutPortin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; ADD\_tb <= '0';

Grb\_tb <= '0'; Gra\_tb <= '0';

Clear\_tb <= '0';

WHEN preLoadRegA => -- see if you need to de-assert these signals

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

Clear\_tb <= '1';

WHEN preLoadRegB =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

-- 00800055: 32-bit instruction for LDI R1, $55

WHEN preLoadRegC =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN preLoadRegD =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Grb\_tb <= '1'; BAout\_tb <= '1'; Yin\_tb <= '1';

WHEN preLoadRegE =>

Grb\_tb <= '0'; BAout\_tb <= '0'; Yin\_tb <= '0';

Cout\_tb <= '1'; ADD\_tb <= '1'; Zin\_tb <= '1';

WHEN preLoadRegF =>

Cout\_tb <= '0'; ADD\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; Gra\_tb <= '1'; Rin\_tb <= '1';

WHEN T0 => -- see if you need to de-assert these signals

Zlowout\_tb <= '0'; Gra\_tb <= '0'; Rin\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Gra\_tb <= '1'; Rout\_tb <= '1'; OutPortin\_tb <= '1';

END CASE;

END PROCESS;

END ARCHITECTURE tb\_out\_port\_arch;

## In

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_in\_port IS

END ENTITY tb\_in\_port;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_in\_port\_arch OF tb\_in\_port IS -- Add any other signals to see in your simulation

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL PCout\_tb, PCin\_tb, IncPC\_tb, IRin\_tb, InPortin\_tb: std\_logic;

SIGNAL MARin\_tb, Read\_tb, MDRin\_tb, MDRout\_tb: std\_logic;

SIGNAL Gra\_tb, Grb\_tb, BAout\_tb, Rin\_tb, Rout\_tb, Cout\_tb, InPortout\_tb: std\_logic;

SIGNAL Yin\_tb, Zin\_tb, ADD\_tb, Zlowout\_tb: std\_logic;

SIGNAL BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, preLoadInPort, T0, T1, T2, T3);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Write : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

Grb : IN STD\_LOGIC;

Gra : IN STD\_LOGIC;

Grc : IN STD\_LOGIC;

Rin : IN STD\_LOGIC;

Rout : IN STD\_LOGIC;

BAout : IN STD\_LOGIC;

CONin : IN STD\_LOGIC;

InputDeviceData : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

CONout : OUT STD\_LOGIC;

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutputDeviceData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

HIin => '0',

LOin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => ADD\_tb,

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

Write => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

InPortin => InPortin\_tb,

PCout => PCout\_tb,

HIout => '0',

LOout => '0',

Zhighout => '0',

Zlowout => Zlowout\_tb,

MDRout => MDRout\_tb,

Cout => Cout\_tb,

InPortout => InPortout\_tb,

IRin => IRin\_tb,

Grb => Grb\_tb,

Gra => Gra\_tb,

Grc => '0',

Rin => Rin\_tb,

Rout => Rout\_tb,

BAout => BAout\_tb,

CONin => '0',

InputDeviceData => x"0F0F0F0F",

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

Clock\_tb <= '1', '0' after 20 ns;

wait for 40 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= preLoadInPort;

WHEN preLoadInPort =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

Cout\_tb <= '0'; Rout\_tb <= '0'; BAout\_tb <= '0'; InPortout\_tb <= '0';

MARin\_tb <= '0'; PCin\_tb <='0'; MDRin\_tb <= '0'; InPortin\_tb <= '0';

IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0'; Rin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; ADD\_tb <= '0';

Grb\_tb <= '0'; Gra\_tb <= '0';

Clear\_tb <= '0';

WHEN preLoadInPort => -- see if you need to de-assert these signals

InPortin\_tb <= '1';

Clear\_tb <= '1';

WHEN T0 => -- see if you need to de-assert these signals

InPortin\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1'; Read\_tb <= '1';

WHEN T1 =>

PCout\_tb <= '0'; MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; MDRin\_tb <= '1';

WHEN T2 =>

IncPC\_tb <= '0'; PCin\_tb <= '0'; MDRin\_tb <= '0'; Read\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

Gra\_tb <= '1'; InPortout\_tb <= '1'; Rin\_tb <= '1';

END CASE;

END PROCESS;

END ARCHITECTURE tb\_in\_port\_arch;

# Output Waveforms

## Load R1, $55

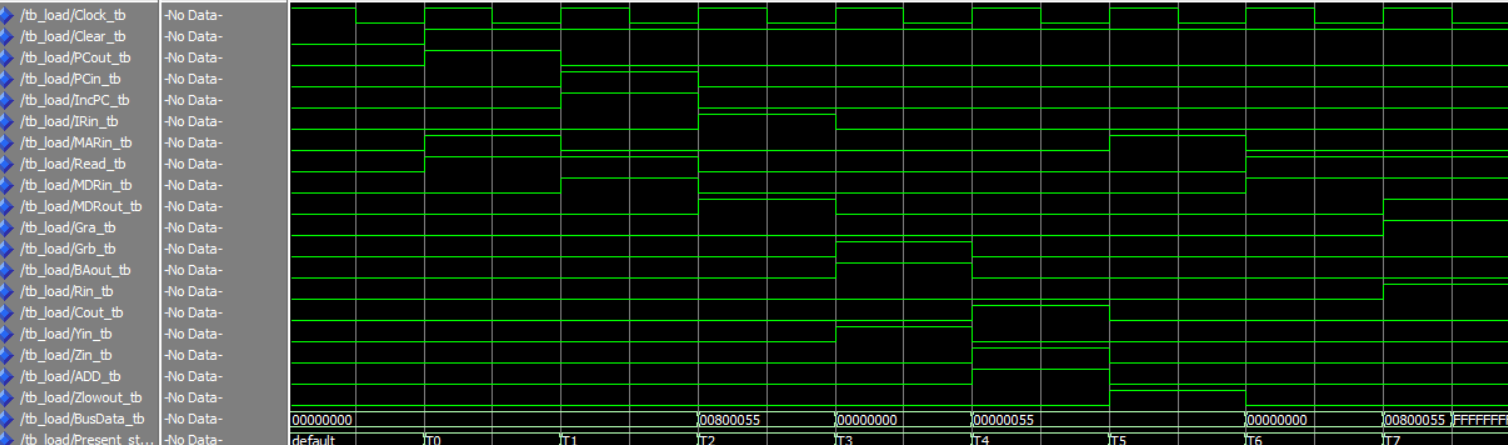
**RAM Initializations (this is the same for all load instructions)**

ram(0) <= x"00800055"; -- Load first instruction at address 0 of ram

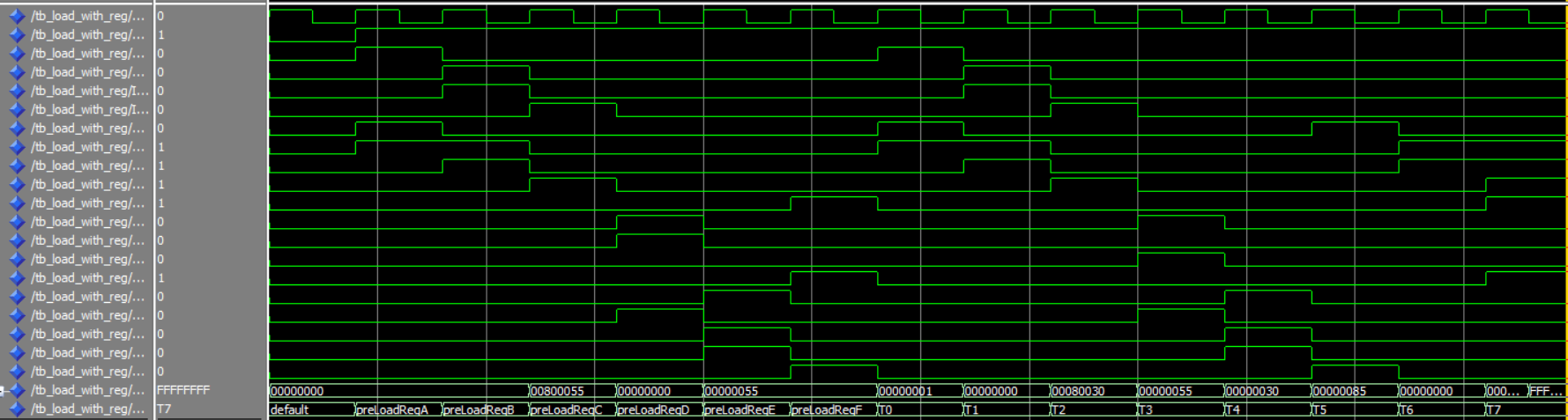
ram(1) <= x"00080030"; -- Load next instructions at the next ram address the next ram address

ram(85) <= x"FFFFFFFF"; -- This will change based on what type of instruction we're doing, used to verify correctness

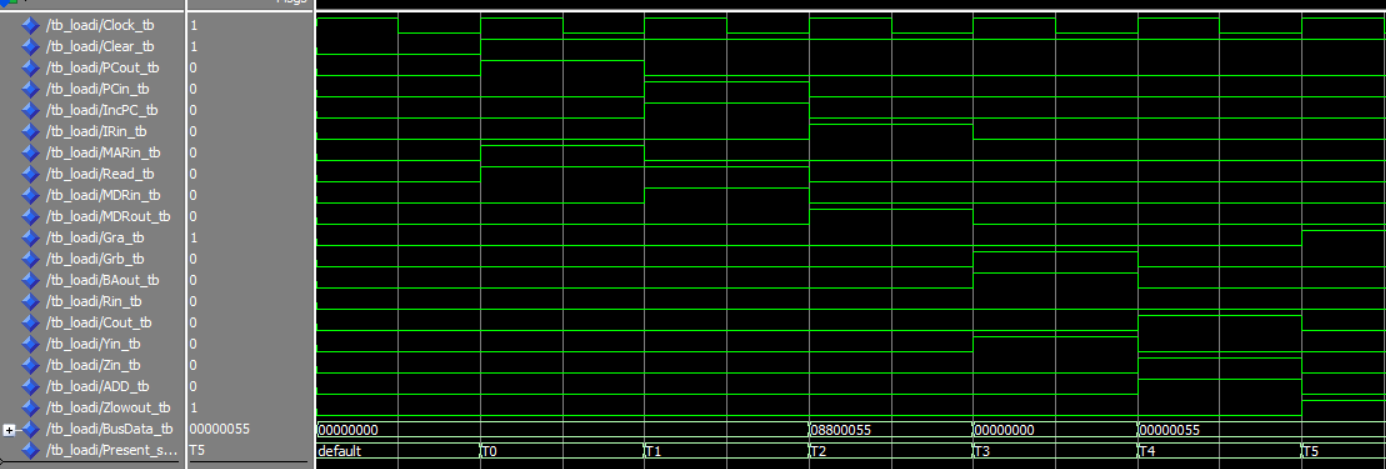
ram(133) <= x"FFFFFFFF";



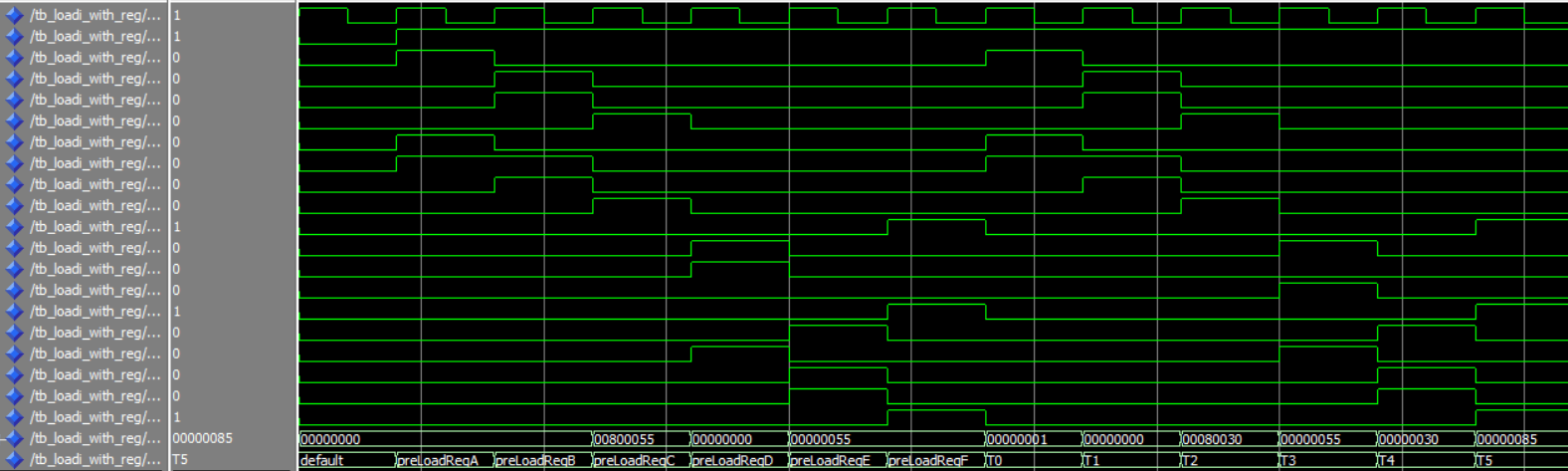
## Load R0, $35(R1)



## Loadi R1, $55



## Loadi R0, $35(R1)



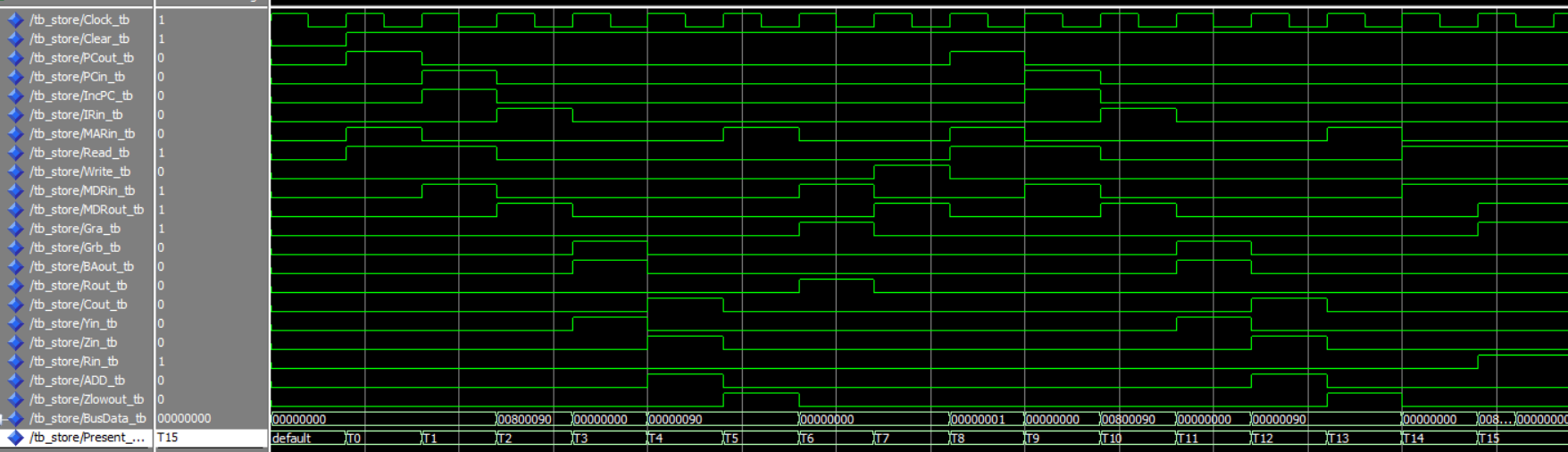
## Store $90, R1

**RAM Initializations**

ram(0) <= x"00800090"; -- Load first instruction at address 0 of ram

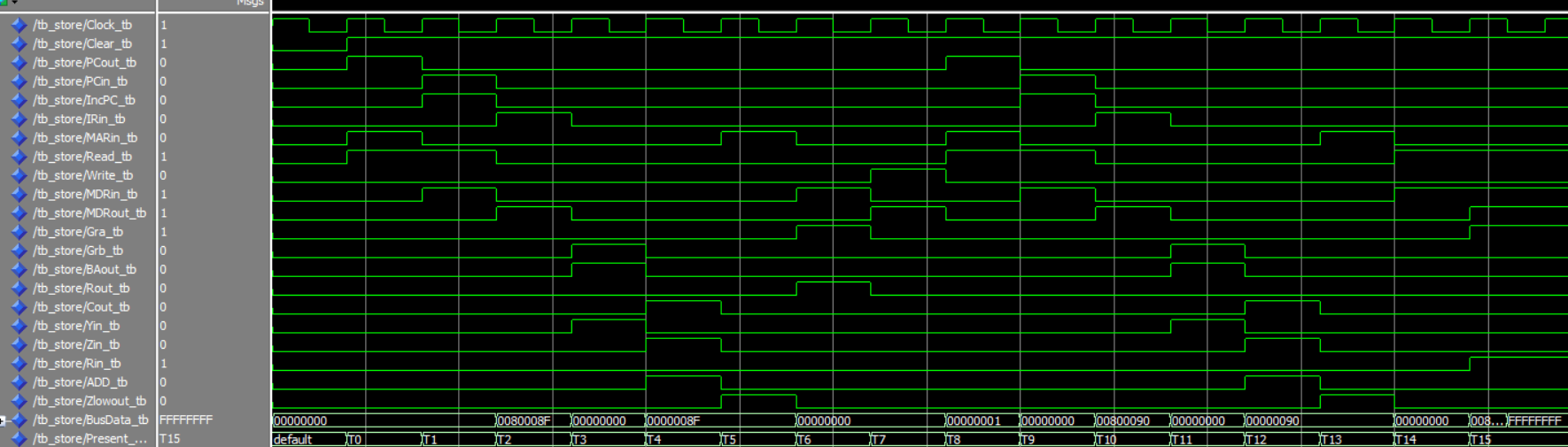
ram(1) <= x"00800090"; -- Load next instructions at the next ram address the next ram address

ram(144) <= x"FFFFFFFF"; -- This should be overwritten to 0x00000000 if store works correctly



## Store $8F, R1

I’m showing this waveform to prove that the store instruction does work



## Store $90, (R1)R1

**RAM Initialization**

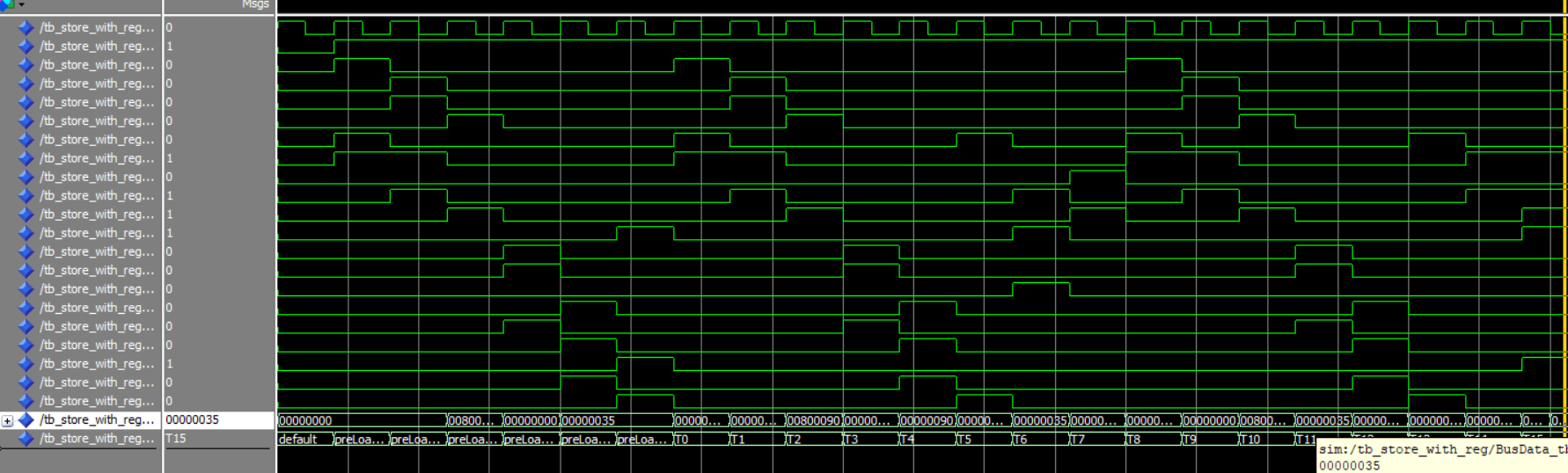
ram(0) <= x"00800035"; -- Load first instruction at address 0 of ram

ram(1) <= x"00800090"; -- Load next instructions at the next ram address the next ram address

ram(2) <= x"00800090";

ram(53) <= x"00000001";

ram(145) <= x"FFFFFFFF"; -- This should be overwritten to 0x00000000 if store works correctly

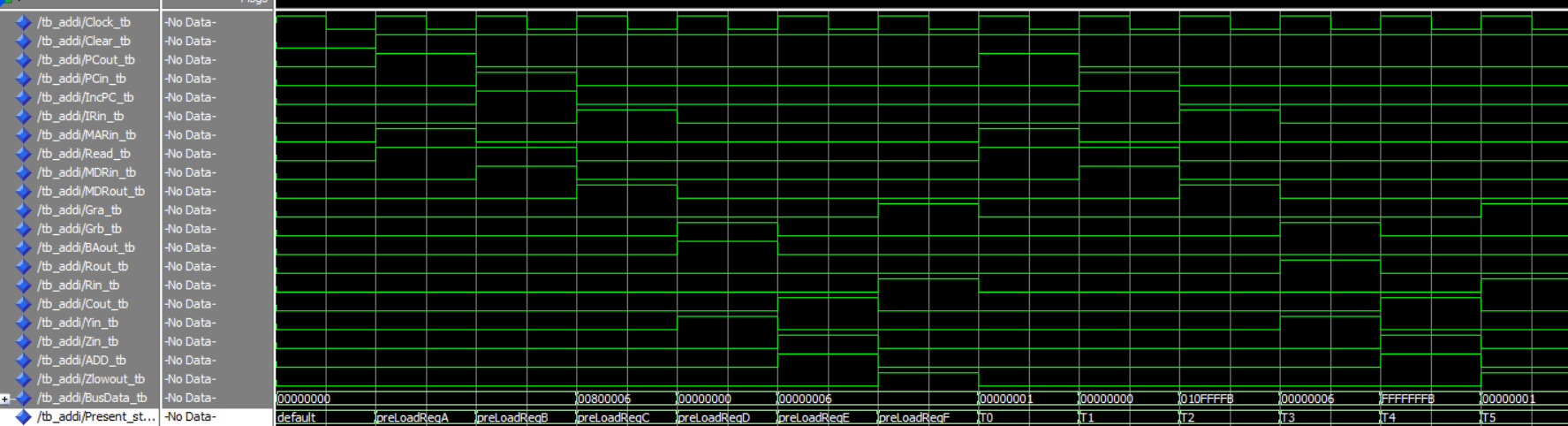


## Addi R2, R1, -5

**RAM Initialization**

ram(0) <= x"00800006"; -- Load first instruction at address 0 of ram

ram(1) <= x"010FFFFB"; -- Load next instructions at the next ram address the next ram address

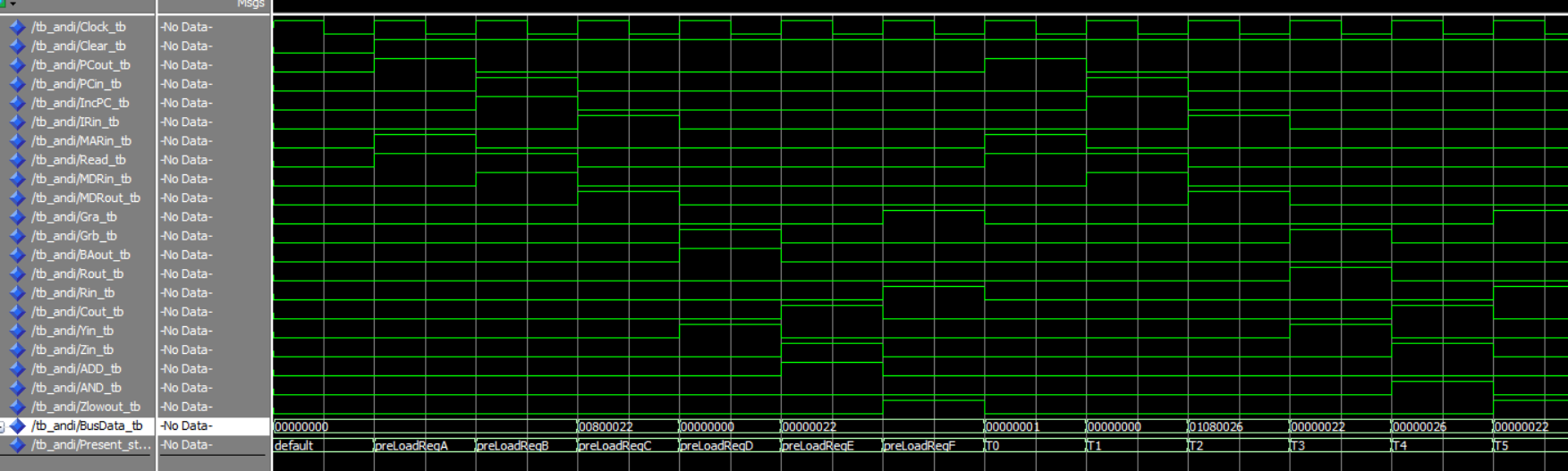


## Andi R2, R1, $26

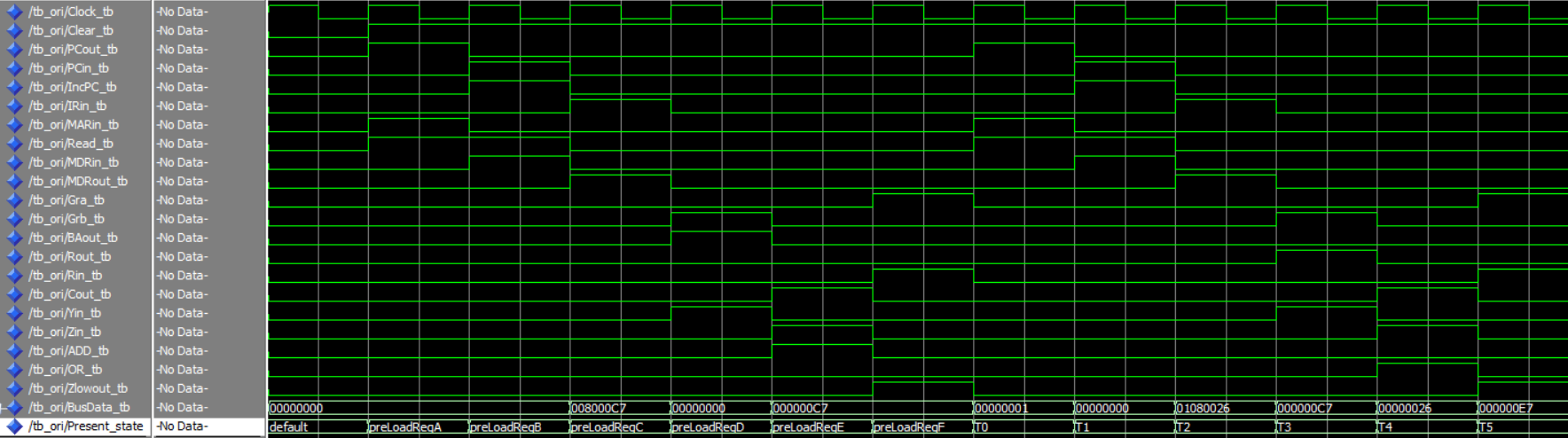
**RAM Initialization (this is the same for ANDi and ORi)**

ram(0) <= x"00800022"; -- Load first instruction at address 0 of ram

ram(1) <= x"01080026"; -- Load next instructions at the next ram address the next ram address



## Ori R2, R1, $26



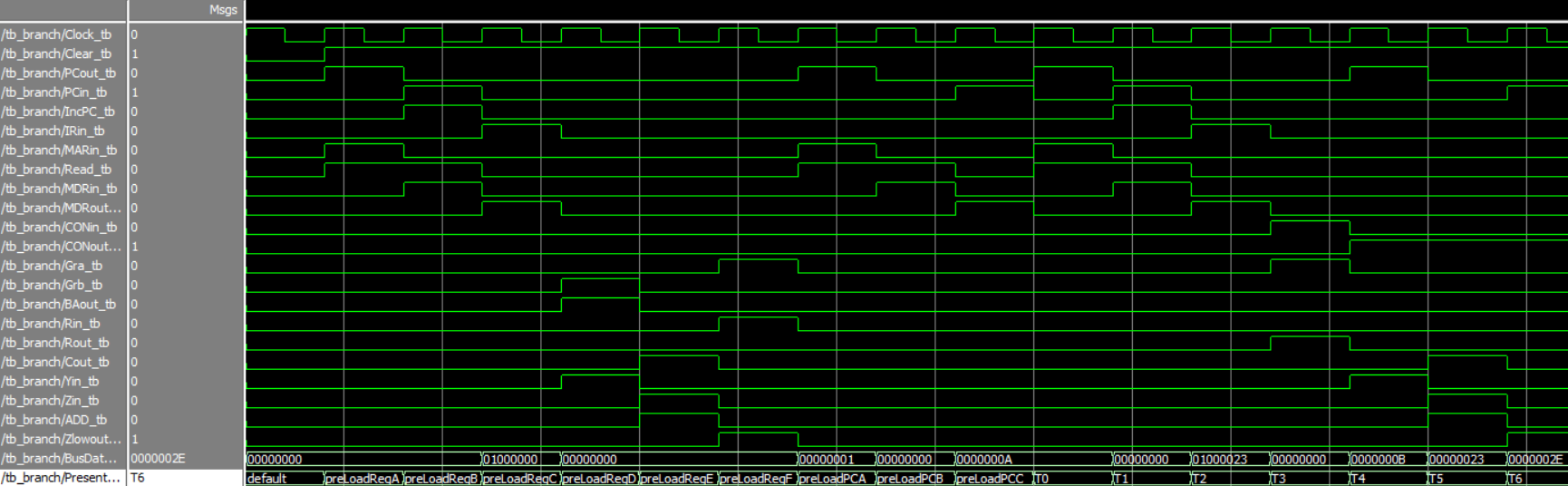
## Brzr R2, 35 (branch correctly taken)

**RAM Initialization**

ram(0) <= x"01000000"; -- Load first instruction at address 0 of ram

ram(1) <= x"0000000A"; -- Load next instructions at the next ram address the next ram address

ram(10) <= x"01000023";



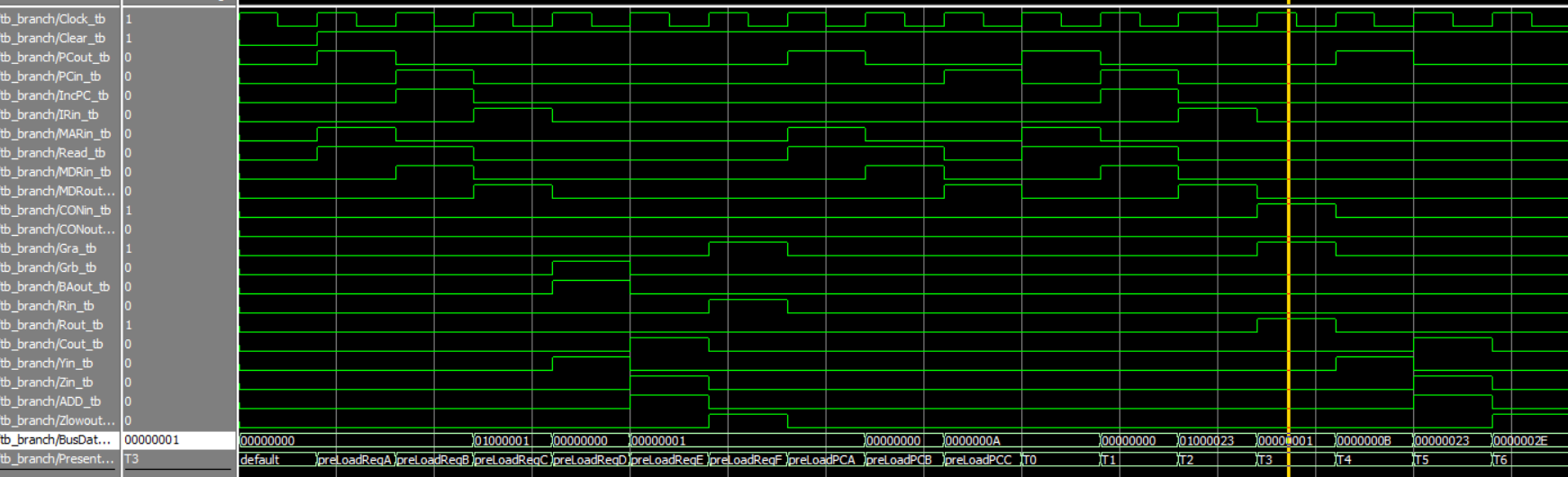
## Brzr R2, 35 (branch not taken)

**RAM Initialization**

ram(0) <= x"01000001"; -- Load first instruction at address 0 of ram

ram(1) <= x"0000000A"; -- Load next instructions at the next ram address the next ram address

ram(10) <= x"01000023";



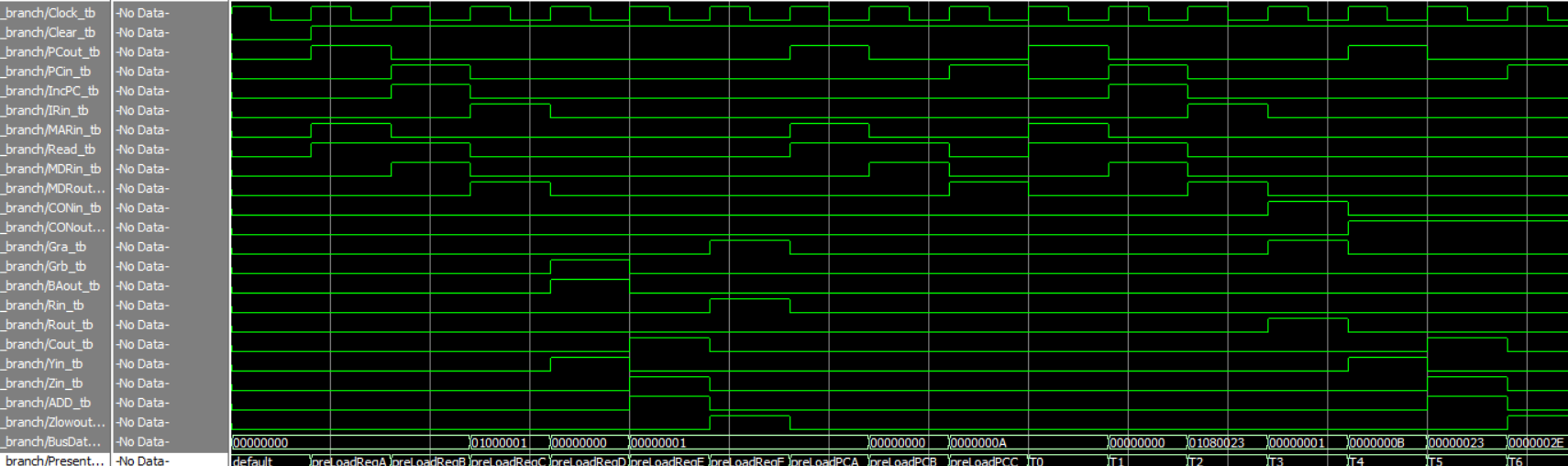
## Brnz R2, 35 (branch correctly taken)

**RAM Initialization**

ram(0) <= x"01000001"; -- Load first instruction at address 0 of ram

ram(1) <= x"0000000A"; -- Load next instructions at the next ram address the next ram address

ram(10) <= x"01080023";



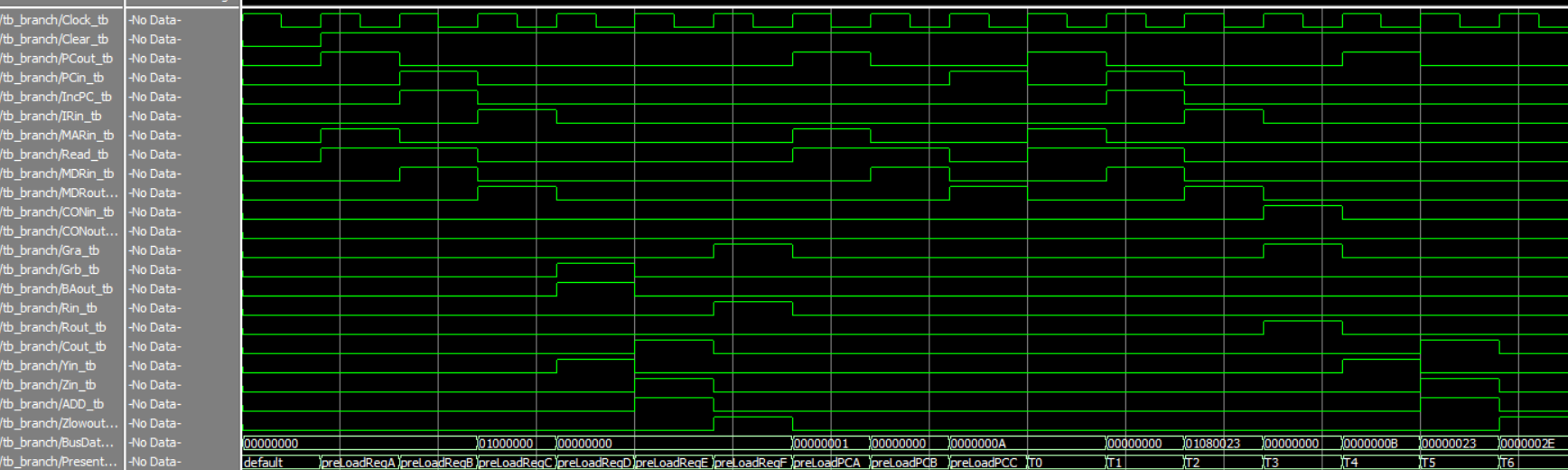
## Brnz R2, 35 (branch not taken)

**RAM Initialization**

ram(0) <= x"01000000"; -- Load first instruction at address 0 of ram

ram(1) <= x"0000000A"; -- Load next instructions at the next ram address the next ram address

ram(10) <= x"01080023";



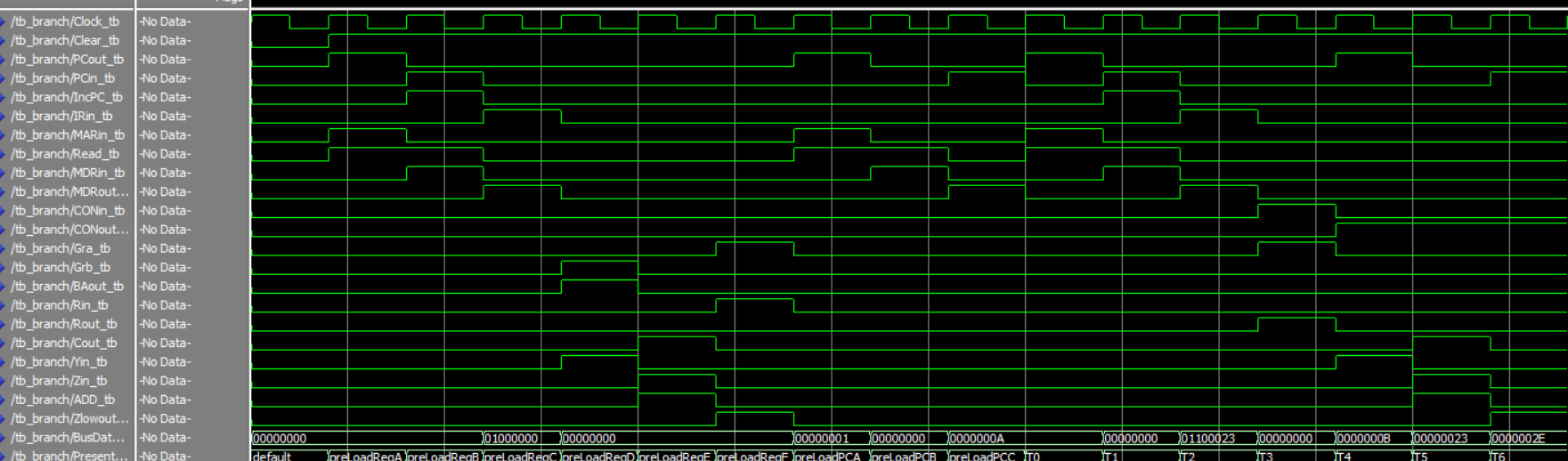
## Brpl R2, 35 (branch correctly taken)

**RAM Initialization**

ram(0) <= x"01000001"; -- Load first instruction at address 0 of ram

ram(1) <= x"0000000A"; -- Load next instructions at the next ram address the next ram address

ram(10) <= x"01080023";



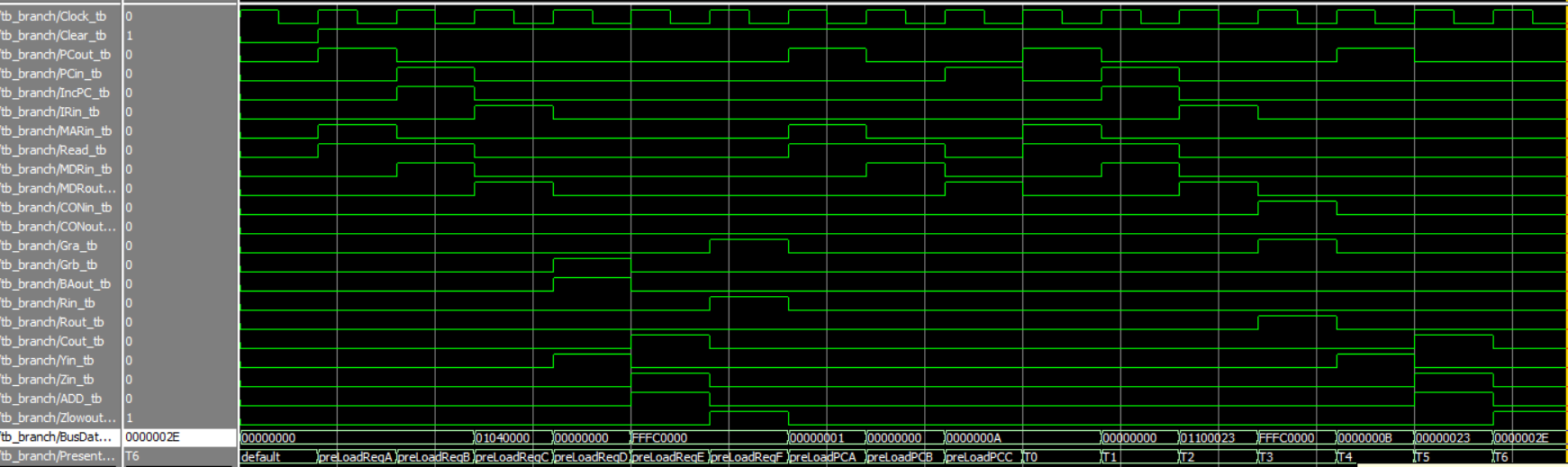
## Brpl R2, 35 (branch not taken)

**RAM Initialization**

ram(0) <= x"01040000"; -- Load first instruction at address 0 of ram

ram(1) <= x"0000000A"; -- Load next instructions at the next ram address the next ram address

ram(10) <= x"01100023";



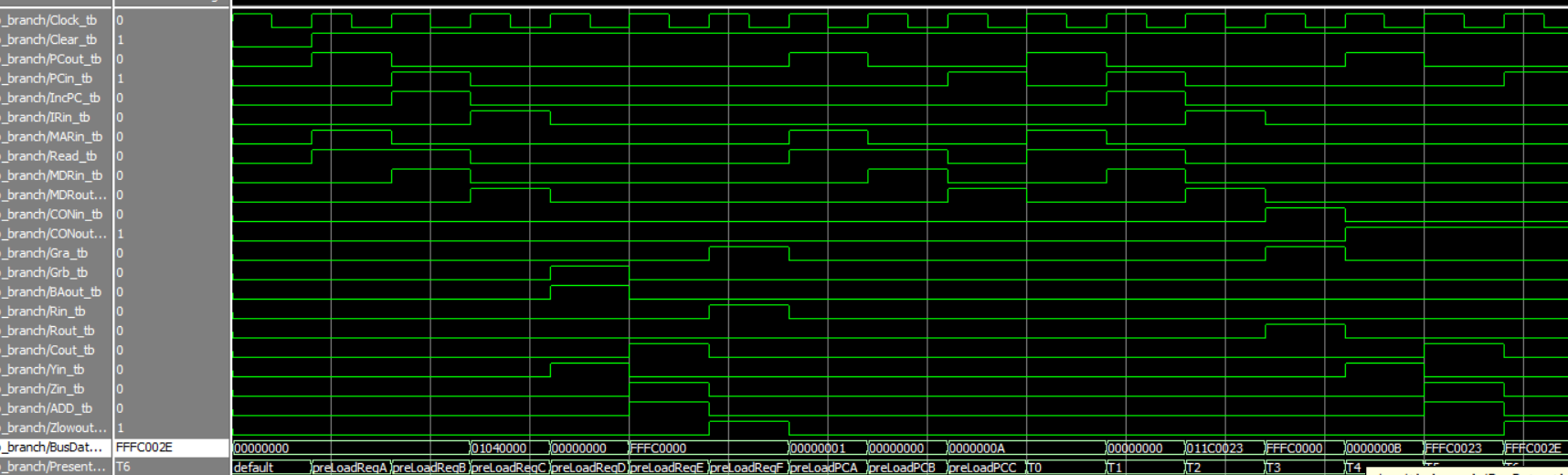
## Brmi R2, 35 (branch correctly taken)

**RAM Initialization**

ram(0) <= x"01040000"; -- Load first instruction at address 0 of ram

ram(1) <= x"0000000A"; -- Load next instructions at the next ram address the next ram address

ram(10) <= x"011C0023";



## Brmi R2, 35 (branch not taken)

**RAM Initialization**

ram(0) <= x"01000000"; -- Load first instruction at address 0 of ram

ram(1) <= x"0000000A"; -- Load next instructions at the next ram address the next ram address

ram(10) <= x"011C0023";

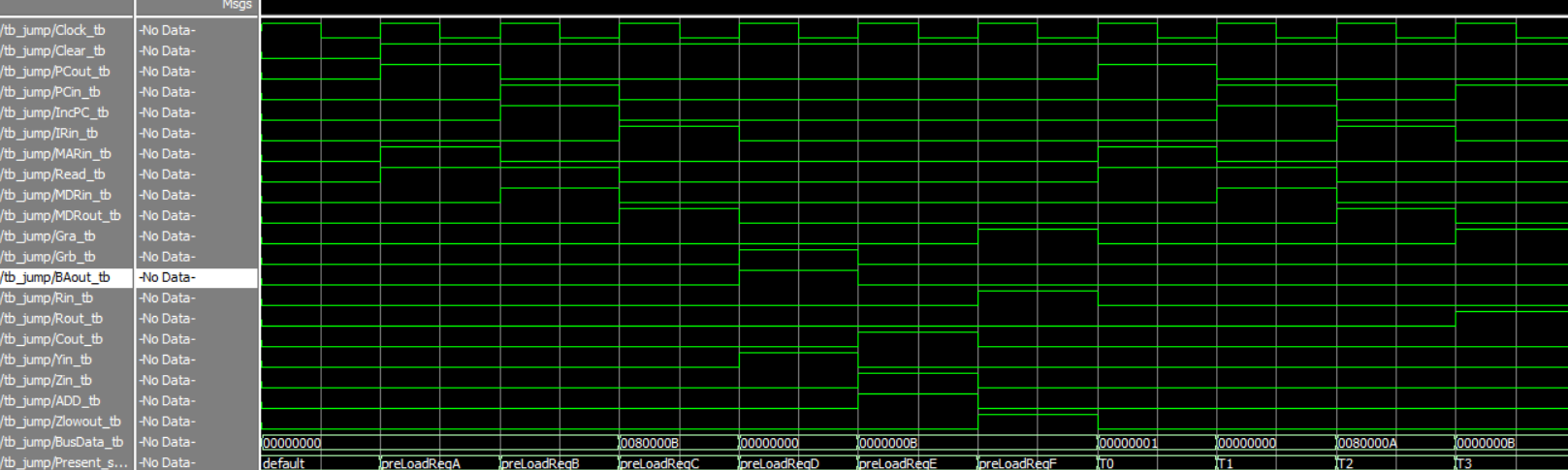


## Jr R1

**RAM Initialization**

ram(0) <= x"0080000B"; -- Load first instruction at address 0 of ram

ram(1) <= x"0080000A"; -- Load next instructions at the next ram address the next ram address

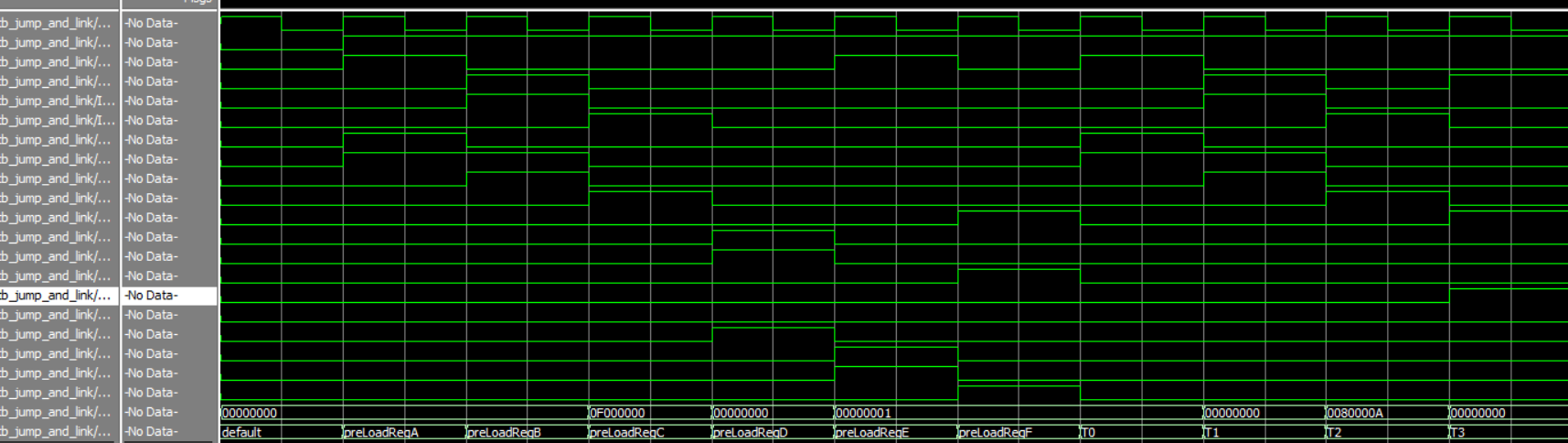


## Jal R1

**RAM Initialization**

ram(0) <= x"0F000000"; -- Load first instruction at address 0 of ram

ram(1) <= x"0080000A"; -- Load next instructions at the next ram address the next ram address



## Mfhi R2

**RAM Initialization (same for Mflo)**

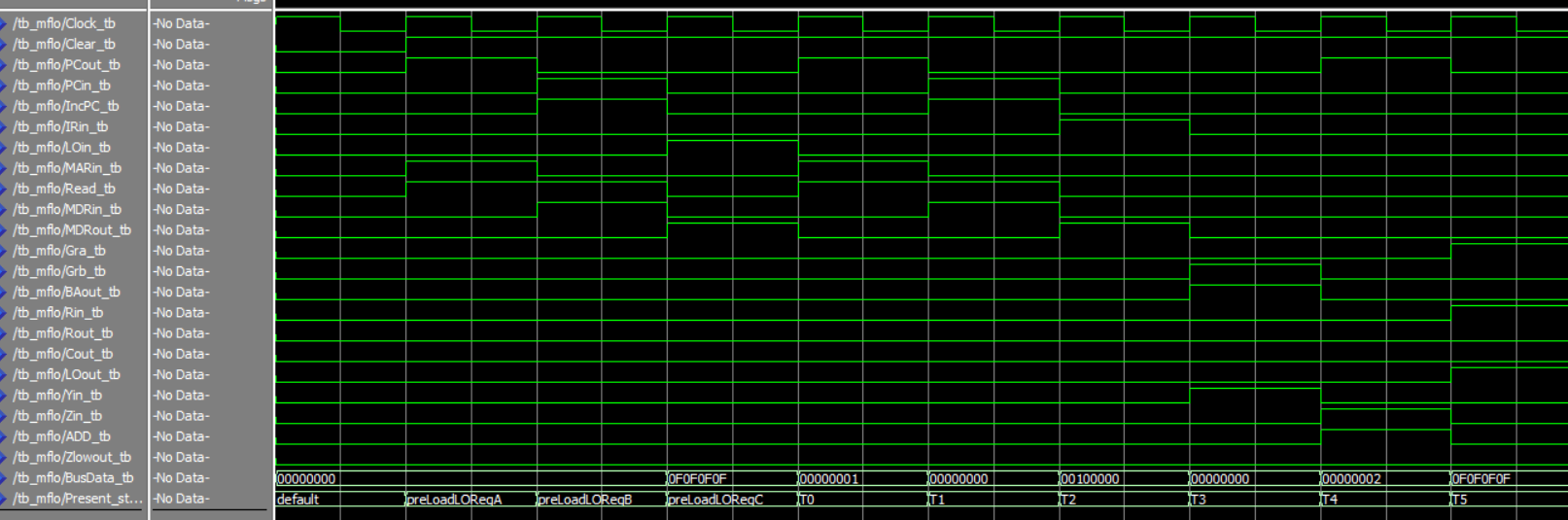
ram(0) <= x"0F0F0F0F"; -- Load first instruction at address 0 of ram

ram(1) <= x"00100000"; -- Load next instructions at the next ram address the next ram address

A picture containing fabric

Description automatically generated

## Mflo R2

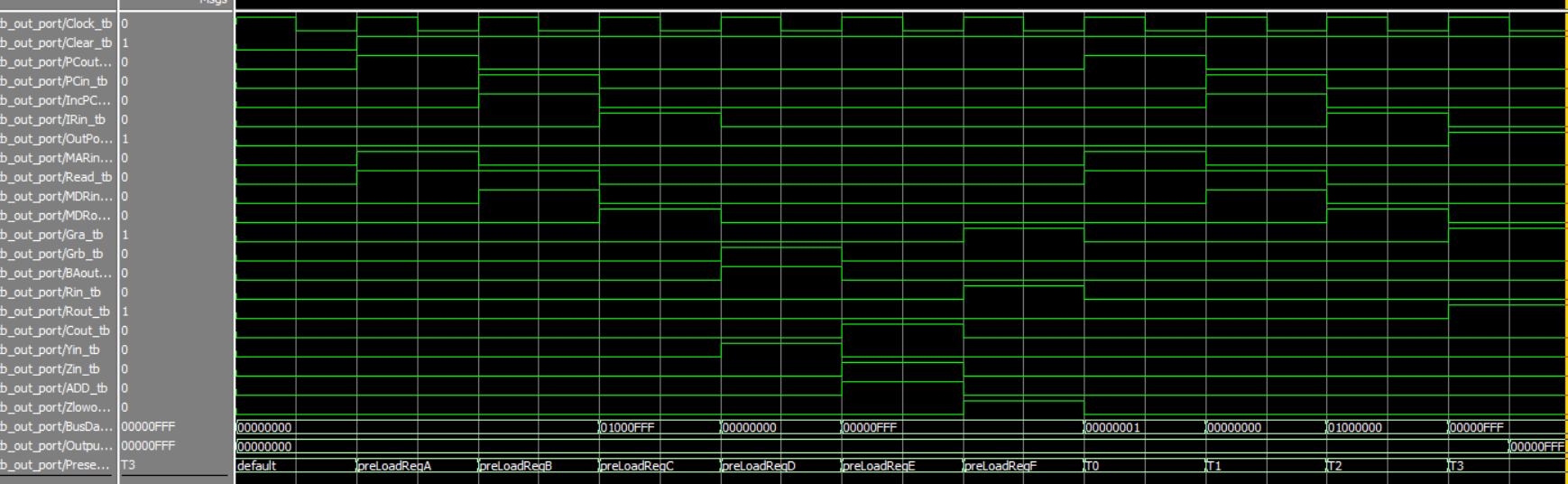


## Out R2

**RAM Initialization**

ram(0) <= x"01000FFF"; -- Load first instruction at address 0 of ram

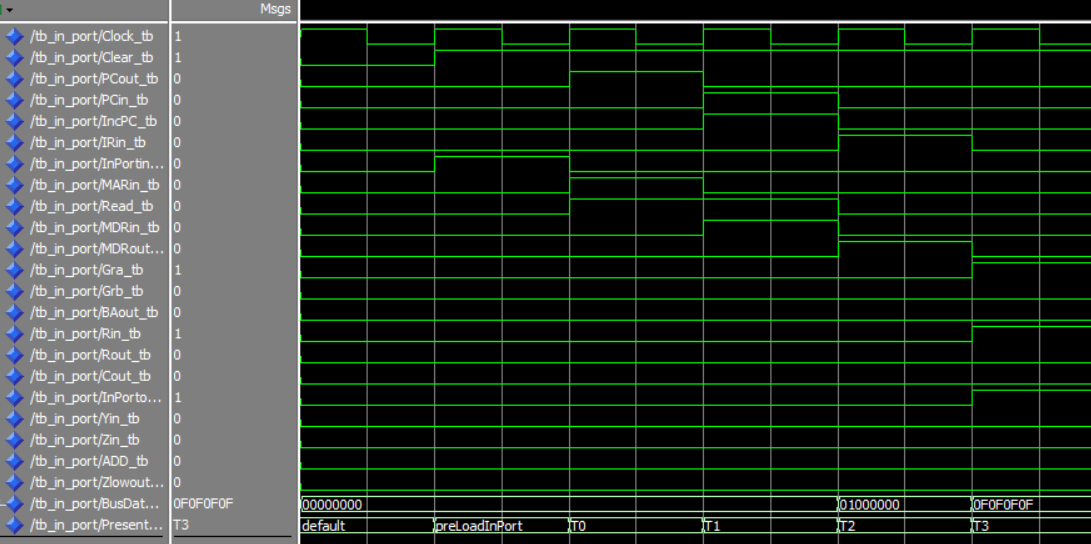
ram(1) <= x"01000000"; -- Load next instructions at the next ram address the next ram address



## In R1

**RAM Initialization**

ram(0) <= x"01000000"; -- Load first instruction at address 0 of ram



# Memory Contents

The following waveforms display the contents of RAM after a simulated store and load instruction have taken place (I have already proved in the sections above that these operations work), Note that it takes 2 clock cycles for the RAM to be written to/read from

