UNIVERSITY of York

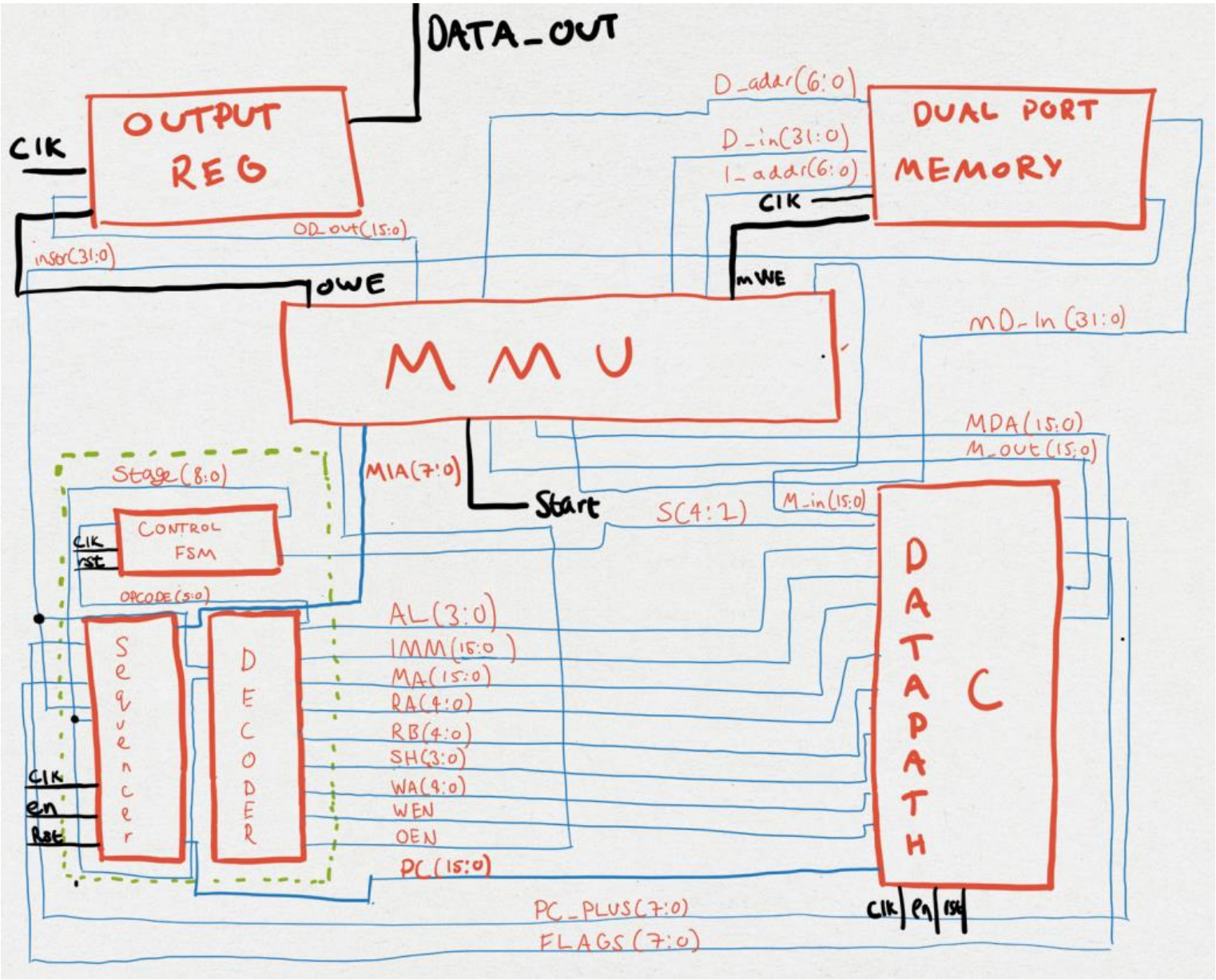
Computer Architectures Project

32/16 BIT PROCESSOR

Y3840426 & Y3839090 | ELE00009I | 22/05/2017

Contents

Processor Block Diagram



Instruction Coding

There are several things to consider when doing instruction coding. First of all, it must be considered that the instructions need to have a certain amount of uniformity. As seen in the table below displaying our instruction sets, the operands are placed in the same place throughout. This means that it is easier to use them for decoding, as well as allowing for a faster, smaller and more efficient design of processor. Also seen below is that we have split the operands for the instruction from the opcode. This adds to readability of the instruction as the opcode is the first 6 bits that the reader will see.

Another important aspect of the instruction coding is preventing overlap. This is very important as assigning certain bits of an operand to the same location in the instruction as another will cause serious errors in the architecture as the instructions will not be carried through correctly. This is the most important aspect, so if necessary, uniformity can be lost in order to stop overlapping of operands.

One final thing to consider is grouping instructions together that contain the same operands. This is useful as if they have exactly the same operands, they can have exactly the same instruction formations. This is seen in the table below, for example with 'shl/shr/rol/ror '.

add/sub/and/or/xor rt, ra, rb	OPCODE	x	x	x	x	x	B	B	B	B	B	x	x	x	x	x	x	A	A	A	A	A	T	T	T	T	T
inc/dec/not/move/loadr rt,ra ; storr rb,ra	OPCODE	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	A	A	A	A	A	T	T	T	T	T
addi/subi/amdi/ori/xori rt, ra, imm	OPCODE	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	A	A	A	A	A	T	T	T	T	T
shl/shr/rol/ror rt,ra,n	OPCODE	x	x	x	x	x	x	n	n	n	n	x	x	x	x	x	x	A	A	A	A	A	T	T	T	T	T
loadi rt, imm ; stori rb, imm	OPCODE	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	x	x	x	x	x	T/B	T/B	T/B	T/B	T/B
loado rt, ra, off; storo rt, ra, off	OPCODE	x	x	x	x	x	x	O	O	O	O	O	O	O	O	O	O	A	A	A	A	A	T/B	T/B	T/B	T/B	T/B
jmp off	OPCODE	x	O	O	O	O	O	O	O	O	O	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
brc ra, cond, off	OPCODE	x	O	O	O	O	O	O	O	O	O	x	x	x	x	x	x	A	A	A	A	A	x	x	x	x	x

Table showing the instruction sets used

Instruction	OPCODE	STEP	STATE[4,1]	AL	OEN	WEN
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx00	1010	0	0
add rt, ra, rb	000100	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx00	1011	0	0
sub rt, ra, rb	000101	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx01	1010	0	0
addi rt, ra, imm	000110	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx01	1011	0	0
subi rt, ra, imm	000111	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx0x	1000	0	0
inc rt, ra	001000	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx0x	1001	0	0
dec rt, ra	001001	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx0x	0111	0	0
not rt, ra	01 0000	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx00	0100	0	0
and rt, ra, rb	01 0001	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx00	0101	0	0
or rt, ra, rb	01 0010	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx00	0110	0	0
xor rt, ra, rb	01 0011	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx01	0100	0	0
andi rt, ra, imm	01 0100	RegWr	0xxx	xxxx	0	1
ori rt, ra, imm	01 0101	FETCH	xx1x	1000	0	0

		RegRd	xxxx	xxxx	0	0
		ALU	xx01	0101	0	0
		RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx01	0110	0	0
xori rt, ra, imm	01 0110	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx0x	1100	0	0
shl rt, ra, n	01 0000	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx0x	1101	0	0
shr rt, ra, n	01 1001	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx0x	1110	0	0
rotl rt, ra, n	01 1010	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx0x	1111	0	0
rotl rt, ra, n	01 1011	RegWr	0xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx0x	0000	0	0
move rt, ra	10 0000	RegWr	0xxx	xxxx	0	0
		FETCH	xx1x	1000	0	0
		MemAcc	x1xx	xxxx	0	0
loadi rt, imm	10 0001	RegWr	1xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx0x	0000	0	0
		MemAcc	x0xx	xxxx	0	0
loadr rt, ra	10 0010	RegWr	1xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx01	1010	0	0
		MemAcc	x0xx	xxxx	0	0
loado rt, ra, off	10 0011	Regwr	1xxx	xxxx	0	1
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
stori rb, imm	10 0101	MemAcc	x1xx	xxxx	1	0
		FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx0x	0000	0	0
storr rb, ra	10 0110	MemAcc	x1xx	xxxx	1	0
storo rb, ra, off	10 0111	FETCH	xx1x	1000	0	0

		RegRd	xxxx	xxxx	0	0
		ALU	xx01	1010	0	0
		MemAcc	x1xx	xxxx	1	0
jmp off	11 0111	FETCH	xx1x	1000	0	0
		ALU	xx11	1010	0	0
brc ra, cond, off	11 0ccc	FETCH	xx1x	1000	0	0
		RegRd	xxxx	xxxx	0	0
		ALU	xx0x	0000	0	0

Table showing the control signals for all instruction types

Category	Instruction	Operands	Opcode
No-operation	nop	N/A	00 0000
Arithmetic	add	rt,ra,rb	00 0100
	sub	rt,ra,rb	00 0101
	addi	rt,ra,imm	00 0110
	subi	rt,ra,imm	00 0111
	inc	rt,ra	00 1000
	dec	rt,ra	00 1001
Logic	not	rt,ra	01 0000
	and	rt,ra,rb	01 0001
	or	rt,ra,rb	01 0010
	xor	rt,ra,rb	01 0011
	andi	rt,ra,imm	01 0100
	ori	rt,ra,imm	01 0101
	xori	rt,ra,imm	01 0110
	shl	rt,ra,n	01 1000
	shr	rt,ra,n	01 1001
	rol	rt,ra,n	01 1010
	ror	rt,ra,n	01 1011
Transfer	move	rt,ra	10 0000
	loadi	rt,imm	10 0001
	loadr	rt,ra	10 0010
	loado	rt,ra,off	10 0011
	stori	rb,imm	10 0101
	storr	rb,ra	10 0110
	storo	rb,ra,off	10 0111
control	jmp	off	11 0111
	brc ra = 0	ra,cond,off	11 0000
	brc ra != 0	ra,cond,off	11 0001
	brc ra = 1	ra,cond,off	11 0010
	brc ra < 0	ra,cond,off	11 0011
	brc ra > 0	ra,cond,off	11 0100
	brc ra <= 0	ra,cond,off	11 0101
	brc ra >= 0	ra,cond,off	11 0101

Table showing Opcodes for every instruction type

Test Program

Assembler	Machine Language (Hexadecimal)
Nop	00000000
loadi r15, h01f0	8407C00F
brz r15,-1	C1FF800F
addi r31, r0, 000000011111	18AD9C1F
add r1, r0, r0	00000001
inc r1, r1	20000021
brlz r31, +5	CC02801F
storr r1, r31	980003E1
inc r1, r1	20000021
dec r31, r31	240003FF
jmp -4 (L1)	1DDFE0000
loadi r2, 0000000000000100	184019002
ori r30, r0, 0000000000000100	15400101E
loadr r3, r30	1880003C3
loado r4, r30, 3	18C000FC4
xori r29, r0, 1111111111111111	15BFFFC1D
move r7, r0	180000007
andi r5, r29, 0000000000010000	1500043A5
andi r6, r4, 0000000000000001	150000486
brz r6, +2	1C0010006
add r7, r3, r7	200038067
shr r4, r4, 1	264010084
shl r3, r3, 1	260010063
dec r5, r5	2240000A5
brnz r5, -6	2C5FD0005
stori r7, 0	294000007
rol r7, r7, 9	2680900E7
ror r7, r7, 3	26C0300E7
not r8, r7	2400000E8
xor r8, r29, r8	24C0403A8
sub r9, r8, r7	314038109
subi r10, r9, 0000000000000001	31C00052A
brgz r10, +9	3D004800A
brz r10, +8	3C004000A
addi r11, r10, 0000000000000010	31800294B
brz r11, +6	3C003000B
or r12, r7, r11	3480580EC
storo r12, r0, 1	39C00040C
and r12, r12, r11	34405818C
bro r12, =1, +3	3C800800C
jmp +0	4DC000000
jmp +0	4DC000000
stori r7, 000000011111000	4940FFC07
jmp +0	4DC000000

VHDL CODE

PROCESSOR.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- ====
-- Processor
-- Top level schematic that connects all of the main blocks of the
-- CPU together.
-- ====

entity Processor is
    Port (
        clk : in  STD_LOGIC;
        en : in STD_LOGIC;
        rst : in  STD_LOGIC;

        start : in  STD_LOGIC;
        data_out : out  STD_LOGIC_VECTOR (15 downto 0)
    );
end Processor;



architecture Behavioral of Processor is

    --signal data_in : STD_LOGIC_VECTOR(15 downto 0);
    signal mWE : STD_LOGIC;
    signal oWE : STD_LOGIC;
    signal mDAddress : STD_LOGIC_VECTOR(6 downto 0);
    signal DPD_out : STD_LOGIC_VECTOR(31 downto 0);
    signal DPD_in : STD_LOGIC_VECTOR(31 downto 0);
    signal I_addr : STD_LOGIC_VECTOR(6 downto 0);
    signal I_out : STD_LOGIC_VECTOR(31 downto 0);
    signal oD_out : STD_LOGIC_VECTOR(15 downto 0);
    signal Instr_Addr : STD_LOGIC_VECTOR(7 downto 0);



    signal D_addr : STD_LOGIC_VECTOR(6 downto 0);

    signal Flags : STD_LOGIC_VECTOR(7 downto 0);
    signal PC : STD_LOGIC_VECTOR(7 downto 0);
    signal S : STD_LOGIC_VECTOR (4 downto 1);
    signal RA : STD_LOGIC_VECTOR (4 downto 0);
    signal RB : STD_LOGIC_VECTOR (4 downto 0);
    signal WA : STD_LOGIC_VECTOR (4 downto 0);
    signal MA : STD_LOGIC_VECTOR (15 downto 0);
    signal IMM : STD_LOGIC_VECTOR (15 downto 0);
    signal AL : STD_LOGIC_VECTOR (3 downto 0);
    signal SH : STD_LOGIC_VECTOR (3 downto 0);

    signal OEN : STD_LOGIC;
    signal WEN : STD_LOGIC;
```

```
    signal MDA : STD_LOGIC_VECTOR (15 downto 0);
    signal M_in : STD_LOGIC_VECTOR (15 downto 0);
    signal M_out : STD_LOGIC_VECTOR (15 downto 0);

    signal PC_PLUS_16 : STD_LOGIC_VECTOR(15 downto 0) := (others => '0' );
    signal PC_16 : STD_LOGIC_VECTOR(15 downto 0) := (others => '0' );
    signal PC_PLUS : STD_LOGIC_VECTOR(7 downto 0);

begin

    Inst_OutputReg: entity work.OutputReg PORT MAP(
        clk => clk,
        data_in => oD_out,
        WE => oWE,
        data_out => data_out
    );

    --DUAL PORT MEMORY
    Inst_DualPortMemory: entity work.DualPortMemory PORT MAP(
        clk => clk,
        I_addr => I_addr,
        D_addr => mDAddress,
        D_in => DPD_in,
        WE => mWE,
        D_out => DPD_out,
        I_out => I_out
    );

    -- MMU
    Inst_MMU: entity work.MMU PORT MAP(
        I_addr => Instr_addr,
        OEn => OEN,
        D_in => M_out,
        D_out => M_in,
        D_addr => MDA,
        start => start,
        oWE => oWE,
        oD_out => oD_out,
        mWE => mWE,
        mD_addr => mDAddress,
        mD_in => DPD_out,
        mD_out => DPD_in,
        mI_addr => I_addr
    );

    Inst_DataPath_C: entity work.DataPath_C PORT MAP(
        clk => clk,
        rst => rst,
        en => en,
        R_A => RA,
        R_B => RB,
        W_EN => WEN,
        W_A => WA,
        IMM => IMM,
        M_A => MA,
        M_in => M_in,
        PC => PC_16,
        S => S,
        AL => AL,
        SH => SH,
```

```
        PC_plus => PC_Plus_16,
        Flags => Flags,
        M_DA => MDA,
        M_out => M_out
    );
    PC_16(7 downto 0) <= PC;
    PC_Plus <= PC_Plus_16(7 downto 0);

    Inst_ControlUnit: entity work.ControlUnit PORT MAP(
        clk => clk,
        rst => rst,
        en => en,
        PC_Plus => PC_Plus,
        instr => I_out,
        Flags => Flags,
        Instr_addr => Instr_addr,
        PC => PC,
        S => S,
        RA => RA,
        RB => RB,
        WA => WA,
        MA => MA,
        IMM => IMM,
        AL => AL,
        SH => SH,
        WEN => WEN,
        OEN => OEN
    );



end Behavioral;
```

OUTPUTREG.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- ====
-- OutputReg
-- A simple 16bit register
-- ====

entity OutputReg is
    Port (
        clk : in STD_LOGIC;
        data_in : in  STD_LOGIC_VECTOR (15 downto 0);
        WE : in  STD_LOGIC;
        data_out : out  STD_LOGIC_VECTOR (15 downto 0)
    );
end OutputReg;

architecture Behavioral of OutputReg is

begin

    -- simple register proccess

    update : process(clk, WE)
    begin
        if (rising_edge(clk) and WE = '1') then
            data_out <= data_in;
        end if;
    end process;

end Behavioral;
```

DUALPORTMEMORY.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

-- ====
-- DualPortMemory
-- A Dual ASnyc read, single Sync write RAM block. With 128, 32 bit words.
-- Pre loaded with the instruction data.
-- ====
entity DualPortMemory is
    Port (
        clk : in std_logic; -- clock
        I_addr : in  STD_LOGIC_VECTOR (6 downto 0); -- Instruction read address
        D_addr : in  STD_LOGIC_VECTOR (6 downto 0); -- Data address
        D_in : in  STD_LOGIC_VECTOR (31 downto 0); -- Data to write
        WE : in  STD_LOGIC; -- Data write enable
        D_out : out  STD_LOGIC_VECTOR (31 downto 0); -- Data output
        I_out : out  STD_LOGIC_VECTOR (31 downto 0) -- Instruction output

    );
end DualPortMemory;

architecture Behavioral of DualPortMemory is
    type ram_type is array (0 to (2**7)-1) of std_logic_vector(31 downto 0);
    signal ram : ram_type := (
-- simple branches test program
--      0 => X"00000000",
--      1 => X"00000000",
--      2 => X"CDFF8000",
--      3 => X"00000000",
--      4 => X"C1FF8000",

-- simple adds test program
--      0 => X"00000000",
--      1 => X"18003C01",
--      2 => X"00000000",
--      3 => X"00000022",
--      4 => X"C1FF8002",

-- main test program
        0 => X"00000000",
        1 => X"8407C00F",
        2 => X"C1FF800F",
        3 => X"18007C1F",
        4 => X"00000001",
        5 => X"20000021",
        6 => X"CC02801F",
        7 => X"980003E1",
        8 => X"20000021",
        9 => X"240003FF",
        10 => X"DDFE0000",
        11 => X"84019002",
        12 => X"5400101E",
        13 => X"880003C3",
        14 => X"8C000FC4",
```

```
        15 => X"5BFFFC1D",
        16 => X"80000007",
        17 => X"500043A5",
        18 => X"50000486",
        19 => X"C0010006",
        20 => X"00038067",
        21 => X"64010084",
        22 => X"60010063",
        23 => X"240000A5",
        24 => X"C5FD0005",
        25 => X"94000007",
        26 => X"680900E7",
        27 => X"6C0300E7",
        28 => X"400000E8",
        29 => X"4C0403A8",
        30 => X"14038109",
        31 => X"1C00052A",
        32 => X"D004800A",
        33 => X"C004000A",
        34 => X"1800294B",
        35 => X"C003000B",
        36 => X"480580EC",
        37 => X"9C00040C",
        38 => X"4405818C",
        39 => X"C800800C",
        40 => X"DC000000",
        41 => X"DC000000",
        42 => X"940FFC07",
        43 => X"DC000000",
        others => X"00000000"
    );
begin

    -- Instruction reads
    I_out <= ram(to_integer(unsigned(I_addr)));

    -- Data reads
    D_out <= ram(to_integer(unsigned(D_addr)));

    -- Data writes
    write_proc : process(clk, WE)
    begin
        if (rising_edge(clk) and WE = '1') then
            ram(to_integer(unsigned(D_addr))) <= D_in;
        end if;
    end process;

end Behavioral;
```

MMU.VHD

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

-- ====
-- MMU
-- The memory management unit is resposible for mapping the datapath's full 16 bit
-- Addresses to the hardware's smaller adrres space and memory mapped peripherals
-- ====

entity MMU is
    Port (
        I_addr : in  STD_LOGIC_VECTOR (7 downto 0); -- Instruction address from control
unit

        OEn : in  STD_LOGIC; -- Memory write enable from data path
        D_in : in  STD_LOGIC_VECTOR (15 downto 0); -- data in from datapath
        D_out : out  STD_LOGIC_VECTOR (15 downto 0); -- data out to datapath
        D_addr : in  STD_LOGIC_VECTOR (15 downto 0); --  data address from datapath

        start : in  STD_LOGIC; --  memory mapped push button

        oWE : out  STD_LOGIC; -- GPIO write enable
        oD_out : out  STD_LOGIC_VECTOR (15 downto 0); -- data out to GPIO

        mWE : out  STD_LOGIC; -- Physical memory write enable
        mD_addr : out  STD_LOGIC_VECTOR (6 downto 0); -- Physical memory address
        mD_in : in  STD_LOGIC_VECTOR (31 downto 0); -- data in from pyhsical memory
        mD_out : out  STD_LOGIC_VECTOR (31 downto 0); -- data out to physical memory
        mI_addr : out  STD_LOGIC_VECTOR (6 downto 0) -- instruction address for physical
memory

    );
end MMU;

architecture Behavioral of MMU is
    signal to_write : STD_LOGIC_VECTOR (31 downto 0);

    signal isMemAccess : STD_LOGIC;
begin

    -- Flag that determins weather this will be an accress to physical memory
    isMemAccess <=
        '1' when unsigned(D_addr) >= 64 and  unsigned(D_addr) < 64+(64*2) else
        '0';

    --Instructions
    mI_addr <= I_addr(6 downto 0);

    -- Output Reg
    oD_out <= D_in;
    oWE <= '1' when D_addr = X"01F8" and OEn = '1' else '0';

    -- Data to write to memory
```

```
    to_write(31 downto 16) <=
        D_in when D_addr(0) = '0' else
        mD_in(31 downto 16);

    to_write(15 downto 0) <=
        D_in when D_addr(0) = '0' else
        mD_in(15 downto 0);

    -- map internal signal to output
    mD_out <= to_write;

    -- if its a memory access and datapath wants to write then set the physical memorys
write enable
    mWE <=
        '1' when isMemAccess = '1' and OEn = '1' else
        '0';

    -- set the physical memorys write enable
    mD_addr <= std_logic_vector(unsigned(D_addr(7 downto 1)) + to_unsigned(64,7));

    -- Data to processor
    D_out <=
        mD_in(31 downto 16)     when isMemAccess = '1' and D_addr(0) = '1'  else
        mD_in(15 downto 0)  when isMemAccess = '1' and D_addr(0) = '0'  else
        X"ffff"             when D_addr = X"01F0" and start = '1'       else
        X"0000"             when D_addr = X"01F0" and start = '0'       else
        (others =>'U');

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- ====
-- ControlFSM
-- The finate state machine that controls which stage of instruction processing
-- We are currently in. As well as deciding which stages of proccessing the
-- current instruction need to pass through. It also provide the STAGE and S
-- flags used else where in the CPU
-- ====

entity ControlFSM is
    Port (
        clk : in  STD_LOGIC; -- clock
        rst : in STD_LOGIC; -- active high sync reset
        opcode : in  STD_LOGIC_Vector(5 downto 0); -- the opcode of the current
instruction
        S : out  STD_LOGIC_VECTOR (4 downto 1); -- Selection flags which control
multiplexers in the data path
        STAGE : out STD_LOGIC_VECTOR(8 downto 0) -- The current stage of exicution
    );
end ControlFSM;

architecture Behavioral of ControlFSM is

    type STATE_TYPE is ( S0, S1, S2, S3, S4, S5, S6, S7, S8);
    signal curr_state, next_state: STATE_TYPE;

    signal optype : STD_LOGIC_VECTOR(1 downto 0);

begin

    -- The next state assignment process for this FSM
    restart : process(clk,rst) is
    begin
        if rising_edge(clk) then
            if (rst='1') then -- sync reset
                curr_state <= S0; -- S0 is the reset state
            else
                curr_state <= next_state; -- if were not reseting move to the next state
            end if;
        end if;
    end process;

    -- The next state calculation process
    control : process(curr_state, opcode, optype) is
    begin

        -- Top two bits of the opcode tell us which type of instruction this is.
        -- e.g. arithmetic or branch
        optype <= opcode(opcode'length -1 downto opcode'length -2);

        -- Decide on what the next state is based of of the opcode and optype
        case curr_state is
            when S0 =>
```

```
                next_state <= S1;
            when S1 =>
                if (optype = "11") then
                    -- Branch
                    next_state <= S8;
                elsif (optype = "10") then
                    -- Memory
                    next_state <= S4;
                else
                    -- Registers
                    next_state <= S2;
                end if;
            when S2 =>
                next_state <= S3;
            when S3 =>
                next_state <= S0;
            when S4 =>
                if (opcode(1) = '1') then
                    -- Store
                    next_state <= S5;
                else
                    -- Load
                    next_state <= S6;
                end if;
            when S5 =>
                next_state <= S0;
            when S6 =>
                next_state <= S7;
            when S7 =>
                next_state <= S0;
            when S8 =>
                next_state <= S0;
        end case;
    end process;

    -- Set the bit flags for the STAGE signal
    STAGE(0) <= '1' when curr_state = S0 else '0';
    STAGE(1) <= '1' when curr_state = S1 else '0';
    STAGE(2) <= '1' when curr_state = S2 else '0';
    STAGE(3) <= '1' when curr_state = S3 else '0';
    STAGE(4) <= '1' when curr_state = S4 else '0';
    STAGE(5) <= '1' when curr_state = S5 else '0';
    STAGE(6) <= '1' when curr_state = S6 else '0';
    STAGE(7) <= '1' when curr_state = S7 else '0';
    STAGE(8) <= '1' when curr_state = S8 else '0';

    -- Set the flags for the select (S) signal
    S(1) <=
        '-' when curr_state = S0 else
        '-' when curr_state = S1 and optype /= "11" else
        '1' when curr_state = S1 and optype = "11" else
        '0' when curr_state = S2 else
        '-' when curr_state = S3 else
        '1' when curr_state = S4 else
        '-' when curr_state = S5 else
        '-' when curr_state = S6 else
        '0' when curr_state = S7 else
        '0' when curr_state = S8 else
        'X';

    S(2) <=
```

```
        '1' when curr_state = S0 else
        '-' when curr_state = S1 and optype /= "11" else
        '1' when curr_state = S1 and optype = "11" else
        '0' when curr_state = S2 else
        '0' when curr_state = S3 else
        '0' when curr_state = S4 else
        '-' when curr_state = S5 else
        '-' when curr_state = S6 else
        '0' when curr_state = S7 else
        '0' when curr_state = S8 else
        'X';

    S(3) <= '1' when opcode = "100001" and curr_state = S7 else
            '1' when opcode = "100101" and curr_state = S7 else
            '0';

    S(4) <=
        '-' when curr_state = S0 else
        '-' when curr_state = S1 else
        '0' when curr_state = S2 else
        '0' when curr_state = S3 else
        '-' when curr_state = S4 else
        '-' when curr_state = S5 else
        '1' when curr_state = S6 else
        '-' when curr_state = S7 else
        '-' when curr_state = S8 else
        'X';

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- ====
-- Sequencer
-- Decides on what instruction to exicute next.
-- Handels branches and incrimenting the PC
-- ====

entity Sequencer is
    Port (
        clk : in  STD_LOGIC; -- clock
        rst : in STD_LOGIC; -- active high sync reset
        en : in STD_LOGIC; -- enable
        STAGE : in STD_LOGIC_VECTOR(8 downto 0); -- stage of exicution flags
        instr : in  STD_LOGIC_VECTOR (31 downto 0); -- instruction
        PC_plus : in  STD_LOGIC_VECTOR (7 downto 0); -- PC + offset
        flags : in  STD_LOGIC_VECTOR (7 downto 0); -- flags
        PC : out  STD_LOGIC_VECTOR (7 downto 0); -- program counter
        MIA : out  STD_LOGIC_VECTOR (7 downto 0) -- memory address of instruction
    );
end Sequencer;

architecture Behavioral of Sequencer is
    signal PC_internal : STD_LOGIC_VECTOR (7 downto 0) := (others => '0'); -- current
internal PC state
    signal PC_next : STD_LOGIC_VECTOR (7 downto 0); -- the next PC state

    signal cond_met : STD_LOGIC; -- whether the branching condition has been met

    signal opcode : STD_LOGIC_VECTOR(5 downto 0); -- the opcode part of the instruction
    signal cond : STD_LOGIC_VECTOR(2 downto 0); -- the condition part of the opcode

    signal cond_met_i : STD_LOGIC;
begin

    -- assign opcode and cond from the instruction
    opcode <= instr(31 downto 26);
    cond <= opcode(2 downto 0);

    -- Calculate weather the condition to branch has been met
    cond_met <=
        '1' when cond = "000" and flags(0) = '1' and STAGE(8) = '1' else  -- ra = 0
        '1' when cond = "001" and flags(1) = '1' and STAGE(8) = '1' else  -- ra != 0
        '1' when cond = "010" and flags(2) = '1' and STAGE(8) = '1' else  -- ra = 1
        '1' when cond = "011" and flags(4) = '1' and STAGE(8) = '1' else  -- ra < 0
        '1' when cond = "100" and flags(3) = '1' and STAGE(8) = '1' else  -- ra > 0
        '1' when cond = "101" and flags(6) = '1' and STAGE(8) = '1' else  -- ra <= 0
        '1' when cond = "110" and flags(5) = '1' and STAGE(8) = '1' else  -- ra >= 0
        '1' when cond = "111" else -- jump
        '0';

    -- map internal signals to outputs
    PC <= PC_internal;
    MIA <= PC_internal;
```

```
    -- The Instruction register update process
    register_proc : process(clk, rst, en, STAGE, cond_met, opcode) is
    begin
        if rising_edge(clk) then
            if rst = '1' then
                PC_internal <= (others => '0');
            elsif en = '1' and STAGE(0) = '1' then
                if opcode(5 downto 4) = "11" and cond_met_i = '1' then
                    PC_internal <= PC_next;
                else
                    PC_internal <= std_logic_vector(unsigned(PC_internal) + 1);
                end if;
            end if;
        end if;
    end process;

    -- Add a register for PC_PLUS so we can keep the value until after
    -- the condition is evaluated.
    stick_proc : process(clk, PC_plus, STAGE) is
    begin
        if rising_edge(clk) and STAGE(8) = '1' then
            PC_next <= PC_plus;
        end if;
    end process;

    -- Add a register for cond met so we can keep the value until we need
    -- to branch.
    met_proc : process(clk, cond_met, STAGE) is
    begin
        if rising_edge(clk) and STAGE(8) = '1' then
            cond_met_i <= cond_met;
        end if;
    end process;

end Behavioral;
```

```
library IEEE;
USE ieee.numeric_std.ALL;
use IEEE.STD_LOGIC_1164.ALL;

-- ====
-- Decoder
-- Decode from the current instruction to STAGE to the
-- Individual control signals need to drive the datapath.
-- ====

entity Decoder_Block is
    Port (
        instr : in  STD_LOGIC_VECTOR (31 downto 0); -- Instruction

        STAGE : in STD_LOGIC_VECTOR(8 downto 0); -- Currennt stage flags
        OPCODE : out STD_LOGIC_VECTOR(5 downto 0); -- Opcode from instruction

        RA : out  STD_LOGIC_VECTOR (4 downto 0); -- Index of working register A
        RB : out  STD_LOGIC_VECTOR (4 downto 0); -- Index of working register B
        WA : out  STD_LOGIC_VECTOR (4 downto 0); -- Index of register to be written to.

        MA : out  STD_LOGIC_VECTOR (15 downto 0); -- Memory address from instruction.
        IMM : out  STD_LOGIC_VECTOR (15 downto 0); -- Intermediate value form
instruction

        AL : out  STD_LOGIC_VECTOR (3 downto 0); -- Control code of the ALU
        SH : out  STD_LOGIC_VECTOR (3 downto 0); -- Amount ot shift by

        WEN : out  STD_LOGIC; -- Write enable for the registers
        OEN : out  STD_LOGIC --  Write enable for the memory
    );

end Decoder_Block;

architecture Behavioral of Decoder_Block is
    signal OPCODE_int : STD_LOGIC_VECTOR(5 downto 0);

    signal IMM_internal : STD_LOGIC_VECTOR(15 downto 0);

    signal InstrInternal1 : std_logic_vector(IMM_internal'range) := (others => '0');
    signal InstrInternal2 : std_logic_vector(IMM_internal'range) := (others => '0');
begin

    -- grab the opcode part of the instruction
    OPCODE_int <= instr(31 downto 26);

    -- map internal signal to output
    OPCODE <= OPCODE_int;

    -- RA and WA are always in the same place
    -- so they can be assigned with no logic
    RA <= instr(9 downto 5);
    WA <= instr(4 downto 0);

    -- RB moves so it needs
    RB <=
        instr(20 downto 16) when OPCODE_int(5 downto 4) = "00" else
```

```
        instr(4 downto 0);

    -- The IMM/Offset is the most variable of the data that is encoded into the
    -- Instructions. it also changes it size. SO it has the most complicated decode
    -- Logic
    InstrInternal1 <= std_logic_vector(resize(signed(instr(19 downto 10)), 16));
    InstrInternal2 <= std_logic_vector(resize(signed(instr(24 downto 16)), 16));

    IMM_internal <=
        InstrInternal1 when OPCODE_int(5 downto 4) = "10" and  OPCODE_int(1 downto 0) =
"11" else
        InstrInternal2 when OPCODE_int(5 downto 4) = "11" else
        instr(25 downto 10);

    -- Map internal signals to the ouputs
    IMM <= IMM_internal;
    MA <= IMM_internal;

    -- The instruction coding given to cannot be directly mapped to the
    AL <=
        "1010" when STAGE(1) = '1' else -- calc branch
        "1010" when STAGE(8) = '1' else -- calc branch
        "0100" when OPCODE_int(5 downto 3) = "010" and OPCODE_int(1 downto 0) = "01"
else -- A & B
        "0101" when OPCODE_int(5 downto 3) = "010" and OPCODE_int(1 downto 0) = "10"
else -- A || B
        "0110" when OPCODE_int(5 downto 3) = "010" and OPCODE_int(1 downto 0) = "11"
else -- A xor B
        "0111" when OPCODE_int(5 downto 3) = "010" and OPCODE_int(1 downto 0) = "00"
else -- not A
        "1000" when OPCODE_int(5 downto 4) = "00" and  OPCODE_int(3) = '1' and
OPCODE_int(4) = '0' else -- A + 1
        "1001" when OPCODE_int(5 downto 4) = "00" and  OPCODE_int(3) = '1' and
OPCODE_int(4) = '0' else -- A - 1
        "1010" when OPCODE_int(5 downto 4) = "00" and OPCODE_int(2) = '1' and
OPCODE_int(0) = '0' else -- A + B
        "1011" when OPCODE_int(5 downto 4) = "00" and OPCODE_int(2) = '1' and
OPCODE_int(0) = '1' else -- A - B
        "1100" when OPCODE_int(5 downto 3) = "011" and OPCODE_int(2 downto 0) = "000"
else -- A sla X
        "1101" when OPCODE_int(5 downto 3) = "011" and OPCODE_int(2 downto 0) = "001"
else -- A sra X
        "1110" when OPCODE_int(5 downto 3) = "011" and OPCODE_int(2 downto 0) = "010"
else -- A rotl X
        "1111" when OPCODE_int(5 downto 3) = "011" and OPCODE_int(2 downto 0) = "011"
else -- A rotr X
        "0000"; -- A

    -- Get the amount ot shift by.
    SH <= instr(19 downto 16);
    -- OEN is high for any opcode starting with 1001, i.e store instructions
    OEN <=
        '1' when OPCODE_int(5 downto 2) = "1001" else
        '0';

    -- Write enable is high for certian stages of exicution
    WEN <=
        '1' when STAGE(3) = '1' else
        '1' when STAGE(7) = '1' else
        '0';
```

```
end Behavioral;
```

CONTROL UNIT

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- ====
-- ControlUnit
-- The main control unit for the processor.
-- Contains the decode, instruction register, sequencer and FSM.
-- ====

entity ControlUnit is
    Port (

           clk : in std_logic; -- clock
           rst : in std_logic; --  sync active high reset
           en : in STD_LOGIC; -- active high enable

           PC_Plus : in  STD_LOGIC_VECTOR(7 downto 0); -- the PC plus offset
           instr : in  STD_LOGIC_VECTOR (31 downto 0); -- the current instruction
           Flags : in STD_LOGIC_VECTOR(7 downto 0); --  the flags from the ALU


           Instr_addr : out  STD_LOGIC_VECTOR (7 downto 0); -- The address of the next
instruction
           PC : out  STD_LOGIC_VECTOR(7 downto 0); -- The program counter

           S : out  STD_LOGIC_VECTOR (4 downto 1); -- Selects for the ALU multiplexers

           RA : out  STD_LOGIC_VECTOR (4 downto 0); -- Index of working register A
           RB : out  STD_LOGIC_VECTOR (4 downto 0); -- Index of working register B
           WA : out  STD_LOGIC_VECTOR (4 downto 0); -- Index of working register to be
written to (Rt)

           MA : out  STD_LOGIC_VECTOR (15 downto 0); -- Memory address from instruction
           IMM : out  STD_LOGIC_VECTOR (15 downto 0); -- Intermediate value from
instruction

           AL : out  STD_LOGIC_VECTOR (3 downto 0); -- The ALU control code
           SH : out  STD_LOGIC_VECTOR (3 downto 0); -- The amount of bits to shift by

           WEN : out  STD_LOGIC; -- Write enable for the registers
           OEN : out  STD_LOGIC --  Write enable for the memory
         );
end ControlUnit;

architecture Behavioral of ControlUnit is

    signal OPCODE : STD_LOGIC_VECTOR (5 downto 0);

    signal STAGE : STD_LOGIC_VECTOR (8 downto 0);
begin

    -- The control FSM
    Inst_ControlFSM: entity work.ControlFSM PORT MAP(
        clk => clk ,
```

```
        rst => rst,
        opcode => OPCODE ,
        S => S,
        STAGE => STAGE
    );

    -- The PC sequencer
    Inst_Sequencer: entity work.Sequencer PORT MAP(
        clk => clk ,
        rst => rst,
        en => en,
        STAGE => STAGE,
        instr => instr ,
        PC_plus => PC_Plus,
        flags => Flags,
        PC => PC,
        MIA => Instr_addr
    );

    -- The combonational decoder
    Inst_Decoder_Block: entity work.Decoder_Block PORT MAP(
        instr => instr,
        STAGE => STAGE,
        OPCODE => OPCODE,
        RA => RA,
        RB => RB,
        WA => WA,
        MA => MA,
        IMM => IMM,
        AL => AL,
        SH => SH,
        WEN => WEN,
        OEN => OEN
    );


end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity DataPath_C is
    GENERIC(
        data_size : natural := 16;
        num_registers : natural := 32
    );
    Port (
        clk : in  STD_LOGIC;
        rst : in STD_LOGIC;
        en : in STD_LOGIC;

        -- Inputs
        R_A : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Read address A
        R_B : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Read address B
        W_EN : in  STD_LOGIC;                                        -- Register write
enable
        W_A : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Write address

        IMM : in  STD_LOGIC_VECTOR (data_size-1 downto 0);           -- Itermediate
value
        M_A : in  STD_LOGIC_VECTOR (data_size-1 downto 0);           -- Memory address
        M_in : in  STD_LOGIC_VECTOR (data_size-1 downto 0);          -- Memory input
(read)
        PC : in  STD_LOGIC_VECTOR (15 downto 0);                     -- Current Program
Counter
        S : in  STD_LOGIC_VECTOR (4 downto 1);                       -- Selector control
        AL : in  STD_LOGIC_VECTOR (3 downto 0);                      -- ALU control
        SH : in  STD_LOGIC_VECTOR (log2(data_size)-1  downto 0);     -- Shift amount

        -- Outputs
        PC_plus : out  STD_LOGIC_VECTOR (15 downto 0);               -- Next Program
Counter
        Flags : out  STD_LOGIC_VECTOR (7 downto 0);                  -- ALU flags
        M_DA : out  STD_LOGIC_VECTOR (data_size-1 downto 0);         -- Memory address
        M_out : out  STD_LOGIC_VECTOR (data_size-1 downto 0)         -- Memory output
(write)
    );
end DataPath_C;

architecture Behavioral of DataPath_C is

    -- Data to write to the registers
    signal reg_in : STD_LOGIC_VECTOR (data_size-1 downto 0);

    -- Outputs of the registers
    signal A_data : STD_LOGIC_VECTOR (data_size-1 downto 0);
    signal B_data : STD_LOGIC_VECTOR (data_size-1 downto 0);

    -- Output of the register on the A and B buses
    signal A_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);
    signal B_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);
```

```
    -- The Outputs of the two Muxes on the input to the ALU
    signal A_mux : STD_LOGIC_VECTOR (data_size-1 downto 0);
    signal B_mux : STD_LOGIC_VECTOR (data_size-1 downto 0);

    -- The output of the combined ALU and shifter
    signal ALU_out : STD_LOGIC_VECTOR (data_size-1 downto 0);
    signal ALU_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);

    -- The output of the memory in register
    signal M_in_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);

begin

    -- The two multiplexers on the input to the ALU
    A_mux <= A_reg_out when S(2) = '0' else PC;
    B_mux <= B_reg_out when S(1) = '0' else IMM;

    -- The address multiplexer for the memory
    M_DA <= M_A when s(3) = '1' else ALU_reg_out;

    M_out <= B_reg_out;
    PC_plus <= ALU_reg_out;

    -- The register write multiplexer
    reg_in <= ALU_reg_out when s(4) = '0' else M_in_reg_out;

    -- The register on the A bus
    A_reg: entity work.Reg Generic Map (data_size => data_size)
    PORT MAP(
        clk => clk,
        rst => rst,
        en => en,
        data_in => A_data,
        data_out => A_reg_out
    );

    -- The register on the B bus
    B_reg: entity work.Reg Generic Map (data_size => data_size)
    PORT MAP(
        clk => clk,
        rst => rst,
        en => en,
        data_in => B_data,
        data_out => B_reg_out
    );

    -- The Register on the out put of the ALU
    ALU_reg: entity work.Reg Generic Map (data_size => data_size)
    PORT MAP(
        clk => clk,
        rst => rst,
        en => en,
        data_in => ALU_out,
        data_out => ALU_reg_out
    );

    -- The register on the memory read bus
    M_in_reg: entity work.Reg Generic Map (data_size => data_size)
    PORT MAP(
        clk => clk,
```

```
        rst => rst,
        en => en,
        data_in => M_in,
        data_out => M_in_reg_out
    );

    -- The ALU and shifter from Lab 1
    ALU: entity work.ALU_param GENERIC MAP( N => data_size )
    PORT MAP(
        A => A_mux,
        B => B_mux,
        X => SH,
        ctrl => AL,
        O => ALU_out,
        flags => flags
    );

    -- The register bank
    Registers: entity work.regbank
    PORT MAP(
        RSELA => R_A,
        RSELB => R_B,
        WSEL => W_A,
        D => reg_in,
        WEN => W_EN,
        clk => clk,
        A => A_data,
        B => B_data,
        rst => '0'
    );

end Behavioral;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- A paramateriable synchronous reset D-type registers with enable
entity Reg is
    Generic(
        data_size: natural := 8 -- How many bits will the register deal with
    );
    Port (
        clk : in  STD_LOGIC;
        rst : in STD_LOGIC; -- synchronus reset
        en : in STD_LOGIC; -- synchronus reset
        data_in : in  STD_LOGIC_VECTOR (data_size-1 downto 0); -- input
        data_out : out  STD_LOGIC_VECTOR (data_size-1 downto 0) -- output
    );
end Reg;

architecture Behavioral of Reg is
begin

    process(clk)
    begin
        -- Synchronise to the clock
        if (rising_edge(clk)) then
            if (rst = '1') then
                -- Reset to zero
                data_out <= (others => '0');
            elsif (en = '1') then
                -- Pass the input to the output
                data_out <= data_in;
            end if;
        end if;

    end process;


end Behavioral;
```

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    18:20:58 02/23/2009
-- Design Name:
-- Module Name:    regbank - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity regbank is
    Port ( A : out  STD_LOGIC_VECTOR (15 downto 0);
           B : out  STD_LOGIC_VECTOR (15 downto 0);
           RSELA : in  STD_LOGIC_VECTOR (4 downto 0);
           RSELB : in  STD_LOGIC_VECTOR (4 downto 0);
           D : in  STD_LOGIC_VECTOR (15 downto 0);
           WSEL : in  STD_LOGIC_VECTOR (4 downto 0);
              WEN: in STD_LOGIC;
              CLK, RST: in STD_LOGIC);
end regbank;

architecture Behavioral of regbank is

--type rbank is array (0 to 31) of std_logic_vector (15 downto 0);
signal REG00 : std_logic_vector (15 downto 0);
signal REG01 : std_logic_vector (15 downto 0);
signal REG02 : std_logic_vector (15 downto 0);
signal REG03 : std_logic_vector (15 downto 0);
signal REG04 : std_logic_vector (15 downto 0);
signal REG05 : std_logic_vector (15 downto 0);
signal REG06 : std_logic_vector (15 downto 0);
signal REG07 : std_logic_vector (15 downto 0);
signal REG08 : std_logic_vector (15 downto 0);
signal REG09 : std_logic_vector (15 downto 0);
signal REG10 : std_logic_vector (15 downto 0);
signal REG11 : std_logic_vector (15 downto 0);
signal REG12 : std_logic_vector (15 downto 0);
signal REG13 : std_logic_vector (15 downto 0);
signal REG14 : std_logic_vector (15 downto 0);
```

```
signal REG15 : std_logic_vector (15 downto 0);
signal REG16 : std_logic_vector (15 downto 0);
signal REG17 : std_logic_vector (15 downto 0);
signal REG18 : std_logic_vector (15 downto 0);
signal REG19 : std_logic_vector (15 downto 0);
signal REG20 : std_logic_vector (15 downto 0);
signal REG21 : std_logic_vector (15 downto 0);
signal REG22 : std_logic_vector (15 downto 0);
signal REG23 : std_logic_vector (15 downto 0);
signal REG24 : std_logic_vector (15 downto 0);
signal REG25 : std_logic_vector (15 downto 0);
signal REG26 : std_logic_vector (15 downto 0);
signal REG27 : std_logic_vector (15 downto 0);
signal REG28 : std_logic_vector (15 downto 0);
signal REG29 : std_logic_vector (15 downto 0);
signal REG30 : std_logic_vector (15 downto 0);
signal REG31 : std_logic_vector (15 downto 0);

begin

    A <=    REG00 when RSELA = "00000" else
            REG01 when RSELA = "00001" else
            REG02 when RSELA = "00010" else
            REG03 when RSELA = "00011" else
            REG04 when RSELA = "00100" else
            REG05 when RSELA = "00101" else
            REG06 when RSELA = "00110" else
            REG07 when RSELA = "00111" else
            REG08 when RSELA = "01000" else
            REG09 when RSELA = "01001" else
            REG10 when RSELA = "01010" else
            REG11 when RSELA = "01011" else
            REG12 when RSELA = "01100" else
            REG13 when RSELA = "01101" else
            REG14 when RSELA = "01110" else
            REG15 when RSELA = "01111" else
            REG16 when RSELA = "10000" else
            REG17 when RSELA = "10001" else
            REG18 when RSELA = "10010" else
            REG19 when RSELA = "10011" else
            REG20 when RSELA = "10100" else
            REG21 when RSELA = "10101" else
            REG22 when RSELA = "10110" else
            REG23 when RSELA = "10111" else
            REG24 when RSELA = "11000" else
            REG25 when RSELA = "11001" else
            REG26 when RSELA = "11010" else
            REG27 when RSELA = "11011" else
            REG28 when RSELA = "11100" else
            REG29 when RSELA = "11101" else
            REG30 when RSELA = "11110" else
            REG31 when RSELA = "11111" else
            "XXXXXXXXXXXXXXXX";

    B <=    REG00 when RSELB = "00000" else
            REG01 when RSELB = "00001" else
            REG02 when RSELB = "00010" else
            REG03 when RSELB = "00011" else
            REG04 when RSELB = "00100" else
            REG05 when RSELB = "00101" else
            REG06 when RSELB = "00110" else
```

```
            REG07 when RSELB = "00111" else
            REG08 when RSELB = "01000" else
            REG09 when RSELB = "01001" else
            REG10 when RSELB = "01010" else
            REG11 when RSELB = "01011" else
            REG12 when RSELB = "01100" else
            REG13 when RSELB = "01101" else
            REG14 when RSELB = "01110" else
            REG15 when RSELB = "01111" else
            REG16 when RSELB = "10000" else
            REG17 when RSELB = "10001" else
            REG18 when RSELB = "10010" else
            REG19 when RSELB = "10011" else
            REG20 when RSELB = "10100" else
            REG21 when RSELB = "10101" else
            REG22 when RSELB = "10110" else
            REG23 when RSELB = "10111" else
            REG24 when RSELB = "11000" else
            REG25 when RSELB = "11001" else
            REG26 when RSELB = "11010" else
            REG27 when RSELB = "11011" else
            REG28 when RSELB = "11100" else
            REG29 when RSELB = "11101" else
            REG30 when RSELB = "11110" else
            REG31 when RSELB = "11111" else
            "XXXXXXXXXXXXXXXX";

   REG00 <= "0000000000000000";

process (CLK)
begin
    if (CLK'event and CLK = '1') then
        if RST = '1' then
           REG01 <= "0000000000000000";
           REG02 <= "0000000000000000";
           REG03 <= "0000000000000000";
           REG04 <= "0000000000000000";
           REG05 <= "0000000000000000";
           REG06 <= "0000000000000000";
           REG07 <= "0000000000000000";
           REG08 <= "0000000000000000";
           REG09 <= "0000000000000000";
           REG10 <= "0000000000000000";
           REG11 <= "0000000000000000";
           REG12 <= "0000000000000000";
           REG13 <= "0000000000000000";
           REG14 <= "0000000000000000";
           REG15 <= "0000000000000000";
           REG16 <= "0000000000000000";
           REG17 <= "0000000000000000";
           REG18 <= "0000000000000000";
           REG19 <= "0000000000000000";
           REG20 <= "0000000000000000";
           REG21 <= "0000000000000000";
           REG22 <= "0000000000000000";
           REG23 <= "0000000000000000";
           REG24 <= "0000000000000000";
           REG25 <= "0000000000000000";
           REG26 <= "0000000000000000";
           REG27 <= "0000000000000000";
           REG28 <= "0000000000000000";
```

```
            REG29 <= "0000000000000000";
            REG30 <= "0000000000000000";
            REG31 <= "0000000000000000";
        else
            if (WEN = '1') then
                case WSEL is
--                  when "00000" => REG00 <= D;
                    when "00001" => REG01 <= D;
                    when "00010" => REG02 <= D;
                    when "00011" => REG03 <= D;
                    when "00100" => REG04 <= D;
                    when "00101" => REG05 <= D;
                    when "00110" => REG06 <= D;
                    when "00111" => REG07 <= D;
                    when "01000" => REG08 <= D;
                    when "01001" => REG09 <= D;
                    when "01010" => REG10 <= D;
                    when "01011" => REG11 <= D;
                    when "01100" => REG12 <= D;
                    when "01101" => REG13 <= D;
                    when "01110" => REG14 <= D;
                    when "01111" => REG15 <= D;
                    when "10000" => REG16 <= D;
                    when "10001" => REG17 <= D;
                    when "10010" => REG18 <= D;
                    when "10011" => REG19 <= D;
                    when "10100" => REG20 <= D;
                    when "10101" => REG21 <= D;
                    when "10110" => REG22 <= D;
                    when "10111" => REG23 <= D;
                    when "11000" => REG24 <= D;
                    when "11001" => REG25 <= D;
                    when "11010" => REG26 <= D;
                    when "11011" => REG27 <= D;
                    when "11100" => REG28 <= D;
                    when "11101" => REG29 <= D;
                    when "11110" => REG30 <= D;
                    when "11111" => REG31 <= D;
                    when others => REG31 <= REG31;
                end case;
            end if;
        end if;
    end if;
end process;

end Behavioral;
```

REGISTER.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity register is
    Port ( clk : in  STD_LOGIC;
           data_in : in  STD_LOGIC_VECTOR (3 downto 0);
           data_out : out  STD_LOGIC_VECTOR (3 downto 0));
end register;

architecture Behavioral of register is

begin


end Behavioral;
```

```
----------------------------------------------------------------------------------
--
-- Uni           :   University of York
-- Course        :   Electronic Engineering
-- Module        :   Computer Architectures
-- Engineers     :   Y3839090 & Y3840426
--
-- Create Date   :   13:19:20 02/17/2017
-- Design Name   :   ALU_param - Behavioral
-- Description   :   A paramateriable integer ALU.
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity ALU_param is
    Generic (
        N : natural := 8 -- data size in bits
    );
    Port (
        A : in  STD_LOGIC_VECTOR (N-1 downto 0);
        B : in  STD_LOGIC_VECTOR (N-1 downto 0);
        X : in  STD_LOGIC_VECTOR (log2(N)-1 downto 0); -- shift/rotate amount input
        ctrl : in  STD_LOGIC_VECTOR (3 downto 0); -- control signals from opcode
        O : out  STD_LOGIC_VECTOR (N-1 downto 0);
        flags :out  STD_LOGIC_VECTOR (7 downto 0) -- flags
    );
end ALU_param;

architecture Behavioral of ALU_param is

    -- internal signed signal for A and B inpy=uts
    signal A_itrn : SIGNED (N-1 downto 0);
    signal B_itrn : SIGNED (N-1 downto 0);

    -- internal integer for X
    signal X_itrn : integer;

    -- internal signed signal for output
    signal O_itrn : SIGNED (N-1 downto 0);


    -- max positive and negitive N bit signed numbers
    constant max_pos : SIGNED (N-1 downto 0) := to_signed(( 2 ** (N-1) ) - 1, N);
    constant max_neg : SIGNED (N-1 downto 0) := to_signed( -2 ** (N-1) , N);

begin

        A_itrn <= signed(A); -- converts A to signed and maps the result to A_itrn
        B_itrn <= signed(B); -- converts A to signed and maps the result to B_itrn

        -- converts X to integer and maps the result to X_itrn
        X_itrn <= to_integer(unsigned(X));
```

```
        -- converts O_itrn to a plain std_logic_vector and maps it to O
        O <= std_logic_vector(O_itrn);

        -- Main ALU multiplexer for each possible command
        O_itrn <=
            A_itrn                  when ctrl = "0000" else     -- Output A
            A_itrn and B_itrn       when ctrl = "0100" else     -- Output A & B
            A_itrn or B_itrn        when ctrl = "0101" else     -- Output A || B
            A_itrn xor B_itrn       when ctrl = "0110" else     -- Output A xor B
            not A_itrn              when ctrl = "0111" else     -- Output not A
            A_itrn + 1              when ctrl = "1000" else     -- Output A + 1
            A_itrn - 1              when ctrl = "1001" else     -- Output A - 1
            A_itrn + B_itrn         when ctrl = "1010" else     -- Output A + B
            A_itrn - B_itrn         when ctrl = "1011" else     -- Output A - B
            SHIFT_LEFT (A_itrn , X_itrn)    when ctrl = "1100" else -- Output A sla X
            SHIFT_RIGHT (A_itrn , X_itrn)   when ctrl = "1101" else -- Output A sra X
            ROTATE_LEFT (A_itrn , X_itrn)   when ctrl = "1110" else -- Output A rotl X
            ROTATE_RIGHT (A_itrn , X_itrn)  when ctrl = "1111" else -- Output A rotr X
            (others =>'U');

    -- Overflow flag
    flags(7) <=
        -- Will overflow if you add one to the max positive value
        '1' when ctrl = "1000" and A_itrn = max_pos else

        -- Will overflow if you minus one to the max negitive value
        '1' when ctrl = "1001" and A_itrn = max_neg else

        -- Will overflow if two neg values added give a pos result
        '1' when ctrl = "1010" and A_itrn(N-1) = '1' and  B_itrn(N-1) = '1' and
O_itrn(N-1) = '0' else
        -- Will overflow if two pos values added give a neg result
        '1' when ctrl = "1010" and A_itrn(N-1) = '0' and  B_itrn(N-1) = '0' and
O_itrn(N-1) = '1' else

        -- Will overflow if a pos value is subtracted from a neg value gives a pos
result
        '1' when ctrl = "1011" and A_itrn(N-1) = '1' and  B_itrn(N-1) = '0' and
O_itrn(N-1) = '0' else
        -- Will overflow if a neg value is subtracted from a pos value gives a neg
result
        '1' when ctrl = "1011" and A_itrn(N-1) = '0' and  B_itrn(N-1) = '1' and
O_itrn(N-1) = '1'

        -- If none of the above are true then the result hasn't overflown
        else '0';

    -- Other flags
    flags(6) <= '1'     when O_itrn >= 0    else '0'; -- grater than or equal to zero
    flags(5) <= '1'     when O_itrn <= 0    else '0'; -- less than or equal to zero
    flags(4) <= '1'     when O_itrn >  0    else '0'; -- grater than zero
    flags(3) <= '1'     when O_itrn <  0    else '0'; -- less than zero
    flags(2) <= '1'     when O_itrn =  1    else '0'; -- one flag
    flags(1) <= '1'     when O_itrn /= 0    else '0'; -- not zero flag
    flags(0) <= '1'     when O_itrn =  0    else '0'; -- zero flag


end Behavioral;
```

```
--------------------------------------------------
-- PACKAGE FOR DIGITAL ENGINEERING LABS
--
-- To use:
--
-- - Download file
-- - Use "Add copy" to add to Xilinx project
-- - Add "use work.DigEng.all" on top of entity
--
--------------------------------------------------

package DigEng is

function log2 (x : natural ) return natural;
function size (x : natural ) return natural;

end DigEng;

package body DigEng is

--------------------------------------------------
-- LOG BASE 2 FUNCTION
-- returns the ceiling of log base 2 of a (non-zero) integer
-- (1->0; 2->1; 3->2; 4->2; 5->3 ...)
--
-- This function is NOT SYNTHESIZABLE
-- should be used for indices, not circuit description
--
-- Examples:
-- - signal A : STD_LOGIC_VECTOR(log2(data_size)-1 downto 0);
--
--------------------------------------------------
function log2 ( x : natural ) return natural is
        variable temp : natural := x ;
        variable n : natural := 0 ;
    begin
        while temp > 1 loop
            temp := temp / 2 ;
            n := n + 1 ;
        end loop ;
       if (x > 2**n) then
        n := n + 1;
       end if;
        return n ;
end function log2;

--------------------------------------------------
-- SIZE FUNCTION
-- returns the size of a vector that can encode a (non-zero) integer
-- (1->1; 2->2; 3->2; 4->3; 5->3 ...)
--
-- This function is NOT SYNTHESIZABLE
-- should be used for indices, not circuit description
--
-- Examples:
-- - signal A : STD_LOGIC_VECTOR(size(n)-1 downto 0);
--
```

```
-------------------------------------------------
function size ( x : natural ) return natural is
        variable temp : natural := x ;
        variable n : natural := 0 ;
    begin
        while temp >= 1 loop
            temp := temp / 2 ;
            n := n + 1 ;
        end loop ;
        return n ;
end function size;

end DigEng;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
USE ieee.numeric_std.ALL;

package easyprint is

function s_tostr(val : std_logic_vector) return string;
function u_tostr(val : std_logic_vector) return string;

function to_bstring(sl : std_logic) return string;
function to_bstring(slv : std_logic_vector) return string;


end easyprint;

package body easyprint is

    -- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====
    -- From http://stackoverflow.com/a/24336034 By Morten Zilmer
    -- Allows printing a std_logic_vector as a string that represents it's binary form.
    -- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====
    function to_bstring(sl : std_logic) return string is
        variable sl_str_v : string(1 to 3);  -- std_logic image with quotes around
    begin
        sl_str_v := std_logic'image(sl);
        return "" & sl_str_v(2);  -- "" & character to get string
    end function;

    function to_bstring(slv : std_logic_vector) return string is
        alias    slv_norm : std_logic_vector(1 to slv'length) is slv;
        variable sl_str_v : string(1 to 1);  -- String of std_logic
        variable res_v    : string(1 to slv'length);
    begin
        for idx in slv_norm'range loop
            sl_str_v := to_bstring(slv_norm(idx));
            res_v(idx) := sl_str_v(1);
        end loop;
        return res_v;
    end function;
    -- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====


    -- converts an std_logic_vector to a string that represents it's signed value
    function s_tostr(val : std_logic_vector) return string is
    begin
        return integer'image( to_integer(signed(val)) );
    end function;

    -- converts an std logic vector to a string that represents it's unsigned value
    function u_tostr(val : std_logic_vector) return string is
    begin
        return integer'image( to_integer(unsigned(val)) );
    end function;
```

```
end easyprint;
```

PROCESSORTB.VHD

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.easyprint.ALL;

ENTITY ProcessorTB IS
END ProcessorTB;

ARCHITECTURE behavior OF ProcessorTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Processor
    PORT(
         clk : IN  std_logic;
         en : IN  std_logic;
         rst : IN  std_logic;
         start : IN  std_logic;
         data_out : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal en : std_logic := '0';
   signal rst : std_logic := '0';
   signal start : std_logic := '0';

    --Outputs
   signal data_out : std_logic_vector(15 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: Processor PORT MAP (
      clk => clk,
      en => en,
      rst => rst,
      start => start,
      data_out => data_out
    );

  -- Clock process definitions
  clk_process :process
  begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
  end process;
```

```
  -- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100 ns.
    wait for 100 ns;


    -- do an inital reset
    rst <= '1';

    wait until rising_edge(clk);

    rst <= '0';

    en <= '1';

    -- wait a while whilst the cpu is in a loop before
    --  taking start high
    wait for 50*clk_period;

    start <= '1';

    wait until rising_edge(clk);
    wait for 15*clk_period;
    start <= '0';

    wait;
  end process;

END;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.easyprint.ALL;

ENTITY OutputRegTB IS
END OutputRegTB;

ARCHITECTURE behavior OF OutputRegTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT OutputReg
    PORT(
         clk : IN  std_logic;
         data_in : IN  std_logic_vector(15 downto 0);
         WE : IN  std_logic;
         data_out : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal data_in : std_logic_vector(15 downto 0) := (others => '0');
   signal WE : std_logic := '0';
   signal data_out : std_logic_vector(15 downto 0) := (others => '0');

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: OutputReg PORT MAP (
          clk => clk,
          data_in => data_in,
          WE => WE,
          data_out => data_out
        );

   -- Clock process definitions
   clk_process :process
   begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
```

```
      wait for clk_period*10;

      -- insert stimulus here
      WE <= '1';
      data_in <= "1100110011001100";
      wait for clk_period;

      assert std_match(data_out, "1100110011001100")
      report lf &
      "DATA IN/DATA OUT BROKED"
      severity error;

      WE <= '0';
      data_in <= "0011100000111111";
      wait for clk_period;

      assert std_match(data_out, "1100110011001100")
      report lf &
      "WE BROKED"
      severity error;

      wait;
   end process;

END;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.easyprint.ALL;


ENTITY DualPortMemoryTB IS
END DualPortMemoryTB;

ARCHITECTURE behavior OF DualPortMemoryTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT DualPortMemory
    PORT(
         clk : IN  std_logic;
         I_addr : IN  std_logic_vector(6 downto 0);
         D_addr : IN  std_logic_vector(6 downto 0);
         D_in : IN  std_logic_vector(31 downto 0);
         WE : IN  std_logic;
         D_out : OUT  std_logic_vector(31 downto 0);
         I_out : OUT  std_logic_vector(31 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal I_addr : std_logic_vector(6 downto 0) := (others => '0');
   signal D_addr : std_logic_vector(6 downto 0) := (others => '0');
   signal D_in : std_logic_vector(31 downto 0) := (others => '0');
   signal WE : std_logic := '0';

    --Outputs
   signal D_out : std_logic_vector(31 downto 0);
   signal I_out : std_logic_vector(31 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: DualPortMemory PORT MAP (
          clk => clk,
          I_addr => I_addr,
          D_addr => D_addr,
          D_in => D_in,
          WE => WE,
          D_out => D_out,
          I_out => I_out
        );

   -- Clock process definitions
   clk_process :process
   begin
        clk <= '0';
```

```
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;


    -- Stimulus process
    stim_proc: process
    begin
        -- hold reset state for 100 ns.
        wait for 100 ns;

        wait for clk_period*10;

        -- insert stimulus here

        I_addr <= "0000100";

        wait for clk_period;

        assert std_match(I_out, X"00000100")
        report lf &"BROKED"
        severity error;


        wait for clk_period;

        D_addr <= "0000010";
        WE  <= '1';
        D_in <= "00011000110011001100110101011110";

        wait for clk_period;

        assert std_match(D_out, "00011000110011001100110101011110")
        report lf & "BROKED"
        severity error;

        wait;
    end process;

END;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.easyprint.ALL;

ENTITY MMU_TB IS
END MMU_TB;

ARCHITECTURE behavior OF MMU_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT MMU
    PORT(
         clk : IN  std_logic;
         I_addr : IN  std_logic_vector(7 downto 0);
         OEn : IN  std_logic;
         D_in : IN  std_logic_vector(15 downto 0);
         D_out : OUT  std_logic_vector(15 downto 0);
         D_addr : IN  std_logic_vector(15 downto 0);
         start : IN  std_logic;
         oWE : OUT  std_logic;
         oD_out : OUT  std_logic_vector(15 downto 0);
         mWE : OUT  std_logic;
         mD_addr : OUT  std_logic_vector(6 downto 0);
         mD_in : IN  std_logic_vector(31 downto 0);
         mD_out : OUT  std_logic_vector(31 downto 0);
         mI_addr : OUT  std_logic_vector(6 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal I_addr : std_logic_vector(7 downto 0) := (others => '0');
   signal OEn : std_logic := '0';
   signal D_in : std_logic_vector(15 downto 0) := (others => '0');
   signal D_addr : std_logic_vector(15 downto 0) := (others => '0');
   signal start : std_logic := '0';
   signal mD_in : std_logic_vector(31 downto 0) := (others => '0');

    --Outputs
   signal D_out : std_logic_vector(15 downto 0);
   signal oWE : std_logic;
   signal oD_out : std_logic_vector(15 downto 0);
   signal mWE : std_logic;
   signal mD_addr : std_logic_vector(6 downto 0);
   signal mD_out : std_logic_vector(31 downto 0);
   signal mI_addr : std_logic_vector(6 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

    -- Test Data
    type TEST_RECORD is record

        --Inputs
```

```
        I_addr : std_logic_vector(7 downto 0);
        OEn : std_logic;
        D_in : std_logic_vector(15 downto 0);
        D_addr : std_logic_vector(15 downto 0);
        start : std_logic;
        mD_in : std_logic_vector(31 downto 0);

        --Outputs
        D_out : std_logic_vector(15 downto 0);
        oWE : std_logic;
        oD_out : std_logic_vector(15 downto 0);
        mWE : std_logic;
        mD_addr : std_logic_vector(6 downto 0);
        mD_out : std_logic_vector(31 downto 0);
        mI_addr : std_logic_vector(6 downto 0);

    end record TEST_RECORD;

    type TEST_RECORD_ARRAY is array (NATURAL range <>) of TEST_RECORD;

    constant test_data : TEST_RECORD_ARRAY := (
    --   I_addr,     Oen,  D_in,               D_addr,   Start, mD_in,       D_out,
oWE, oD_out,            mWE, mD_addr,   mD_out,                          mI_addr
        ("01010101", '0' , "----------------",  X"007e", '0',   X"00FF00FF", X"00FF",
'0', "----------------", '0', "1111111",  "--------------------------------",
"1010101"),
        ("11010101", '1' , "----------------",  X"007e", '0',   X"00FF00FF", "----------
------", '0', "----------------", '1', "1111111",  X"00FF00EE",
"1010101"),

    --   I_addr,     Oen,   D_in,               D_addr,  Start, mD_in,
D_out,                          oWE, oD_out,  mWE, mD_addr,    mD_out,
mI_addr
        ("11011111", '1',   X"00CC",            X"01F8", '0',   "----------------------
---------", "----------------", '1', X"00CC", '0', "-------", "-------------------------
-------", "1011111"),
    --   I_addr,     Oen, D_in,              D_addr,  Start,   mD_in,
D_out,   oWE, oD_out,            mWE, mD_addr,   mD_out,
mI_addr
        ("01110111", '0', "----------------", X"01F0", '1',     "----------------------
---------", X"FFFF", '0', "----------------", '0', "-------", "-------------------------
-------", "1110111"),
        ("01110111", '0', "----------------", X"01F0", '0',     "----------------------
---------", X"0000", '0', "----------------", '0', "-------", "-------------------------
-------", "1110111")
        );


BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: MMU PORT MAP (
          clk => clk,
          I_addr => I_addr,
          OEn => OEn,
          D_in => D_in,
          D_out => D_out,
          D_addr => D_addr,
          start => start,
          oWE => oWE,
          oD_out => oD_out,
```

```
          mWE => mWE,
          mD_addr => mD_addr,
          mD_in => mD_in,
          mD_out => mD_out,
          mI_addr => mI_addr
        );

   -- Clock process definitions
   clk_process :process
   begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

      wait until rising_edge(clk);

      for i in test_data'range loop

        -- assign test inputs
        I_addr <= test_data(i).I_addr;
        OEn <= test_data(i).OEn;
        D_in <= test_data(i).D_in;
        D_addr <= test_data(i).D_addr;
        start <= test_data(i).start;
        mD_in <= test_data(i).mD_in;

        wait until falling_edge(clk);

        -- Check to see if the out put was what we were expecting

        assert std_match(D_out, test_data(i).D_out)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual D_out did not equal expected D_out."&
        " Actual [ " & to_bstring(D_out) & " ]" &
        " Expected [ " & to_bstring(test_data(i).D_out) & " ]" & lf
        severity error;

        assert std_match(oWE, test_data(i).oWE)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual oWE did not equal expected oWE."&
        " Actual [ " & to_bstring(oWE) & " ]" &
        " Expected [ " & to_bstring(test_data(i).oWE) & " ]" & lf
        severity error;

        assert std_match(mWE, test_data(i).mWE)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual mWE did not equal expected mWE."&
        " Actual [ " & to_bstring(mWE) & " ]" &
        " Expected [ " & to_bstring(test_data(i).mWE) & " ]" & lf
        severity error;

        assert std_match(mD_addr, test_data(i).mD_addr)
```

```
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual mD_addr did not equal expected mD_addr."&
        " Actual [ " & to_bstring(mD_addr) & " ]" &
        " Expected [ " & to_bstring(test_data(i).mD_addr) & " ]" & lf
        severity error;

        assert std_match(mD_out, test_data(i).mD_out)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual mD_out did not equal expected mD_out."&
        " Actual [ " & to_bstring(mD_out) & " ]" &
        " Expected [ " & to_bstring(test_data(i).mD_out) & " ]" & lf
        severity error;

        assert std_match(mI_addr, test_data(i).mI_addr)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual mI_addr did not equal expected mI_addr."&
        " Actual [ " & to_bstring(mI_addr) & " ]" &
        " Expected [ " & to_bstring(test_data(i).mI_addr) & " ]" & lf
        severity error;

        -- if there were no isses report that the test was successful
        assert not (
            std_match(D_out, test_data(i).D_out) and
            std_match(oWE, test_data(i).oWE) and
            std_match(oD_out, test_data(i).oD_out) and
            std_match(mWE, test_data(i).mWE) and
            std_match(mD_addr, test_data(i).mD_addr) and
            std_match(mD_out, test_data(i).mD_out) and
            std_match(mI_addr, test_data(i).mI_addr)
        )
        report lf & " [ OK ] Test " & integer'image(i)& " was successful!" & lf
        severity note;
        wait until rising_edge(clk);

        end loop;
      wait;
   end process;

END;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.DigEng.ALL;
USE work.easyprint.ALL;
USE ieee.numeric_std.ALL;


-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY ControlFSM_TB IS
END ControlFSM_TB;

ARCHITECTURE behavior OF ControlFSM_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT ControlFSM
    PORT(
        clk : IN  std_logic; --  clk
        rst : IN  std_logic; -- sync active high reset
        opcode : IN  std_logic_vector(5 downto 0); -- opcode from instruction
        S : OUT  std_logic_vector(3 downto 0) --  Selectors for muxes
    );
    END COMPONENT;


    --Inputs
    signal clk : std_logic := '0';
    signal rst : std_logic := '0';
    signal opcode : std_logic_vector(5 downto 0) := (others => '0');

    --Outputs
    signal S : std_logic_vector(3 downto 0);

    -- Clock period definitions
    constant clk_period : time := 10 ns;

    -- Test Data
    type TEST_RECORD is record
        opcode  : std_logic_vector(5 downto 0);
        S : std_logic_vector(4 downto 1);
    end record TEST_RECORD;

    type TEST_RECORD_ARRAY is array (NATURAL range <>) of TEST_RECORD;

    constant test_data : TEST_RECORD_ARRAY := (
        -- Register
        ("000100", "--1-"), --add rt, ra, rb
        ("000100", "----"),
        ("000100", "--00"),
        ("000100", "0---"),

        ("010000", "--1-"), -- and rt, ra, rb
        ("010000", "----"),
        ("010000", "--00"),
```

```
        ("010000", "0---"),
        -- Memory
        ("100111", "--1-"), -- stor
        ("100111", "----"),
        ("100111", "--01"),
        ("100111", "-0--"),

        ("100101", "--1-"), -- load
        ("100101", "----"),
        ("100101", "--01"),
        ("100101", "-0--"),
        ("100101", "1---")
        -- Branch
    );

    BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: ControlFSM PORT MAP (
        clk => clk,
        rst => rst,
        opcode => opcode,
        S => S
    );

    -- Clock process definitions
    clk_process :process
        begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;


    -- Stimulus process
    stim_proc: process
        begin
        -- hold reset state for 100 ns.
        wait for 100 ns;

        rst <= '1';
        wait for 2*clk_period;
        rst <= '0';

        for i in test_data'range loop


        -- assign test inputs
        opcode <= test_data(i).opcode;

        wait until falling_edge(clk);

        -- Check to see if the out put was what we were expecting
        assert std_match(S, test_data(i).S)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual STATE did not equal expected STATE."&
        " Actual [ " & to_bstring(S) & " ]" &
        " Expected [ " & to_bstring(test_data(i).S) & " ]" & lf
        severity error;
```

```
        -- if there were no isses report that the test was successful
        assert not (std_match(S, test_data(i).S))
        report lf & " [ OK ] Test " & integer'image(i)& " was successful!" & lf
        severity note;
        wait until rising_edge(clk);

        end loop;

        -- End of test
        wait;
    end process;

END;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.easyprint.ALL;

ENTITY SequencerTB IS
END SequencerTB;

ARCHITECTURE behavior OF SequencerTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Sequencer
    PORT(
         clk : IN  std_logic;
         rst : IN  std_logic;
         en : IN std_logic;
         FETCH : in STD_LOGIC;
         PC_plus : IN  std_logic_vector(7 downto 0);
         instr : IN  std_logic_vector(31 downto 0);
         flags : IN  std_logic_vector(7 downto 0);
         PC : OUT  std_logic_vector(7 downto 0);
         MIA : OUT  std_logic_vector(7 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal rst : std_logic := '1';
   signal en : std_logic := '0';
   signal FETCH : std_logic := '0';
   signal PC_plus : std_logic_vector(7 downto 0) := (others => '0');
   signal instr : std_logic_vector(31 downto 0) := (others => '0');
   signal flags : std_logic_vector(7 downto 0) := (others => '0');

    --Outputs
   signal PC : std_logic_vector(7 downto 0);
   signal MIA : std_logic_vector(7 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

   -- Test Data
    type TEST_RECORD is record

        instr : std_logic_vector(31 downto 0);
        PC_plus  : std_logic_vector(7 downto 0);
        flags : std_logic_vector(7 downto 0);

        PC : std_logic_vector(7 downto 0);
        MIA : std_logic_vector(7 downto 0);

    end record;

    type TEST_RECORD_ARRAY is array (NATURAL range <>) of TEST_RECORD;
```

```
    constant test_data : TEST_RECORD_ARRAY := (
    -- Instruction  , PCplus        , Flags     , PC    , MIA
        (X"01010101", "--------", "--------", X"00", X"00"),
        (X"200000A5", "--------", "--------", X"01", X"01"),
        (X"100C00A8", "--------", "--------", X"02", X"02"),
        (X"C40A00C0", X"0C"     , "------1-", X"03", X"03"),
        (X"00000000", "--------", "--------", X"0C", X"0C")
    );

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: Sequencer PORT MAP (
        clk => clk,
        rst => rst,
        en => en,
        FETCH => FETCH,
        PC_plus => PC_plus,
        instr => instr,
        flags => flags,
        PC => PC,
        MIA => MIA
    );

   -- Clock process definitions
   clk_process :process
   begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      wait for 100 ns;
      rst <= '1';
      en <= '1';
      wait until rising_edge(clk);

        for i in test_data'range loop

            rst <= '0';

            -- assign test inputs
            instr <= test_data(i).instr;
            PC_plus <= test_data(i).PC_plus;
            flags <= test_data(i).flags;
            FETCH <= '1';

            wait for clk_period;
            FETCH <= '0';
            wait for 5*clk_period;

            -- Check to see if the out put was what we were expecting
            assert std_match(PC, test_data(i).PC)
            report lf & " [ERR!] Test " & integer'image(i)&
            " Actual PC did not equal expected PC."&
            " Actual [ " & to_bstring(PC) & " ]" &
```

```
            " Expected [ " & to_bstring(test_data(i).PC) & " ]" & lf
            severity error;

            assert std_match(MIA, test_data(i).MIA)
            report lf & " [ERR!] Test " & integer'image(i)&
            " Actual MIA did not equal expected MIA."&
            " Actual [ " & to_bstring(MIA) & " ]" &
            " Expected [ " & to_bstring(test_data(i).MIA) & " ]" & lf
            severity error;

            -- if there were no isses report that the test was successful
            assert not (std_match(PC, test_data(i).PC) and std_match(MIA,
test_data(i).MIA))
            report lf & " [ OK ] Test " & integer'image(i)& " was successful!" & lf
            severity note;


        end loop;

        -- End of test
      wait;
   end process;

END;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.easyprint.ALL;

ENTITY Decoder_TB IS
END Decoder_TB;

ARCHITECTURE behavior OF Decoder_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Decoder_Block
    PORT(
         instr : IN  std_logic_vector(31 downto 0);
         OPCODE : OUT  std_logic_vector(5 downto 0);
         RA : OUT  std_logic_vector(4 downto 0);
         RB : OUT  std_logic_vector(4 downto 0);
         WA : OUT  std_logic_vector(4 downto 0);
         MA : OUT  std_logic_vector(15 downto 0);
         IMM : OUT  std_logic_vector(15 downto 0);
         AL : OUT  std_logic_vector(3 downto 0);
         SH : OUT  std_logic_vector(3 downto 0);
         WEN : OUT  std_logic;
         OEN : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal instr : std_logic_vector(31 downto 0) := (others => '0');

    --Outputs
   signal OPCODE : std_logic_vector(5 downto 0);
   signal RA : std_logic_vector(4 downto 0);
   signal RB : std_logic_vector(4 downto 0);
   signal WA : std_logic_vector(4 downto 0);
   signal MA : std_logic_vector(15 downto 0);
   signal IMM : std_logic_vector(15 downto 0);
   signal AL : std_logic_vector(3 downto 0);
   signal SH : std_logic_vector(3 downto 0);
   signal WEN : std_logic;
   signal OEN : std_logic;

   -- Test Data
    type TEST_RECORD is record
        --Inputs
         instr : std_logic_vector(31 downto 0);

         --Outputs
         OPCODE : std_logic_vector(5 downto 0);
         RA : std_logic_vector(4 downto 0);
         RB : std_logic_vector(4 downto 0);
         WA : std_logic_vector(4 downto 0);
         MA : std_logic_vector(15 downto 0);
         IMM : std_logic_vector(15 downto 0);
         AL : std_logic_vector(3 downto 0);
```

```
         SH : std_logic_vector(3 downto 0);
         WEN : std_logic;
         OEN : std_logic;

    end record;

    type TEST_RECORD_ARRAY is array (NATURAL range <>) of TEST_RECORD;

    constant test_data : TEST_RECORD_ARRAY := (
--  INSTRUCTION,                                OPCODE,     RA,          RB,          WA,
MA,                   IMM,                  ALU,    SH,     WEN, OEN
        ("011000------0010------1001110011",    "011000",   "10011",    "-----",
"10011", "----------------",    "----------------", "1100", "0010", '1', '0'),
        ("00011000110011001100110101011110",    "000110",   "01010",    "-----",
"11110", "----------------",    "0011001100110011", "1010", "----", '1', '0'),
        ("000100-----00011------0001000010",    "000100",   "00010",    "00011",
"00010", "----------------",    "----------------", "1010", "----", '1', '0'),
        ("10000100000001111100000000001111",    "100001",   "-----",    "-----",
"01111",            X"01F0",    "----------------", "----", "----", '1', '0')
    );

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: Decoder_Block PORT MAP (
          instr => instr,
          OPCODE => OPCODE,
          RA => RA,
          RB => RB,
          WA => WA,
          MA => MA,
          IMM => IMM,
          AL => AL,
          SH => SH,
          WEN => WEN,
          OEN => OEN
        );

   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

      for i in test_data'range loop


            -- assign test inputs
            instr <= test_data(i).instr;

            wait for 100 ns;

            -- Check to see if the out put was what we were expecting
            assert std_match(OPCODE, test_data(i).OPCODE)
            report lf & " [ERR!] Test " & integer'image(i)&
            " Actual OPCODE did not equal expected OPCODE."&
            " Actual [ " & to_bstring(OPCODE) & " ]" &
            " Expected [ " & to_bstring(test_data(i).OPCODE) & " ]" & lf
            severity error;

            assert std_match(RA, test_data(i).RA)
```

```
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual RA did not equal expected RA."&
        " Actual [ " & to_bstring(RA) & " ]" &
        " Expected [ " & to_bstring(test_data(i).RA) & " ]" & lf
        severity error;

        assert std_match(RB, test_data(i).RB)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual RB did not equal expected RB."&
        " Actual [ " & to_bstring(RB) & " ]" &
        " Expected [ " & to_bstring(test_data(i).RB) & " ]" & lf
        severity error;

        assert std_match(WA, test_data(i).WA)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual PC did not equal expected RB."&
        " Actual [ " & to_bstring(WA) & " ]" &
        " Expected [ " & to_bstring(test_data(i).WA) & " ]" & lf
        severity error;

        assert std_match(MA, test_data(i).MA)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual MA did not equal expected MA."&
        " Actual [ " & to_bstring(MA) & " ]" &
        " Expected [ " & to_bstring(test_data(i).MA) & " ]" & lf
        severity error;

        assert std_match(IMM, test_data(i).IMM)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual IMM did not equal expected IMM."&
        " Actual [ " & to_bstring(IMM) & " ]" &
        " Expected [ " & to_bstring(test_data(i).IMM) & " ]" & lf
        severity error;

        assert std_match(AL, test_data(i).AL)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual AL did not equal expected AL."&
        " Actual [ " & to_bstring(AL) & " ]" &
        " Expected [ " & to_bstring(test_data(i).AL) & " ]" & lf
        severity error;

        assert std_match(SH, test_data(i).SH)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual SH did not equal expected SH."&
        " Actual [ " & to_bstring(SH) & " ]" &
        " Expected [ " & to_bstring(test_data(i).SH) & " ]" & lf
        severity error;

        assert std_match(WEN, test_data(i).WEN)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual WEN did not equal expected WEN."&
        " Actual [ " & to_bstring(WEN) & " ]" &
        " Expected [ " & to_bstring(test_data(i).WEN) & " ]" & lf
        severity error;

        assert std_match(OEN, test_data(i).OEN)
        report lf & " [ERR!] Test " & integer'image(i)&
        " Actual OEN did not equal expected OEN."&
        " Actual [ " & to_bstring(WEN) & " ]" &
        " Expected [ " & to_bstring(test_data(i).OEN) & " ]" & lf
        severity error;
```

```
            -- if there were no isses report that the test was successful
            assert not (
                std_match(OPCODE, test_data(i).OPCODE) and
                std_match(RA, test_data(i).RA) and
                std_match(RB, test_data(i).RB) and
                std_match(WA, test_data(i).WA) and
                std_match(MA, test_data(i).MA) and
                std_match(IMM, test_data(i).IMM) and
                std_match(AL, test_data(i).AL) and
                std_match(SH, test_data(i).SH) and
                std_match(OEN, test_data(i).OEN) and
                std_match(WEN, test_data(i).WEN)
            )
            report lf & " [ OK ] Test " & integer'image(i)& " was successful!" & lf
            severity note;


        end loop;

        -- End of test

      -- insert stimulus here

      wait;
   end process;

END;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.DigEng.ALL;
USE work.easyprint.ALL;
USE ieee.numeric_std.ALL;

ENTITY DataPath_C_TB IS
END DataPath_C_TB;

ARCHITECTURE behavior OF DataPath_C_TB IS

    -- Constants
    constant data_size : NATURAL := 16;
    constant num_registers : NATURAL := 32;

    -- Clock period definitions
    constant clk_period : time := 10 ns;

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT DataPath_C
        GENERIC(
            data_size : natural;
            num_registers : natural
        );
        PORT(
            clk     : IN    std_logic;
            rst     : IN    std_logic;
            en   : IN    std_logic;

            R_A     : IN    std_logic_vector    (log2(num_registers)-1 downto 0);
            R_B     : IN    std_logic_vector    (log2(num_registers)-1 downto 0);
            W_EN    : IN    std_logic;
            W_A     : IN    std_logic_vector    (log2(num_registers)-1 downto 0);
            IMM     : IN    std_logic_vector    (data_size-1 downto 0);
            M_A     : IN    std_logic_vector    (data_size-1 downto 0);
            M_in    : IN    std_logic_vector    (data_size-1 downto 0);
            PC      : IN    std_logic_vector    (15 downto 0);
            S       : IN    std_logic_vector    (4 downto 1);
            AL      : IN    std_logic_vector    (3 downto 0);
            SH      : IN    std_logic_vector    (log2(data_size)-1 downto 0);

            PC_plus : OUT   std_logic_vector    (15 downto 0);
            Flags   : OUT   std_logic_vector    (7 downto 0);
            M_DA    : OUT   std_logic_vector    (data_size-1 downto 0);
            M_out   : OUT   std_logic_vector    (data_size-1 downto 0)
        );
    END COMPONENT;


    --Inputs
    signal clk  : std_logic                                             := '0';
    signal rst  : std_logic                                             := '1';
    signal en   : std_logic                                             := '0';

    signal R_A  : std_logic_vector  (log2(num_registers)-1 downto 0)    := (others =>
'0');
```

```
    signal R_B  : std_logic_vector  (log2(num_registers)-1 downto 0)     := (others =>
'0');
    signal W_EN : std_logic                                              := '0';
    signal W_A  : std_logic_vector  (log2(num_registers)-1 downto 0 )    := (others =>
'0');
    signal IMM  : std_logic_vector  (data_size-1 downto 0)               := (others =>
'0');
    signal M_A  : std_logic_vector  (data_size-1 downto 0)               := (others =>
'0');
    signal M_in : std_logic_vector  (data_size-1 downto 0)               := (others =>
'0');
    signal PC   : std_logic_vector  (15 downto 0)                        := (others =>
'0');
    signal S    : std_logic_vector  (4 downto 1)                         := (others =>
'0');
    signal AL   : std_logic_vector  (3 downto 0)                         := (others =>
'0');
    signal SH   : std_logic_vector  (log2(data_size)-1 downto 0)         := (others =>
'0');

    --Outputs
    signal PC_plus  : std_logic_vector  (15 downto 0);
    signal Flags    : std_logic_vector  (7 downto 0);
    signal M_DA     : std_logic_vector  (data_size-1 downto 0);
    signal M_out    : std_logic_vector  (data_size-1 downto 0);
    signal OEN : std_logic := '0';

    -- Test data definitions
    type TEST_VECTOR is RECORD
         R_A    : STD_LOGIC_VECTOR(log2(num_registers)-1 downto 0);
         R_B    : std_logic_vector(log2(num_registers)-1 downto 0);

         W_EN   : std_logic;
         W_A    : std_logic_vector(log2(num_registers)-1 downto 0);

         IMM    : std_logic_vector(data_size-1 downto 0);
         M_A    : std_logic_vector(data_size-1 downto 0);
         M_in   : std_logic_vector(data_size-1 downto 0);
         PC     : std_logic_vector(15 downto 0);
         S      : std_logic_vector(4 downto 1);
         AL     : std_logic_vector(3 downto 0);
         SH     : std_logic_vector(log2(data_size)-1 downto 0) ;

         PC_plus: std_logic_vector(15 downto 0);
         flags  : std_logic_vector(7 downto 0);
         M_DA   : std_logic_vector(data_size-1 downto 0);
         M_out  : std_logic_vector(data_size-1 downto 0);

         OEN : std_logic;
    end RECORD;
    type TEST_VECTOR_ARRAY is ARRAY(NATURAL RANGE <>) of TEST_VECTOR;

    -- Test Data
    constant test_vectors : TEST_VECTOR_ARRAY := (
        --R_A,      R_B,        W_EN,   W_A,        IMM,                M_A,
M_in,               PC,         S,      AL,     SH,         PC_plus,            flags,
M_DA,               M_out

        -- inc R1, R0
```

```
        ( "-----",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"00FF",    "--1-", "1000", "----", "----------------",
"01010010", "----------------", "----------------", '0' ),
        ( "00000",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"00FF",    "----", "----", "----", X"0100",            "-------
-", "----------------", "----------------", '0' ),
        ( "-----",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"00FF",    "--0-", "1000", "----", "----------------",
"01010110", "----------------", "----------------", '0' ),
        ( "-----",  "-----",    '1',    "00001",    "----------------", "----------------
-", "----------------", X"00FF",    "0---", "----", "----", "----------------", "-------
-", "----------------", "----------------", '0' ),

        -- addi R2, R0, 0x0005
        ( "-----",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0100",    "--1-", "1000", "----", "----------------",
"01010010", "----------------", "----------------", '0' ),
        ( "00000",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0100",    "----", "----", "----", X"0101",            "-------
-", "----------------", "----------------", '0' ),
        ( "-----",  "-----",    '0',    "-----",    X"0005",            "----------------
-", "----------------", X"0100",    "--01", "1010", "----", "----------------",
"01010010", "----------------", "----------------", '0' ),
        ( "-----",  "-----",    '1',    "00010",    "----------------", "----------------
-", "----------------", X"0100",    "0---", "----", "----", "----------------", "-------
-", "----------------", "----------------", '0' ),

        -- shl R3, R1, 3
        ( "-----",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0101",    "--1-", "1000", "----", "----------------",
"01010010", "----------------", "----------------", '0' ),
        ( "00001",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0101",    "----", "----", "----", X"0102",            "-------
-", "----------------", "----------------", '0' ),
        ( "-----",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0101",    "--0-", "1100", "0011", "----------------",
"01010010", "----------------", "----------------", '0' ),
        ( "-----",  "-----",    '1',    "00011",    "----------------", "----------------
-", "----------------", X"0101",    "0---", "----", "----", "----------------", "-------
-", "----------------", "----------------", '0' ),

        -- storr R2, R3
        ( "-----",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0102",    "--1-", "1000", "----", "----------------",
"01010010", "----------------", "----------------", '0' ),
        ( "00011",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0102",    "----", "----", "----", X"0103",            "-------
-", "----------------", "----------------", '0' ),
        ( "-----",  "00010",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0102",    "--00", "0000", "----", "----------------",
"01010010", "----------------", "----------------", '0' ),
        ( "-----",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0102",    "-0--", "----", "----", "----------------", "-------
-", X"0008",            X"0005"           , '1' ),

        -- loadi R5, 1f1f
        ( "-----",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0103",    "--1-", "1000", "----", "----------------",
"01010010", "----------------", "----------------", '0' ),
```

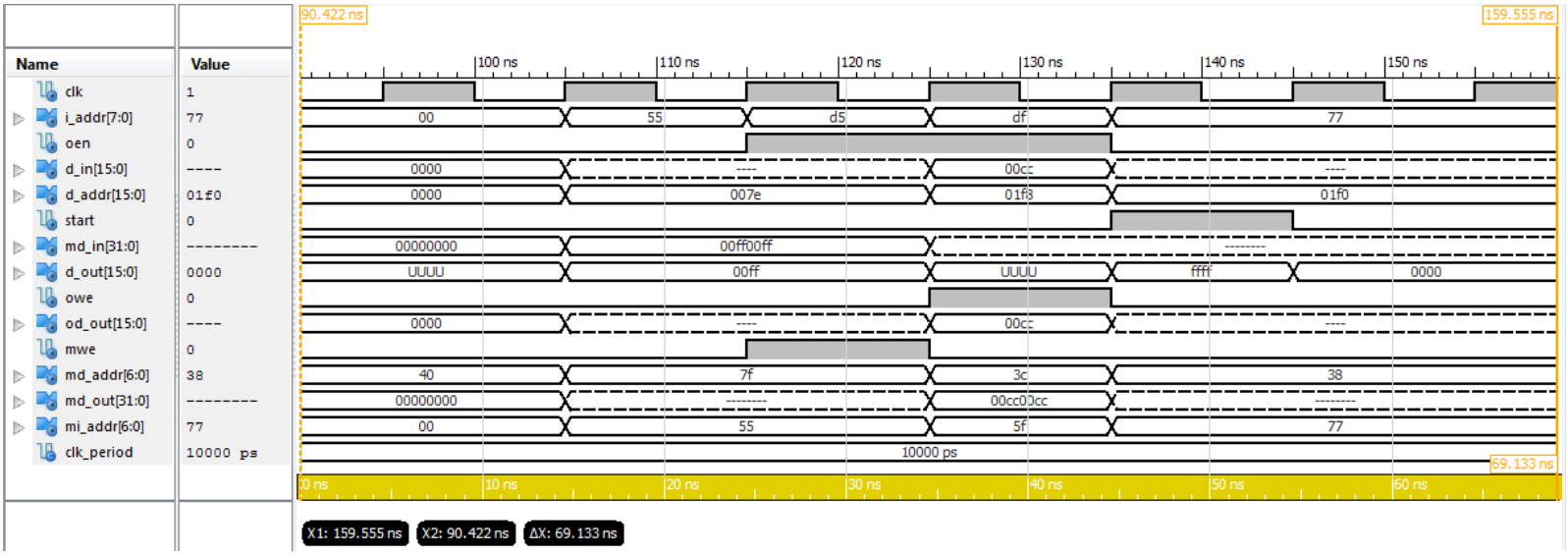
```
        ( "-----",  "-----",    '0',    "-----",    "----------------", X"1F1F",
X"CCCC",            X"0103",    "-1--", "----", "----", X"0104",           "--------",
X"1f1f",            "----------------", '0' ),
        ( "-----",  "-----",    '1',    "00101",    "----------------", "----------------
-", "----------------", X"0103",    "1---", "----", "----", "----------------", "-------
-", "----------------", "----------------", '0' ),

        -- brneq R2, 010F
        ( "-----",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0104",    "--1-", "1000", "----", "----------------",
"01010010", "----------------", "----------------", '0' ),
        ( "00010",  "00000",    '0',    "-----",    X"010f",           "----------------
-", "----------------", X"0104",    "--11", "1010", "----", X"0105",
"01010010", "----------------", "----------------", '0' ),
        ( "-----",  "-----",    '0',    "-----",    "----------------", "----------------
-", "----------------", X"0104",    "--00", "1010", "----", X"0213",           "------
10", "----------------", "----------------", '0' )

    );

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: DataPath_C
    Generic Map(
        data_size => data_size,
        num_registers => num_registers
    )
    PORT MAP (
        clk => clk,
        rst => rst,
        en => en,

        -- Inputs
        R_A => R_A,
        R_B => R_B,

        W_EN => W_EN,
        W_A => W_A,

        IMM => IMM,
        M_A => M_A,
        M_in => M_in,
        PC => PC,
        S => S,
        AL => AL,
        SH => SH,

        -- Outputs
        PC_plus => PC_plus,
        Flags => Flags,
        M_DA => M_DA,
        M_out => M_out
    );

   -- Clock process definitions
   clk_process :process
   begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
```

```
        wait for clk_period/2;
    end process;


    -- Stimulus process
    stim_proc: process
    begin

        -- hold reset state for 100 ns.
        wait for 100 ns;

        -- enable the system
        rst <= '0';
        en <= '1';
        wait for 2*clk_period;

        -- run the test for every set of data
        for i in test_vectors'range loop

            wait until rising_edge(clk);

            -- assign test inputs
            R_A <= test_vectors(i).R_A;
            R_B <= test_vectors(i).R_B;

            W_EN <= test_vectors(i).W_EN;
            W_A <= test_vectors(i).W_A;

            IMM <= test_vectors(i).IMM;
            M_A <= test_vectors(i).M_A;
            M_in <= test_vectors(i).M_in;
            PC <= test_vectors(i).PC;
            S <= test_vectors(i).S;
            AL <= test_vectors(i).AL;
            SH <= test_vectors(i).SH;

            OEN <= test_vectors(i).OEN;

            wait until falling_edge(clk);

            -- Check to see if the out put was what we were expecting
            -- Have to use std_match() instead of = when comparing meta values like '-'
            assert std_match(test_vectors(i).flags, flags)
            report lf & " [ERR!] Test " & integer'image(i)&
                " Actual flags did not equal expected flags."&
                " Actual [ " & to_bstring(flags) & " ]" &
                " Expected [ " & to_bstring(test_vectors(i).flags) & " ]" & lf
            severity error;

            assert std_match(test_vectors(i).M_out, M_out)
            report lf & " [ERR!] Test " & integer'image(i)&
                " Actual value to memory did not equal expected value to memory."&
                " Actual [ " & u_tostr(M_out) & " ]" &
                " Expected [ " & u_tostr(test_vectors(i).M_out) & " ]" & lf
            severity error;

            assert std_match(M_DA , test_vectors(i).M_DA)
            report lf & " [ERR!] Test " & integer'image(i)&
                " Actual memory address did not equal expected memory address."&
                " Actual [ " & u_tostr(M_DA) & " ]" &
                " Expected [ " & u_tostr(test_vectors(i).M_DA) & " ]" & lf
```

```
            severity error;

            assert std_match(PC_plus, test_vectors(i).PC_plus)
            report lf & " [ERR!] Test " & integer'image(i)&
                " Actual program counter did not equal expected program counter."&
                " Actual [ " & u_tostr(PC_plus) & " ]" &
                " Expected [ " & u_tostr(test_vectors(i).PC_plus) & " ]" & lf
            severity error;

            -- if there were no isses report that the test was successful
            assert not (
                std_match(flags, test_vectors(i).flags) and
                std_match(M_out, test_vectors(i).M_out) and
                std_match(M_DA, test_vectors(i).M_DA) and
                std_match(PC_plus, test_vectors(i).PC_plus)
            )
            report lf & " [ OK ] Test " & integer'image(i)& " was successful!" & lf
            severity note;

        end loop;

        -- End of test
        wait;
    end process;


END;
```

Component Simulations

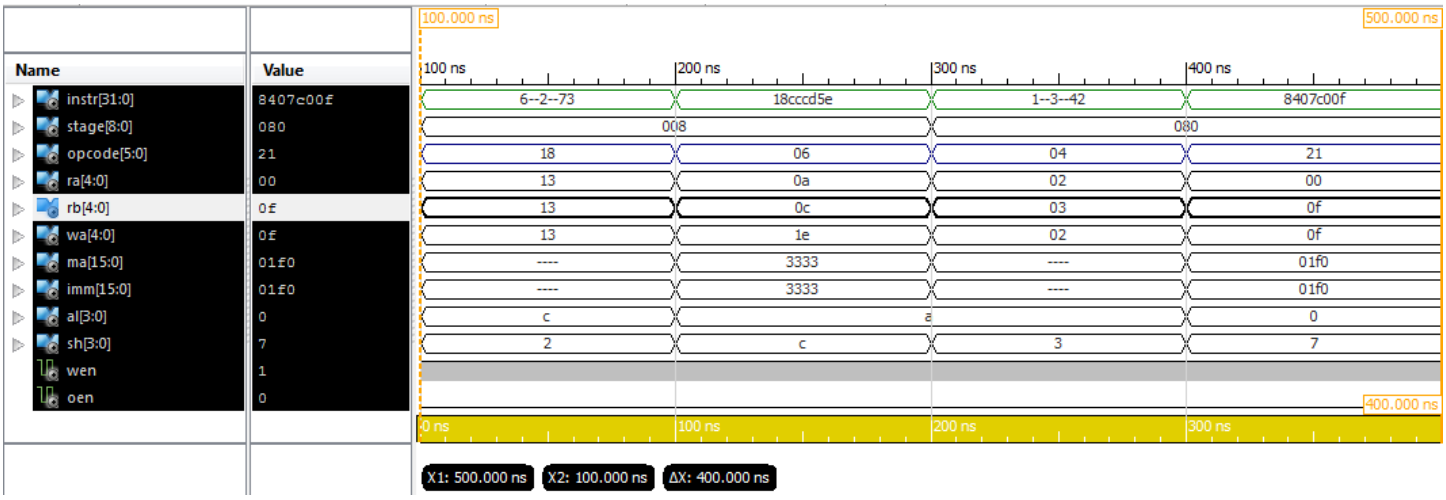
MMU TESTBENCH SIMULATION



ISIM CONSOLE LOG (MMU TB)

```
ISim P.28xd (signature 0xa0883be4)
This is a Full version of ISim.
Time resolution is 1 ps
WARNING: Simulation object /mmu_tb/test_data was not traceable in the design for the
following reason:
ISim does not yet support tracing of constant and generic multi-dimensional arrays.
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 110 ns(1): Note:
 [ OK ] Test 0 was successful!
 (/mmu_tb/).
at 120 ns(1): Note:
 [ OK ] Test 1 was successful!
 (/mmu_tb/).
at 130 ns(1): Note:
 [ OK ] Test 2 was successful!
 (/mmu_tb/).
at 140 ns(1): Note:
 [ OK ] Test 3 was successful!
 (/mmu_tb/).
at 150 ns(1): Note:
 [ OK ] Test 4 was successful!
 (/mmu_tb/).
ISim>
```

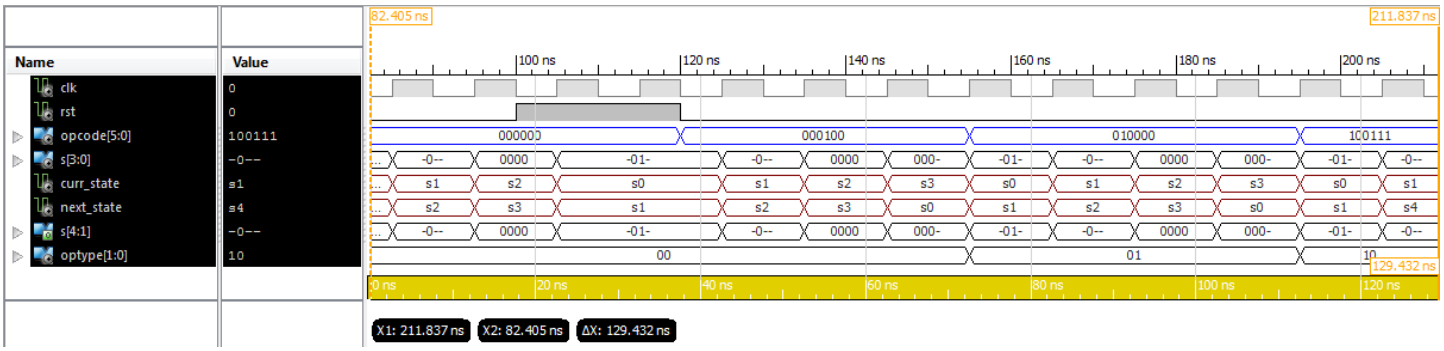
DECODER TESTBENCH SIMULATION



ISIM CONSOLE LOG (DECODER TB)

```
ISim P.28xd (signature 0xa0883be4)
This is a Full version of ISim.

# run 1000 ns
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 200 ns: Note:
 [ OK ] Test 0 was successful!
 (/decoder_tb/).
at 300 ns: Note:
 [ OK ] Test 1 was successful!
 (/decoder_tb/).
at 400 ns: Note:
 [ OK ] Test 2 was successful!
 (/decoder_tb/).
at 500 ns: Note:
 [ OK ] Test 3 was successful!
 (/decoder_tb/).
ISim>
```

CONTROL-FSM TESTBENCH SIMULATION

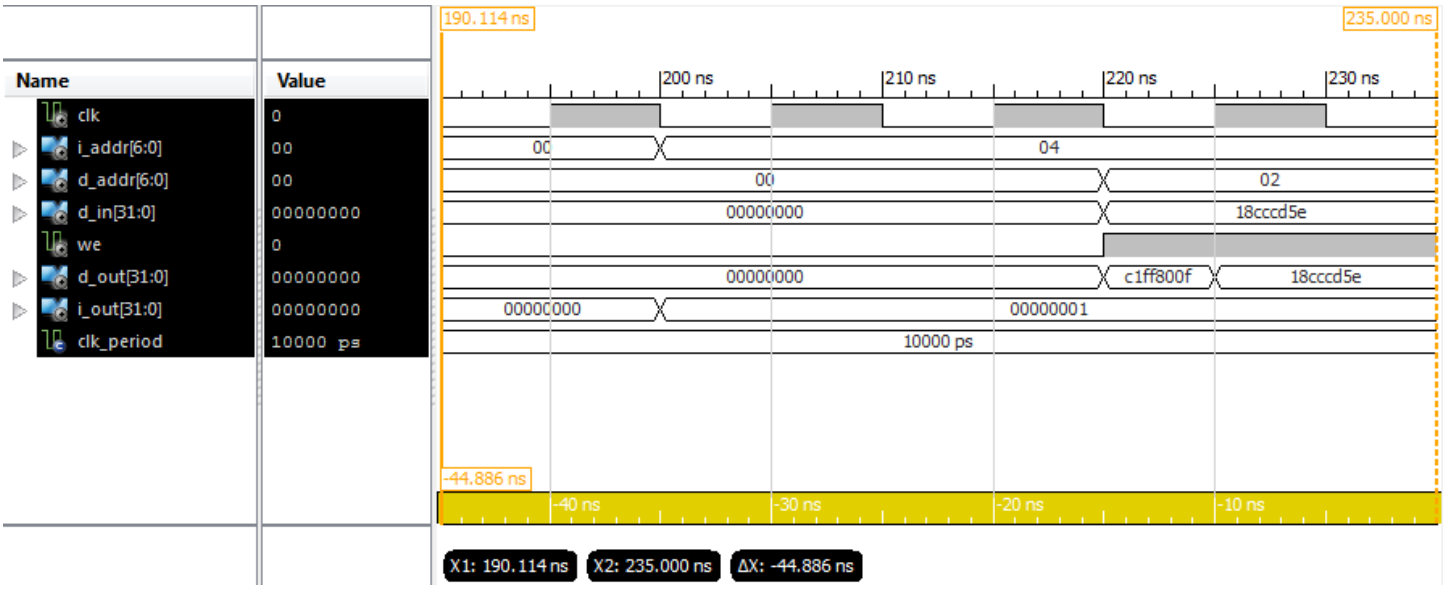


ISIM CONSOLE LOG (CONTROL FSM TB)

```
ISim P.28xd (signature 0xa0883be4)
This is a Full version of ISim.
Time resolution is 1 ps
WARNING: Simulation object /controlfsm_tb/test_data was not traceable in the design for
the following reason:
ISim does not yet support tracing of constant and generic multi-dimensional arrays.
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 120 ns(1): Note:
 [ OK ] Test 0 was successful!
 (/controlfsm_tb/).
at 130 ns(1): Note:
 [ OK ] Test 1 was successful!
 (/controlfsm_tb/).
at 140 ns(1): Note:
 [ OK ] Test 2 was successful!
 (/controlfsm_tb/).
```

```
at 150 ns(1): Note:
 [ OK ] Test 3 was successful!
 (/controlfsm_tb/).
at 160 ns(1): Note:
 [ OK ] Test 4 was successful!
 (/controlfsm_tb/).
at 170 ns(1): Note:
 [ OK ] Test 5 was successful!
 (/controlfsm_tb/).
at 180 ns(1): Note:
 [ OK ] Test 6 was successful!
 (/controlfsm_tb/).
at 190 ns(1): Note:
 [ OK ] Test 7 was successful!
 (/controlfsm_tb/).
at 200 ns(1): Note:
 [ OK ] Test 8 was successful!
 (/controlfsm_tb/).
at 210 ns(1): Note:
 [ OK ] Test 9 was successful!
 (/controlfsm_tb/).
at 220 ns(1): Note:
 [ OK ] Test 10 was successful!
 (/controlfsm_tb/).
at 230 ns(1): Note:
 [ OK ] Test 11 was successful!
 (/controlfsm_tb/).
at 240 ns(1): Note:
 [ OK ] Test 12 was successful!
 (/controlfsm_tb/).
at 250 ns(1): Note:
 [ OK ] Test 13 was successful!
 (/controlfsm_tb/).
at 260 ns(1): Note:
 [ OK ] Test 14 was successful!
 (/controlfsm_tb/).
at 270 ns(1): Note:
 [ OK ] Test 15 was successful!
 (/controlfsm_tb/).
at 280 ns(1): Note:
 [ OK ] Test 16 was successful!
 (/controlfsm_tb/).
ISim>
```

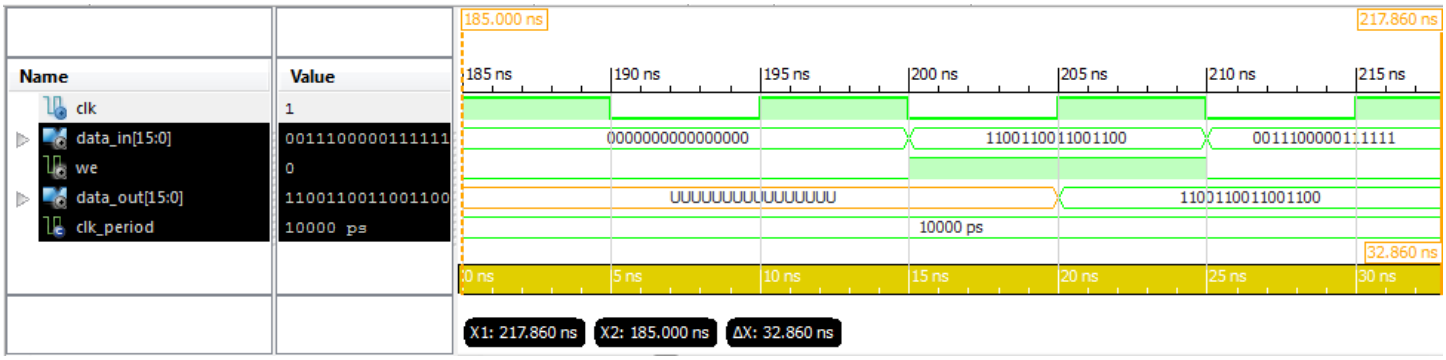
DUAL PORT MEMORY TESTBENCH SIMULATION



ISIM CONSOLE LOG (DUAL PORT MEMORY TB)

```
ISim P.28xd (signature 0xa0883be4)
This is a Full version of ISim.

# run 1000 ns
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 210 ns: Note:
Works (/dualportmemorytb/).
at 230 ns: Note:
Works (/dualportmemorytb/).
ISim>
```

OUTPUT REG TESTBENCH SIMULATION

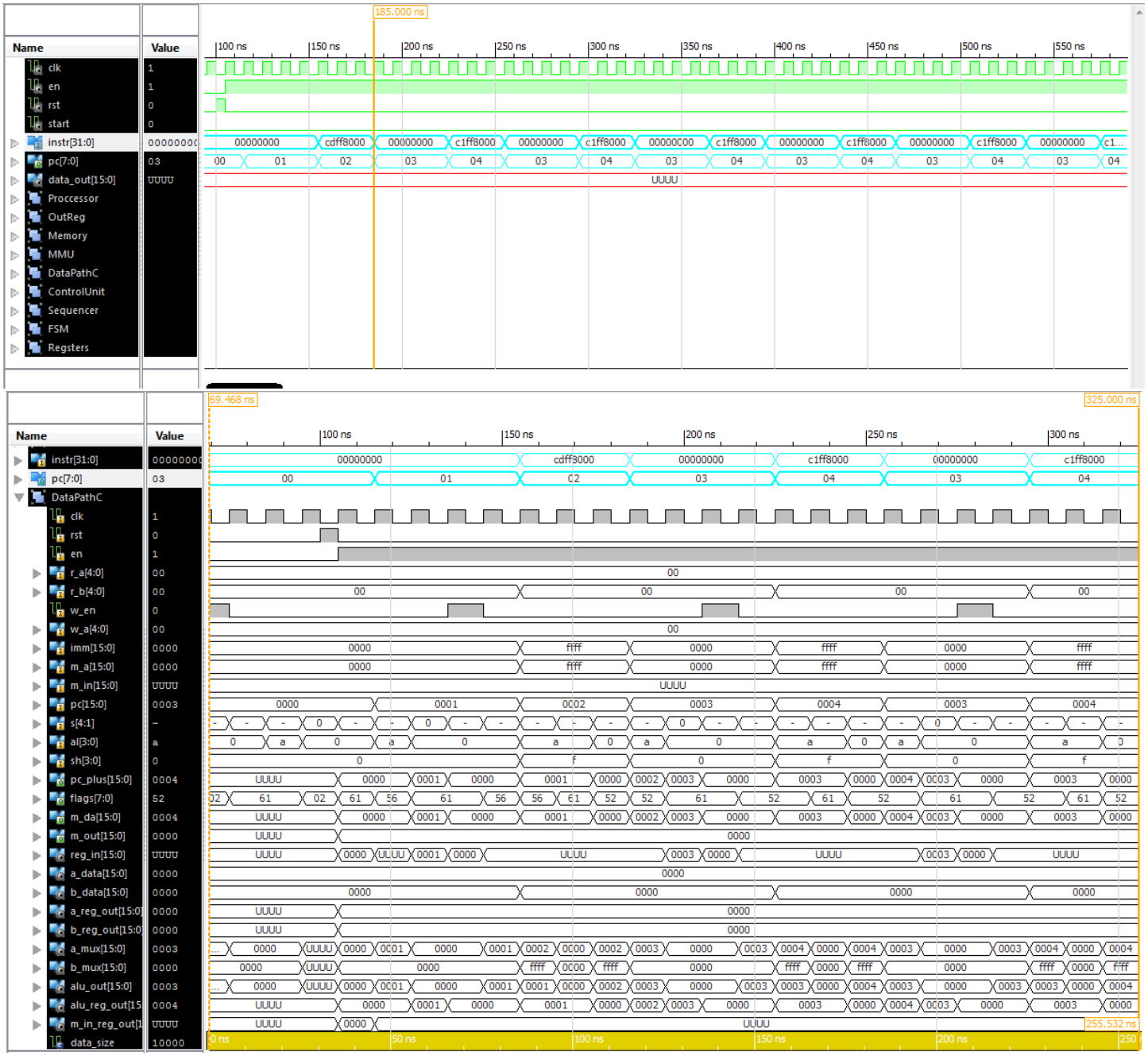


ISIM CONSOLE LOG (OUTPUT REG TB)

```
ISim P.28xd (signature 0xa0883be4)
This is a Full version of ISim.

# run 1000 ns
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 210 ns: Note:
DATA IN/DATA OUT WORKS (/outputregtb/).
at 220 ns: Note:
WE Works (/outputregtb/).
ISim>
```

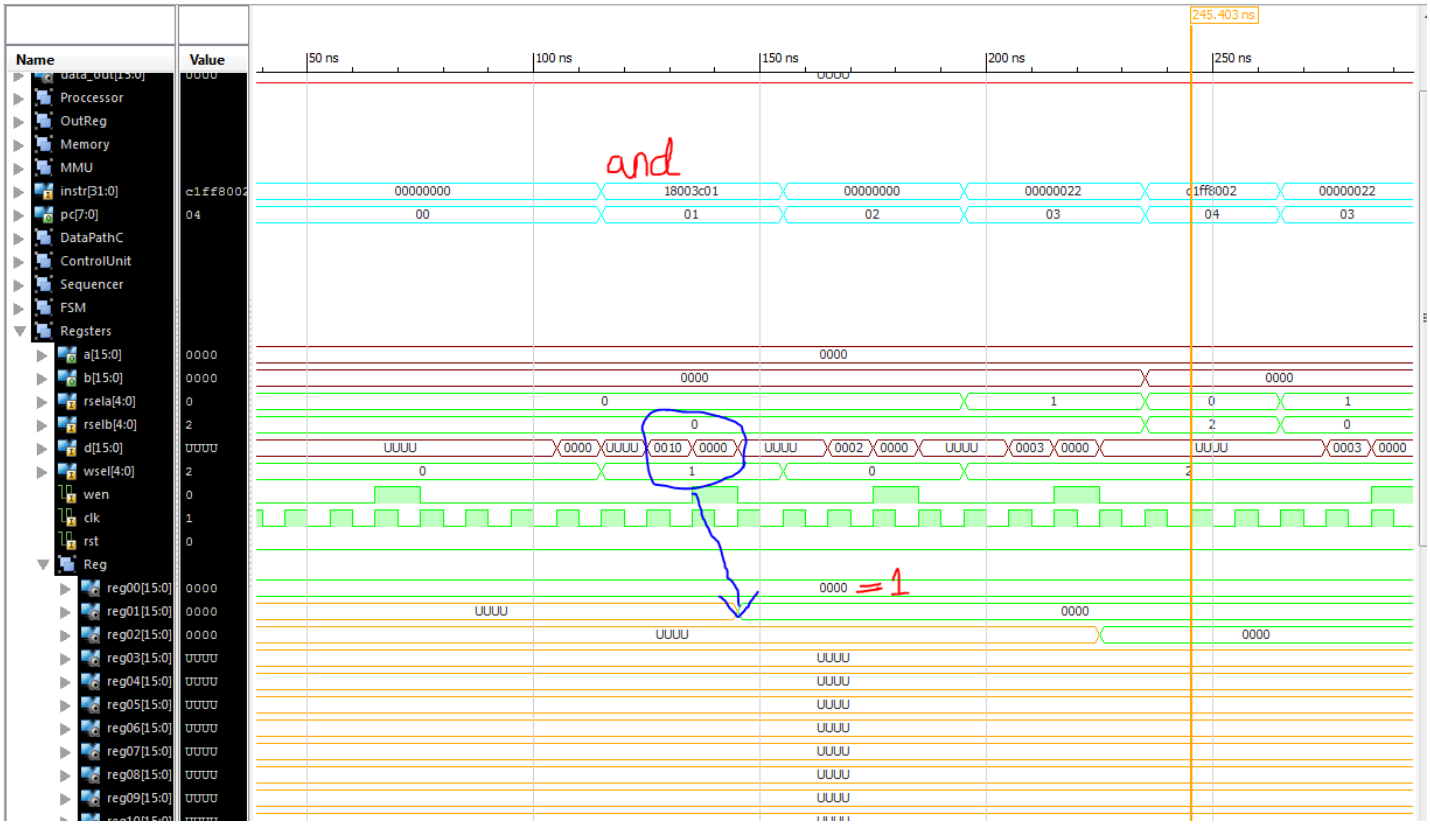
Top Level Processor Testing

SIMPLE BRANCH PROGRAM




```
-- simple branches test program
00000000
00000000
CDFF8000
00000000
C1FF8000
```

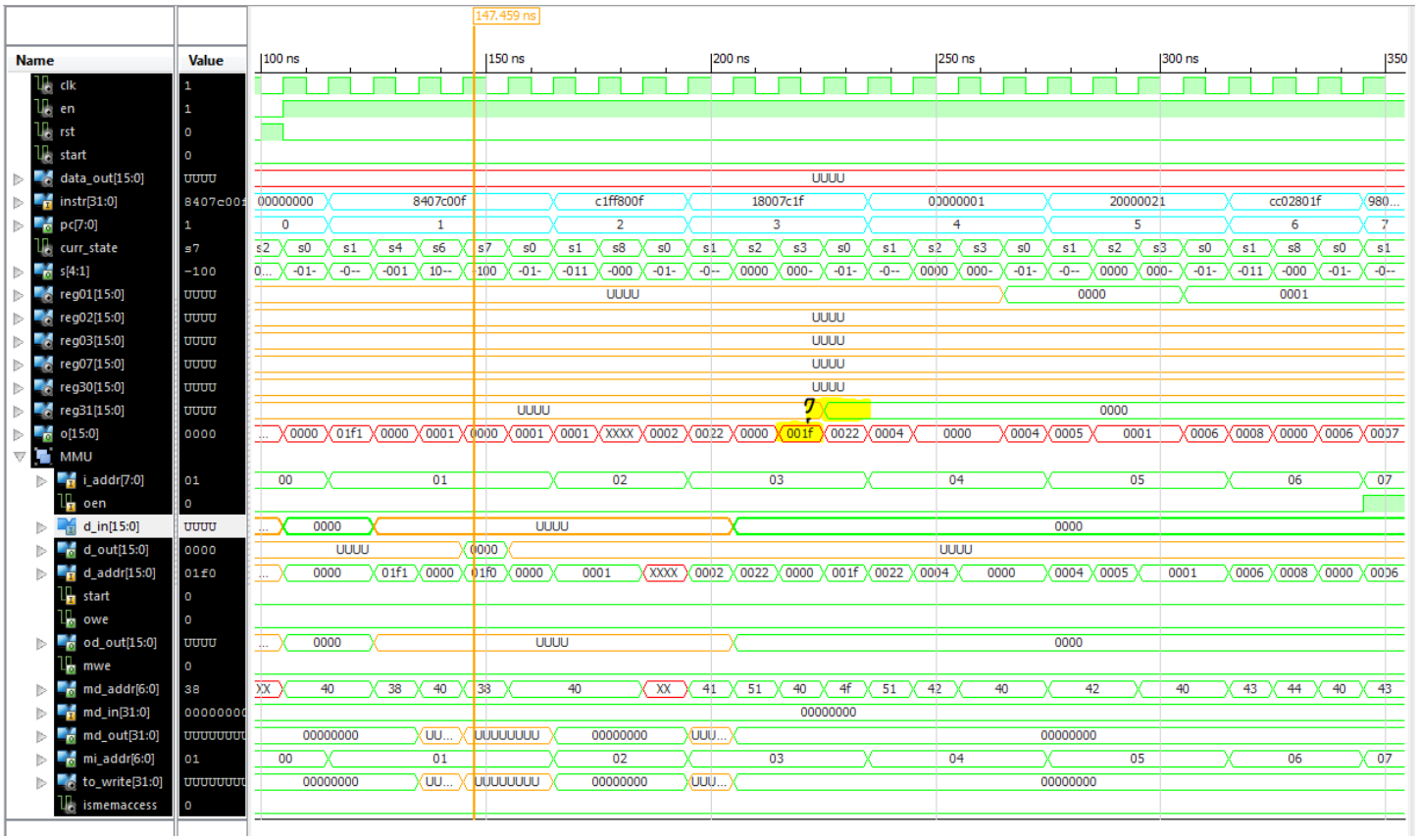
Simple ‘And’ Program (Processor TB)



`-- simple adds test program`
`00000000`
`18003C01`
`00000000`
`00000022`
`C1FF8002`	

In the simulation above you can see the output of the data bus, “0010” is not being written into register one as we would expect. Instead, the value at Register 1 is equal to 0. The problem we are having in this test is probably the same as the one we are having in our full program simulation, as both times the registers get set to 0.

Full Program (Processor TB)



As highlighted in the simulation above, we encountered a bug in our implementation that stopped our processor from writing anything but zeros to memory. We tried isolating this into a simple test case, which is where the two simple programs above came from. After spending a lot of time trying to fix this, we were not able to find the solution. We belive that this is caused by a timing issue with the signal 'S'. We are expecting the value of 31 (01F) to be written to memory but instead we get zero. The value of 31 makes it all the way to the data input of the register banks. At some point we were able to get 31 written to memory but it was overridden by 0000 on the next clock cycle.

HDL Synthesis

```
=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Synthesizing Unit <Processor>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\Proccessor.vhd".
WARNING:Xst:2935 - Signal 'PC_16<15:8>', unconnected in block 'Processor', is tied to
its initial value (00000000).
    Summary:
        no macro.
Unit <Processor> synthesized.

Synthesizing Unit <OutputReg>.
```

```
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\OutputReg.vhd".
    Found 16-bit register for signal <data_out>.
    Summary:
       inferred  16 D-type flip-flop(s).
Unit <OutputReg> synthesized.

Synthesizing Unit <DualPortMemory>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\DualPortMemory.vhd".
    Found 128x32-bit dual-port RAM <Mram_ram> for signal <ram>.
    Summary:
       inferred   1 RAM(s).
Unit <DualPortMemory> synthesized.

Synthesizing Unit <MMU>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\MMU.vhd".
WARNING:Xst:647 - Input <I_addr<7:7>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
    Found 7-bit adder for signal <mD_addr> created at line 70.
    Found 16-bit comparator lessequal for signal <n0000> created at line 42
    Found 16-bit comparator greater for signal <D_addr[15]_GND_7_o_LessThan_2_o>
created at line 42
    Summary:
       inferred   1 Adder/Subtractor(s).
       inferred   2 Comparator(s).
       inferred   3 Multiplexer(s).
Unit <MMU> synthesized.

Synthesizing Unit <DataPath_C>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\DataPath_C.vhd".
        data_size = 16
        num_registers = 32
    Summary:
       inferred   4 Multiplexer(s).
Unit <DataPath_C> synthesized.

Synthesizing Unit <Reg>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\Reg.vhd".
        data_size = 16
    Found 16-bit register for signal <data_out>.
    Summary:
       inferred  16 D-type flip-flop(s).
Unit <Reg> synthesized.

Synthesizing Unit <ALU_param>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\ALU_param.vhd".
        N = 16
    Found 16-bit adder for signal <A_itrn[15]_B_itrn[15]_add_27_OUT> created at line
69.
    Found 16-bit adder for signal <A_itrn[15]_GND_11_o_add_31_OUT> created at line
1253.
```

```
    Found 16-bit subtractor for signal <A_itrn[15]_B_itrn[15]_sub_26_OUT<15:0>>
created at line 70.
    Found 16-bit subtractor for signal <A_itrn[15]_GND_11_o_sub_30_OUT<15:0>> created
at line 1320.
    Found 16-bit shifter rotate right for signal
<A_itrn[15]_X_itrn[30]_rotate_right_17_OUT> created at line 3021
    Found 16-bit shifter rotate left for signal
<A_itrn[15]_X_itrn[30]_rotate_left_19_OUT> created at line 3012
    Found 16-bit shifter arithmetic right for signal
<A_itrn[15]_X_itrn[30]_shift_right_21_OUT> created at line 2982
    Found 16-bit shifter logical left for signal
<A_itrn[15]_X_itrn[30]_shift_left_23_OUT> created at line 2973
    Found 16-bit 13-to-1 multiplexer for signal <O_itrn> created at line 42.
    Found 16-bit comparator greater for signal <flags<3>> created at line 99
    Found 16-bit comparator greater for signal <flags<4>> created at line 100
    Summary:
       inferred   1 Adder/Subtractor(s).
       inferred   2 Comparator(s).
       inferred  16 Multiplexer(s).
       inferred   4 Combinational logic shifter(s).
Unit <ALU_param> synthesized.

Synthesizing Unit <regbank>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\otherRegBank.vhd".
    Found 16-bit register for signal <REG02>.
    Found 16-bit register for signal <REG03>.
    Found 16-bit register for signal <REG04>.
    Found 16-bit register for signal <REG05>.
    Found 16-bit register for signal <REG06>.
    Found 16-bit register for signal <REG07>.
    Found 16-bit register for signal <REG08>.
    Found 16-bit register for signal <REG09>.
    Found 16-bit register for signal <REG10>.
    Found 16-bit register for signal <REG11>.
    Found 16-bit register for signal <REG12>.
    Found 16-bit register for signal <REG13>.
    Found 16-bit register for signal <REG14>.
    Found 16-bit register for signal <REG15>.
    Found 16-bit register for signal <REG16>.
    Found 16-bit register for signal <REG17>.
    Found 16-bit register for signal <REG18>.
    Found 16-bit register for signal <REG19>.
    Found 16-bit register for signal <REG20>.
    Found 16-bit register for signal <REG21>.
    Found 16-bit register for signal <REG22>.
    Found 16-bit register for signal <REG23>.
    Found 16-bit register for signal <REG24>.
    Found 16-bit register for signal <REG25>.
    Found 16-bit register for signal <REG26>.
    Found 16-bit register for signal <REG27>.
    Found 16-bit register for signal <REG28>.
    Found 16-bit register for signal <REG29>.
    Found 16-bit register for signal <REG30>.
    Found 16-bit register for signal <REG31>.
    Found 16-bit register for signal <REG01>.
    Found 16-bit 32-to-1 multiplexer for signal <A> created at line 30.
    Found 16-bit 32-to-1 multiplexer for signal <B> created at line 31.
```

```
    Summary:
       inferred 496 D-type flip-flop(s).
       inferred   2 Multiplexer(s).
Unit <regbank> synthesized.

Synthesizing Unit <ControlUnit>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\ControlUnit.vhd".
    Summary:
       no macro.
Unit <ControlUnit> synthesized.

Synthesizing Unit <ControlFSM>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\ControlFSM.vhd".
    Found 4-bit register for signal <curr_state>.
    Found finite state machine <FSM_0> for signal <curr_state>.
    -----------------------------------------------------------------------
    | States             | 9                                              |
    | Transitions        | 12                                             |
    | Inputs             | 3                                              |
    | Outputs            | 10                                             |
    | Clock              | clk (rising_edge)                              |
    | Reset              | rst (positive)                                 |
    | Reset type         | synchronous                                    |
    | Reset State        | s0                                             |
    | Power Up State     | s0                                             |
    | Encoding           | auto                                           |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Summary:
       inferred   4 Multiplexer(s).
       inferred   1 Finite State Machine(s).
Unit <ControlFSM> synthesized.

Synthesizing Unit <Sequencer>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\Sequencer.vhd".
WARNING:Xst:647 - Input <STAGE<7:1>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <instr<25:0>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <flags<7:7>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
    Found 8-bit register for signal <PC_next>.
    Found 1-bit register for signal <cond_met_i>.
    Found 8-bit register for signal <PC_internal>.
    Found 8-bit adder for signal <PC_internal[7]_GND_46_o_add_13_OUT> created at line
1241.
    Summary:
       inferred   1 Adder/Subtractor(s).
       inferred  17 D-type flip-flop(s).
       inferred   7 Multiplexer(s).
Unit <Sequencer> synthesized.
```

```
Synthesizing Unit <Decoder_Block>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\Decoder.vhd".
WARNING:Xst:647 - Input <STAGE<0:0>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <STAGE<2:2>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <STAGE<6:4>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
    Summary:
       inferred  17 Multiplexer(s).
Unit <Decoder_Block> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# RAMs                                                 : 1
 128x32-bit dual-port RAM                              : 1
# Adders/Subtractors                                   : 3
 16-bit addsub                                         : 1
 7-bit adder                                           : 1
 8-bit adder                                           : 1
# Registers                                            : 39
 1-bit register                                        : 1
 16-bit register                                       : 36
 8-bit register                                        : 2
# Comparators                                          : 4
 16-bit comparator greater                             : 3
 16-bit comparator lessequal                           : 1
# Multiplexers                                         : 53
 1-bit 2-to-1 multiplexer                              : 17
 16-bit 2-to-1 multiplexer                             : 18
 16-bit 32-to-1 multiplexer                            : 2
 32-bit 2-to-1 multiplexer                             : 1
 4-bit 2-to-1 multiplexer                              : 13
 5-bit 2-to-1 multiplexer                              : 1
 8-bit 2-to-1 multiplexer                              : 1
# Logic shifters                                       : 4
 16-bit shifter arithmetic right                       : 1
 16-bit shifter logical left                           : 1
 16-bit shifter rotate left                            : 1
 16-bit shifter rotate right                           : 1
# FSMs                                                 : 1
# Xors                                                 : 1
 16-bit xor2                                           : 1

=========================================================================
```

Synthesis Warnings

```
WARNING:HDLCompiler:634 - "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Project\Project\Proccessor.vhd" Line 62:
Net <PC_16[15]> does not have a driver.
WARNING:Xst:2935 - Signal 'PC_16<15:8>', unconnected in block 'Processor', is tied to
its initial value (00000000).
WARNING:Xst:647 - Input <I_addr<7:7>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <STAGE<7:1>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <instr<25:0>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <flags<7:7>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <STAGE<0:0>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <STAGE<2:2>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
WARNING:Xst:647 - Input <STAGE<6:4>> is never used. This port will be preserved and
left unconnected if it belongs to a top-level block or it belongs to a sub-block and
the hierarchy of this sub-block is preserved.
```

UCF File

```
# PlanAhead Generated physical constraints

NET "data_out[15]" LOC = N12;
NET "data_out[14]" LOC = P16;
NET "data_out[13]" LOC = D4;
NET "data_out[12]" LOC = M13;
NET "data_out[11]" LOC = L14;
NET "data_out[10]" LOC = N14;
NET "data_out[9]" LOC = M14;
NET "data_out[8]" LOC = U18;
NET "clk" LOC = L15;
NET "en" LOC = T15;
NET "rst" LOC = P3;
NET "start" LOC = F6;
```