DESIGN PROJECT

(MATRIX OPERATION UNIT)

PROCESSOR DESIGN

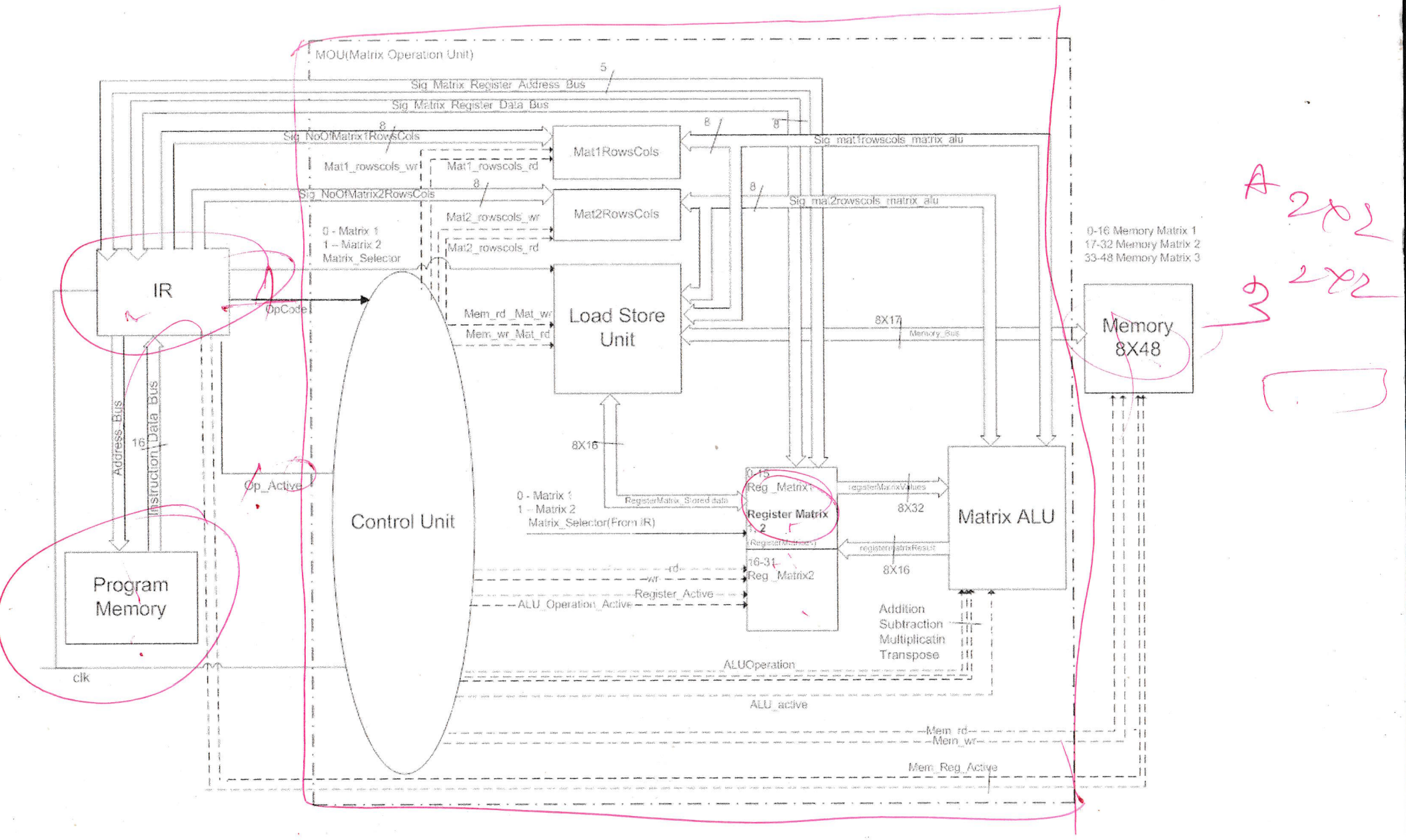
ECX 6236

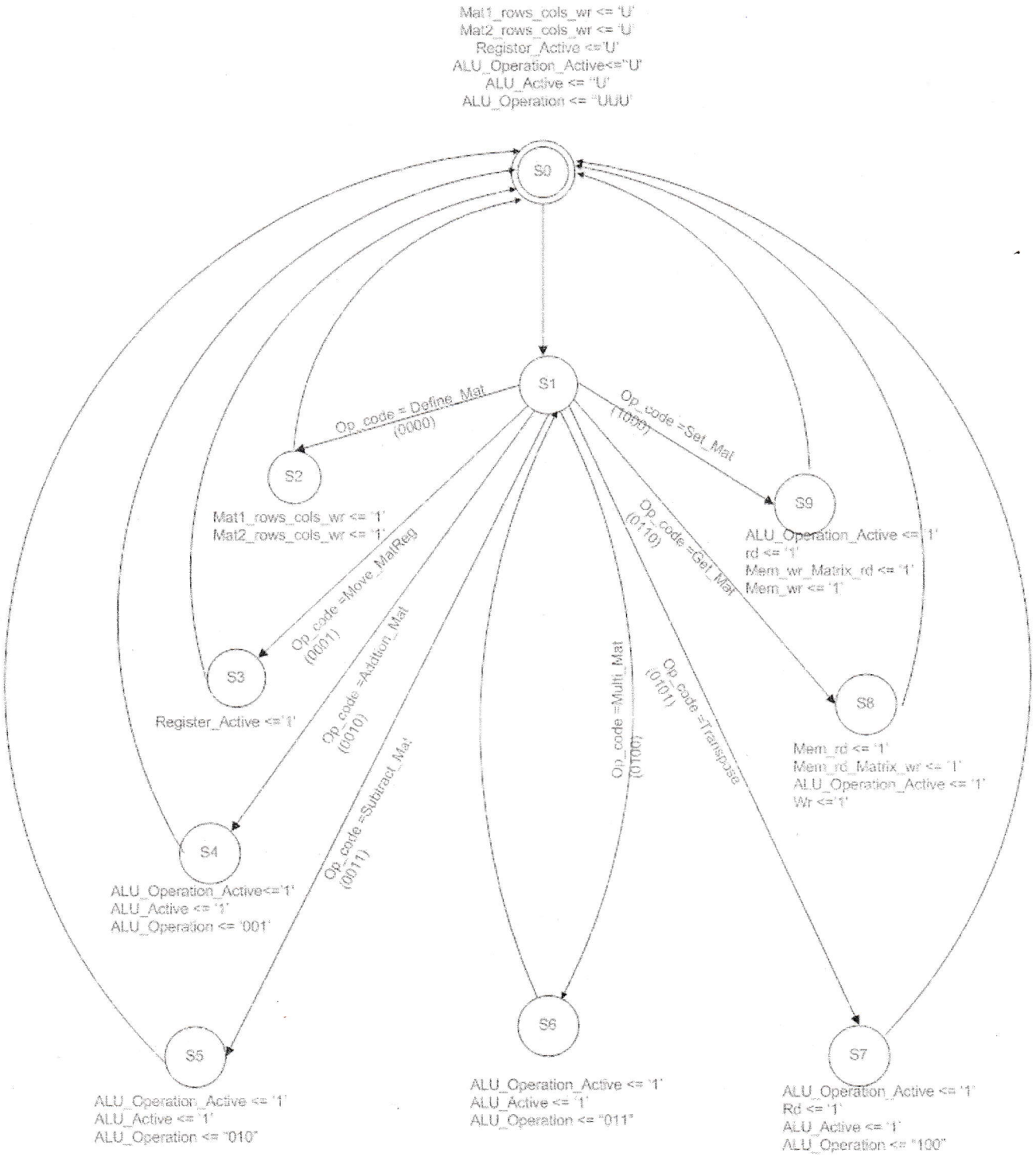
NAME : C.K KODITHUWAKKU

REG:NO : 40365222

CENTER : MATARA

Table of Content


MOU(Matrix Operation Unit)
Sig_Matrix_Register_Address_Bus
Sig_Matrix_Register_Data_Bus
Sig_NoOfMatrix1RowsCols
Mat1_rowscols_wr
Mat1_rowscols_rd
Mat1RowsCols
Sig_mat1rowscols_matrix_alu
Sig_NoOfMatrix2RowsCols
Mat2_rowscols_wr
Mat2_rowscols_rd
Mat2RowsCols
Sig_mat2rowscols_matrix_alu
0 - Matrix 1
1 – Matrix 2
Matrix_Selector
IR
OpCode
Mem_rd_Mat_wr
Mem_wr_Mat_rd
Load Store Unit
8X17
Memory_Bus
0-16 Memory Matrix 1
17-32 Memory Matrix 2
33-48 Memory Matrix 3
Memory 8X48
Address_Bus
Instruction_Data_bus
16
8X16
Op_Active
Control Unit
0 - Matrix 1
1 – Matrix 2
Matrix_Selector(From IR)
RegisterMatrix_Stored data
0-15
Reg_Matrix1
Register Matrix
2
registerMatrixValues
8X32
Matrix ALU
registerMatrixResult
8X16
16-31
Reg_Matrix2
rd
wr
Register_Active
ALU_Operation_Active
Addition
Subtraction
Multiplicatin
Transpose
ALUOperation
ALU_active
Program Memory
clk
Mem_rd
Mem_wr
Mem_Reg_Active



State Diagram of Control Unit

VHDL Coding

Pro_Memory

```
----------------------------------------------------------------------------
-- Create Date:    21:49:46 12/01/2009
-- Design Name:
-- Module Name:    Pro_Memory - Behavioral
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.Processor_Package.ALL;

--Pro_Memory entity is Program Memory
--This is used to store Instructions

entity Pro_Memory is
   Port (          Instr_Address :in STD_LOGIC_VECTOR(7 downto 0);
                                    Instr_Out : out  STD_LOGIC_VECTOR(16 downto 0));
end Pro_Memory;

architecture Behavioral of Pro_Memory is

CONSTANT pro_mem_data : ProgramMemeoryArray :=(
                                                "00000000111100000",
                                                "00000000000000000",
                                                "00000000000000000",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
                                                "00000000000111110",
```

```
                                        "00000000000111110",
                                        "00000000000111110",
                                        "00000000000111110",
                                        "00000000000111110",
                                        "00000000000111110",
                                        "00000000000111110",
                                        "00000000000111110",
                                        "00000000000111110",
                                        "00000000000111110",
                                        "00000000000111110"
                                        );

begin

instr_Out <= pro_mem_data(conv_integer(Instr_Address));
end Behavioral;
```

Instruction Register

```
----------------------------------------------------------------------------------
-- Create Date:    17:45:35 12/17/2009
-- Design Name:
-- Module Name:    IR - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity IR is
   Port (
                        clk : in STD_LOGIC;
                        Inst_Address : out  STD_LOGIC_VECTOR (7 downto 0);-- address should be
out but i used in for input address of memory
                        Instr_Data : in  STD_LOGIC_VECTOR (16 downto 0);
                        OpCode : out STD_LOGIC_VECTOR(3 downto 0);
                        OpActive: out STD_LOGIC;
                        MatrixRegister_Address : out STD_LOGIC_VECTOR (4 downto 0);
                        MatrixRegister_Data :out STD_LOGIC_VECTOR (7 downto 0);
                        Matrix1RowsCols : out STD_LOGIC_VECTOR (7 downto 0);
                        Matrix2RowsCols : out STD_LOGIC_VECTOR (7 downto 0);
                        Mem_RegActive : out STD_LOGIC_VECTOR (1 downto 0);
                        Matrix_selector:out STD_LOGIC
                       );
end IR;

architecture Behavioral of IR is
signal tempData :STD_LOGIC_VECTOR (Instr_Data'HIGH downto 0);
signal tempValuesForRowsCols :STD_LOGIC_VECTOR(7 downto 0);
signal tempOpCode :STD_LOGIC_VECTOR(3 downto 0);
signal tempRegisterAddress:STD_LOGIC_VECTOR(4 downto 0);
signal tempRegisterValue :STD_LOGIC_VECTOR(7 downto 0);
signal tempMatrixSelector:STD_LOGIC_VECTOR(1 downto 0);-- to select matrix to define values per
row of the matrix

begin

--      this process is used to send instruction memory address
        process(clk)
        variable counter:integer :=0;
        begin
                if(clk'EVENT and clk = '1') then
                        Inst_Address <= conv_std_logic_vector(counter,8);
                        counter := counter +1;
                end if;
        end process;
```

```
--      this process is used to grab input Instr_Data to tempData
        process(clk,Instr_Data)
        begin
                if(clk'EVENT and clk = '1') then
                tempData <= Instr_Data;
                end if;
        end process;

--get the opcode
        tempOpCode(3) <= tempData(16);
        tempOpCode(2) <= tempData(15);
        tempOpCode(1) <= tempData(14);
        tempOpCode(0) <= tempData(13);

--      this process is used to identify valied instructions for MOU
        process(clk,tempOpCode)
        begin
         if(clk'Event and clk = '1') then
         case tempOpCode is
                when    "0000" |
                                "0001" |
                                "0010" |
                                "0011" |
                                "0100" |
                                "0110" |
                                "1000" |
                                "1001" |
                                "1010" =>  OpCode <= tempOpCode ;
                when others => null;
        end case;

         case tempOpCode is
                when    "0000" |
                                "0001" |
                                "0010" |
                                "0011" |
                                "0100" |
                                "0110" |
                                "1000" |
                                "1001" |
                                "1010" => OpActive <='1' ;
                when others => null;

        end case;
        end if;
        end process;

--this process is used to matrix values for rows and cols of matrix
--7to4 is Matrix no:Of Rows 3 to 0 is Matrix no of Cols
```

```
        tempValuesForRowsCols(7) <= tempData(7);
        tempValuesForRowsCols(6) <= tempData(6);
        tempValuesForRowsCols(5) <= tempData(5);
        tempValuesForRowsCols(4) <= tempData(4);
        tempValuesForRowsCols(3) <= tempData(3);
        tempValuesForRowsCols(2) <= tempData(2);
        tempValuesForRowsCols(1) <= tempData(1);
        tempValuesForRowsCols(0) <= tempData(0);

        tempMatrixSelector(1) <= tempData(12);
        tempMatrixSelector(0) <= tempData(11);

        process(clk,tempMatrixSelector)
        begin
                if(clk'Event and clk = '1') then
                case tempMatrixSelector is
                when "01" => Matrix1RowsCols  <= tempValuesForRowsCols;
                when "10" => Matrix2RowsCols  <= tempValuesForRowsCols;
                when others =>  Matrix1RowsCols <= "ZZZZZZZZ" ;  Matrix2RowsCols <= "ZZZZZZZZ";
                end case;
                end if;
        end process;
--      ValuesPerRow <= tempValuesPerRow;

--this process is used to pass resgister addresses of  register matrix
        tempRegisterAddress(4) <= tempData(12);
        tempRegisterAddress(3) <= tempData(11);
        tempRegisterAddress(2) <= tempData(10);
        tempRegisterAddress(1) <= tempData(9);
        tempRegisterAddress(0) <= tempData(8);

        MatrixRegister_Address <= tempRegisterAddress;

--this process is used to pass values to register matrix
        tempRegisterValue(7) <= tempData(7);
        tempRegisterValue(6) <= tempData(6);
        tempRegisterValue(5) <= tempData(5);
        tempRegisterValue(4) <= tempData(4);
        tempRegisterValue(3) <= tempData(3);
        tempRegisterValue(2) <= tempData(2);
        tempRegisterValue(1) <= tempData(1);
        tempRegisterValue(0) <= tempData(0);

        MatrixRegister_Data <= tempRegisterValue;

--this process is used to pass values to memory for select memory matrix from 3 memory matrices
        Mem_RegActive(1) <= tempData(11);
        Mem_RegActive(0) <= tempData(10);

--this process is used to pass values to LoadStoreUnit for select matrix from 2 register matrices
        Matrix_selector <= tempData(12);
```

```
end Behavioral;
Mat1RowsCols
----------------------------------------------------------------------------------
-- Create Date:    11:12:37 12/31/2009
-- Design Name:
-- Module Name:    Mat1RowsCols - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.Processor_Package.ALL;

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

entity Mat1RowsCols is
  Port (
        clk : in STD_LOGIC;
        NoOfMatrix1RowsCols : in  STD_LOGIC_VECTOR (7 downto 0);
        mat1_rowscols_wr : in  STD_LOGIC;
        mat1_rowscols_rd : in  STD_LOGIC;
        StoredValue : inout  STD_LOGIC_VECTOR (7 downto 0));
end Mat1RowsCols;

architecture Behavioral of Mat1RowsCols is
signal tempStoredValue :STD_LOGIC_VECTOR (7 downto 0);
begin

process(clk, NoOfMatrix1RowsCols,mat1_rowscols_wr, mat1_rowscols_rd)
begin
if(clk'EVENT and clk= '1') then
tempStoredValue <= NoOfMatrix1RowsCols;

if(mat1_rowscols_wr = '1') then
tempStoredValue <= StoredValue;
end if;

if(mat1_rowscols_rd = '1') then
StoredValue <= tempStoredValue;
end if;
end if;
end process;
end Behavioral;
```

Mat2RowsCol

```
----------------------------------------------------------------------------------
-- Create Date:    11:30:45 12/31/2009
-- Design Name:
-- Module Name:    Mat2RowsCols - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.Processor_Package.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Mat2RowsCols is
Port (
                        clk : in STD_LOGIC;
                        NoOfMatrix2RowsCols : in  STD_LOGIC_VECTOR (7 downto 0);
      mat2_rowscols_wr : in  STD_LOGIC;
      mat2_rowscols_rd : in  STD_LOGIC;
      StoredValue : inout  STD_LOGIC_VECTOR (7 downto 0));

end Mat2RowsCols;

architecture Behavioral of Mat2RowsCols is
signal tempStoredValue :STD_LOGIC_VECTOR (7 downto 0);
begin

process(clk, NoOfMatrix2RowsCols,mat2_rowscols_wr, mat2_rowscols_rd)
begin
if(clk'EVENT and clk= '1') then
tempStoredValue <= NoOfMatrix2RowsCols;

if(mat2_rowscols_wr = '1') then
tempStoredValue <= StoredValue;
end if;

if(mat2_rowscols_rd = '1') then
StoredValue <= tempStoredValue;
end if;
end if;
```

```
end process;
end Behavioral;
LoadStoreUnit
----------------------------------------------------------------------------------
-- Create Date:   10:36:32 12/29/2009
-- Design Name:
-- Module Name:   LoadStoreUnit - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.Processor_Package.ALL;

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

entity LoadStoreUnit is
   Port ( clk : in STD_LOGIC;
        Matrix1RowsCols : inout STD_LOGIC_VECTOR (7 downto 0);
        Matrix2RowsCols : inout STD_LOGIC_VECTOR (7 downto 0);
                          MemoryBus: inout MemoryType;
                          RegisterMatrixStoredData: inout Matrix_Register;
                          Mem_rd_Mat_wr: in STD_LOGIC;
                          Mem_wr_Mat_rd: in STD_LOGIC;
                          Matrix_Selector: in STD_LOGIC
                          );
end LoadStoreUnit;

architecture Behavioral of LoadStoreUnit is

begin

Load:process(clk,MemoryBus,mem_rd_mat_wr,Matrix_Selector)
                begin
                        if(clk'EVENT and clk ='1') then
                                if(mem_rd_mat_wr= '1') then
                                        if(Matrix_Selector = '0') then
                                                Matrix1RowsCols <= MemoryBus(16);
                                                for i in 0 to 15 loop
                                                 RegisterMatrixStoredData(i) <= MemoryBus(i);
                                                end loop;
                                        elsif(Matrix_Selector = '1') then
```

```
                                        Matrix2RowsCols <= MemoryBus(16);
                                        for i in 0 to 15 loop
                                         RegisterMatrixStoredData(i) <= MemoryBus(i);
                                        end loop;
                                        end if;
                                end if;
                        end if;
        end process;

Store:process(clk,RegisterMatrixStoredData,mem_wr_mat_rd,Matrix_Selector)
        begin
                if(clk'EVENT and clk ='1') then
                        if(mem_wr_mat_rd = '1') then
                                if(Matrix_Selector = '0') then
                                        MemoryBus(16)<= Matrix1RowsCols;
                                        for i in 0 to 15 loop
                                         MemoryBus(i) <= RegisterMatrixStoredData(i);
                                        end loop;
                                elsif(Matrix_Selector = '1') then
                                         MemoryBus(16) <=Matrix2RowsCols;
                                        for i in 0 to 15 loop
                                         MemoryBus(i) <= RegisterMatrixStoredData(i);
                                        end loop;
                                        end if;
                                end if;
                        end if;
        end process;

end Behavioral;
```

Register_Matrices

```
----------------------------------------------------------------------------------
-- Create Date:    14:44:33 12/19/2009
-- Design Name:
-- Module Name:    Register_Matrices - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.Processor_Package.ALL;

entity Register_Matrices is
   Port ( clk : in STD_LOGIC;
                           Register_Address : in  STD_LOGIC_VECTOR (4 downto 0);
          Register_Data : in  STD_LOGIC_VECTOR (7 downto 0);
          RegisterMatrix_Stored_Data : inout Matrix_Register;
                           RegisterMatrix_Values: out Register_Bank;
                           RegisterMatrix_Result : in Matrix_Register;
                           ALU_Operation_Active: in STD_LOGIC;
                           Matrix_Selector : in STD_LOGIC;
                           Register_Active : in STD_LOGIC;
                           rd: in STD_LOGIC;
                           wr: in STD_LOGIC );
end Register_Matrices;

architecture Behavioral of Register_Matrices is
signal Register_Matrix_Bank :Register_Bank;-- this is 8X32 register banck for 2 register matrix
signal Register_Matrix_1 : Matrix_Register;
signal Register_Matrix_2 : Matrix_Register;

signal tempRegister_Matrix_Bank:Register_Bank;
begin


        process(clk,wr,Matrix_Selector,Register_Active,RegisterMatrix_Stored_Data,RegisterMatrix_
Result)
                                begin
                                if(clk'EVENT and clk ='1') then
                                --this is used to store values into Matrix's Registers
                                if(Register_Active ='1') then
                                        Register_Matrix_Bank( conv_integer( Register_Address)) <=
Register_Data;
```

```
elsif(wr = '1' and Matrix_Selector = '0') then
        for i in 0 to 15 loop
        Register_Matrix_Bank(i)<= RegisterMatrix_Stored_Data(i);
        end loop;
elsif(wr = '1' and Matrix_Selector = '1') then
        for i in 16 to 31 loop
Register_Matrix_Bank(i) <= RegisterMatrix_Stored_Data(i-16);
        end loop;
elsif(wr ='1' and ALU_Operation_Active ='1') then
        for i in 0 to 15 loop
        Register_Matrix_Bank(i) <= RegisterMatrix_Result(i);
        end loop;
end if;
end if;
end process;

process(clk,rd,Matrix_Selector,Register_Matrix_Bank)
begin
if(clk'EVENT and clk ='1') then
if(rd = '1' and Matrix_Selector = '0') then
for i in 0 to 15 loop
                RegisterMatrix_Stored_Data(i) <=
Register_Matrix_Bank(i);
end loop;
elsif(rd = '1' and Matrix_Selector = '1') then
for i in 16 to 31 loop
                RegisterMatrix_Stored_Data(i-16) <=
Register_Matrix_Bank(i);
end loop;
elsif(rd = '1' and ALU_Operation_Active ='1') then
for i in 0 to 31 loop
                RegisterMatrix_values(i) <=
Register_Matrix_Bank(i);
end loop;
end if;
end if;
end process;
end Behavioral;
```

Matrix_ALU

```
----------------------------------------------------------------------------------
-- Create Date:   07:31:57 12/23/2009
-- Design Name:
-- Module Name:   Matrix_ALU - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.Processor_Package.ALL;

entity Matrix_ALU is
  Port ( clk : in STD_LOGIC;
                        Matrix_In : in  Register_Bank;
       Matrix1RowsCols : in  STD_LOGIC_VECTOR (7 downto 0);
       Matrix2RowsCols : in  STD_LOGIC_VECTOR (7 downto 0);
       Matrix_Out : out  Matrix_Register;
       Matrix_ALU_Active : in  STD_LOGIC;
       ArithmeticOp_Active : in  STD_LOGIC_VECTOR (2 downto 0));
end Matrix_ALU;

architecture Behavioral of Matrix_ALU is
--Matrix1 and Matrix2 use to addition and subtractions
signal Matrix1: Matrix_Register;
signal Matrix2: Matrix_Register;
signal Matrix1Rows:STD_LOGIC_VECTOR (3 downto 0);
signal Matrix1Cols:STD_LOGIC_VECTOR (3 downto 0);
signal Matrix2Rows:STD_LOGIC_VECTOR (3 downto 0);
signal Matrix2Cols:STD_LOGIC_VECTOR (3 downto 0);
--tempMatrix1 and tempMatrix2 use to Multiplication
signal tempMatrix1: MatrixType(0 to 3,0 to 3);
signal tempMatrix2: MatrixType(0 to 3,0 to 3);
signal tempMatrixResult :MatrixType(0 to 3,0 to 3);

begin
--input matrices register values separate to two matrices called Matrix1 and Matrix2
StoreMatrices:process(clk,Matrix_In)
                                    begin
                                        if(clk'EVENT and clk='1') then
                                            for i in 0 to 15 loop
                                            Matrix1(i)<= Matrix_In(i);
```

```
                                        Matrix2(i)<= Matrix_In(i + 16);
                                        end loop;
                                        end if;
                                end process;

-- use to get matrix1 and matrix2 rows and cols
        Matrix1Rows(3) <= Matrix1RowsCols(7);
        Matrix1Rows(2) <= Matrix1RowsCols(6);
        Matrix1Rows(1) <= Matrix1RowsCols(5);
        Matrix1Rows(0) <= Matrix1RowsCols(4);

        Matrix1Cols(3) <= Matrix1RowsCols(3);
        Matrix1Cols(2) <= Matrix1RowsCols(2);
        Matrix1Cols(1) <= Matrix1RowsCols(1);
        Matrix1Cols(0) <= Matrix1RowsCols(0);

        Matrix2Rows(3) <= Matrix2RowsCols(7);
        Matrix2Rows(2) <= Matrix2RowsCols(6);
        Matrix2Rows(1) <= Matrix2RowsCols(5);
        Matrix2Rows(0) <= Matrix2RowsCols(4);

        Matrix2Cols(3) <= Matrix2RowsCols(3);
        Matrix2Cols(2) <= Matrix2RowsCols(2);
        Matrix2Cols(1) <= Matrix2RowsCols(1);
        Matrix2Cols(0) <= Matrix2RowsCols(0);

--fill the values into Matrix1 and matrix2 according to given MatrixRows and MatrixCols
MatrixMappingForMatrix1:process(clk,Matrix1Rows,Matrix1Cols)
begin
        if(clk'EVENT and clk='1') then
        for i in 0 to 3 loop
        if(i =0) then
                for j in 0 to conv_integer(Matrix1Cols)-1 loop
                        tempMatrix1(0,j) <= Matrix1(j);
                        end loop;
        elsif(i=1) then
                for j in 0 to conv_integer(Matrix1Cols)-1 loop
                        tempMatrix1(1,j) <= Matrix1(j+4);
                end loop;
        elsif(i=2) then
                for j in 0 to conv_integer(Matrix1Cols)-1 loop
                        tempMatrix1(2,j) <= Matrix1(j+8);
                end loop;
        elsif(i =3) then
                for j in 0 to conv_integer(Matrix1Cols)-1 loop
```

```
                tempMatrix1(3,j) <= Matrix1(j+12);
            end loop;
                                        end if;
                                        end loop;
                                        end if;
                                    end process;

MatrixMappingForMatrix2:process(clk,Matrix2Rows,Matrix2Cols)
                        begin
                    if(clk'EVENT and clk='1') then
                        for i in 0 to 3 loop
                        if(i =0) then
                        for j in 0 to conv_integer(Matrix2Cols)-1 loop
                        tempMatrix2(0,j) <= Matrix2(j);
                        end loop;
                    elsif(i=1) then
                        for j in 0 to conv_integer(Matrix2Cols)-1 loop
                        tempMatrix2(1,j) <= Matrix2(j+4);
                        end loop;
                    elsif(i=2) then
                        for j in 0 to conv_integer(Matrix2Cols)-1 loop
                        tempMatrix2(2,j) <= Matrix2(j+8);
                        end loop;
                    elsif(i =3) then
                        for j in 0 to conv_integer(Matrix2Cols)-1 loop
                        tempMatrix2(3,j) <= Matrix2(j+12);
                        end loop;
                    end if;
                    end loop;
                    end if;
            end process;


MatrixArithmeticOperations:process(clk,ArithmeticOp_Active)
            variable mat1value,mat2value,resultvalue:integer;
            begin
            if(clk'EVENT and clk='1') then
                if(ArithmeticOp_Active = "001") then
                for i in 0 to 15 loop
                Matrix_Out(i) <= Matrix1(i) + Matrix2(i);
                end loop;
            elsif(ArithmeticOp_Active = "010") then
                for i in 0 to 15 loop
                Matrix_Out(i) <= Matrix1(i) - Matrix2(i);
                end loop;
```

```
elsif(ArithmeticOp_Active = "011") then
        for i in 0 to       conv_integer(Matrix1Rows)-1 loop
                for j in 0 to conv_integer(Matrix2Cols)-1 loop
                        for k in 0 to       conv_integer(Matrix1Cols)-1 loop
        resultvalue := conv_integer(tempMatrixResult(i,j)) +
        conv_integer(tempMatrix1(i,k))*conv_integer(tempMatrix2(k,j));
        tempMatrixResult(i,j) <= conv_std_logic_vector(resultvalue,8);
                end loop;
        end loop;
        end loop;

        for i in 0 to 3 loop
                if(i =0) then
                        for j in 0 to conv_integer(Matrix2Cols)-1 loop
                        Matrix_Out(j) <= tempMatrixResult(0,j);
                        end loop;
                elsif(i=1) then
                        for j in 0 to conv_integer(Matrix2Cols)-1 loop
                        Matrix_Out(j+4) <= tempMatrixResult(1,j);
                        end loop;
                elsif(i=2) then
                        for j in 0 to conv_integer(Matrix2Cols)-1 loop
                        Matrix_Out(j+8) <= tempMatrixResult(2,j);
                        end loop;
                elsif(i =3) then
                        for j in 0 to conv_integer(Matrix2Cols)-1 loop
                        Matrix_Out(j+12) <= tempMatrixResult(3,j);
                        end loop;
                end if;
                end loop;
    end if;
    end if;
    end process;
    end Behavioral;
```

```
                end if;
                end if;
end process;

process(clk,wr,mem_active,Memory_Bus)
begin
        if(clk'EVENT and clk='1') then
        if(wr = '1') then
                if(mem_active ="01") then
                        MemoryMatrix1 <= Memory_Bus;
                elsif(mem_active = "10") then
                        MemoryMatrix2 <= Memory_Bus;
                elsif(mem_active = "11") then
                        MemoryMatrix3 <= Memory_Bus;
                end if;
                end if;
        end if;
end process;
end Behavioral;
```

Matrix_Controller

```
----------------------------------------------------------------------------------
-- Create Date:    16:16:17 12/29/2009
-- Design Name:
-- Module Name:    Matrix_Controller - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.Processor_Package.ALL;

entity Matrix_Controller is
  Port (
                        clk : in STD_LOGIC;
                        opActive:in STD_LOGIC;
                        opCode:in STD_LOGIC_VECTOR(3 downto 0);
                        mat1_rows_cols_wr:out STD_LOGIC;
                        mat1_rows_cols_rd:out STD_LOGIC;
                        mat2_rows_cols_wr:out STD_LOGIC;
                        mat2_rows_cols_rd:out STD_LOGIC;
                        mem_rd_mat_wr: out STD_LOGIC;
                        mem_wr_mat_rd:out STD_LOGIC;
                        rd:out STD_LOGIC;
```

```
                    wr:out STD_LOGIC;
                    register_Active:out STD_LOGIC;
                    alu_Operation_Active:out STD_LOGIC;
                    ALUOperation:out STD_LOGIC_VECTOR(2 downto 0);
                    alu_Active:out STD_LOGIC;
                    mem_rd:out STD_LOGIC;
                    mem_wr:out STD_LOGIC
                    );
end Matrix_Controller;

architecture Behavioral of Matrix_Controller is
TYPE STATES is (S0,S1,S2,S3,S4,S5,S6,S7,S8,S9);
signal currentState,nxtState : STATES;
begin

stateSequence:process(clk,OpActive)
                                    begin
                                    if(OpActive ='0') then
                                    currentState <= S0;
                                    else
                                    if(clk'EVENT and clk = '1') then
                                    currentState <= nxtState;
                                    end if;
                                    end if;
                                    end process;

stateMachine:process(currentState,OpCode)
                                    begin
                                    --initialised all of control signals

                                     case currentState is
                                     when S0 =>
                                                mat1_rows_cols_wr <= 'U';
                                                mat1_rows_cols_rd <= 'U';
                                                mat2_rows_cols_wr <= 'U';
                                                mat2_rows_cols_rd <= 'U';
                                                mem_rd_mat_wr <= 'U';
                                                mem_wr_mat_rd <= 'U';
                                                rd <= 'U';
                                                wr <= 'U';
                                                register_Active <= 'U';
                                                alu_Operation_Active <= 'U';
                                                ALUOperation <= "UUU";
                                                alu_Active <= 'U';
                                                mem_rd <= 'U';
```

```
                mem_wr <= 'U';
                        nxtState <= S1;
when S1 =>
                        case OpCode is
                        when "0000" => nxtState <= S2;
                        when "0001" => nxtState <= S3;
                        when "0010" => nxtState <= S4;
                        when "0011" => nxtState <= S5;
                        when "0100" => nxtState <= S6;
                        when "0101" => nxtState <= S7;
                        when "0110" => nxtState <= S8;
                        when "0111" => nxtState <= S9;
                        when others => nxtState <= S0;
                        end case;

when S2 =>
                        mat1_rows_cols_wr <= '1';
                        mat2_rows_cols_wr <= '1';
                        nxtState <= S0;

when S3 =>
                        register_Active <='1';
                        nxtState <= S0;

when S4 =>
                        alu_Operation_Active <='1' ;
                        alu_Active <= '1';
                        ALUOperation <= "001";
                        nxtState <= S0;
when S5 =>
                        alu_Operation_Active <='1' ;
                        alu_Active <= '1';
                        ALUOperation <= "010";
                        nxtState <= S0;
when S6 =>
                        alu_Operation_Active <='1' ;
                        alu_Active <= '1';
                        ALUOperation <= "011";
                        nxtState <= S0;
when S7 =>
--                      alu_Operation_Active <='1' ;
--                      alu_Active <= '1';
--                      ALUOperation <= "001";
                        nxtState <= S0;
when S8 =>
```

```
                    mem_rd <= '1';
                    mem_rd_mat_wr <= '1';
                    alu_Operation_Active <= '1';
                    wr <= '1';
                when S9 =>
                    alu_Operation_Active <= '1';
                    rd <= '1';
                    mem_wr_mat_rd <= '1';
                    mem_wr <='1';
                    nxtState <= S0;
            end case;
        end process;

end Behavioral;
```

Performance

The number of **transistors** available has a huge effect on the performance of a processor. As seen earlier, a typical instruction in a processor like an 8088 took 15 clock cycles to execute. Because of the design of the multiplier, it took approximately 80 cycles just to do one 16-bit multiplication on the 8088. With more transistors, much more powerful multipliers capable of single-cycle speeds become possible.

There are used two register matrices to arithmetic operations. We know registers are fast hardware than traditional memory. It is used to enhance instruction execution delay. It gives high execution result than used with memory. Also, result of the ALU is stored in same Register matrix. Therefore, that doesn't want to design another memory to store the result.

More transistors also allow for a technology called **pipelining**. In a pipelined architecture, instruction execution overlaps. So even though it might take five clock cycles to execute each instruction, there can be five instructions in various stages of execution simultaneously. That way it looks like one instruction completes every clock cycle.

We can use pipelining technique for Matrix multiplication operation. Then it can be avoided to multiplication

References

http://www.vhdl.org/vhdlsynth/vhdlexamples/

http://www.csee.umbc.edu/help/VHDL/samples/samples.shtml

http://webdocs.cs.ualberta.ca/~amaral/courses/329/labs/VHDL_Guideline.html

http://books.google.lk/books?id=g2KMClWIafkC&dq=vhdl+coding&printsec=frontcover&source=bl&ots=LCJcefBe9-&sig=EfnFQkczgZqpLgte_pJG7vSi2a0&hl=en&ei=eqg9S8uLG8qlkAWpn4C3Ag&sa=X&oi=book_result&ct=result&resnum=10&ved=0CCsQ6AEwCQ#v=onepage&q=&f=false

IEEE Press RTL Hardware Design Using VHDL by Pong P.CHU

the_designer_s_guide_to_vhdl by Peter J. Ashenden

McGraw.Hill.VHDL.Programming.by.Example.4th.Ed by douglues L. Perry