รายการอ้างอิง

ภาษาไทย

วรรณรัช สันติอมรทัต, เจริญ วงษ์ชุ่มเย็น, สมศักดิ์ มิตะถา และ อาทิตย์ ทองทักษ์, "การทวนสอบไมโครโปรเซสเซอร์ ARM7 ที่ออกแบบด้วยภาษา VHDL." วารสารลาดกระบัง, มิถุนายน, หน้า 52-55, 2542.

วรรณรัช สันติอมรทัต, เจริญ วงษ์ชุ่มเย็น, อภิเนตร อูนากูล และ อาทิตย์ ทองทักษ์, "การออกแบบ 32 บิตไมโครโปรเซสเซอร์ ARM 7." การประชุมวิชาการวิศวกรรมไฟฟ้าครั้งที่ 21, ธนบุรี, กรุงเทพมหานคร, หน้า.581-584, 2541.

วรรณรัช สันติอมรทัต, สมศักดิ์ มิตะถะ, อาทิตย์ ทองทักษ์, "การสร้างชุดทดสอบสำหรับการทวนสอบแบบจำลองการทำงานของไมโครโปรเซสเซอร์ ARM 7." การประชุมวิชาการทางวิศวกรรมไฟฟ้าครั้งที่ 22, มหาวิทยาลัยเกษตรศาสตร์, กรุงเทพมหานคร, 2542.

วรรณรัช สันติอมรทัต, อภิเนตร อูนากูล และปัญจภา เรืองสินทรัพย์, " การออกแบบและพัฒนาไมโครโปรเซสเซอร์ 32 บิต." การประชุมวิชาการประจำปีวิทยาศาสตร์เทคโนโลยีกับการปรับโครงสร้างเศรษฐกิจไทย ณ. ศูนย์ประชุมสหประชาชาติ กรุงเทพมหานคร, หน้า. 100-107, 2542.

ภาษาอังกฤษ

Boussebha, N. Giambiasi and J. Magnier. Temporal Verification of Behavioral Descriptions in VHDL. In IEEE transaction Computer-Aided Design vol.3, 1992.

Chrysalis Symbolic Design. Replacing Functional Gate-level Simulation with Equivalence checking, white paper, Feburary, 1998.

D. L.Perry. VHDL. McGraw-Hill International Editions, second Edition, 1996.

E.M. Clarke and J.M. Wing. Formal Methods : State of the Art and Future Directions. In IEEE transactions on computers on Computer-Aided Design, April, 1996, pp. 46-50.

E. M. Clark , and R Kurshan. Computer-Aided Verification. In IEEE Specturm Vol. 33, No. 2, 1996, pp. 61-67.

F. Casaubieilh, A. McIsaac, M. Benjamin, M. Bartley, F. Pogodalla, F. Rocheteau, M. Belhadj, J. Eggletion, G. Mas, G. Barrett, and C. Berther. Functional Verification Methodology of Chameleon Processor. In Proceeding of the 33rd ACM/IEEE Design Automation Conference, 1996, pp. 273-278.

G. De Micheli. Synthesis and Optimization of Digital Circuits. McGraw-Hill International Editions, 1994.

G. Milne. Formal Specification and Verification of Digital Systems. School of Computer and Information Science, University of Sounth Australia, 1994. pp. 1-19.

H.Al Asaad, D.Van Campenhout, J.P.Hayes, T.Mudge and R.B. Brown. High Level Design Verification of microprocessors via Error modeling. In IEEE International High Level Design Validation and Test Workshop Oakland, California, November 14-15, 1997, pp.194-201.

I. Houston, And King. S. Experiences and results from using Z. In Formal Development Methods, vol. 551 of Springer Lecture notes in Computer Science, 1991, pp. 86-90.

I. Sommervile. Software Engineering. A Wiley-Interscience Publication., 1992.

J. Dave. Advanced RISC Machine Architectural Reference Manual. Prentice Hall, Englewood Clifs, N.J., 1996.

J. K. Huggins, and D. Van Campenhout. Specification and Verification of Pipelining in the ARM2 RISC Microprocessor. In IEEE International High Level Design Validation and Test Workshop Oakland, California, November 14-15, 1997, pp. 186-193.

J. Kljaich, B. Smith , and A . Woicik. Formal Verification of fault tolerance using theorem-proving techniques. In IEEE transactions on computers on Computer-Aided Design. Vol. 38, No.3, 1992, pp. 366-376.

J. L. Hennessy & Davis A Patterson. Computer Architecture A Quantitative Approach. Morgan Kaufmann Publishers Inc., 1992, pp250-343.

J. V. Oldfield and Richard C. Dorf. Field Programmable Gate Arrays reconfigurable logic for rapid prototyping and implementation of digital systems. A Wiley-Interscience Publication, 1995.

K. L. McMillan. Symbolic Model Checking. An Approach to the State Explosion Problem. Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1992.

M. Kentrowitz and L.M. Noack. Verification Coverage Analysis and Correctness Checking of the DECchip 21164 Alpha Microprocessor. In Proceeding of the 33rd ACM/IEEE Design Automation Conference, 1996, pp.325-330.

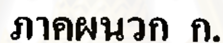
R. Lipsett, Carl F. schaefer and Cary Usser. VHDL : Hardware Description and Design. Kluwer Academic Publishers, 1996.

S. G. Shiva. Computer Design & Architecture. Morgan Kaufmann Publishers Inc., Second Edition, 1997.

Y. Gurevich. Logic and the challenge of computer science. In Current Trends in Theoretical Computer Science, E. Borger, Ed. Computer Science Press, 1988, pp. 1-57.

ภาคผนวก

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย



ภาคผนวก ก.

ไมโครโพรเซสเซอร์ที่ทำการออกแบบด้วยภาษา VHDL

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
-- func (from Control Unit)
-- 0000 -> AND          OUT_ALU = (OP1 AND OP2)
-- 0001 -> XOR (EOR)    OUT_ALU = (OP1 XOR OP2)
-- 0010 -> SUB          OUT_ALU = (OP1 - OP2)
-- 0011 -> RSB          OUT_ALU = (OP2 - OP1)
-- 0100 -> ADD          OUT_ALU = (OP1 + OP2)
-- 0101 -> ADC          OUT_ALU = (OP1 + OP2 + CARRY)
-- 0110 -> SBC          OUT_ALU = (OP1 - OP2 + CARRY - 1)
-- 0111 -> RSC          OUT_ALU = (OP2 - OP1 + CARRY - 1)
-- 1000 -> TST          SAME AND BUT NOT WRITE RESULT
-- 1001 -> TEQ          SAME EOR BUT NOT WRITE RESULT
-- 1010 -> CMP          SAME SUB BUT NOT WRITE RESULT
-- 1011 -> CMN          SAME ADD BUT NOT WRITE RESULT
-- 1100 -> ORR          OUT_ALU = (OP1 OR OP2)
-- 1101 -> MOV          OUT_ALU = OP2
-- 1110 -> BIC          OUT_ALU = OP1 AND (NOT OP2)
-- 1111 -> MVN          OUT_ALU = NOT OP2

-- alu_in1 from Register File
-- alu_in2 from    Shifter or Booth's Multiplier
-- sh_flag from Shifter (carry flag of Shifter)
-- alu_out -> result that finish do that operation
-- cflag  -> carry flag from alu
-- vflag  -> over flag from alu that control unit must use in to AND with old vflag

library IEEE;
use     IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

Entity alu is
        port(reset          : in std_logic;
                clk             : in std_logic;
                rclk            : in std_logic;
                set_fg          : in std_logic;
                sh_flag         : in std_logic;
```

```
            alu_rd          : in std_logic;
            alu_in1         : in std_logic_vector(31 downto 0);
            alu_in2         : in std_logic_vector(31 downto 0);
            alu_md          : in std_logic_vector(3 downto 0);
            cond            : in std_logic_vector(3 downto 0);
            cond_ok         : out std_logic;
            carry_f         : out std_logic;
            psw             : out std_logic_vector(3 downto 0);
            alu_out         : out std_logic_vector(31 downto 0));
end alu;

architecture rtl_alu of alu is
        signal in1,result : std_logic_vector(31 downto 0);
        signal over_f,cflag,vflag,cin,ok_cond,neg,carry,over,zero : std_logic;
        signal func : std_logic_vector(3 downto 0);
        signal ze : std_logic_vector(31 downto 0) := (others => '0');
        signal extra : std_logic;
        signal cf,vf,cary : std_logic;

begin
        -- Read ALU mode and ALU input1 into buffer (bus change always so must keep)
        -- Read in falling edge of clk (falling clk = rising rclk)
        READ_INPUT:
        Process (rclk)
        begin
                if rising_edge(rclk) then
                        func <= alu_md;
                        cin     <= carry;
                        if alu_rd = '1' then
                                in1 <= alu_in1;
                        cnd if;
                end if;
        end process READ_INPUT;
        -- ALU work
        DO_ALU:
        process (in1,alu_in2,func,cin)
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
        variable res        : std_logic_vector(32 downto 0);
        variable carry      : std_logic;
begin
        cf <= '0';
        vf <= '0';
        case func is
                when "0000" | "1000" =>
                -- AND | TST
                  res(31 downto 0) := in1 and alu_in2;
                  extra <= '0';

                when "0001" | "1001" =>
                -- XOR (EOR) | TEQ
                  extra <= '0';
                  res(31 downto 0) := in1 xor alu_in2;

                when "0010" | "0110" | "1010" =>
                -- SUB | SBC | CMP
                  extra <= '1';
                  carry := cin or (not func(2));
                  res       := ('0' & in1) + ('1' & not alu_in2) + carry;
                  cf <= res(32);
                  vf <= (in1(31) xor alu_in2(31)) and (in1(31) xor res(31));

                when "0100" | "0101" | "1011" =>
                -- ADD | ADC | CMN
                  extra <= '1';
                  carry := (func(2) and func(0)) and cin;
                  res       := ('0' & in1) + ('0' & alu_in2) + carry;
                  cf <= res(32);
                  vf <= (not (in1(31) xor alu_in2(31))) and (in1(31) xor res(31));


                when "1100" =>
                -- ORR
                  extra <= '0';
```

```
                res(31 downto 0) := in1 or alu_in2;

            when "0011" | "0111" =>
            -- RSB | RSC
                extra <= '1';
                carry := cin or (not func(2));
                res := ('0' & alu_in2) + ('1' & not in1) + carry;
                cf <= res(32);
                vf <= (in1(31) xor alu_in2(31)) and (alu_in2(31) xor res(31));

            when "1101" =>
            -- MOV
                extra <= '0';
                res(31 downto 0) := alu_in2;

            when "1110" =>
            -- BIC
                extra <= '0';
                res(31 downto 0) := in1 and (not alu_in2);

            when others =>
            -- MVN
                extra <= '0';
                res(31 downto 0) := not alu_in2;
        end case;
        alu_out   <= res(31 downto 0);
        result    <= res(31 downto 0);
end process DO_ALU;

-- Check Condition for set cond_ok signal
CHECK_COND:
Process (cond,carry,zero,neg,ovcr)
begin
        case cond is
            when "0000" => ok_cond    <= zero;
            when "0001" => ok_cond    <= not zero;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
when "0010" => ok_cond    <= carry;
when "0011" => ok_cond    <= not carry;
when "0100" => ok_cond    <= neg;
when "0101" => ok_cond    <= not neg;
when "0110" => ok_cond    <= over;
when "0111" => ok_cond    <= not over;
when "1000" =>
if (carry = '1' and zero = '0') then
            ok_cond <= '1';
      else
            ok_cond <= '0';
      end if;
when "1001" =>
if (carry = '0' and zero = '1') then
            ok_cond  <= '1';
      else
            ok_cond <= '0';
      end if;
when "1010" =>
      if ((neg = '1' and over = '1') or (neg = '0' and over = '0')) then
            ok_cond <= '1';
      else
            ok_cond  <= '0';
      end if;
when "1011" =>
if ((neg = '1' and over = '0') or (neg = '0' and over = '1')) then
            ok_cond  <= '1';
      else
            ok_cond  <= '0';
      end if;
when "1100" =>
if ((zero = '0') and ((neg = '1' and over = '1') or (neg = '0' and over = '0'))) then
            ok_cond <= '1';
      else
            ok_cond  <= '0';
      end if;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
when "1101" =>
if ((zero = '1') or ((neg = '1' and over = '0') or (neg = '0' and over = '1')))          then
                    ok_cond  <= '1';
          else
                    ok_cond  <= '0';
          end if;
when "1110" =>
          ok_cond  <= '1';
when others  => ok_cond <= '0';
end case;
end process CHECK_COND;
cond_ok <= ok_cond;
cflag <= cf when extra = '1' else sh_flag;
vflag <= vf when extra = '1' else over_f;
-- Set Cond_ok and Flag signal
SET_FLAG:
Process (reset,clk)
begin
          if (reset = '0') then
                    neg          <= '0';
                    carry        <= '0';
                    cary         <= '0';
                    over         <= '0';
                    zero         <= '0';
          elsif rising_edge(clk) then
                    if (set_fg = '1') then
                              neg   <= result(31);
                              carry <= cflag;
                              cary  <= cflag;
                              over  <= vflag;
                              if result = ze then
                                        zero <= '1';
                              else
                                        zero <= '0';
                              end if;
                    end if;
```

```
                end if;
        end process SET_FLAG;
        carry_f  <= carry;
        over_f   <= over;
        psw      <= neg & zero & cary & over;
end rtl_alu;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
library IEEE;
use IEEE.std_logic_1164.all;
Entity datamem is
        Port    (clk            : in std_logic;
                 rclk           : in std_logic;
                 mem_rd         : in std_logic;
                 noByte         : in std_logic_vector(1 downto 0);
                 data_wr        : in std_logic_vector(31 downto 0);
                 nBW            : in std_logic;
                 data_ot        : out std_logic_vector(31 downto 0);
                 instruc        : out std_logic_vector(31 downto 0);
                 mdat           : in std_logic_vector(31 downto 0);
                 dat            : out std_logic_vector(31 downto 0));
end datamem;

Architecture rtl_datamem of datamem is
        signal tmp,temp,tmpdat : std_logic_vector(31 downto 0);
begin
        -- Get input in falling edge of clk (falling clk = rising rclk)
        GETINPUT:
        Process (rclk)
        begin
                if rising_edge(rclk) then
                        if mem_rd = '1' then
                                tmp <= data_wr;
                        end if;
                end if;
        end process GETINPUT;

        -- Store instruction must store word aligned address
        SELECT_DATA_STORE:
        Process (nBW,tmp)
        begin
                if nBW = '0' then
                        tmpdat <= tmp;
                else
```

```
                tmpdat <= tmp(7 downto 0) & tmp(7 downto 0) & tmp(7 downto 0) & tmp(7 downto 0);
            end if;
    end process SELECT_DATA_STORE;
    -- Load instruction can load both not word aligned or word aligned address
    ARRANGE_DATA_LOAD:
    Process (nBW,noByte,mdat)
            variable t : std_logic_vector(31 downto 0);
    begin
            if (nBW = '0') then -- word
             case noByte is
                    when "00" => t := mdat;
                    when "01" => t := mdat(7 downto 0) & mdat(31 downto 8);
                    when "10" => t := mdat(15 downto 0) & mdat(31 downto 16);
                    when others => t := mdat(23 downto 0) & mdat(31 downto 24);
             end case;
            else
             case noByte is
                    when "00" => t(7 downto 0) := mdat(7 downto 0);
                    when "01" => t(7 downto 0) := mdat(15 downto 8);
                    when "10" => t(7 downto 0) := mdat(23 downto 16);
                    when others => t(7 downto 0) := mdat(31 downto 24);
             end case;
             t(31 downto 8) := (others => '0');
            end if;
            temp <= t;
    end process ARRANGE_DATA_LOAD;
            SEND_AND_RECEIVE:
    Process (clk)
    begin
            if rising_edge(clk) then
                    dat     <= tmpdat;
                    instruc <= mdat;
                    data_ot <= temp;
            end if;
    end Process SEND_AND_RECEIVE;
end rtl_datamem;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

Entity mar is
        Port    (reset          : in std_logic;
                 rclk           : in std_logic;
                 mar_in         : in std_logic_vector(31 downto 0);
                 mar_rd         : in std_logic;
                 mreq           : in std_logic;
                 seq            : in std_logic;
                 alu_use        : in std_logic;
                 not_inc        : in std_logic;
                 maddr          : out std_logic_vector(31 downto 0);
                 pc             : out std_logic_vector(29 downto 0);
                 st_de          : out std_logic);
end mar;

Architecture rtl_mar of mar is
        signal tpc,tmar : std_logic_vector(31 downto 0);

begin
        -- Calculate address in falling clk (falling clk = rising rclk)
  ADDRESS:
  process(reset,rclk)
                variable to_mar : std_logic_vector(31 downto 0);
  begin
        if (reset = '0') then
                        tmar    <= (others => '0');
                        tpc     <= (others => '0');
                        st_de   <= '0';
    elsif rising_edge(rclk) then
                        st_de   <= '1';
                        if mar_rd = '1' then
                                to_mar := mar_in;
```

```
                    end if;

                    if mreq = '0' then
                            if seq = '1' then
                if (not_inc = '1') then -- Load and Store multiple
          tmar  <=  tmar + "100";
        else
                tmar        <= tpc;
                tpc         <= tpc + "100";
        end if;
                else
                    if (alu_use = '1') then         -- instruction follow from store instruction
                                            tmar <= mar_in;
                                            if (not_inc = '0') then
                                                    tpc <= mar_in + "100";
                                            end if;
                                    elsif (not_inc = '0') then
                                            tmar <= tpc;
                                            tpc         <= tpc + "100";
                                    else
                                            tmar <= to_mar;
                                    end if;
                            end if;
                    end if;
    end if;
  end process ADDRESS;
        maddr <= tmar;
        pc <= tpc(31 downto 2);
end rtl_mar;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use     IEEE.std_logic_arith.all;

entity reg_file is
      port(rf_in1    : in std_logic_vector(3 downto 0);
          rf_in2     : in std_logic_vector(3 downto 0);
          rf_des     : in std_logic_vector(3 downto 0);
          rf_dataw   : in std_logic_vector(31 downto 0);
          clk        : in std_logic;
          rf_write   : in std_logic;
          rf_din1    : out std_logic_vector(31 downto 0);
          rf_din2    : out std_logic_vector(31 downto 0));
end reg_file;

architecture regf of reg_file is
   type  dat32 is array (15 downto 0) of std_logic_vector(31 downto 0);
         signal regs : dat32;
begin
   rf_din1 <= regs(CONV_INTEGER(rf_in1));
   rf_din2 <= regs(CONV_INTEGER(rf_in2));

   WRITE_REG:
   process(clk)
   begin
         if rising_edge(clk) then
                        if rf_write = '1' then
                                  regs(CONV_INTEGER(rf_des)) <= rf_dataw;
                        end if;
               end if;
   end process WRITE_REG;
end regf;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

Entity shifter is
        port(sh_in1             : in std_logic_vector(7 downto 0);
             sh_in2             : in std_logic_vector(31 downto 0);
             sh_in3             : in std_logic_vector(12 downto 0);
             sh_md              : in std_logic_vector(1 downto 0);
             op_regs  : in std_logic_vector(1 downto 0);
             rclk               : in std_logic;
             sh_rd              : in std_logic;
             carry_f  : in std_logic;
             sh_flag  : out std_logic;
             shift_out: out std_logic_vector(31 downto 0));
end shifter;

architecture rtl_shifter of shifter is

    signal idx                  : natural range 0 to 31;
    signal dshift       : std_logic_vector(95 downto 0);
    signal zero,ovf,eq32        : std_logic;
    signal in2,din,result       : std_logic_vector(31 downto 0);
    signal in1,idx_vec  : std_logic_vector(7 downto 0);
    signal regs,func    : std_logic_vector(1 downto 0);
    signal in3 : std_logic_vector(12 downto 0);

begin

    din(31 downto 8) <=     (others=>'0') when regs = "10" else in2(31 downto 8);
    din(7 downto 0) <= in3(7 downto 0)  when regs = "10" else in2(7 downto 0);

    dshift(95 downto 64) <=  din(31 downto 0) when func = "11" else (others=>'0');
    dshift(63 downto 32) <=  din(31 downto 0);
    dshift(31 downto 0)      <=     (others=>'0');
```

```
idx_vec  <=     in1 when regs = "00" else "000" & in3(12 downto 8);
ovf      <=     '1' when idx_vec > "00011111" else '0';
eq32     <=     '1' when idx_vec = "00100000" else '0';
zero     <=     '1' when idx_vec = "00000000" else '0';
idx      <=     conv_integer(idx_vec(4 downto 0));

INPUT_SHIFT:
Process (rclk)
begin
        if rising_edge(rclk) then
                func <= sh_md;
                regs <= op_regs;
                in3  <= sh_in3;
                if (sh_rd = '1') then
                in1 <= sh_in1;
                in2 <= sh_in2;
                end if;
        end if;
end process INPUT_SHIFT;

SHIFT_PROC:
process(idx,dshift,func,ovf)
begin
        for i in 0 to 31 loop
                case func is
                        when "00" => -- Logical shift left
                                if ovf = '0' then
                                        result(i) <= dshift(32-idx+i);
                                else
                                        result(i) <= '0';
                                end if;
                        when "01" => -- Logical shift right
                                if ovf = '0' then
                                        result(i) <= dshift(32+idx+i);
                                else
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
                                        result(i) <= '0';
                              end if;
                    when "10" => -- Arithmetic shift right
                              if ovf = '1' then
                                        result(i) <= dshift(63);
                              else
                                        if i < 32-idx then
                                                  result(i) <= dshift(32+idx+i);
                                        else
                                                  result(i) <= dshift(63);
                                        end if;
                              end if;
                    when others =>
                              result(i) <= dshift(32+idx+i);
          end case;
     end loop;
end process SHIFT_PROC;

shift_out <= result(31 downto 0);

FLAG_PROC:
process(dshift(63 downto 32),carry_f,func,ovf,eq32,idx,zero)
begin
     if zero = '1' then
          sh_flag <= carry_f;
     else
     case func is
          when "00"         =>
                    if ovf = '0' then
                              sh_flag <= dshift(64-idx);
                    elsif eq32 = '1' then
                              sh_flag <= dshift(32);
                    else
                              sh_flag <= '0';
                    end if;
          when "01"         =>
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
                    if ovf = '0' then
                            sh_flag <= dshift(31+idx);
                    elsif eq32 = '1' then
                            sh_flag <= dshift(63);
                    else
                            sh_flag <= '0';
                    end if;
                when "10"       =>
                    if ovf = '0' then
                            sh_flag <= dshift(31+idx);
                    else
                            sh_flag <= dshift(63);
                    end if;
                when others     =>
                    sh_flag <= dshift(63+idx);
            end case;
            end if;
        end process FLAG_PROC;

end rtl_shifter;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

Entity ctl is
  Port        (reset        : in std_logic;
              clk           : in std_logic;
              rclk          : in std_logic;
              instruc       : in std_logic_vector(31 downto 0);
              done          : in std_logic;
              cond_ok       : in std_logic;
              st_de         : in std_logic;
              -------- Control Data flow to Bus ----------
              mem_ot        : out std_logic;
              mar_rd        : out std_logic;
              forw1         : out std_logic;
              forw2         : out std_logic;
              pc1           : out std_logic;
              pc2           : out std_logic;
              reg1          : out std_logic;
              reg2          : out std_logic;
              use_of        : out std_logic;
              ----- Control each Entity read Operand -----
              alu_pc        : out std_logic;
              pc_rg         : out std_logic;
              alu_rd        : out std_logic;
              sh_rd         : out std_logic;
              mem_rd        : out std_logic;

              ----- Control Function of each Entity ------
              rf_write      : out std_logic;
              rf_in1        : out std_logic_vector(3 downto 0);
              rf_in2        : out std_logic_vector(3 downto 0);
              rf_des        : out std_logic_vector(3 downto 0);
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
                alu_md          : out std_logic_vector(3 downto 0);
                cond            : out std_logic_vector(3 downto 0);
                set_fg          : out std_logic;
                 sh_md          : out std_logic_vector(1 downto 0);
                op_regs         : out std_logic_vector(1 downto 0);
                sh_out          : out std_logic;
                start           : out std_logic;
                 not_inc        : out std_logic;
                alu_use         : out std_logic;
                fr_ctl          : out std_logic_vector(25 downto 0);
                sh_in3          : out std_logic_vector(12 downto 0);
                nENOUT          : out std_logic;
               lock             : out std_logic;
               nRW    : out std_logic;
               nBW    : out std_logic;
                mreq            : out std_logic;
                seq             : out std_logic);
end ctl;

architecture rtl_ctl of ctl is

-------- Decode Process ----------------------------------
        signal reg_no,inst  : std_logic_vector(4 downto 0);
        signal temp_rf,dest : std_logic_vector(3 downto 0);
        signal pt_inx,shf,nByte,work,inc,link  : std_logic;
        signal wr_back,fg_set,not_wr : std_logic;
        signal top1,top2    : std_logic_vector(3 downto 0);
-------- Execute Process ---------------------------------
        signal cycle : std_logic_vector(4 downto 0);
        signal rf,dest_rf     : std_logic_vector(3 downto 0);
        signal tp_inst        : std_logic_vector(31 downto 0);
        signal pc_reg,not_get,wr_regs,wr_mem,meq,sc  : std_logic;
        signal bypass,Byte,from_tp,can_de,out_mem,new_adr,use_off : std_logic;
-------- Others ------------------------------------------
        signal fin_mul       : std_logic;
        signal fw1,fw2,rg1,rg2,p1,p2 : std_logic;
```

```
--        signal fyes : std_logic; -- use to tell in 2nd cycle of decode not do forwarding
--        signal D_ir : std_logic_vector(31 downto 0);
--        signal nd : std_logic;

begin
        mreq            <= meq;
        seq             <= sc;
        nRW             <= wr_mem and clk;
        nENOUT          <= wr_mem;
        nBW             <= Byte;
        rf_write        <= wr_regs;
        start           <= work;
        fin_mul         <= done when inst = "10001" else '0';
        pc_rg           <= pc_reg;
        use_of          <= use_off;
------------------------------------------------------------------------------------------
-- cstate = N           have ir <= instruction 'coz last cycle of Store inst. is N then
--(mreq,se = 00)        new inst. must start with N cycle not S cycle so last exec. of Store
--                      must set not_get = '0' and from_tp = '1'
---------------- SIGNAL FROM EXECUTE PROCESS. ---------------------
-- not_get              get new instruction or not
-- from_tp              get new instruction from tp_inst or memory

----------------- VARIABLE CONTROL ------------------------------
-- not_dc               next cycle will can decode new instruction or not(Must decide with can_de and
--                      fin_mul that send from Execute process.
-- up                   increment or substraction

------------------ SIGNAL CONTROL ------------------------------
-- mar_rd               '1' when must use address from Reg (Post Index) by keep it in temp
--                      and use it in next cycle.
-- mem_rd               '1' when must send value to memory (swap or store)
-- sh_rd                '1' when must use shifter
-- alu_rd               '1' when must use alu
-- use_off              '1' when must use offset value to branch how from base (Another will not send
value into Bus.)
```

```
-- shf            exec. that inst. not have shift. use only data processing exec.
-- not_wr         exec. that inst. is about compare that not write result to register
-- pt_inx         exec. that is post indexing
-------------------------------------------------------------------------------------

        -- Decode in falling clk (falling clk = rising rclk)

 DECODE_1:
 process(reset,rclk,st_de)
                variable count      : std_logic_vector(4 downto 0);
        variable ir          : std_logic_vector(31 downto 0);
                variable list        : std_logic_vector(15 downto 0);
                variable off_reg: std_logic_vector(11 downto 0);
   variable temp,tp_md : std_logic_vector(3 downto 0);
                variable tp_byte,mul_acc,ioffset,ld_st,up,not_de : std_logic;
                variable op1,op2 : std_logic_vector(3 downto 0);
 begin
        if (reset = '0' or st_de = '0') then
                        not_de   := '0';
                        ioffset  := '0';
                        up       := '0';
                        mul_acc  := '0';
                        ld_st    := '0';
                        tp_byte  := '0';
                        count    := "00000";
                        temp     := "0000";
                        tp_md    := "1101";
                        ir       := (others => '0');
                        off_reg  := (others => '0');
                        list     := (others => '0');
                        op1      := "1111";
                        op2      := "1111";
                        work     <= '1';
                        pt_inx   <= '0';
                        shf      <= '0';
                        nByte    <= '0';
                        link     <= '0';
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
				not_wr    <= '0';
				wr_back <= '0';
				fg_set    <= '0';
				inc         <= '0';
				dest        <= "0000";
				temp_rf  <= "0000";
				reg_no    <= "00000";
				inst        <= "11111";
				alu_rd     <= '1';
				sh_rd      <= '1';
				mem_rd   <= '0';
				mar_rd    <= '0';
				sh_in3     <= (others => '0');
				fr_ctl      <= (others => '0');
				cond        <= "1110";
				alu_md    <= "1101";
				sh_md      <= "00";
				op_regs   <= "01";
				use_off    <= '0';
				alu_pc      <= '0';
elsif rising_edge(rclk) then

				if (not_get = '0') then
					if (from_tp = '0') then
						ir := instruc;
					else
						ir := tp_inst;
					end if;
				end if;
				if (cond_ok = '0') then
					not_de := '0';
				end if;
				shf <= '0';
				if (can_de = '1' or not_de = '0' or fin_mul = '1') then
					cond        <= ir(31 downto 28);
case ir(27 downto 26) is
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
when "00" =>
use_off <= '0';
if (((ir(4) and ir(7)) = '0') or ir(25) = '1') then
    -------------------- Data Processing ----------------
    work     <= '0';
    tp_md    := ir(24 downto 21);
    inst     <= '0' & ir(24 downto 21);
    fg_set   <= ir(20);
    sh_rd    <= '1';
    mem_rd   <= '0';
    mar_rd   <= '0';
    dest     <= ir(15 downto 12);
    alu_md   <= ir(24 downto 21);
    if ir(24 downto 23) = "10" then
                                          -- TST,TEQ,CMP,CMN
        not_wr <= '1';
    else
        not_wr <= '0';
    end if;
    if (ir(25) = '0') then
        ------ Register Operand --------------
        sh_md  <= ir(6 downto 5);
        temp_rf<= ir(19 downto 16);
        op2        := ir(3 downto 0);
    if (ir(4) = '1') then
        -- shift amount from register
        op_regs <= "00";
        sh_rd   <= '1';
        op1        := ir(11 downto 8); -- amount shift
        shf        <= '1';
        not_de     := '1';
        alu_rd     <= '0';
    else
    -- shift amount is imm. value
        op1        := ir(19 downto 16);
        op_regs <= "01";
```

```
                sh_in3(12 downto 8) <= ir(11 downto 7);
                        not_de      := '0';
                        alu_rd      <= '1';
                end if;
        else
        ------ Immediate Operand -------------
                sh_in3  <= ir(11 downto 8) & '0' & ir(7 downto 0);
                                                op_regs <= "10";
                                                sh_md   <= "11";
                                                op1     := ir(19 downto 16);
                                                alu_rd  <= '1';
                                        if ir(15 downto 12) = "1111" then
                                                not_de := '1';
                                        else
                                                not_de := '0';
                                        end if;
        end if;
else
        not_de  := '1';
 if (ir(24) = '0') then
---------------------- Multiply ----------------------
        op1     := ir(3 downto 0);
        op2     := ir(11 downto 8);
        inst    <= "10001";
        dest    <= ir(19 downto 16);
        fg_set <= ir(20);
        work    <= '1';
        alu_rd  <= '0';
        sh_rd   <= '0';
        mem_rd  <= '0';
        mar_rd  <= '0';
        sh_md   <= "00";
        op_regs <= "01";
        sh_in3(12 downto 8) <= "00000";
  if (ir(21) = '1') then
        temp_rf <= ir(15 downto 12);
```

```
                    mul_acc := '1';
            else
                    mul_acc  := '0';
            end if;
        else
----------------------- Data Swap ---------------------
                    work      <= '0';
                    nByte     <= ir(22);
                    fg_set    <= '0';
                    dest      <= ir(15 downto 12);
                    temp_rf  <= ir(3 downto 0);
                    inst      <= "10110";
                    alu_md  <= "1101";
                    op_regs <= "01";
                    sh_md    <= "00";
                    sh_rd     <= '1';
                    alu_rd    <= '1';
                    mem_rd  <= '0';
                    mar_rd   <= '0';
                    op2        := ir(19 downto 16);
                    sh_in3(12 downto 8) <= "00000";
        end if;
        end if;

  when "01" =>
----------------------- Single Data Transfer ----------
                                        work      <= '0';
                                        not_de    := '1';
                                        up                := ir(23);
                                        ioffset   := ir(25);
                                        off_reg   := ir(11 downto 0);
                                        tp_byte   := ir(22);
                                        wr_back <= ir(21);
                                        temp_rf  <= ir(19 downto 16);
```

```
-- Set operand that post-index and pre-index difference..
-- ld_st set when want it to check offset
        if (ir(24) = '0') then
                -- post indexing
                pt_inx    <= '1';
                op_regs  <= "01";
                sh_md     <= "00";
                sh_in3(12 downto 8)<= "00000";
                nByte     <= ir(22);
                use_off <= '0';
        if (ir(20) = '0') then
                        -- Store
                        op2        := ir(15 downto 12);
                        op1        := ir(19 downto 16);
                        mem_rd  <= '1';
                        mar_rd   <= '1';
                        alu_rd    <= '0';
                        sh_rd     <= '0';
        else
                        op2        := ir(19 downto 16);
                        mem_rd  <= '0';
                        alu_rd    <= '1';
                        sh_rd     <= '1';
                        mar_rd   <= '0';
                        alu_md   <= "1101";
                end if;
        else
                -- pre indexing
                ld_st       := '1';
                pt_inx     <= '0';
                op1         := ir(19 downto 16);
                mem_rd  <= '0';
                mar_rd    <= '0';
                if (up = '0') then
                        alu_md    <= "0010";
                else
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
                                        alu_md   <= "0100";
                                end if;
                        end if;

                        if (ir(20) = '1') then
                                -- Load
                                inst        <= "10010";
                                dest        <= ir(15 downto 12);
                        else
                                -- Store
                                inst        <= "10011";
                                temp        := ir(15 downto 12);
                        end if;

when "10" =>
                        not_dc    := '1';
                        fg_set    <= '0';
                        work      <= '0';
            if (ir(25) = '0') then
---------------------- Block Data Transfer -----------
                -- use sh_in3(4 downto 0) for calculate distance
                                list        := ir(15 downto 0);
                                op1         := ir(19 downto 16);
                                wr_back     <= ir(21);
                                temp_rf     <= ir(19 downto 16);
                                pt_inx      <= not(ir(24));
                                use_off     <= '0';
                                sh_rd       <= '0';
                                mem_rd      <= '0';
                                alu_rd      <= '1';
                                inc         <= ir(23);
                                count       := "00000";
                                for i in 0 to 15 loop
                                        if (list(i) = '1') then
                                                count := count + "1";
                                        end if;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
end loop;
reg_no <= count;
if (ir(23) = '0') then
        -- Decrement
        alu_md   <= "0010";
        mar_rd   <= '0';
        op_regs  <= "10";
        sh_md    <= "00";
sh_in3(12 downto 5) <= "00010000";
if (ir(24) = '0') then
                -- Post Index
sh_in3(4 downto 0) <= count - "1";
        else
                -- Pre Index
sh_in3(4 downto 0) <= count;
end if;
else
        -- Increment
        alu_md   <= "0100";
if (ir(24) = '0') then
                -- Post Index
                mar_rd   <= '1';
else
                -- Pre Index
                mar_rd   <= '0';
sh_in3(12 downto 0) <= "0001000000001";
            op_regs     <= "10";
            sh_md       <= "00";
        end if;
end if;
if (ir(20) = '1') then
        -- Load
        inst        <= "10100";
else
        -- Store
        inst        <= "10101";
```

```
                                        end if;
                                else
---------------------- Branch -----------------------
                                        link     <= ir(24);
                                        inst     <= "10000";
                                        op_regs  <= "01";
                                        sh_md    <= "00";
                                        alu_md   <= "0100";
                                        op1      := "1111";
                                        use_off  <= '1';
                                        alu_rd   <= '1';
                                        sh_rd    <= '1';
                                        mar_rd   <= '0';
                                        mem_rd   <= '0';
                                        fr_ctl   <= ir(23 downto 0) & "00";
                                        alu_pc   <= ir(24);
                end if;
        when others =>
                                work     <= '0';
                                sh_rd    <= '0';
                                alu_rd   <= '0';
                                mar_rd   <= '0';
                                mem_rd   <= '0';
                                use_off  <= '0';
                                inst     <= "11111";
                        end case;
---------------------- Not First Decode ----------------------
        else
                        alu_pc   <= '0';
        if (inst = "10010" or inst = "10011") then
                -- 2nd Decode of Load-Store
                        if (pt_inx = '1') then
                                ld_st    := '1';
                                pt_inx   <= '0';
                                op1      := temp_rf;
                                mar_rd   <= '0';
```

```
                mem_rd  <= '0';
                if (up = '0') then
                        alu_md <= "0010";
                else
                        alu_md <= "0100";
                end if;
        else
                nByte   <= tp_byte;
                use_off  <= '0';
                if (inst = "10011") then
                        op2        := temp;
                        mem_rd  <= '1';
                        mar_rd  <= '0';
                        alu_rd  <= '0';
                        sh_rd      <= '0';
                else
                        alu_rd     <= '0';
                        sh_rd      <= '0';
                end if;
        end if;
elsif (inst = "10100" or inst = "10101") then
        -- Load or Store Multiple
        if (cycle = "00000") then
                sh_in3(12 downto 5)          <= "00010000";
                sh_in3(4 downto 0)           <= count;
                op_regs  <= "10";
                sh_md    <= "00";
                alu_rd   <= '0';
                sh_rd    <= '0';
                mar_rd   <= '0';
        end if;
        for i in 0 to 15 loop
                if (list(i) = '1') then
                temp := CONV_STD_LOGIC_VECTOR(i,4);
                        reg_no <= reg_no - "1";
                        list(i) := '0';
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
                                        exit;
                                end if;
                        end loop;
                        if (inst = "10100") then -- Load
                                dest     <= temp;
                                mem_rd  <= '0';
                        else -- Store
                                op2      := temp;
                                mem_rd  <= '1';
                        end if;
                else
                 use_off <= '0';
                 if (inst = "10110") then
                        -- Swap inst. must call reg_file to send value to store to mem.
        if (cycle = "00010") then
                                op2      := temp_rf;
                                mem_rd  <= '1';
                                alu_rd   <= '0';
                                sh_rd    <= '0';
                                mar_rd   <= '0';
 end if;

                elsif shf = '1' then
                -- Data Processing that shift amount is reg. so must call reg_file
                -- to send value to alu..(2 operand use for shifter)
                        shf      <= '1';
                        alu_rd   <= '1';
                        sh_rd    <= '0';
                         alu_md  <= tp_md;
                        op1      := temp_rf;
elsif (inst = "10001") then
                        -- Multiply
                        work <= '0';
                        if (mul_acc = '1') then
                        -- Multiply Accumulate that add result with register value.
        alu_md   <= "0100";
```

```
                                        alu_rd    <= '1';
                                        op1       := temp_rf;
                                else
-- Multiply not Accumulate so alu only move result to alu_out
                                        alu_md   <= "1101";
                                end if;
                        else
-- not decode new instruction and no instruction that must call reg_file more that one time
                                alu_rd    <= '0';
                                sh_rd     <= '0';
                                mar_rd    <= '0';
                                mem_rd    <= '0';
    end if;
                        end if;
end if;

                -- bypass tell to send signal to bypass value to shifter and tell alu to move
                -- bypass set only when not have to decode new instruction.
  if (bypass = '1') then
                        sh_md     <= "00";
                        op_regs <= "01";
                        alu_md   <= "1101";
                        sh_rd     <= '1';
                        alu_rd    <= '0';
                        mar_rd   <= '0';
                        mem_rd  <= '0';
                        use_off <= '0';
                        sh_in3(12 downto 8) <= "00000";
                end if;

                -- ld_st tell that use below only load/store instruction
                -- set offset value can use both pre-indexing and post-indexing
                if (ld_st = '1') then
                        ld_st      := '0';
                        sh_rd     <= '1';
                        alu_rd    <= '1';
```

```
                    if (ioffset = '0') then
                              -- offset is imm. value
                              use_off <= '1';
                              fr_ctl(25 downto 12) <= (others => '0');
                              fr_ctl(11 downto 0)  <=       off_reg;
                              op_regs  <= "01";
                              sh_md    <= "00";
                    else
                              -- offset is register
                              use_off <= '0';
                              sh_md    <= off_reg(6 downto 5);
                              op2      := off_reg(3 downto 0);
                              op_regs  <= "01";
                              sh_in3(12 downto 8) <= off_reg(11 downto 7);
                    end if;
               end if;
  end if;
          rf_in1    <= op1;
          rf_in2    <= op2;
          top1      <= op1;
          top2      <= op2;
 end process DECODE_1;

-------------------------------------------------------------------------------
-- forw1 send output from alu when is forwarding or reg. value is address
-- reg1  send reg. value into bus when R0-R14 is operand
-- pc1   send pc value into bus when R15 is operand

     SET_SIGNAL_BUS1:
     Process (top1,dest_rf,new_adr,wr_regs)
          variable nforw : std_logic;
     begin

          if wr_regs = '1' then
               if top1 = dest_rf then
                    nforw := '0';
```

```
                else
                        nforw := '1';
                end if;
        else
                nforw := '1';
        end if;

        if (top1 = "1111" and new_adr = '0') then
                fw1     <= '0';
                rg1     <= '0';
                p1      <= '1';
        elsif (nforw = '0' or new_adr = '1') then
                fw1 <= '1';
                rg1     <= '0';
                p1      <= '0';
        else
                fw1     <= '0';
                rg1     <= '1';
                p1      <= '0';
        end if;
end process SET_SIGNAL_BUS1;

CHECKING_BUS1:
Process (reset,clk)
begin
        if reset = '0' then
                forw1   <= '0';
                reg1    <= '0';
                pc1     <= '1';
        elsif rising_edge(clk) then
                forw1   <= fw1;
                reg1    <= rg1;
                pc1     <= p1;
        end if;
end process CHECKING_BUS1;
--------------------------------------------------------------------------------
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
-- forw2  send output from alu when is forwarding or reg. value is address
-- reg2   send reg. value into bus when R0-R14 is operand
-- pc2    send pc value into bus when R15 is operand
-- mem_ot         send data that read from memory into bus when is Loading

    SET_SIGNAL_BUS2:
    Process (top2,dest_rf,use_off,out_mem,wr_regs)
        variable nforw : std_logic;
    begin
        if wr_regs = '1' then
            if top2 = dest_rf then
                nforw := '0';
            else
                nforw := '1';
            end if;
        else
            nforw := '1';
        end if;

        if (use_off or out_mem) = '1' then
            fw2     <= '0';
            rg2     <= '0';
            p2      <= '0';
        else
            if top2 = "1111" then
                fw2     <= '0';
                rg2     <= '0';
                p2      <= '1';
            elsif nforw = '0' then
                fw2     <= '1';
                rg2     <= '0';
                p2      <= '0';
            else
                fw2     <= '0';
                rg2     <= '1';
                p2      <= '0';
```

```
                end if;
        end if;
end process SET_SIGNAL_BUS2;

CHECKING_BUS2:
Process (reset,clk)
begin
        if reset = '0' then
                forw2   <= '0';
                reg2    <= '0';
                pc2     <= '1';
                mem_ot  <= '0';
        elsif rising_edge(clk) then
                forw2   <= fw2;
                reg2    <= rg2;
                pc2     <= p2;
                mem_ot  <= out_mem;
        end if;
end process CHECKING_BUS2;
--------------------------------------------------------------------------------
-- inst value is
--      "00000" - "01111" Data Processing
--              "10000"         Branch and Branch with Link
--              "10001"         Multiply and Multiply Accumulate
--              "10010"         Load
--              "10011"         Store
--              "10100"         Load Multiple
--              "10101"         Store Multiple
--              "10110"         Swap

----------------- SIGNAL FROM DECODE PROCESS --------------
-- link         set from Decode process tell Branch with Link instruction Exec.
-- dest         destination register to keep output
-- wr_regs      '1' tell to write Register (and with ph)

----------------- SIGNAL CONTROL --------------------------
```

```
-- bypass Decode. tell to send signal control to tell alu,shifter pass value
--                              it use only instruction Load or Swap..
-- not_inc          Mar. to not increment pc
-- alu_use          Mar. to use alu_out to mar..(N must use 2 sig, S use only not_inc)

        EXECUTE1:
        process (reset,rclk)
                variable cyc          : std_logic_vector(4 downto 0);
                variable check,chk : std_logic;
        begin
                if (reset = '0') then
                        chk        := '0';
                        check      := '0';
                        cyc        := "00000";
                        not_inc   <= '0';
                        alu_use   <= '0';
                        pc_reg    <= '0';
                        not_get   <= '0';
                        wr_regs <= '0';
                        wr_mem  <= '0';
                        meq        <= '0';
                        sc          <= '1';
                        can_de    <= '0';
                        lock       <= '0';
                        bypass    <= '0';
                        Byte       <= '0';
                        set_fg     <= '0';
                        from_tp   <= '0';
                        dest_rf    <= "1111";
                        rf           <= "0000";
                        tp_inst    <= (others => '0');
                        sh_out     <= '0';
                        out_mem<= '0';
                        new_adr  <= '0';
    elsif rising_edge(rclk) then
                if (cond_ok = '1') then
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
if (inst(4) = '0') then
        ------------- Data Processing -------------
                        Byte     <= '0';
                        wr_mem <= '0';
                        lock     <= '0';
                        bypass   <= '0';
                        from_tp  <= '0';
                        dest_rf  <= dest;
                        sh_out   <= '0';
                        out_mem<= '0';
                if shf = '0' then
                            -- No shift
                        set_fg   <= fg_set;
                        not_get  <= '0';
        if (cyc= "00000") then
                        alu_use  <= '1';
                        not_inc  <= '0';
                        can_de   <= '0';
                        meq               <= '0';
        if dest = "1111" then
                        se       <= '0';
                        new_adr <= '1';
        else
                        new_adr <= '0';
                        se       <= '1';
                        cyc      := "11111";
        end if;
                        wr_regs  <= ('1' xor not_wr);
elsif (cyc= "00001") then
                        new_adr  <= '0';
                        wr_regs  <= '0';
                        meq      <= '0';
                        se       <= '1';
                        can_de   <= '1';
                        set_fg   <= '0';
  elsif (cyc= "00010") then
```

```
                    can_de   <= '0';
                    cyc      := "11111";
          end if;
                    else
                             -- have shift
                    if (cyc= "00000") then
                                   wr_regs   <= '0';
                                   set_fg    <= '0';
                                   meq       <= '1';
                                   se        <= '0';
                                   if (dest = "1111") then
                                             not_get   <= '0';
                                             can_de    <= '0';
                                   else
                                             not_get   <= '1';
                                             can_de    <= '1';
                                   end if;
                         elsif (cyc= "00001") then
                                   set_fg    <= fg_set;
                                   alu_use   <= '1';
                                   meq       <= '0';
                                   wr_regs   <= ('1' xor not_wr);
                         if (dest = "1111") then
                                   se        <= '0';
                                   not_inc   <= '1';
                                   new_adr   <= '1';
                         else
                                   new_adr   <= '0';
                                   not_inc   <= '0';
                                   se        <= '1';
                                   not_get   <= '0';
                                   can_de    <= '0';
                                   cyc       := "11111";
                         end if;
                         elsif (cyc= "00010") then
                                   new_adr   <= '0';
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
                        wr_regs <= '0';
                        not_inc  <= '0';
                        meq      <= '0';
                        se       <= '1';
                        can_de   <= '1';
                    elsif (cyc= "00011") then
                        can_de   <= '0';
                        cyc      := "11111";
                    end if;
                end if;
            else
        case inst(2 downto 0) is
            when "000" =>
                ---------- Branch -----------
                        Byte     <= '0';
                        set_fg   <= '0';
                        wr_mem   <= '0';
                        not_get  <= '0';
                        from_tp  <= '0';
                        lock     <= '0';
                        bypass   <= '0';
                        dest_rf  <= dest;
                        sh_out   <= '0';
                        out_mem  <= '0';
            if (cyc= "00000") then
                                -- Branch with Link must keep pc into R14
                        alu_use  <= '1';
                        not_inc  <= '0';
                        meq      <= '0';
                        se       <= '0';
                        can_de   <= '0';
                        wr_regs  <= '0';
                        new_adr  <= '1';
    elsif (cyc= "00001") then
                        pc_reg   <= link;
                        wr_regs  <= link;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
                                        new_adr <= '0';
                                        meq     <= '0';
                                        se      <= '1';
                                        can_de  <= '1';
    elsif (cyc= "00010") then
                                        pc_reg  <= '0';
                                        wr_regs <= '0';
                                        meq     <= '0';
                                        se      <= '1';
                                        can_de  <= '0';
                                        cyc     := "11111";
    end if;
when "001" =>
---------- Multiply (not perfectly) ---------
                                        Byte    <= '0';
                                        wr_mem<= '0';
                                        bypass  <= '0';
                                        lock    <= '0';
                                        from_tp <= '0';
                                        dest_rf <= dest;
                                        not_inc <= '0';
                                        out_mem<= '0';
                                        new_adr <= '0';
                               if (cyc= "00000") then
                                        set_fg  <= '0';
                                        wr_regs <= '0';
                                        meq     <= '1';
                                        sc      <= '0';
                                        can_de  <= '0';
                                        not_get <= '1';
                                        sh_out  <= '0';
                               else
                               if (fin_mul = '1') then
                                        sh_out <= '1';
                                        meq     <= '0';
                                        sc      <= '1';
```

```
                    not_inc   <= '0';
                    not_get   <= '0';
                    wr_regs   <= '1';
                    set_fg    <= fg_set;
                    cyc       := "11111";
            end if;
                    end if;
    when "010" =>
        ---------- Load -------------
                    set_fg    <= '0';
                    wr_mem    <= '0';
                    lock      <= '0';
                    sh_out    <= '0';


if (cyc= "00000") then
                    Byte      <= '0';
                    bypass    <= '1';
                    wr_regs <= '0';
                    not_get   <= '1';
                    meq       <= '0';
                    se        <= '0';
                    can_de    <= '0';
                    alu_use   <= '1';
                    not_inc   <= '1';
                    new_adr <= '1';
elsif (cyc= "00001") then
                    new_adr <= '0';
                    Byte      <= nByte;
                    bypass    <= '0';
                    meq       <= '1';
                    se        <= '0';
                    out_mem<= '1';
        if (wr_back = '1') then
                    wr_regs   <= '1';
                    dest_rf   <= temp_rf;
```

```
                        end if;
                                    if (dest /= "1111") then
                                                tp_inst <= instruc;
                                                can_de   <= '1';
                                    end if;
elsif (cyc= "00010") then
                                                Byte      <= '0';
                                                meq       <= '0';
                                                not_inc   <= '0';
                                                wr_regs <= '1';
                                                dest_rf   <= dest;
                                                out_mem<= '0';
            if (dest = "1111") then
                                                se   <= '0';
                                                new_adr <= '1';
            else
                                                not_get   <= '0';
                                                from_tp   <= '1';
                                                se        <= '1';
                                                can_de    <= '0';
                                                cyc       := "11111";
            end if;
    elsif (cyc= "00011") then
                                                new_adr  <= '0';
                                                wr_regs <= '0';
                                                meq       <= '0';
                                                se        <= '1';
                                                not_get   <= '0';
                                                from_tp   <= '0';
                                                can_de    <= '1';
elsif (cyc= "00100") then
                                                can_de    <= '0';
                                                cyc       := "11111";
end if;
    when "011" =>
                        ---------- Store ------------
```

```
                    set_fg      <= '0';
                    lock        <= '0';
                    bypass      <= '0';
                    sh_out      <= '0';

        if (cyc= "00000") then
                    Byte        <= '0';
                    wr_regs <= '0';
                    not_get     <= '1';
                    wr_mem <= '0';
                    meq         <= '0';
                    se          <= '0';
                    not_inc     <= '1';
                    can_de      <= '1';
                    alu_use     <= not pt_inx;
                    new_adr <= not pt_inx;
        elsif (cyc= "00001") then
                    new_adr <= '0';
                    Byte        <= nByte;
                    tp_inst <= instruc;
                    not_get     <= '0';
                    from_tp     <= '1';
                    wr_mem<= '1';
                    can_de      <= '0';
                    not_inc     <= '0';
                    alu_use     <= '0';
                    cyc         := "11111";
            if (wr_back = '1') then
                    wr_regs <= '1';
                    dest_rf     <= temp_rf;
            end if;
    end if;

      when "100" =>
---------- Load Multiple ----
                    set_fg      <= '0';
```

```
                              Byte     <= '0';
                              wr_mem <= '0';
                              lock     <= '0';
                              rf       <= dest;
                              sh_out   <= '0';
                              if (check = '1') then
                  -- do this cycles when last register is PC
                  if (cyc= "00000") then
                        new_adr <= '0';
                        wr_regs <= '0';
                        meq      <= '0';
                        se       <= '1';
                        not_get  <= '0';
                        from_tp  <= '0';
                        can_de   <= '1';
                  else
                        can_de   <= '0';
                        cyc      := "11111";
                        check := '0';
                  end if;
                  elsif (cyc = "00000") then
                        new_adr <= '1';
                        bypass   <= '1';
                        wr_regs <= '0';
                        not_get  <= '1';
                        meq      <= '0';
                        se       <= '0';
                        can_de   <= '0';
                        not_inc  <= '1';
                        alu_use  <= not (pt_inx and inc);
            elsif (cyc= "00001") then
                        out_mem <= '1';
                        new_adr <= '0';
                        tp_inst <= instruc;
                  if (wr_back = '1') then
                        wr_regs  <= '1';
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
            dest_rf    <= temp_rf;
end if;
if (reg_no = "00000") then
            bypass <= '0';
            meq        <= '1';
            se          <= '0';
  if (dest /= "1111") then
                        can_de    <= '1';
  end if;
                        chk := '1';
else
                        meq <= '0';
                        se <= '1';
end if;
elsif (reg_no = "00000") then
                        bypass <= '0';
            if (chk = '0') then
                        meq          <= '1';
                        se            <= '0';
                        dest_rf     <= rf;
                        wr_regs   <= '1';
            if (dest /= "1111") then
                        can_de     <= '1';
            end if;
            else
                        cyc          := "11111";
                        out_mem <= '0';
                        meq         <= '0';
                        not_inc    <= '0';
                        dest_rf     <= rf;
                        wr_regs <= '1';
if (dest = "1111") then
                        new_adr  <= '1';
                        se            <= '0';
                        alu_use   <= '1';
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
                        check    := '1';
                else
                        not_get  <= '0';
                        from_tp  <= '1';
                        se       <= '1';
                        can_de   <= '0';
                end if;
            end if;
                        chk := not chk;
            else
                        wr_regs <= '1';
                        dest_rf  <= rf;
            end if;
      when "101" =>
---------- Store Multiple ---
                        set_fg   <= '0';
                        Byte     <= '0';
                        bypass   <= '0';
                        lock     <= '0';
                        sh_out   <= '0';
                if (cyc = "00000") then
                        new_adr <= '1';
                        meq      <= '0';
                        se       <= '0';
                        not_get  <= '1';
                        wr_regs  <= '0';
                        wr_mem <= '0';
                        can_de   <= '0';
                        alu_use <= not (pt_inx and inc);
                        not_inc  <= '1';
                elsif (cyc = "00001") then
                        new_adr <= '0';
                if (wr_back = '1') then
                        dest_rf <= temp_rf;
                        wr_regs <= '1';
                end if;
```

```
                        tp_inst   <= instruc;
                        wr_mem <= '1';
                        se        <= '1';
                else
                        wr_regs <= '0';
                        dest_rf <= "1111";
                end if;
                if (reg_no = "00001") then
                        can_de <= '1';
                elsif (reg_no = "00000") then
                        se        <= '0';
                        can_de    <= '0';
                        not_get   <= '0';
                        from_tp <= '1';
                        not_inc   <= '0';
                        alu_use   <= '0';
                        cyc       := "11111";
                end if;
when "110" =>
---------- Swap -------------
                        set_fg    <= '0';
                        sh_out    <= '0';
        if (cyc = "00000") then
                new_adr <= '1';
                wr_mem <= '0';
                Byte      <= '0';
                bypass    <= '1';
                lock      <= '0';
                not_get   <= '1';
                from_tp   <= '0';
                wr_regs <= '0';
                meq       <= '0';
                se        <= '0';
                can_de    <= '0';
                alu_use   <= '1';
                not_inc   <= '1';
```

```
elsif (cyc= "00001") then
                    Byte      <= nByte;
                    tp_inst   <= instruc;
                    lock      <= '1';
                    bypass    <= '0';
                    out_mem<= '1';
elsif (cyc= "00010") then
                    out_mem<= '0';
                    new_adr <= '0';
                    meq       <= '1';
                    se        <= '0';
                    can_de    <= '1';
elsif (cyc= "00011") then
                    lock      <= '0';
                    not_get   <= '0';
                    from_tp   <= '1';
                    wr_regs <= '1';
                    dest_rf   <= dest;
                    wr_mem    <= '1';
                    meq       <= '0';
                    se        <= '1';
                    can_de    <= '0';
                    not_inc   <= '0';
                    cyc       := "11111";
end if;
      when others =>
                    Byte      <= '0';
                    set_fg    <= '0';
                    meq       <= '0';
                    se        <= '1';
                    not_inc   <= '0';
                    wr_regs   <= '0';
                    wr_mem <= '0';
                    not_get <= '0';
                    from_tp <= '0';
                    can_de    <= '0';
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
                                lock      <= '0';
                                bypass    <= '0';
                                sh_out    <= '0';
          end case;
        end if;
                           if (inst /= "11111") then
                                cyc := cyc + "1";
                           end if;
                       else
                           Byte      <= '0';
                           set_fg    <= '0';
                           meq       <= '0';
                           se        <= '1';
                           not_inc   <= '0';
                           wr_regs   <= '0';
                           wr_mem <= '0';
                           not_get <= '0';
                           from_tp <= '0';
                           can_de    <= '0';
                           lock      <= '0';
                           bypass    <= '0';
                           sh_out    <= '0';
                      end if;
      end if;
                 cycle <= cyc;
end process EXECUTE1;
WRITE_REGISTER:
          process (pc_reg,dest_rf)
          begin
                 if (pc_reg = '1') then
                           rf_des    <= "1110";
                 else
                           rf_des    <= dest_rf;
                 end if;
          end process WRITE_REGISTER;
end rtl_ctl;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.p_mem.all;

Entity memory is
-- generic parameters
-- port list
    port(rw       : in std_logic;
         bw       : in std_logic;
         addr     : in std_logic_vector(31 downto 0);
         mdata    : inout std_logic_vector(31 downto 0));
end memory;

architecture rtl_mem of memory is

        signal mem  : mem_table := mem_constant;
        alias byte0 : std_logic_vector(7 downto 0) is mdata(7 downto 0);
        alias byte1 : std_logic_vector(7 downto 0) is mdata(15 downto 8);
        alias byte2 : std_logic_vector(7 downto 0) is mdata(23 downto 16);
        alias byte3 : std_logic_vector(7 downto 0) is mdata(31 downto 24);
        alias nb : std_logic_vector(1 downto 0) is addr(1 downto 0);


begin
-- concurrent statements
        MANAGE:
        Process (bw,rw,addr)
                variable post : integer;
        begin
    post := ((conv_integer(addr))/4)*4;
                if rw = '1' then -- Read Memory
                  if bw = '1' then -- Byte
         case nb is
        when "00"   => byte0 <= mem(post);
        when "01"   => byte1 <= mem(post+1);
```

```
            when "10"   => byte2 <= mem(post+2);
            when others => byte3 <= mem(post+3);
            end case;
            else -- word
            mdata <= mem(post+3) & mem(post+2) & mem(post+1) & mem(post);
                                end if; -- if (bw = '1')
                    else
                                mdata <= (others => 'Z');
                    end if;
            end process MANAGE;
end rtl_mem;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
-- ARM begin work in falling edge clk
library IEEE;
use IEEE.std_logic_1164.all;

Entity ARM is
        port(clk        : in std_logic;
        reset           : in std_logic;
        mdata           : inout std_logic_vector(31 downto 0);
        addr            : out std_logic_vector(31 downto 0);
        reg_out  : out std_logic_vector(31 downto 0);
        reg_rw          : out std_logic;
        lock            : out std_logic;
        RW              : out std_logic;
        BW              : out std_logic;
        oc              : out std_logic;
        mreq            : out std_logic;
        seq             : out std_logic;
        psw             : out std_logic_vector(3 downto 0));
end ARM;

Architecture rtl_ARM of ARM is
        component OSC4                          -- 500K onchip oscillator
   Port (F8M      : out std_logic );
        End component;

        component ctl
                port(reset      : in std_logic;
                clk             : in std_logic;
                rclk            : in std_logic;
                instruc         : in std_logic_vector(31 downto 0);
                done            : in std_logic;
                cond_ok         : in std_logic;
                st_de           : in std_logic;
                mem_ot          : out std_logic;
                mar_rd          : out std_logic;
                forw1           : out std_logic;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
            forw2          : out std_logic;
            pc1            : out std_logic;
            pc2            : out std_logic;
            reg1           : out std_logic;
            reg2           : out std_logic;
            use_of         : out std_logic;
            alu_pc         : out std_logic;
            pc_rg          : out std_logic;
            alu_rd         : out std_logic;
            sh_rd          : out std_logic;
            mem_rd         : out std_logic;
            rf_write       : out std_logic;
            rf_in1         : out std_logic_vector(3 downto 0);
            rf_in2         : out std_logic_vector(3 downto 0);
            rf_des         : out std_logic_vector(3 downto 0);
            alu_md         : out std_logic_vector(3 downto 0);
            cond           : out std_logic_vector(3 downto 0);
            set_fg         : out std_logic;
            sh_md          : out std_logic_vector(1 downto 0);
            op_regs        : out std_logic_vector(1 downto 0);
            sh_out         : out std_logic;
            start          : out std_logic;
            not_inc        : out std_logic;
            alu_use        : out std_logic;
            fr_ctl         : out std_logic_vector(25 downto 0);
            sh_in3         : out std_logic_vector(12 downto 0);
            lock           : out std_logic;
            nRW            : out std_logic;
            nBW            : out std_logic;
            nENOUT         : out std_logic;
            mreq           : out std_logic;
            seq            : out std_logic);
end component;

component alu
port(reset          : in std_logic;
```

```
        clk         : in std_logic;
        rclk        : in std_logic;
        set_fg      : in std_logic;
        sh_flag     : in std_logic;
        alu_rd      : in std_logic;
        alu_in1     : in std_logic_vector(31 downto 0);
        alu_in2     : in std_logic_vector(31 downto 0);
        alu_md      : in std_logic_vector(3 downto 0);
        cond        : in std_logic_vector(3 downto 0);
        cond_ok     : out std_logic;
        carry_f     : out std_logic;
        psw         : out std_logic_vector(3 downto 0);
        alu_out     : out std_logic_vector(31 downto 0));
end component;

component datamem
        port(clk    : in std_logic;
        rclk        : in std_logic;
        mem_rd      : in std_logic;
        noByte      : in std_logic_vector(1 downto 0);
        data_wr     : in std_logic_vector(31 downto 0);
        nBW         : in std_logic;
        data_ot     : out std_logic_vector(31 downto 0);
        instruc     : out std_logic_vector(31 downto 0);
        dat         : out std_logic_vector(31 downto 0);
        mdat        : in std_logic_vector(31 downto 0));
end component;

component mar
        port(reset  : in std_logic;
        rclk        : in std_logic;
        mar_in      : in std_logic_vector(31 downto 0);
        mar_rd      : in std_logic;
        mreq        : in std_logic;
        seq         : in std_logic;
        alu_usc     : in std_logic;
```

```
        not_inc         : in std_logic;
        maddr           : out std_logic_vector(31 downto 0);
        pc              : out std_logic_vector(29 downto 0);
        st_de           : out std_logic);
end component;

component mul
        port(clk        : in std_logic;
        rclk            : in std_logic;
        start           : in std_logic;
        mul_in1         : in std_logic_vector(31 downto 0);
        mul_in2         : in std_logic_vector(31 downto 0);
        done            : out std_logic;
        mul_out         : out std_logic_vector(31 downto 0));
end component;

component shifter
port(sh_in1             : in  std_logic_vector(7 downto 0);
        sh_in2          : in  std_logic_vector(31 downto 0);
        sh_in3          : in  std_logic_vector(12 downto 0);
        sh_md           : in  std_logic_vector(1 downto 0);
        op_regs         : in  std_logic_vector(1 downto 0);
        rclk            : in  std_logic;
        sh_rd           : in  std_logic;
        carry_f         : in  std_logic;
        sh_flag         : out std_logic;
        shift_out       : out std_logic_vector(31 downto 0));
end component;

component reg_file
        port(rf_in1     : in std_logic_vector(3 downto 0);
        rf_in2          : in std_logic_vector(3 downto 0);
        rf_des          : in std_logic_vector(3 downto 0);
        rf_dataw        : in std_logic_vector(31 downto 0);
        rf_write        : in std_logic;
        clk             : in std_logic;
```

```
            rf_din1            : out std_logic_vector(31 downto 0);
            rf_din2            : out std_logic_vector(31 downto 0));
        end component;
        signal iclk : std_logic;                -- 500K onchip oscillator
        signal clk : std_logic;                 -- 500K onchip oscillator
        signal rclk: std_logic;                 -- inverse 500K onchip oscillator
        signal done,cond_ok,mem_ot,mar_rd,forw1,forw2 : std_logic;
        signal pc1,pc2,reg1,reg2,use_of,alu_pc,alu_rd,sh_rd,mem_rd,pc_rg : std_logic;
        signal set_fg,sh_out,start,not_inc,alu_use,nRW,nBW,nENOUT : std_logic:
        signal alu_md,cond : std_logic_vector(3 downto 0);
        signal sh_md,op_regs : std_logic_vector(1 downto 0);
        signal bus1,bus2,instruc : std_logic_vector(31 downto 0);
        signal alu_in2,alu_out : std_logic_vector(31 downto 0);
        signal sh_flag,carry_f : std_logic;
        signal sh_in3 : std_logic_vector(12 downto 0);
        signal maddr : std_logic_vector(31 downto 0);
        signal data_ot : std_logic_vector(31 downto 0);
        signal shift_out,mul_out : std_logic_vector(31 downto 0);
        signal meq,se : std_logic;
        signal dat : std_logic_vector(31 downto 0);
        signal fr_ctl : std_logic_vector(25 downto 0);
        signal pc : std_logic_vector(29 downto 0);
        signal temp : std_logic_vector(31 downto 0);
        signal rf_din1,rf_din2,rf_dataw : std_logic_vector(31 downto 0);
        signal rf_in1,rf_in2,rf_des : std_logic_vector(3 downto 0);
        signal rf_write : std_logic;
        signal st_de : std_logic;
begin
-- concurrent statements
        RW <= nRW;
        BW<= nBW;
        addr <= maddr;
        bus1 <= alu_out when forw1 = '1' else (others => 'Z');
        bus1 <= (pc & "00") when pc1 = '1' else (others => 'Z');
        bus1 <= rf_din1 when reg1 = '1' else (others => 'Z');
        bus2 <= data_ot when mem_ot = '1' else (others => 'Z');
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
        bus2 <= alu_out when forw2 = '1' else (others => 'Z');
        bus2 <= (pc & "00") when pc2 = '1' else (others => 'Z');
        bus2 <= rf_din2 when reg2 = '1' else (others => 'Z');
        bus2(25 downto 0) <= fr_ctl when use_of = '1' else (others => 'Z');
        bus2(31 downto 26) <= (others => fr_ctl(25)) when use_of = '1' else (others => 'Z');
        oe<= nENOUT;
        alu_in2 <= mul_out when sh_out = '1' else shift_out;
        mreq<= meq;
        seq<= sc;
        mdata<= dat when nENOUT = '1' else (others => 'Z');
        reg_out <= rf_dataw;
        reg_rw    <= rf_write;

-- alu_pc => enable D-FF to keep value maddr for write to Register in next cycle
-- that is Return-addrees of Branch
-- pc_rg  => select value to rf_dataw (Address (Branch only) or Alu output)
        KEEP_MADDR:
        Process (clk)
        begin
                if rising_edge(clk) then
                        if alu_pc = '1' then
                                temp <= maddr;
                        end if;
                end if;
        end process KEEP_MADDR;
        rf_dataw <= temp when pc_rg = '1' else alu_out;
-- CLOCK INSIDE FPGA
--      UOSC4: OSC4 port map (F8M=>ICLK);  -- 500K onchip oscillator

        clk  <= iclk and '1' ;          -- 500K onchip oscillator
        rclk <= iclk xor '1' ;          -- inverse 500K onchip oscillator

        use_control:ctl port map(reset    => reset,
                                 clk      => clk,
                                 rclk     => rclk,
                                 instruc  => instruc, -- data from memory
```

```
done      => done,
cond_ok   => cond_ok,
st_de     => st_de,
mem_ot    => mem_ot,
mar_rd    => mar_rd,
forw1     => forw1,
forw2     => forw2,
pc1       => pc1,
pc2       => pc2,
reg1      => reg1,
reg2      => reg2,
use_of    => use_of,
alu_pc    => alu_pc,
pc_rg     => pc_rg,
alu_rd    => alu_rd,
sh_rd     => sh_rd,
mem_rd    => mem_rd,
rf_write  => rf_write,
rf_in1    => rf_in1,
rf_in2    => rf_in2,
rf_des    => rf_des,
alu_md    => alu_md,
cond      => cond,
sct_fg    => sct_fg,
sh_md     => sh_md,
op_regs   => op_regs,
sh_out    => sh_out,
start     => start,
not_inc   => not_inc,
alu_use   => alu_use,
fr_ctl    => fr_ctl,
sh_in3    => sh_in3,
lock      => lock,
nENOUT    => nENOUT,
nRW       => nRW,
nBW       => nBW,
```

```
                                        mreq      => meq,
                                        seq       => se);

use_alu:alu port map(reset    =>      reset,
                                   clk       =>      clk,
                                   rclk      =>      rclk,
                                   set_fg    =>      set_fg,
                                   sh_flag =>        sh_flag,
                                   alu_rd    =>      alu_rd,
                                   alu_in1 =>        bus1,
                                   alu_in2 =>        alu_in2,
                                   alu_md  =>        alu_md,
                                   cond      =>      cond,
                                   cond_ok =>        cond_ok,
                                   carry_f =>        carry_f,
                                   psw       => psw,
                                   alu_out =>        alu_out);

use_datamem:datamem port map(clk    =>      clk,
                                   rclk      => rclk,
                                   mem_rd  => mem_rd,
                                   noByte    =>      maddr(1 downto 0),
                                   data_wr =>        bus2,
                                   nBW       =>      nBW,
                                   data_ot =>        data_ot,
                                   instruc => instruc,
                                   dat       => dat,
                                   mdat      =>      mdata);

use_mar:mar port map(reset    => reset,
                                   rclk      => rclk,
                                   mar_in   => bus1,
                                   mar_rd   => mar_rd,
                                   mreq      => meq,
                                   seq       => sc,
                                   alu_use  => alu_use,
```

```
                                        not_inc  => not_inc,
                                        maddr    => maddr,
                                        pc       => pc,
                                        st_de    => st_de);

    use_mul:mul port map(clk    => clk,
                                        rclk     => rclk,
                                        start    => start,
                                        mul_in1  => bus1,
                                        mul_in2  => bus2,
                                        done     => done,
                                        mul_out  => mul_out);

    use_shift:shifter port map(sh_in1      => bus1(7 downto 0),
                                         sh_in2   => bus2,
                                         sh_in3   => sh_in3,
                                         sh_md    => sh_md,
                                         op_regs  => op_regs,
                                         rclk     => rclk,
                                         sh_rd    => sh_rd,
                                         carry_f  => carry_f,
                                         sh_flag  => sh_flag,
                                         shift_out => shift_out);

    use_rf:reg_file port map(rf_in1   => rf_in1,
                                        rf_in2   => rf_in2,
                                        rf_des   => rf_des,
                                        rf_dataw => rf_dataw,
                                        rf_write => rf_write,
                                        clk      => clk,
                                        rf_din1  => rf_din1,
                                        rf_din2  => rf_din2);

end rtl_ARM;
```

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ข. ประยุกต์และวิเคราะห์การใช้ ASM ในการทวนสอบ

1 การทวนสอบการทำงานของหน่วยคำนวณและตรรกะ

Rule : ExecuteALU

```
If  ExecuteOK and ALUInstr(Instr)  then
    If Satisfies(Status, CondCode(Instr))  then
        If  WriteResult(Instr)  then
          Contents(DesReg)  := ALU( ALUop(Instr), Aop, Bop, Carry(Status))
        Endif
    If  SetCondCode(Instr)  then
        Status := UpdateStatus( Status, ALUop(Instr), Aop, Bop, ShiftCarryOp)
    Endif
Endif
```

Specification :

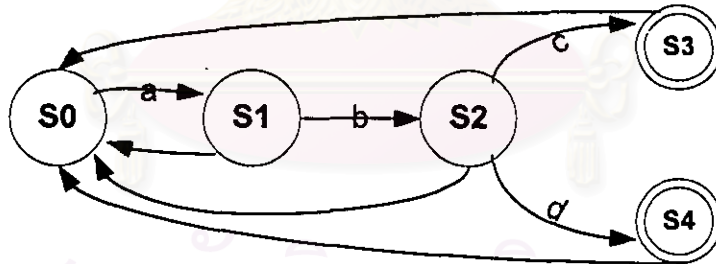
Rule a : ExecuteOK and ALUInstr(Instr)

Rule b : Satisfies(Status, CondCode(Instr))

Rule c : WriteResult(Instr)

Rule d : SetCondCode(Instr)

แทนด้วยสเตตแมชชีน :



รูปที่ 1 Finite State Machine (FSM) ของคุณลักษณะที่ต้องการของหน่วยคำนวณและตรรกะ

การสร้าง : เราสามารถแทนการทำงานของ VHDL ด้วยสายอักขระการทำงานดังนี้

ตารางที่1. 1 ผลจากการแทนโมดูลที่ออกแบบด้วย VHDL เข้าสู่ FSM ของหน่วยคำนวณและตรรกะ

สายอักขระ	การทำงาน	ผลที่ได้
Ac	ExcuteOK and ALUInstr(Instr) และ WriteResult(Instr)	Not accept
Acd	ExcuteOK and ALUInstr(Instr) และ WriteResult(Instr) และ SetCondCode(Instr)	Not accept
Ad	ExcuteOK and ALUInstr(Instr) และ SetCondCode(Instr)	Not accept

<u>สรุปผลที่ได้</u>

พบว่าจากการสร้างส่วนการทำงานของหน่วยคำนวณและตรรกะในส่วนของโมดูลที่ออกแบบเองด้วย VHDL มีข้อผิดพลาดในส่วนที่ไม่ได้มีการตรวจสอบว่าเงื่อนไขค่าแฟลคก่อนที่จะเริ่มเข้าทำงาน ซึ่งเป็นส่วนของสายอักขระ b (Satisfies(Status,Condcode(Instr)) ที่ขาดหายไป ผลจากข้อผิดพลาด ณ. จุดนี้ไม่กระทบต่อการทำงานของหน่วยคำนวณและตรรกะ ทำให้การจำลองการทำงานไม่สามารถตรวจสอบข้อผิดพลาดจุดนี้ได้ และผลจากข้อผิดพลาดนี้ทำให้โมดูลที่ออกแบบมีการทำงานตลอดเวลาจึงทำให้สิ้นเปลืองพลังงานที่ใช้ในการประมวลผลมากกว่าปกติ

2 การทวนสอบการทำงานของรีจิสเตอร์

<u>Rule : Register_file</u>

```
If rf_write and Content(DesReg)  then
        Content(Desop) = Content(DesReg)
elsif rf_read  then
        Aop = Content(AopReg)
        Bop = Content(BopReg)
End if;
```

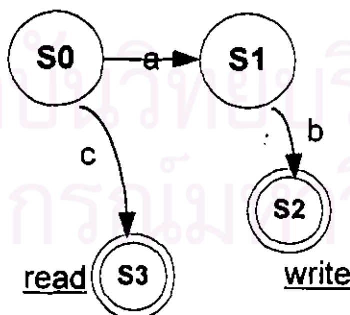
<u>Specification :</u>

Rule a : rf_write

Rule b : Content(DesReg)

Rule c : rf_read

<u>แทนด้วยสเตตแมชชีน</u> :



รูปที่ 2 Finite State Machine (FSM) ของคุณลักษณะที่ต้องการของรีจิสเตอร์

<u>การสร้าง</u> : เราสามารถแทนการทำงานของ VHDL ด้วยสายอักขระการทำงานดังนี้

ตารางที่ 2ผลจากการแทนโมดูลที่ออกแบบด้วย VHDL เข้าสู่ FSM ของหน่วยรีจิสเตอร์

สายอักขระ	การทำงาน	ผลที่ได้
ab	Rf_write and Content(DesReg)	Accept
c	Rf_read	Accept

<u>สรุปผลที่ได้</u>

พบว่าโมดูลที่ออกแบบด้วยภาษา VHDL ของหน่วยรีจิสเตอร์ไม่มีข้อผิดพลาด ทำให้ Finite State Machine ที่เป็นคุณลักษณะที่กำหนด ยอมรับการทำงานที่ป้อนให้ผ่านทางสายอักขระ ทั้งในส่วนของการอ่านและเขียนข้อมูลลงบนรีจิสเตอร์

3 การทวนสอบการทำงานของหน่วยการเก็บข้อมูลจากหน่วยความจำลงสู่รีจิสเตอร์

<u>Rule : Single Load</u>

```
If SingleLoadInstr(ExecuteInstr)  then
  If ExecuteMode = first-step and Satisfies(Status, CondCode(executeInstr) then
        AddrReg :=  MemAddr
        ExecuteMode := load-read-memory
        Bop := offset
  elsif  ExecuteMode = load-read-memory then
        if ByteTransferInstr(ExecuteInstr)  then
              DataIn := PadWord(Memory(AddrReg))
        else  DataIn :=  MemoryWord(AddrReg)
        end if
        if WriteBack(ExecuteInstr)  then
              Contents(BaseOp(ExcuteInstr)) := ALU ("+",Aop,Bop,0)
        End if
        ExecuteMode := load-write-register
  elsif   ExcuteMode := load-write-register
        Contents(DesReg) := DataIn
        ExecuteMode := first-step
  End if
End if
```

Specification :

Rule a : SingleLoadInstr(ExecuteInstr)

Rule b : ExecuteMode = first-step and Satisfies(Status, CondCode (executeInstr))

Rule c : ExecuteMode = load-read-memory

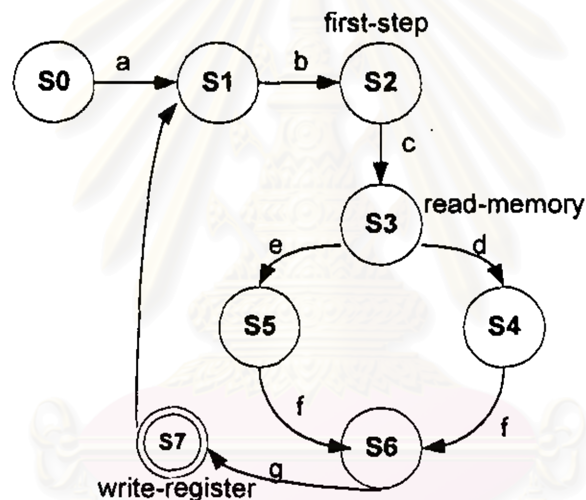
Rule d : ByteTransferInstr(ExecuteInstr)

Rule e : DataIn := MemoryWord(AddrReg)

Rule f : WriteBack(ExecuteInstr)

Rule g : ExcuteMode := load-write-register

แทนด้วยสเตตแมชชีน :



รูปที่ 3 Finite State Machine (FSM) ของคุณลักษณะที่ต้องการของหน่วยเก็บข้อมูลลงรีจิสเตอร์

การสร้าง : เราสามารถแทนการทำงานของ VHDL ด้วยสายอักขระการทำงานดังนี้

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ตารางที่ 3 ผลจากการแทนโมดูลที่ออกแบบด้วย VHDL เข้าสู่ FSM ของหน่วยโหลด

สายอักขระ	การทำงาน	ผลที่ได้
abcdfg	SingleLoadInstr(ExecuteInstr) และ ExecuteMode=first-step and Satisfies(Status, CondCode(ExecuteInstr)) และ ExecuteMode=load-register-memory และ ByteTransferInstr (ExecuteInstr) และ WriteBack(ExecuteInstr) และ ExecuteMode=load-write-register	Accept
abcefg	SingleLoadInstr(ExecuteInstr) และ ExecuteMode=first-step and Satisfies(Status, CondCode(ExecuteInstr)) และ ExecuteMode=load-register-memory และ ByteTransferInstr (ExecuteInstr) และ DataIn := PadWord(Memory(AddrReg)) และ ExecuteMode=load-write-register	Accept

สรุปผลที่ได้

พบว่าโมดูลที่ออกแบบด้วยภาษา VHDL ของหน่วยเก็บข้อมูลจากหน่วยความจำลงสู่รีจิสเตอร์ ไม่มีข้อผิดพลาด มีผลให้ Finite State Machine ที่เป็นคุณลักษณะที่กำหนด ยอมรับการทำงานที่ป้อนให้ผ่านทางสายอักขระ โดยการทดสอบในส่วนนี้การทำงานจะอยู่ในลักษณะที่เรียงตามลำดับขั้น หากเกิดข้อผิดพลาด การทำงานจะคงสถานะเดิมทันที

4 การทวนสอบการทำงานของหน่วยการเก็บข้อมูลจากรีจิสเตอร์ลงสู่หน่วยความจำ

Rule : Single Store

```
If SingleInstr(ExecuteInstr) then
    If ExecuteMode = first-step and Satisfies(Status, CondCode(ExecuteInstr)) then
        MemAddr := AddrReg
        DataOut := Contents(DesReg)
        ExecuteMode := Store
        Bop := offset
    elsif ExecuteMode = store then
        if ByteTransferInstr(ExecuteInstr) then
            Memory(AddrReg) := DataOut
        else AssignWord(AddrReg,DataOut)
        end if
```

```
            if WriteBack(ExcuteInstr) then
                    Contents(Baseop(ExecuteInstr)) := ALU ("+", AOP,BOP,0)
            end if
            ExecuteMode := first-step
        end if
    end if
```

Specification :

Rule a : SingleStoreInstr(ExecuteInstr)

Rule b : ExecuteMode = first-step and Satisfies(Status,CondCode (ExecuteInstr)))

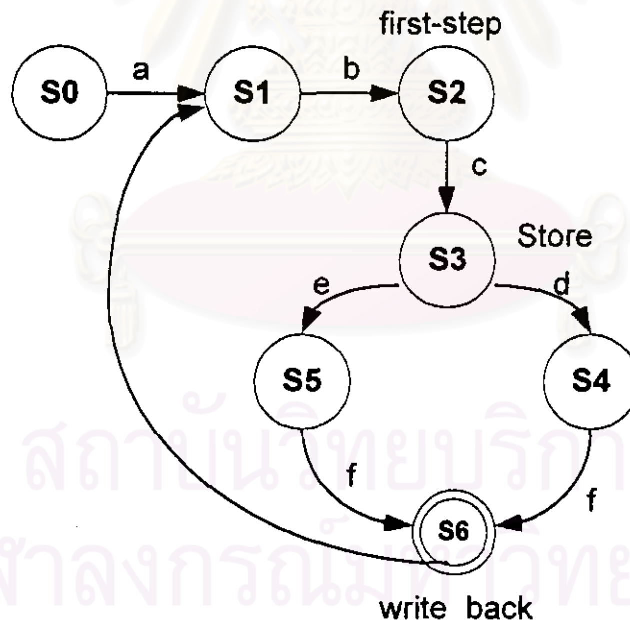
Rule c : ExecuteMode = store

Rule d : ByteTransferInstr(ExecuteInstr)

Rule e : AssignWord(AddrReg,DataOut)

Rule f : WriteBack(ExecuteInstr)

แทนด้วยสเตตแมชชีน :



รูปที่ 4 Finite State Machine (FSM) ของคุณลักษณะที่ต้องการของหน่วยเก็บข้อมูลลงหน่วยความจำ

การสร้าง : เราสามารถแทนการทำงานของ VHDL ด้วยสายอักขระการทำงานดังนี้

ตารางที่ 4 ผลจากการแทนโมดูลที่ออกแบบด้วย VHDL เข้าสู่ FSM ของหน่วยเก็บค่า

สายอักขระ	การทำงาน	ผลที่ได้
Abcdf	SingleStoreInstr(ExecuteInstr) และ ExecuteMode = first-step and Satisfies(Status,CondCode(ExecuteInstr))) และ ExecuteMode = store และ ByteTransferInstr(ExecuteInstr) และ WriteBack(ExecuteInstr)	Accept
Abcef	SingleStoreInstr(ExecuteInstr) และ ExecuteMode = first-step and Satisfies(Status,CondCode(ExecuteInstr))) และ ExecuteMode = store และ AssignWord(AddrReg,DataOut) WriteBack(ExecuteInstr)	Accept

สรุปผลที่ได้

พบว่าโมดูลที่ออกแบบด้วยภาษา VHDL ของหน่วยเก็บข้อมูลจากรีจิสเตอร์ลงสู่หน่วยความจำไม่มีข้อผิดพลาด มีผลให้ Finite State Machine ที่เป็นคุณลักษณะที่กำหนด ยอมรับการทำงานที่ป้อนให้ผ่านทางสายอักขระ โดยการทดสอบในส่วนนี้การทำงานจะอยู่ในลักษณะที่เรียงตามลำดับขั้น หากเกิดข้อผิดพลาด การทำงานจะคงสถานะเดิมทันที

ประวัติผู้เขียน

นางสาว วรรณรัช สันติอมรทัต เกิดวันที่ 15 กรกฎาคม พ.ศ. 2519 ที่อำเภอเมือง จังหวัดภูเก็ต สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตร์บัณฑิต สาขาไฟฟ้า ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ในปีการศึกษา 2540 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตร์มหาบัณฑิต ที่จุฬาลงกรณ์ เมื่อ พ.ศ. 2541 ปัจจุบันเป็นพนักงานมหาวิทยาลัย ตำแหน่งอาจารย์ ระดับ 3 มหาวิทยาลัยสงขลานครินทร์ อ. หาดใหญ่ จ. สงขลา

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย