Listings

Main

Listing 1: main.vhd

```vhdl
-- Authors:
--      Wim Looman, Forrest McKerchar, Henry Jenkins, Joel Koh, Sasha Wang, Tracy Jackson

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;

entity main is
  port (
        clk   : in  std_logic;
        reset : in  std_logic;
        tx    : out std_logic;
        rx    : in  std_logic;
        sw1   : in  std_logic;
        sw2   : in  std_logic
  );
end main;

architecture main_arch of main is
  component cpu IS
    PORT(
      -- instruction bus
      inst_add  : out std_logic_vector(11 downto 0); -- Address lines.
      inst_data : in  std_logic_vector(15 downto 0); -- Data lines.
      inst_req  : out std_logic;                     -- Pulled low to request bus usage.
      inst_ack  : in  std_logic;                     -- Pulled high to inform of request completion.
      -- data bus
      data_add  : out    std_logic_vector(15 downto 0); -- Address lines.
      data_line : inout std_logic_vector(7 downto 0);  -- Data lines.
      data_read : out    std_logic;                     -- High for a read request, low for a write request.
      data_req  : out    std_logic;                     -- Pulled low to request bus usage.
      data_ack  : inout std_logic;                     -- Pulled high to inform of request completion.
      -- extras
      clk       : in  std_logic;
      reset     : in  std_logic
    );
  end component;
  component mmu_main is
    port (
      -- instruction bus
      inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
      inst_data : out std_logic_vector(15 downto 0); -- Data lines.
      inst_req  : in  std_logic;                     -- Pulled low to request bus usage.
      inst_ack  : out std_logic;                     -- Pulled high to inform of request completion.
      -- data bus
      data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
      data_line : inout std_logic_vector(7 downto 0);  -- Data lines.
      data_read : in    std_logic;                     -- High for a read request,
          low for a write request.
      data_req  : in    std_logic;                     -- Pulled low to request bus
          usage.
      data_ack  : inout std_logic;                     -- Pulled high to inform of
          request completion.
      -- extras
      clk          : in  std_logic;
      receive_pin  : in  std_logic;
      transfer_pin : out std_logic
    );
  END component;
  component IO is
       PORT(
            -- data bus --
            data_add    : IN      std_logic_vector(15 DOWNTO 0); -- address lines --
            data_data   : INOUT   std_logic_vector(7 DOWNTO 0);  -- data lines --
            data_read   : INOUT   std_logic;                     -- pulled high for
                read, low for write --
            data_req    : INOUT   std_logic;                     -- pulled low to
                request bus usage --
            data_ack    : INOUT   std_logic;                     -- pulled high to
                inform request completion --
            -- io --
            clk         : IN      std_logic;
            sw1         : IN      std_logic;
            sw2         : IN      std_logic);
            --leds      : OUT std_logic_vector(7 DOWNTO 0);
  END component;
  -- instruction bus
  signal inst_add  : std_logic_vector(11 downto 0); -- Address lines.
  signal inst_data : std_logic_vector(15 downto 0); -- Data lines.
  signal inst_req  : std_logic;                     -- Pulled low to request bus
      usage.
  signal inst_ack  : std_logic;                     -- Pulled high to inform of
      request completion.
  -- data bus
  signal data_add  : std_logic_vector(15 downto 0); -- Address lines.
  signal data_line : std_logic_vector(7 downto 0);  -- Data lines.
  signal data_read : std_logic;                     -- High for a read request, low
      for a write request.
  signal data_req  : std_logic;                     -- Pulled low to request bus
      usage.
  signal data_ack  : std_logic;                     -- Pulled high to inform of
      request completion.

  begin
    c : cpu port map(
      -- instruction bus
      inst_add  => inst_add, -- Instruction address
      inst_data => inst_data,-- Instruction data
      inst_req  => inst_req, -- Request
      inst_ack  => inst_ack, -- Instruction obtained
      -- data bus
      data_add  => data_add, -- Data address
      data_line => data_line,-- Data
      data_read => data_read,-- 1 for read, 0 for write
      data_req  => data_req, -- Request
      data_ack  => data_ack, -- Data written to/ read from
      -- extras
      clk       => clk,
      reset     => reset
      );
    m : mmu_main port map(
      -- instruction bus
      inst_add     => inst_add,  -- Address lines.
      inst_data    => inst_data, -- Data lines.
      inst_req     => inst_req,  -- Pulled low to request bus usage.
      inst_ack     => inst_ack,  -- Pulled high to inform of request completion.
      -- data bus
```

```
        data_add      => data_add,  -- Address lines.
        data_line     => data_line, -- Data lines.
        data_read     => data_read, -- High for a read request, low for a write request.
        data_req      => data_req,  -- Pulled low to request bus usage.
        data_ack      => data_ack,  -- Pulled high to inform of request completion.
        -- extras
        clk           => clk,
        receive_pin   => rx,
        transfer_pin  => tx
      );
      i : io  port map(
              clk          => clk,
              data_add     => data_add,
              data_data    => data_line,
              data_read    => data_read,
              data_req     => data_req,
              data_ack     => data_ack,
              -- io --
              sw1          => sw1,
              sw2          => sw2
        );
end architecture main_arch;
```

IO

Listing 2: IO/debounce.vhd

```
----------------------------------------------------------------------------------
-- Module Name:    debounce
-- Description: Entity to debounce a mechanical switch/button
-- Authors: Tracy Jackson
--          Sasha Wang
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED."+";

library work;

ENTITY debounce IS
       PORT(clk : IN STD_LOGIC;
       switch : IN STD_LOGIC;
       switch_state : OUT STD_LOGIC);
END debounce;

ARCHITECTURE debounced_switch OF debounce IS
    SIGNAL count : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
     -- Debounce the switch using a counter
     PROCESS(clk, switch)
     BEGIN
          IF switch = '0' THEN
             count <= "000";
          ELSIF rising_edge(clk) THEN
                IF count /= "111" THEN
                   count <= count + 1;
                END IF;
          END IF;
          IF count = "111" AND switch = '1' THEN
             switch_state <= '1';
          ELSE
              switch_state <= '0';
          END IF;
     END PROCESS;
END debounced_switch;
```

Listing 3: IO/IO.vhd

```
----------------------------------------------------------------------------------
-- Module Name:    IO
-- Description: Entity to hangle IO
-- Authors: Tracy Jackson
--          Sasha Wang
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;

library work;
use work.debounce;
use work.switch_reg;
use work.led_io;



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

```vhdl

entity IO is
       PORT(
                -- data bus --
                data_add        : IN            std_logic_vector(15 DOWNTO 0);
                    -- address lines --
                data_data   : INOUT         std_logic_vector(7 DOWNTO 0);   -- data
                    lines --
                data_read   : INOUT         std_logic;
                    -- pulled high for read, low for write --
                data_req        : INOUT     std_logic;
                    -- pulled low to request bus usage --
                data_ack        : INOUT     std_logic;
                    -- pulled high to inform request completion --
                -- io --
                    clk         : IN            std_logic;
                    sw1         : IN            std_logic;
                    sw2         : IN            std_logic);
            --leds      : OUT std_logic_vector(7 DOWNTO 0);
end IO;

architecture io of IO is


COMPONENT led_io
    PORT(
            data_add        : IN            std_logic_vector(15 DOWNTO 0);      --
                address lines --
            data_data   : INOUT         std_logic_vector(7 DOWNTO 0);   -- data lines
                 --
            data_read   : INOUT         std_logic;                              --
                pulled high for read, low for write --
            data_req        : INOUT     std_logic;                              --
                pulled low to request bus usage --
            data_ack        : INOUT     std_logic;                              --
                pulled high to inform request completion --

            clock       : IN            std_logic
            );
END COMPONENT;

COMPONENT switch_io IS
    PORT ( data_add         : IN            std_logic_vector(15 DOWNTO 0);
           data_data        : INOUT         std_logic_vector(7 DOWNTO 0);
           data_read        : INOUT         std_logic;
           data_req         : INOUT         std_logic;
           data_ack         : INOUT         std_logic;
           clk              : IN            std_logic;
           sw1              : IN            std_logic;
           sw2              : IN            std_logic
           );
END COMPONENT;



BEGIN

led: led_io PORT MAP(data_add, data_data, data_read, data_req, data_ack, clk);
switch: switch_io PORT MAP(data_add, data_data, data_read, data_req, data_ack,
    clk,sw1,sw2);

-----------------------------------------------------------------------------------------------

END io;
```

Listing 4: IO/leds.vhd

```vhdl
----------------------------------------------------------------------------------
-- Module Name:    led_io
-- Description: Entity control output LEDs
-- Authors: Tracy Jackson
--          Sasha Wang
```

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

library work;

ENTITY led_io IS
        PORT(
                data_add        : IN            std_logic_vector(15 DOWNTO 0);
                    -- address lines --
                data_data   : INOUT         std_logic_vector(7 DOWNTO 0);   -- data
                    lines --
                data_read   : INOUT         std_logic;
                    -- pulled high for read, low for write --
                data_req        : INOUT     std_logic;
                    -- pulled low to request bus usage --
                data_ack        : INOUT     std_logic;
                    -- pulled high to inform request completion --
                --
                clock       : IN            std_logic;
                );
END led_io;

ARCHITECTURE led_arch OF led_io IS
        Signal led_enable       : std_logic;
        Signal led_state        : std_logic_vector(7 DOWNTO 0);
BEGIN

   -- Determine if it is the LEDs being accessed
   PROCESS(clock, data_req, data_add, data_read)
   BEGIN
            IF data_req = '0' AND data_add = "0000000000001110" AND data_read = '0'
                THEN
                    led_enable <= '1';
            ELSE
                    led_enable <= '0';
            END IF;
   END PROCESS;

     -- process of data from the CPU and output to LEDS
    PROCESS(clock, led_enable)
    BEGIN
            IF rising_edge(clock) THEN
                IF led_enable = '1' THEN
                        led_state <= data_data;
                        data_ack <= '0';
                END IF;
            END IF;
    END PROCESS;


END led_arch;
```

Listing 5: IO/switch_register.vhd

```vhdl
---------------------------------------------------------------------------------
-- Module Name:    switch_reg
-- Description: Entity to store switch state (can be extended to more than one)
-- Authors: Tracy Jackson
--          Sasha Wang
--
---------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library work;
```

```vhdl
ENTITY switch_reg IS
PORT( D                 : IN STD_LOGIC;
      clk,enable     : IN STD_LOGIC;
      Q                 : OUT STD_LOGIC);
END switch_reg;

ARCHITECTURE reg_arch OF switch_reg IS
BEGIN
     PROCESS(D, enable, clk)
      BEGIN
            IF rising_edge(clk) THEN --Need else there???
                    IF enable = '1' THEN
                        Q <= D;
                    END IF;
            END IF;
     END PROCESS;
END reg_arch;
```

Listing 6: IO/switches.vhd

```vhdl
----------------------------------------------------------------------------------
-- Module Name:    switch_io
-- Description: Entity to control input from switches
-- Authors: Tracy Jackson
--          Sasha Wang
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

library work;
use work.debounce;
use work.switch_reg;

entity switch_io is
       PORT(
                -- data bus --
                data_add        : IN            std_logic_vector(15 DOWNTO 0);
                    -- address lines --
                data_data    : INOUT         std_logic_vector(7 DOWNTO 0);    -- data
                    lines --
                data_read    : INOUT         std_logic;
                    -- pulled high for read, low for write --
                data_req        : INOUT     std_logic;
                    -- pulled low to request bus usage --
                data_ack        : INOUT     std_logic;
                    -- pulled high to inform request completion --
                -- io --
            clk         : IN            std_logic;
            sw1         : IN            std_logic;
            sw2         : IN            std_logic);
end switch_io;

architecture Behavioral of switch_io is


signal enable1                   : std_logic;
signal switch1_connection    : std_logic;
signal switch1_output        : std_logic;

signal enable2                   : std_logic;
signal switch2_connection    : std_logic;
signal switch2_output        : std_logic;


COMPONENT debounce
         PORT(clk, switch : IN STD_LOGIC;
                switch_state: OUT STD_LOGIC);
END COMPONENT;

COMPONENT switch_reg
```

```
         PORT( D            : IN STD_LOGIC;
                clk, enable : IN STD_LOGIC;
                Q               : OUT STD_LOGIC);
END COMPONENT;


BEGIN

sw1_debouncer: debounce PORT MAP(clk, sw1, switch1_connection);
sw1_status: switch_reg PORT MAP(switch1_connection,clk, enable1, switch1_output);


sw2_debouncer: debounce PORT MAP(clk, sw2,switch2_connection);
sw2_status: switch_reg PORT MAP(switch2_connection,clk, enable2, switch2_output);


PROCESS(clk,switch1_output,switch2_output, data_ack)
BEGIN
IF rising_edge(clk) THEN
    IF switch1_output = '1' AND data_ack = 'Z' THEN --when the switch_reg has stored
        1, disable switch_reg from getting any more info
        enable1 <= '0';

    --ELSIF data_ack = '0' AND data_add = "0000000000001110" THEN -- when the data is
         sent to the CPU, enable the switch_reg again
    --  enable1 <= '1';
    ELSE
        enable1 <= '1';
    END IF;

    IF switch2_output = '1' AND data_ack = 'Z' THEN --when the switch_reg has stored
        1, disable switch_reg from getting any more info
        enable2 <= '0';

    --ELSIF data_ack = '0' AND data_add = "0000000000001100" THEN -- when the data is
         sent to the CPU, enable the switch_reg again
    --  enable2 <= '1';
    ELSE
        enable2 <= '1';
    END IF;


END IF;
END PROCESS;


PROCESS(clk, data_add, data_read)
BEGIN
    IF rising_edge(clk) THEN
        IF data_req = '0' AND data_read = '1' THEN
            IF data_add = "0000000000001110" THEN -- switch1 address
                IF switch1_output = '1' THEN
                    data_data <= "00000001";
                ELSE
                    data_data <= "00000000";
                END IF;
                data_ack <= '0';
            END IF;
            IF data_add = "0000000000001100" THEN -- switch2 address
                IF switch2_output = '1' THEN
                    data_data <= "00000001";
                ELSE
                    data_data <= "00000000";
                END IF;
                data_ack <= '0';
            END IF;
        ELSIF data_req = '1' AND data_ack = '0' THEN
            data_ack <= 'Z';
        END IF;
    END IF;
 END PROCESS;
```

```
-------------------------------------------------------------------------------------
END Behavioral;
```

CPU

Listing 7: processor/alu.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:  18:59:20 09/18/2010
-- Design Name:
-- Module Name:  alu - alu_arch
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;


library work;
use work.fulladder8;
--use work.cpu.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

entity alu is
  Port (f   : in   STD_LOGIC_VECTOR (3 downto 0);  -- Function (opcode)
        rx  : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input x (Rx)
        ry  : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input y (Ry)
        ro  : out  STD_LOGIC_VECTOR (7 downto 0);  -- Output Normaly (Ry)
        Cin : in   STD_LOGIC;                      -- Carry in
        sr  : out  STD_LOGIC_VECTOR (15 downto 0)); -- Status register out Z(0),
            C(1), N(2)
end alu;


architecture alu_arch of alu is
  component fulladder8 IS
  Port (A    : in   STD_LOGIC_VECTOR( 7 downto 0);
        B    : in   STD_LOGIC_VECTOR( 7 downto 0);
        Cin  : in   STD_LOGIC;
        Sum  : out  STD_LOGIC_VECTOR( 7 downto 0);
        Cout : out  STD_LOGIC
        );
  end component;
  signal   A          : std_logic_vector(7 downto 0);
  signal   B          : std_logic_vector(7 downto 0);
  signal   AdderCin   : std_logic;
  signal   Sum        : std_logic_vector(7 downto 0);
  signal   AdderCout  : std_logic;
  signal   Z,C,N      : std_logic; -- Make the code easier to read
  signal   output     : std_logic_vector(7 downto 0); -- used to allow reading of ro
BEGIN
  Adder: fulladder8 port map(A, B, AdderCin, Sum, AdderCout);
```

```vhdl
  process(f, rx, ry, Cin, Sum, AdderCout)
    --signal Z,C,N  : std_logic; -- Make the code easier to read
  BEGIN
    -- use case statement to achieve
    -- different operations of ALU

      AdderCin <= '0';
      A <= (others => '0');
      B <= (others => '0');
      output <= (others => '0');
      C <= '0';
      N <= '0';
      IF f = "0001" THEN -- Do AND operation
         output <= ry and rx;
      ELSIF f = "0011" THEN -- Do OR  operation
        output <= ry or rx;
      ELSIF f = "0101" THEN
        output <= not rx;
      ELSIF f = "0111" THEN -- Do XOR operation
        output <= ry xor rx;
      ELSIF f = "1001" THEN -- Do ADD operation
        AdderCin <= '0';
        A <= ry;
        B <= rx;
        output <= Sum;
      ELSIF f = "1011" THEN -- Do ADC operation
        AdderCin <= Cin;
        A <= ry;
        B <= rx;
        output <= Sum;
      ELSIF f = "1101" THEN -- Do SUB operation
        AdderCin <= '1';
        A <= ry;
        B <= (not rx);
        output <= Sum;
      ELSIF f = "1111" THEN -- Do SBB operation
        AdderCin <= (not Cin);
        A <= ry;
        B <= (not rx);
        output <= Sum;
      ELSIF f = "0100" THEN -- Do NEG operation ( two's complement )
        AdderCin <= '1';
        A <= (others => '0');
        B <= (not rx);
        output <= Sum;
        C <= AdderCout;
        N <= output(7);
      ELSIF f = "0110" THEN -- Do CMP operation
        AdderCin <= '1';
        A <= rx;
        B <= (not ry);
        output <= Sum;
        C <= AdderCout;
        N <= output(7);
      ELSE
        AdderCin <= '0';
        A <= (others => '0');
        B <= (others => '0');
        output <= (others => '0');
        C <= '0';
        N <= '0';
      END IF;
--    if (output = "00000000") then -- Set the Zero in status register
--      sr(0) <= '1';
--    ELSE
--      sr(0) <= '0';
--    end if;

      C <= AdderCout; -- Carry is always 0
      N <= output(7); -- This might need to be changed to '0'
      ro <= output;
  end process;
    Z <= not (output(0) AND output(1) AND output(2) AND output(3) AND output(4)
                AND output(5) AND output(6) AND output(7));
    sr(0) <= Z; --Z(0)
    sr(1) <= C; --C(1)
    sr(2) <= N; --N(2)
    sr(15 downto 3) <= (others => '0');

end alu_arch;
```

Listing 8: processor/ar.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:  18:59:20 09/18/2010
-- Design Name:
-- Module Name:  ar - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;
use work.reg16;

entity ar is
  Port (clk          : in   STD_LOGIC;
        enable       : in   STD_LOGIC;
        Sel8Bit      : in   STD_LOGIC;
        SelHighByte  : in   STD_LOGIC;
        ByteInput    : in   STD_LOGIC_VECTOR (7 downto 0);
        SelRi        : in   STD_LOGIC_VECTOR (1 downto 0);   -- Select the address
            register
        SelRo        : in   STD_LOGIC_VECTOR (1 downto 0);   -- Select the address
            register
        Ri           : in   STD_LOGIC_VECTOR (15 downto 0);  -- The input
        Ro           : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output
end ar;

architecture Behavioral of ar is
  component reg16 IS
    port(I       : in  std_logic_vector(15 downto 0);
         clock   : in  std_logic;
         enable  : in  std_logic;
         reset   : in  std_logic;
         Q       : out std_logic_vector(15 downto 0)
        );
  end component;

  signal  R0E   : std_logic;  -- Enable signals
  signal  R1E   : std_logic;
  signal  R2E   : std_logic;
  signal  input : std_logic_VECTOR (15 downto 0);
  signal  Q0    : std_logic_VECTOR (15 downto 0);
  signal  Q1    : std_logic_VECTOR (15 downto 0);
  signal  Q2    : std_logic_VECTOR (15 downto 0);
BEGIN
    reg_0 : reg16 port map(input, clk, R0E, '0', Q0);
    reg_1 : reg16 port map(input, clk, R1E, '0', Q1);
    reg_2 : reg16 port map(input, clk, R2E, '0', Q2);

  SetInput: process(clk, enable, SelRi, Ri)
  BEGIN
    R0E <= '0';
    R1E <= '0';
    R2E <= '0';
    IF enable = '1' THEN
      case SelRi IS
        WHEN "00" =>
          R0E <= '1';
        WHEN "01" =>
          R1E <= '1';
        WHEN "10" =>
          R2E <= '1';
        WHEN others =>
          NULL; -- None of them are enabled
      END CASE;
    END IF;
  end process;

  -- Select if 1 or 2 Bytes is to be written and if
  SetNumBytes: process(clk, Ri, SelRi, ByteInput, Sel8Bit, SelHighByte, Q0, Q1, Q2)
  BEGIN
    IF Sel8Bit = '0' THEN
      input <= Ri;
    ELSE
      if SelHighByte = '1' THEN
        input(15 downto 8) <= ByteInput;
        case SelRi IS
          WHEN "00" =>
            input(7 downto 0) <= Q0(7 downto 0);
          WHEN "01" =>
            input(7 downto 0) <= Q1(7 downto 0);
          WHEN others =>
            input(7 downto 0) <= Q2(7 downto 0);
        END CASE;
      else
        input(7 downto 0) <= ByteInput;
        case SelRi IS
          WHEN "00" =>
            input(15 downto 8) <= Q0(15 downto 8);
          WHEN "01" =>
            input(15 downto 8) <= Q1(15 downto 8);
          WHEN others =>
            input(15 downto 8) <= Q2(15 downto 8);
        END CASE;
      END IF;
    END IF;
  end process;

  -- Set the output Ro
  WITH SelRo SELECT
  Ro <= Q0 WHEN "00",
        Q1 WHEN "01",
        Q2 WHEN others;
end Behavioral;
```

Listing 9: processor/cpu.vhd

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    16:09:46 09/15/2010
-- Design Name:
-- Module Name:    cpu - cpu_arch
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
```

```vhdl
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;

library work;
use work.alu;
use work.cu;
use work.ar;
use work.gpr;
use work.sr;
use work.pc;


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

entity cpu is

  PORT(
      -- instruction bus
      inst_add  : out std_logic_vector(11 downto 0); -- Address lines.
      inst_data : in  std_logic_vector(15 downto 0); -- Data lines.
      inst_req  : out std_logic;                     -- Pulled low to request bus usage.
      inst_ack  : in  std_logic;                     -- Pulled high to inform of request completion.
      -- data bus
      data_add  : out    std_logic_vector(15 downto 0); -- Address lines.
      data_line : inout std_logic_vector(7 downto 0);   -- Data lines.
      data_read : out    std_logic;                     -- High for a read request, low for a write request.
      data_req  : out    std_logic;                     -- Pulled low to request bus usage.
      data_ack  : inout std_logic;                     -- Pulled high to inform of request completion.
      -- extras
      clk       : in  std_logic;
      reset     : in  std_logic
    );

end cpu;

architecture cpu_arch of cpu is
  component alu IS
    Port (f   : in   STD_LOGIC_VECTOR (3 downto 0);   -- Function (opcode)
          rx  : in   STD_LOGIC_VECTOR (7 downto 0);   -- Input x (Rx)
          ry  : in   STD_LOGIC_VECTOR (7 downto 0);   -- Input y (Ry)
          ro  : out  STD_LOGIC_VECTOR (7 downto 0);   -- Output Normaly (Ry)
          Cin : in   STD_LOGIC;                       -- Carry in
          sr  : out  STD_LOGIC_VECTOR (15 downto 0)); -- Status register out Z(0), C(1), N(2)
  END component;
  component ar is
    Port (clk          : in   STD_LOGIC;
          enable       : in   STD_LOGIC;
          Sel8Bit      : in   STD_LOGIC;
          SelHighByte : in   STD_LOGIC;
          ByteInput    : in   STD_LOGIC_VECTOR (7 downto 0);
          SelRi        : in   STD_LOGIC_VECTOR (1 downto 0);    -- Select the address register
          SelRo        : in   STD_LOGIC_VECTOR (1 downto 0);    -- Select the address register
          Ri           : in   STD_LOGIC_VECTOR (15 downto 0);  -- The input
          Ro           : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output
  END component;
  component cu IS
```

```
Port (reset       : in STD_LOGIC;                          -- '0' for reset
      clock       : in STD_LOGIC;                          -- clock

      alu_f       : out STD_LOGIC_VECTOR (3 downto 0);  -- Function
      alu_Cin     : out STD_LOGIC;                         -- Carry in to ALU

      -- General Purpose Registers
      gpr_InSel   : out STD_LOGIC;                         -- select the input path (0
           - cu, 1 - ALU)
      gpr_en      : out STD_LOGIC;                         -- enable write to GPR
      gpr_SelRx   : out STD_LOGIC_VECTOR (2 downto 0);  -- select GPR output x
      gpr_SelRy   : out STD_LOGIC_VECTOR (2 downto 0);  -- select GPR output y
      gpr_SelRi   : out STD_LOGIC_VECTOR (2 downto 0);  -- select GPR input
      gpr_Ri      : out STD_LOGIC_VECTOR (7 downto 0);  -- input to GPR
      gpr_Rx      : in STD_LOGIC_VECTOR (7 downto 0);   -- output Rx from GPR
      --gpr_Ry      : in STD_LOGIC_VECTOR (7 downto 0);   -- output Ry from GPR ,
          not used

      -- Status Register
      sr_en       : out STD_LOGIC;                         -- enable write to SR
      sr_reset    : out STD_LOGIC;                         -- reset SR
      sr_Ro       : in STD_LOGIC_VECTOR (15 downto 0);  -- output from SR
      -- control unit doesnt write to SR, the ALU does

      -- Program Counter
      pc_en       : out STD_LOGIC;                         -- enable write to PC
      pc_reset    : out STD_LOGIC;                         -- reset PC
      pc_Ri       : out STD_LOGIC_VECTOR (15 downto 0); -- input to PC
      pc_Ro       : in STD_LOGIC_VECTOR (15 downto 0);  -- output from PC

      -- Address Registers
      ar_en       : out STD_LOGIC;                         -- enable write to AR
      ar_SelRi    : out STD_LOGIC_VECTOR (1 downto 0);  -- select AR in
      ar_SelRo    : out STD_LOGIC_VECTOR (1 downto 0);  -- select AR out
      ar_Ri       : out STD_LOGIC_VECTOR (15 downto 0); -- input to AR
      ar_Ro       : in STD_LOGIC_VECTOR (15 downto 0);  -- output from AR
      ar_sel8Bit  : out STD_LOGIC;                         -- only write half the AR
      ar_selHByte : out STD_LOGIC;                         -- high or low half of the
          AR to write
      ar_ByteIn   : out STD_LOGIC_VECTOR (7 downto 0);  -- 8 bit input to write
          half of AR

      -- Instruction memory
      inst_add    : out STD_LOGIC_VECTOR (11 downto 0); -- Instruction address
      inst_data   : in STD_LOGIC_VECTOR (15 downto 0);  -- Instruction data
      inst_req    : out STD_LOGIC;                         -- Request
      inst_ack    : in STD_LOGIC;                          -- Instruction obtained

      data_add    : out STD_LOGIC_VECTOR (15 downto 0); -- Data address
      data_data   : inout STD_LOGIC_VECTOR (7 downto 0);-- Data
      data_read   : out STD_LOGIC;                         -- 1 for read, 0 for write
      data_req    : out STD_LOGIC;                         -- Request
      data_ack    : in STD_LOGIC                           -- Data written to/ read
          from

      );
END component;
component gpr is
  Port (clk       : in   STD_LOGIC;
        enable    : in   STD_LOGIC;
        SelRx     : in   STD_LOGIC_VECTOR (2 downto 0);  -- The Rx output selection
            value
        SelRy     : in   STD_LOGIC_VECTOR (2 downto 0);  -- The Ry output selection
            value
        SelRi     : in   STD_LOGIC_VECTOR (2 downto 0);  -- The Ri input selection
            value
        SelIn     : in   STD_LOGIC;  -- Select where the input should be from the CU
             or CDB
        RiCU      : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input from the Control
            Unit
        RiCDB     : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input from the Common
            Data Bus
        Rx        : out  STD_LOGIC_VECTOR (7 downto 0);  -- The Rx output
```

```
          Ry        : out  STD_LOGIC_VECTOR (7 downto 0)); -- The Ry output
  END component;
  component sr is
    Port (clk       : in   STD_LOGIC;
          enable    : in   STD_LOGIC;
          reset     : in   STD_LOGIC;
          Ri        : in   STD_LOGIC_VECTOR (15 downto 0);  -- The input to the SR
          Ro        : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output from SR
  END component;
  component pc is
    Port (clk       : in   STD_LOGIC;
          enable    : in   STD_LOGIC;
          reset     : in   STD_LOGIC;
          Ri        : in   STD_LOGIC_VECTOR (15 downto 0);  -- The input to the SR
          Ro        : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output from SR
  END component;
  signal alu_Cin     : std_logic;
  signal alu_f       : std_logic_vector(3 downto 0);
  signal alu_rx      : std_logic_vector(7 downto 0);
  signal alu_ry      : std_logic_vector(7 downto 0);

  signal sr_reset    : std_logic;
  signal sr_enable   : std_logic;
  signal sr_Ro       : std_logic_vector(15 downto 0);
  signal sr_input    : std_logic_vector(15 downto 0);

  signal ar_enable    : STD_LOGIC;                        -- enable write to AR
  signal ar_SelRi     : STD_LOGIC_VECTOR (1 downto 0);   -- select AR in
  signal ar_SelRo     : STD_LOGIC_VECTOR (1 downto 0);   -- select AR out
  signal ar_Ri        : STD_LOGIC_VECTOR (15 downto 0); -- input to AR
  signal ar_Ro        : STD_LOGIC_VECTOR (15 downto 0);  -- output from AR
  signal ar_sel8Bit   : STD_LOGIC;                        -- only write half the AR
  signal ar_selHByte  : STD_LOGIC;                        -- high or low half of the AR to write
  signal ar_ByteIn    : STD_LOGIC_VECTOR (7 downto 0);   -- 8 bit input to write half of AR

  signal pc_reset    : std_logic;
  signal pc_enable   : std_logic;
  signal pc_Ri       : std_logic_vector(15 downto 0);
  signal pc_Ro       : std_logic_vector(15 downto 0);

  signal gpr_InSel   : std_logic;
  signal gpr_enable  : std_logic;
  signal gpr_SelRx   : std_logic_vector(2 downto 0);
  signal gpr_SelRy   : std_logic_vector(2 downto 0);
  signal gpr_SelRi   : std_logic_vector(2 downto 0);
  signal gpr_RiCU    : std_logic_vector(7 downto 0);
  signal gpr_RiCDB   : std_logic_vector(7 downto 0);
begin

  a: alu port map(
            f    => alu_f,
            rx   => alu_rx,
            ry   => alu_ry,
            ro   => gpr_RiCDB,
            Cin  => alu_Cin,
            sr   => sr_input
           );
  c: cu port map(

            reset     => reset,-- '0' for reset
            clock     => clk,-- clock

            alu_f     => alu_f,-- Function
            alu_Cin   => alu_Cin, -- Carry into the ALU

            -- General Purpose Registers
            gpr_InSel => gpr_InSel,-- select the input path (0 - cu, 1 - ALU)
            gpr_en    => gpr_enable,-- enable write to GPR
            gpr_SelRx => gpr_SelRx,-- select GPR output x
            gpr_SelRy => gpr_SelRy,-- select GPR output y
            gpr_SelRi => gpr_SelRi,-- select GPR input
            gpr_Ri     => gpr_RiCU,-- input to GPR
            gpr_Rx     => alu_rx,-- Rx from GPR
            --gpr_Ry     => alu_ry,-- Ry from GPR

            -- Status Register
            sr_en      => sr_enable,-- enable write to SR
            sr_reset   => sr_reset,-- reset SR
            sr_Ro      => sr_Ro,-- output from SR
            -- control unit doesnt write to SR, the ALU does

            -- Program Counter
            pc_en      => pc_enable,-- enable write to PC
            pc_reset   => pc_reset,-- reset PC
            pc_Ri      => pc_Ri,-- input to PC
            pc_Ro      => pc_Ro,-- output from PC

            -- Address Registers
            ar_en        => ar_enable,    -- enable write to AR
            ar_SelRi     => ar_SelRi,     -- select AR in
            ar_SelRo     => ar_SelRo,     -- select AR out
            ar_sel8Bit   => ar_sel8Bit,
            ar_selHByte  => ar_selHByte,
            ar_ByteIn    => ar_ByteIn,
            ar_Ri        => ar_Ri,        -- input to AR
            ar_Ro        => ar_Ro,        -- output from AR

            -- Instruction memory
            inst_add   => inst_add ,-- Instruction address
            inst_data  => inst_data,-- Instruction data
            inst_req   => inst_req ,-- Request
            inst_ack   => inst_ack ,-- Instruction obtained

            data_add   => data_add ,-- Data address
            data_data  => data_line,-- Data
            data_read  => data_read,-- 1 for read, 0 for write
            data_req   => data_req ,-- Request
            data_ack   => data_ack  -- Data written to/ read from
          );
  address : ar port map(
            clk          => clk,
            enable       => ar_enable,
            Sel8Bit      => ar_Sel8Bit,
            SelHighByte  => ar_selHByte,
            ByteInput    => ar_ByteIn,
            SelRi        => ar_SelRi,
            SelRo        => ar_SelRo,
            Ri           => ar_Ri,
            Ro           => ar_Ro
          );
g : gpr port map(
            clk     => clk,
            enable  => gpr_enable,
            SelRx   => gpr_SelRx,
            SelRy   => gpr_SelRy,
            SelRi   => gpr_SelRi,
            SelIn   => gpr_InSel,
            RiCU    => gpr_RiCU,
            RiCDB   => gpr_RiCDB,
            Rx      => alu_rx,
            Ry      => alu_ry
          );
s : sr port map(
            clk     => clk,
            enable  => sr_enable,
            reset   => sr_reset,
            Ri      => sr_input,
            Ro      => sr_Ro
            );
programcounter: pc port map(
            clk      => clk,
            enable   => pc_enable,
            reset    => pc_reset,
            Ri       => pc_Ri,
            Ro       => pc_Ro
          );
end cpu_arch;
```

Listing 10: processor/cu.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:  18:59:20 09/18/2010
-- Design Name:
-- Module Name:  cu - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description: The control unit
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.std_logic_arith.all;
--use ieee.std_logic_unsigned.all;


library work;
--use work.fulladder;
--use work.cpu.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

entity cu is
  Port (reset        : in STD_LOGIC;                        -- '0' for reset
        clock        : in STD_LOGIC;                        -- clock

        alu_f        : out STD_LOGIC_VECTOR (3 downto 0);  -- Function
        alu_Cin      : out STD_LOGIC;                       -- Carry in to ALU

        -- General Purpose Registers
        gpr_InSel    : out STD_LOGIC;                       -- select the input path (0
             - cu, 1 - ALU)
        gpr_en       : out STD_LOGIC;                       -- enable write to GPR
        gpr_SelRx    : out STD_LOGIC_VECTOR (2 downto 0);  -- select GPR output x
        gpr_SelRy    : out STD_LOGIC_VECTOR (2 downto 0);  -- select GPR output y
        gpr_SelRi    : out STD_LOGIC_VECTOR (2 downto 0);  -- select GPR input
        gpr_Ri       : out STD_LOGIC_VECTOR (7 downto 0);  -- input to GPR
        gpr_Rx       : in STD_LOGIC_VECTOR (7 downto 0);   -- output Rx from GPR
        --gpr_Ry       : in STD_LOGIC_VECTOR (7 downto 0);   -- output Ry from GPR ,
            not used

        -- Status Register
        sr_en        : out STD_LOGIC;                       -- enable write to SR
        sr_reset     : out STD_LOGIC;                       -- reset SR
        sr_Ro        : in STD_LOGIC_VECTOR (15 downto 0);  -- output from SR
        -- control unit doesnt write to SR, the ALU does

        -- Program Counter
        pc_en        : out STD_LOGIC;                       -- enable write to PC
        pc_reset    : out STD_LOGIC;                      -- reset PC
        pc_Ri       : out STD_LOGIC_VECTOR (15 downto 0); -- input to PC
        pc_Ro       : in STD_LOGIC_VECTOR (15 downto 0);  -- output from PC

        -- Address Registers
        ar_en       : out STD_LOGIC;                      -- enable write to AR
        ar_SelRi    : out STD_LOGIC_VECTOR (1 downto 0);  -- select AR in
        ar_SelRo    : out STD_LOGIC_VECTOR (1 downto 0);  -- select AR out
        ar_Ri       : out STD_LOGIC_VECTOR (15 downto 0); -- input to AR
        ar_Ro       : in STD_LOGIC_VECTOR (15 downto 0);  -- output from AR
        ar_sel8Bit  : out STD_LOGIC;                      -- only write half the AR
        ar_selHByte : out STD_LOGIC;                      -- high or low half of the
            AR to write
        ar_ByteIn   : out STD_LOGIC_VECTOR (7 downto 0);  -- 8 bit input to write
            half of AR

        -- Instruction memory
        inst_add    : out STD_LOGIC_VECTOR (11 downto 0); -- Instruction address
        inst_data   : in STD_LOGIC_VECTOR (15 downto 0);  -- Instruction data
        inst_req    : out STD_LOGIC;                      -- Request
        inst_ack    : in STD_LOGIC;                       -- Instruction obtained

        data_add    : out STD_LOGIC_VECTOR (15 downto 0); -- Data address
        data_data   : inout STD_LOGIC_VECTOR (7 downto 0);-- Data
        data_read   : out STD_LOGIC;                      -- 1 for read, 0 for write
        data_req    : out STD_LOGIC;                      -- Request
        data_ack    : in STD_LOGIC                        -- Data written to/ read
            from

        );
end cu;


architecture Behavioral of cu is
  component fulladder16 IS
  Port (A     : in    STD_LOGIC_VECTOR(15 downto 0);
        B     : in    STD_LOGIC_VECTOR(15 downto 0);
        Cin   : in    STD_LOGIC;
        Sum   : out   STD_LOGIC_VECTOR(15 downto 0);
        Cout  : out   STD_LOGIC
        );
  end component;

  type states is (reset_state, fetch, decode, execute);
  signal state      : states := reset_state;
  signal next_state : states := reset_state;

  signal opcode     : std_logic_vector(15 downto 0);  -- unprocessed instruction

  -- Decoded data
  signal rx : std_logic_vector(2 downto 0);
  signal ry : std_logic_vector(2 downto 0);
  signal ay : std_logic_vector(1 downto 0);

  -- Indicates what needs to be executed
  signal write_gpr    : std_logic;
  signal write_sr     : std_logic;
  signal write_pc     : std_logic;
  signal write_ar     : std_logic;
  signal write_memory : std_logic;

  -- full adders
  signal   A16          : std_logic_vector(15 downto 0);
  signal   B16          : std_logic_vector(15 downto 0);
  signal   AdderCin16   : std_logic;
  signal   Sum16        : std_logic_vector(15 downto 0);
  signal   AdderCout16  : std_logic;

  signal v    : STD_LOGIC_VECTOR(7 downto 0);  -- 8-bit immediate


BEGIN
  Adder16: fulladder16 port map(A16, B16, AdderCin16, Sum16, AdderCout16);
```

```

-- Process instruction
-- Assumes all instructions are valid
process(clock, state, opcode, gpr_Rx, sr_Ro, pc_Ro, ar_Ro, inst_data, inst_ack,
    data_data, data_ack,
        rx, ry, ay, v, write_gpr, write_sr, write_pc, write_ar, write_memory)
BEGIN
  if rising_edge(clock) then
    case state is
      when reset_state =>
        sr_reset <= '0';
        pc_reset <= '0';
        next_state <= fetch;

      when fetch =>
        sr_reset <= '1';
        pc_reset <= '1';

        gpr_en <= '0';
        sr_en <= '0';
        pc_en <= '0';
        ar_en <= '0';

        write_gpr <= '0';
        write_sr <= '0';
        write_pc <= '0';
        write_ar <= '0';
        write_memory <= '0';

        inst_add <= pc_Ro(11 downto 0);
        if inst_ack = '0' then
          inst_req <= '1';
        else
          opcode <= inst_data;
          inst_req <= '0';

          -- increment program counter
          AdderCin16 <= '1';
          A16 <= PC_Ro;
          B16 <= "0000000000000000";
          pc_Ri <= Sum16;
          pc_en <= '1';

          next_state <= decode;
        end if;
      when decode =>
        pc_en <= '0';

        -- ALU
        if opcode(15) = '0' and opcode(10) = '0' and not opcode(14 downto 11) =
            "0010" then

          ry <= opcode(7 downto 5);
          rx <= opcode(2 downto 0);

          gpr_SelRy <= ry;
          gpr_SelRx <= rx;
          gpr_SelRi <= ry;
          gpr_InSel <= '1';
          alu_f <= opcode(14 downto 11);
          alu_Cin <= sr_Ro(1);  -- Carry

          if not opcode(14 downto 11) = "0110" then -- CMP doesnt write to gpr, all
              others do
            write_gpr <= '1';
          end if;

          write_sr <= '1';
          next_state <= execute;

        -- Branching
        elsif opcode(11 downto 10) = "11" then
```

```vhdl
        v <= "00000000";  -- initialise v

        if opcode(15) = '1' then
          case opcode(14 downto 12) is
            when "000" => -- BEQ
              if sr_Ro(0) = '1' then -- Z=1
                v <= opcode(9 downto 2);
              end if;
            when "001" => -- BNE
              if sr_Ro(0) = '0' then -- Z=0
                v <= opcode(9 downto 2);
              end if;
            when "010" => -- BLT
              if sr_Ro(0) = '0' and sr_Ro(2) = '1' then -- Z=0 and N=1
                v <= opcode(9 downto 2);
              end if;
            when "011" => -- BGT
              if sr_Ro(0) = '0' and sr_Ro(2) = '0' then -- Z=0 and N=0
                v <= opcode(9 downto 2);
              end if;
            when "100" => -- BC
              if sr_Ro(1) = '1' then -- C=1
                v <= opcode(9 downto 2);
              end if;
            when "101" => -- BNC
              if sr_Ro(1) = '0' then -- C=0
                v <= opcode(9 downto 2);
              end if;
            when "110" => -- RJMP
              v <= opcode(9 downto 2);

            when others =>
              v <= "00000000";
          end case;

          -- PC <- PC + v
          AdderCin16 <= '0';
          A16 <= PC_Ro;
          B16 <= "00000000" & v;
          pc_Ri <= Sum16;

        elsif opcode(15 downto 12) = "0111" then -- JMP
          ay <= opcode(6 downto 5);

          -- PC <- ay
          ar_SelRo <= ay;
          pc_Ri <= ar_Ro;
        else
          -- should not reach here
          pc_Ri <= pc_Ro; -- no change
        end if;

        write_pc <= '1';
        next_state <= execute;


      -- Addressing
      else
        gpr_Insel <= '0';

        case opcode(12 downto 10) is

          when "001" => -- Load
            if opcode(15) = '1' then  -- immediate
              rx <= '0' & opcode(1 downto 0);
              v <= opcode(9 downto 2);

              -- rx <- v
              gpr_SelRi <= rx;
              gpr_Ri <= v;
              write_gpr <= '1';
              next_state <= execute;
            else                      -- direct
              rx <= opcode(2 downto 0);
              ay <= opcode(6 downto 5);

              -- rx <- [ay]
              gpr_SelRi <= rx;
              ar_selRo <= ay;
              data_add <= ar_Ro;
              data_read <= '1';
              if data_ack = '0' then  -- request data
                data_req <= '1';
              else                    -- data obtained
                gpr_Ri <= data_data;
                data_req <= '0';
                write_gpr <= '1';

                case opcode(14 downto 13) is
                  when "01" =>            -- auto increment
                    AdderCin16 <= '1';
                    A16 <= ar_Ro;
                    B16 <= "0000000000000000";
                    ar_selRi <= ay;
                    ar_sel8bit <= '0';
                    ar_Ri <= Sum16;
                    write_ar <= '1';
                  when "10" =>            -- auto decrement
                    AdderCin16 <= '0';
                    A16 <= ar_Ro;
                    B16 <= "1111111111111111";
                    ar_selRi <= ay;
                    ar_sel8bit <= '0';
                    ar_Ri <= Sum16;
                    write_ar <= '1';
                  when others =>
                    -- do nothing
                end case;
                next_state <= execute;
              end if;
            end if;

          when "101" => -- Store
            if opcode(15) = '1' then  -- immediate
              ay <= opcode(1 downto 0);
              v <= opcode(9 downto 2);

              -- [ay] <- v
              ar_selRo <= ay;
              data_add <= ar_Ro;
              data_read <= '0';
              data_data <= v;
              write_memory <= '1';
              next_state <= execute;
            else                      -- direct
              rx <= opcode(2 downto 0);
              ay <= opcode(6 downto 5);

              -- [ay] <- rx
              gpr_selRx <= rx;
              ar_selRo <= ay;
              data_add <= ar_Ro;
              data_read <= '0';
              data_data <= gpr_Rx;
              write_memory <= '1';

              case opcode(14 downto 13) is
                when "01" =>            -- auto increment
                  AdderCin16 <= '1';
                  A16 <= ar_Ro;
                  B16 <= "0000000000000000";
                  ar_selRi <= ay;
                  ar_sel8bit <= '0';
                  ar_Ri <= Sum16;
                  write_ar <= '1';
                when "10" =>            -- auto decrement
```

```
                      AdderCin16 <= ’0’;
                      A16 <= ar_Ro;
                      B16 <= "1111111111111111";
                      ar_selRi <= ay;
                      ar_sel8bit <= ’0’;
                      ar_Ri <= Sum16;
                      write_ar <= ’1’;
                    when others =>
                      -- do nothing
                  end case;
                  next_state <= execute;
                end if;

              when "100" => -- Move
                if opcode(9) = ’1’ then
                  rx <= opcode(2 downto 0);
                  ay <= opcode(6 downto 5);

                  -- ayn <- rx
                  gpr_selRx <= rx;
                  ar_selRi <= ay;
                  ar_sel8bit <= ’1’;
                  ar_ByteIn <= gpr_Rx;

                  if opcode(8) = ’1’ then      -- high
                    ar_selHByte <= ’1’;
                  else                         -- low
                    ar_selHByte <= ’0’;
                  end if;

                  write_ar <= ’1’;
                  next_state <= execute;

                elsif opcode(4) = ’1’ then
                  rx <= opcode(7 downto 5);
                  ay <= opcode(1 downto 0);

                  -- rx <- ayn
                  gpr_selRi <= rx;
                  ar_selRo <= ay;

                  if opcode(3) = ’1’ then      -- high
                    gpr_Ri <= ar_Ro(15 downto 8);
                  else                         -- low
                    gpr_Ri <= ar_Ro(7 downto 0);
                  end if;

                  write_gpr <= ’1’;
                  next_state <= execute;

                else
                  rx <= opcode(2 downto 0);
                  ry <= opcode(7 downto 5);

                  -- ry <- rx
                  gpr_SelRx <= rx;
                  gpr_SelRi <= ry;
                  gpr_Ri <= gpr_Rx;

                  write_gpr <= ’1’;
                  next_state <= execute;

                end if;


              when others =>
                -- should not reach here

            end case;

          end if;

        when execute =>
```

```
          if write_memory = '1' then
            if data_ack = '0' then  -- request write
              data_req <= '1';
            else                    -- data written
              data_req <= '0';
        gpr_en <= write_gpr;
        sr_en <= write_sr;
        pc_en <= write_pc;
        ar_en <= write_ar;
              next_state <= fetch;
            end if;
          else
      gpr_en <= write_gpr;
      sr_en <= write_sr;
      pc_en <= write_pc;
      ar_en <= write_ar;
            next_state <= fetch;
          end if;

        when others =>
          -- shouldnt reach here
          next_state <= reset_state;
      end case;
    end if;
  end process;

  process(clock, reset, next_state)
  BEGIN
    if reset = '0' then
      state <= reset_state;
    elsif rising_edge(clock) then
      state <= next_state;
    end if;
  end process;

end Behavioral;
```

Listing 11: processor/fulladder.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:  18:59:20 09/18/2010
-- Design Name:
-- Module Name:  fulladder - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fulladder is
  Port (Ax   : in   STD_LOGIC;
        Bx   : in   STD_LOGIC;
        Ci   : in   STD_LOGIC;
        Sx   : out  STD_LOGIC;
        Co   : out  STD_LOGIC
        );
end fulladder;


architecture arch_fulladder of fulladder is
```

```vhdl
BEGIN
  process(Ax, Bx, Ci)
  BEGIN
    Sx <= (Ax XOR Bx) XOR Ci;
    Co <= (Ax and Bx) or (Ax and Ci) OR (Bx AND Ci);
  end process;
end arch_fulladder;

-------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fulladder8 is
  Port (A     : in   STD_LOGIC_VECTOR( 7 downto 0);
        B     : in   STD_LOGIC_VECTOR( 7 downto 0);
        Cin   : in   STD_LOGIC;
        Sum   : out  STD_LOGIC_VECTOR( 7 downto 0);
        Cout  : out  STD_LOGIC
        );
end fulladder8;

architecture arch_fulladder8 of fulladder8 is
  component fulladder IS
  Port (Ax    : in   STD_LOGIC;
        Bx    : in   STD_LOGIC;
        Ci    : in   STD_LOGIC;
        Sx    : out  STD_LOGIC;
        Co    : out  STD_LOGIC
        );
  end component;
  signal Carry   : std_logic_vector(8 downto 0);
BEGIN
  Carry(0) <= Cin;

    FA0: fulladder PORT MAP(A(0), B(0), Carry(0), Sum(0), Carry(1));
    FA1: fulladder PORT MAP(A(1), B(1), Carry(1), Sum(1), Carry(2));
    FA2: fulladder PORT MAP(A(2), B(2), Carry(2), Sum(2), Carry(3));
    FA3: fulladder PORT MAP(A(3), B(3), Carry(3), Sum(3), Carry(4));
    FA4: fulladder PORT MAP(A(4), B(4), Carry(4), Sum(4), Carry(5));
    FA5: fulladder PORT MAP(A(5), B(5), Carry(5), Sum(5), Carry(6));
    FA6: fulladder PORT MAP(A(6), B(6), Carry(6), Sum(6), Carry(7));
    FA7: fulladder PORT MAP(A(7), B(7), Carry(7), Sum(7), Carry(8));

  Cout <= Carry(8);
end arch_fulladder8;

-------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fulladder16 is
  Port (A     : in   STD_LOGIC_VECTOR( 15 downto 0);
        B     : in   STD_LOGIC_VECTOR( 15 downto 0);
        Cin   : in   STD_LOGIC;
        Sum   : out  STD_LOGIC_VECTOR( 15 downto 0);
        Cout  : out  STD_LOGIC
        );
end fulladder16;

architecture arch_fulladder16 of fulladder16 is
  component fulladder IS
  Port (Ax    : in   STD_LOGIC;
        Bx    : in   STD_LOGIC;
        Ci    : in   STD_LOGIC;
        Sx    : out  STD_LOGIC;
        Co    : out  STD_LOGIC
        );
  end component;
  signal Carry   : std_logic_vector(16 downto 0);
BEGIN
  Carry(0) <= Cin;
```

```
    FA0:  fulladder PORT MAP(A(0), B(0), Carry(0), Sum(0), Carry(1));
    FA1:  fulladder PORT MAP(A(1), B(1), Carry(1), Sum(1), Carry(2));
    FA2:  fulladder PORT MAP(A(2), B(2), Carry(2), Sum(2), Carry(3));
    FA3:  fulladder PORT MAP(A(3), B(3), Carry(3), Sum(3), Carry(4));
    FA4:  fulladder PORT MAP(A(4), B(4), Carry(4), Sum(4), Carry(5));
    FA5:  fulladder PORT MAP(A(5), B(5), Carry(5), Sum(5), Carry(6));
    FA6:  fulladder PORT MAP(A(6), B(6), Carry(6), Sum(6), Carry(7));
    FA7:  fulladder PORT MAP(A(7), B(7), Carry(7), Sum(7), Carry(8));
    FA8:  fulladder PORT MAP(A(8), B(8), Carry(8), Sum(8), Carry(9));
    FA9:  fulladder PORT MAP(A(9), B(9), Carry(9), Sum(9), Carry(10));
    FA10: fulladder PORT MAP(A(10), B(10), Carry(10), Sum(10), Carry(11));
    FA11: fulladder PORT MAP(A(11), B(11), Carry(11), Sum(11), Carry(12));
    FA12: fulladder PORT MAP(A(12), B(12), Carry(12), Sum(12), Carry(13));
    FA13: fulladder PORT MAP(A(13), B(13), Carry(13), Sum(13), Carry(14));
    FA14: fulladder PORT MAP(A(14), B(14), Carry(14), Sum(14), Carry(15));
    FA15: fulladder PORT MAP(A(15), B(15), Carry(15), Sum(15), Carry(16));

  Cout <= Carry(16);
end arch_fulladder16;
```

Listing 12: processor/gpr.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:  18:59:20 09/18/2010
-- Design Name:
-- Module Name:  GPR - gpr_arch
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;
use work.reg8;

entity gpr is
  Port (clk      : in   STD_LOGIC;
        enable   : in   STD_LOGIC;
        SelRx    : in   STD_LOGIC_VECTOR (2 downto 0);  -- The Rx output selection
            value
        SelRy    : in   STD_LOGIC_VECTOR (2 downto 0);  -- The Ry output selection
            value
        SelRi    : in   STD_LOGIC_VECTOR (2 downto 0);  -- The Ri input selection
            value
        SelIn    : in   STD_LOGIC;  -- Select where the input should be from the CU
            or CDB
        RiCU     : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input from the Control
            Unit
        RiCDB    : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input from the Common Data
             Bus
        Rx       : out  STD_LOGIC_VECTOR (7 downto 0);  -- The Rx output
        Ry       : out  STD_LOGIC_VECTOR (7 downto 0)); -- The Ry output
end gpr;


architecture gpr_arch of gpr is
  component reg8 IS
    port(I      : in  std_logic_vector(7 downto 0);
         clock  : in  std_logic;
         enable : in  std_logic;
         reset  : in  std_logic;
         Q      : out std_logic_vector(7 downto 0)
        );
  end component;

  signal  reset: std_logic := '0';
  signal  input: std_logic_VECTOR (7 downto 0);
  signal  R0E  : std_logic;  -- Enable signals
  signal  R1E  : std_logic;
  signal  R2E  : std_logic;
  signal  R3E  : std_logic;
  signal  R4E  : std_logic;
  signal  R5E  : std_logic;
  signal  R6E  : std_logic;
  signal  R7E  : std_logic;
  signal  Q0   : std_logic_VECTOR (7 downto 0);
  signal  Q1   : std_logic_VECTOR (7 downto 0);
  signal  Q2   : std_logic_VECTOR (7 downto 0);
  signal  Q3   : std_logic_VECTOR (7 downto 0);
  signal  Q4   : std_logic_VECTOR (7 downto 0);
  signal  Q5   : std_logic_VECTOR (7 downto 0);
  signal  Q6   : std_logic_VECTOR (7 downto 0);
  signal  Q7   : std_logic_VECTOR (7 downto 0);
BEGIN
    reg_0 : reg8 port map(input, clk, R0E, reset, Q0);
    reg_1 : reg8 port map(input, clk, R1E, reset, Q1);
    reg_2 : reg8 port map(input, clk, R2E, reset, Q2);
    reg_3 : reg8 port map(input, clk, R3E, reset, Q3);
    reg_4 : reg8 port map(input, clk, R4E, reset, Q4);
    reg_5 : reg8 port map(input, clk, R5E, reset, Q5);
    reg_6 : reg8 port map(input, clk, R6E, reset, Q6);
    reg_7 : reg8 port map(input, clk, R7E, reset, Q7);

  -- Select where the input should come from
    SelectInput: process(SelIn, RiCDB, RiCU)
    BEGIN
      IF SelIn = '1' THEN
          input <= RiCDB;
      ELSE
        input <= RiCU;
      END IF;
    END process;

  -- Set Ri the input
  SetInput: process(clk, enable, SelRi)
  BEGIN
    R0E <= '0';
    R1E <= '0';
    R2E <= '0';
    R3E <= '0';
    R4E <= '0';
    R5E <= '0';
    R6E <= '0';
    R7E <= '0';
   IF enable = '1' THEN
      case SelRi IS
        WHEN "000" =>
          R0E <= '1';
        WHEN "001" =>
          R1E <= '1';
        WHEN "010" =>
          R2E <= '1';
        WHEN "011" =>
          R3E <= '1';
        WHEN "100" =>
          R4E <= '1';
        WHEN "101" =>
          R5E <= '1';
        WHEN "110" =>
          R6E <= '1';
        WHEN "111" =>
          R7E <= '1';
        WHEN others =>
          NULL; -- None of them are enabled
      end case;
   END IF;
  end process;

  -- Set the Rx output
  WITH SelRx SELECT
  Rx <= Q0 WHEN "000",
        Q1 WHEN "001",
        Q2 WHEN "010",
        Q3 WHEN "011",
        Q4 WHEN "100",
        Q5 WHEN "101",
        Q6 WHEN "110",
        Q7 WHEN others;

-- Set the Ry output
  WITH SelRy SELECT
  Ry <= Q0 WHEN "000",
        Q1 WHEN "001",
        Q2 WHEN "010",
        Q3 WHEN "011",
        Q4 WHEN "100",
        Q5 WHEN "101",
        Q6 WHEN "110",
        Q7 WHEN others;

end gpr_arch;
```

Listing 13: processor/reg.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    20:08:41 10/11/2010
-- Design Name:
-- Module Name:    register - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
port(I      : in  std_logic_vector(7 downto 0);
     clock  : in  std_logic;
     enable : in  std_logic;
     reset  : in  STD_LOGIC;
     Q      : out std_logic_vector(7 downto 0)
    );
end reg8;

architecture behv of reg8 is
begin

  process(I, clock, enable, reset)
  begin
    IF reset = '1' THEN
      Q <= (others => '0');
    ELSIF rising_edge(clock) then
      if enable = '1' then
```

```
        Q <= I;
      end if;
    end if;

  end process;

end behv;

--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity reg16 is
port(I      : in  std_logic_vector(15 downto 0);
     clock  : in  std_logic;
     enable : in  std_logic;
     reset  : in  STD_LOGIC;
     Q      : out std_logic_vector(15 downto 0)
    );
end reg16;

architecture behv of reg16 is
begin

  process(I, clock, enable, reset)
  begin
    IF reset = '1' THEN
      Q <= (others => '0');
    ELSIF rising_edge(clock) then
      if enable = '1' then
        Q <= I;
      end if;
    end if;

  end process;

end behv;
```

Listing 14: processor/spr.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:  18:59:20 09/18/2010
-- Design Name:
-- Module Name:  sr - sr_arch
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description: The Special Purpose Register is 3, 16bit registers. One for the PC,
-- another for the SR and the third is the IR.
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;
use work.reg16;

entity sr is
  Port (clk      : in  STD_LOGIC;
        enable   : in  STD_LOGIC;
        reset    : in  STD_LOGIC;
        Ri       : in  STD_LOGIC_VECTOR (15 downto 0);  -- The input to the SR
        Ro       : out STD_LOGIC_VECTOR (15 downto 0)); -- The output from SR
```

```
end sr;

architecture sr_arch of sr is
  component reg16 IS
  port(I      : in  std_logic_vector(15 downto 0);
       clock  : in  std_logic;
       enable : in  std_logic;
       reset  : in  STD_LOGIC;
       Q      : out std_logic_vector(15 downto 0)
     );
  end component;
BEGIN
  reg_sr : reg16 port map(Ri, clk, enable, reset, Ro);
end sr_arch;

-----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity pc is
  Port (clk       : in  STD_LOGIC;
        enable    : in  STD_LOGIC;
        reset     : in  STD_LOGIC;
        Ri        : in  STD_LOGIC_VECTOR (15 downto 0);  -- The input to the SR
        Ro        : out STD_LOGIC_VECTOR (15 downto 0)); -- The output from SR
end pc;


architecture pc_arch of pc is
  component reg16 IS
  port(I      : in  std_logic_vector(15 downto 0);
       clock  : in  std_logic;
       enable : in  std_logic;
       reset  : in  STD_LOGIC;
       Q      : out std_logic_vector(15 downto 0)
     );
  end component;
BEGIN
  reg_pc : reg16 port map(Ri, clk, enable, reset, Ro);
end pc_arch;
```

MMU

Listing 15: mmu/control_unit.vhd

```
-- Authors:
--      Wim Looman, Forrest McKerchar

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;
use work.mmu_types.all;

entity mmu_control_unit is
  port (
     eoc                : in  std_logic; -- High on muart has finished collecting data
     eot                : in  std_logic; -- High on muart has finished transmitting
     ready              : in  std_logic; -- High if the muart is ready for new transfer
     data_read          : in  std_logic; -- High if the cpu requests a read, else write
     data_req           : in  std_logic; -- Low to start a transfer
     data_add_0         : in  std_logic; -- High for memory address, else IO
     inst_req           : in  std_logic; -- Low to start a transfer
     fr                 : in  std_logic; -- Input headers fetch request bit
     inst_or_data_in    : in  std_logic; -- Input headers inst or data bit
     rw                 : in  std_logic; -- Input headers read/!write bit
     write              : out std_logic; -- Pulled high to start muart writing data
     inst_or_data_out   : out std_logic; -- Ouput headers inst or data bit
     inst_ack           : out std_logic; -- Idles high, pulled low when data ready
     data_ack           : inout std_logic; -- Idles 'Z', high when data not ready,
         pulled low when data ready
     muart_input        : out muart_input_state; -- Signal connected to muart input
     muart_output       : out muart_output_state; -- Signal connected to muart output
     clk                : in  std_logic
  );
end mmu_control_unit;

architecture mmu_control_unit_arch of mmu_control_unit is
  component data_control_unit is
    port (
     eoc          : in  std_logic; -- High on muart has finished collecting data
     eot          : in  std_logic; -- High on muart has finished transmitting
     ready        : in  std_logic; -- High if the muart is ready for new transfer
     data_read    : in  std_logic; -- High if the cpu requests a read, else write
     data_req     : in  std_logic; -- Low to start a transfer
     data_add_0   : in  std_logic; -- High for memory address, else IO
     write        : out std_logic; -- Pulled high to start muart writing data.
     data_ack     : inout std_logic; -- Idles 'Z', high when data not ready, pulled
         low when data ready
     muart_input  : out muart_input_state; -- Signal connected to muart input
     muart_output : out muart_output_state; -- Signal connected to muart output
     clk          : in  std_logic
    );
  end component;

  component inst_control_unit is
    port (
     eoc          : in  std_logic; -- High on muart has finished collecting data
     eot          : in  std_logic; -- High on muart has finished transmitting
     ready        : in  std_logic; -- High if the muart is ready for new transfer
     inst_req     : in  std_logic; -- Low to start a transfer
     write        : out std_logic; -- Pulled high to start muart writing data
     inst_or_data : out std_logic; -- Ouput headers inst or data bit
     inst_ack     : out std_logic; -- Idles high, pulled low when data ready
     muart_input  : out muart_input_state; -- Signal connected to muart input
     muart_output : out muart_output_state; -- Signal connected to muart output
     clk          : in  std_logic
    );
  end component;

  signal data_write, inst_write, inst_inst_or_data_out : std_logic;
  signal data_muart_input, inst_muart_input : muart_input_state;
  signal data_muart_output, inst_muart_output : muart_output_state;
```

```
begin
  data_cu : data_control_unit port map (
    eoc,
    eot,
    ready,
    data_read,
    data_req,
    data_add_0,
    data_write,
    data_ack,
    data_muart_input,
    data_muart_output,
    clk
  );
  inst_cu : inst_control_unit port map (
    eoc,
    eot,
    ready,
    inst_req,
    inst_write,
    inst_inst_or_data_out,
    inst_ack,
    inst_muart_input,
    inst_muart_output,
    clk
  );

  inst_or_data_out <= inst_inst_or_data_out ;
  write             <= inst_write or data_write;
  muart_input   <= inst_muart_input  when inst_inst_or_data_out = '1' else
                   data_muart_input;
  muart_output <= inst_muart_output when inst_inst_or_data_out = '1' else
                  data_muart_output;
end mmu_control_unit_arch;
```

Listing 16: mmu/data_control_unit.vhd

```
-- Authors:
--      Wim Looman, Forrest McKerchar

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;
use work.mmu_types.all;

entity data_control_unit is
  port (
     eoc          : in  std_logic; -- High on muart has finished collecting data
     eot          : in  std_logic; -- High on muart has finished transmitting
     ready        : in  std_logic; -- High if the muart is ready for new transfer
     data_read    : in  std_logic; -- High if the cpu requests a read, else write
     data_req     : in  std_logic; -- Low to start a transfer
     data_add_0   : in  std_logic; -- High for memory address, else IO
     write        : out std_logic; -- Pulled high to start muart writing data.
     data_ack     : inout std_logic; -- Idles 'Z', high when data not ready, pulled
         low when data ready
     muart_input  : out muart_input_state; -- Signal connected to muart input
     muart_output : out muart_output_state; -- Signal connected to muart output
     clk          : in  std_logic
  );
end data_control_unit;

architecture data_control_unit_arch of data_control_unit is
  type m_state_type is (
    idle,
    send_header, send_add_high, send_add_low, send_data,
    get_header,  get_add_high,  get_add_low,  get_data,
    finished
  );
  type state_type          is (idle, get_data, wait_clear);
  type read_state_type     is (idle, wait_data, read_data, pause, finished);
  type transmit_state_type is (idle, set_data, trans_data, pause, finished);

  signal state,              next_state              : state_type          := idle;
  signal get_state,          next_get_state          : m_state_type        := idle;
  signal reader_state,       next_reader_state       : read_state_type     := idle;
  signal transmitter_state, next_transmitter_state : transmit_state_type := idle;
begin
  data_fsm : process(state, data_req, clk) begin
    if (rising_edge(clk)) then
      case state is
        when idle =>
          if (data_req = '0' and data_add_0 = '1') then
            next_state <= get_data;
          end if;

        when get_data =>
          if (get_state = finished) then
            next_state <= wait_clear;
          end if;

        when wait_clear =>
          if (data_req = '1') then
            next_state <= idle;
          end if;

        when others =>
          NULL;
      end case;
    end if;
  end process data_fsm;

  get_data_fsm : process(state, clk) begin
    if rising_edge(clk) then
      if state = get_data then
        case get_state is
          when idle =>
            next_get_state <= send_header;

          when send_header =>
            if transmitter_state = finished then
              next_get_state <= send_add_low;
            end if;

          when send_add_low =>
            if transmitter_state = finished then
              next_get_state <= send_add_high;
            end if;

          when send_add_high =>
            if transmitter_state = finished then
              if data_read = '1' then
                next_get_state <= get_header;
              else
                next_get_state <= send_data;
              end if;
            end if;

          when send_data =>
            if transmitter_state = finished then
              next_get_state <= finished;
            end if;

          when get_header =>
            if reader_state = finished then
              next_get_state <= get_add_low;
            end if;

          when get_add_low =>
            if reader_state = finished then
              next_get_state <= get_add_high;
            end if;

          when get_add_high =>
            if reader_state = finished then
              next_get_state <= get_data;
            end if;

          when get_data =>
            if reader_state = finished then
              next_get_state <= finished;
            end if;

          when finished =>
            next_get_state <= idle;

          when others =>
            NULL;
        end case;
      end if;
    end if;
  end process get_data_fsm;

  transmit_fsm : process(clk, get_state, transmitter_state, eot) begin
    if rising_edge(clk) then
      if ((get_state = send_header) or
          (get_state = send_add_low) or
          (get_state = send_add_high) or
          (get_state = send_data)) then
        case transmitter_state is
          when idle =>
          if ready = '1' and eot = '0' then
              next_transmitter_state <= set_data;
            end if;

          when set_data =>
            next_transmitter_state <= trans_data;

          when trans_data =>
            next_transmitter_state <= pause;

          when pause =>
            if eot = '1' then
              next_transmitter_state <= finished;
            end if;

          when finished =>
            next_transmitter_state <= idle;

          when others =>
            NULL;
        end case;
      end if;
    end if;
  end process transmit_fsm;

  read_fsm : process(clk, get_state, reader_state, eoc) begin
    if rising_edge(clk) then
      if ((get_state = get_header) or
          (get_state = get_add_low) or
          (get_state = get_add_high) or
          (get_state = get_data)) then
        case reader_state is
          when idle =>
            next_reader_state <= wait_data;

          when wait_data =>
            if eoc = '1' then
              next_reader_state <= read_data;
            end if;

          when read_data =>
            next_reader_state <= pause;

          when pause =>
            if eoc = '0' then
              next_reader_state <= finished;
            end if;

          when finished =>
            next_reader_state <= idle;

          when others =>
            NULL;
        end case;
      end if;
    end if;
  end process read_fsm;

  switch_states : process(
    clk,
    next_state,
    next_get_state,
    next_reader_state,
    next_transmitter_state
  ) begin
    if rising_edge(clk) then
      state              <= next_state;
      get_state          <= next_get_state;
      reader_state       <= next_reader_state;
      transmitter_state <= next_transmitter_state;
    end if;
  end process switch_states;

  -- Outputs
  with state select
    data_ack <= '1' when wait_clear,
                    'Z' when idle,
                    '0' when others;

  with transmitter_state select
    write <= '1' when trans_data,
                  '0' when others;

  muart_input <= idle          when transmitter_state /= set_data      else
                 idle          when transmitter_state /= trans_data    else
                 header        when get_state          = send_header   else
                 data_add_high when get_state          = send_add_high else
                 data_add_low  when get_state          = send_add_low  else
                 data_data     when get_state          = send_data     else
                 idle;

  muart_output <= clear_data when state          = idle       else
                  idle       when reader_state /= read_data   else
                  header     when get_state      = get_header else
                  data_data  when get_state      = get_data   else
                  idle;
end data_control_unit_arch;
```

Listing 17: mmu/header_builder.vhd

```
-- Author:
--      Forrest McKerchar

-- builds a header to feed into the RS-232 link

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;

entity header_builder is
  port (
    read_write : in  std_logic; -- 1 = read, 0 = write
    inst_data  : in  std_logic; -- 1 = inst, 0 = data
    header     : out std_logic_vector(7 downto 0)
  );
end header_builder;
```

```
architecture header_builder_arch of header_builder is
begin
  header(7) <= read_write or inst_data; -- reading from or writing to
                                          -- memory? (can't write
                                          -- instructions)
  header(6) <= '0'; -- reserved
  header(5) <= '0'; -- reserved
  header(4) <= '0'; -- reserved
  header(3) <= '0'; -- diagnostic
  header(2) <= '0'; -- diagnostic;
  header(1) <= '0'; --  1 if data is being retrieved from RS-232 link
  header(0) <= inst_data; -- instruction data or data data?
end header_builder_arch;
```

Listing 18: mmu/header_decoder.vhd

```
-- Author:
--      Forrest McKerchar

-- decodes a header received from the RS-232 link

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;

entity header_decoder is
  port (
    read_write    : out  std_logic; -- 1 = read, 0 = write
    fetch_request : out std_logic;
    inst_data     : out  std_logic; -- 1 = inst, 0 = data
    header        : in std_logic_vector(7 downto 0)
  );
end header_decoder;

architecture header_decoder_arch of header_decoder is
begin
  read_write <= header(7); -- reading or writing? (should be 1 in this case)
  fetch_request <= header(1); -- fetch request? (should be 1n this case)
  inst_data <= header(0); -- instruction data or data data?
end header_decoder_arch;
```

Listing 19: mmu/inst_control_unit.vhd

```
-- Authors:
--      Wim Looman, Forrest McKerchar

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;
use work.mmu_types.all;

entity inst_control_unit is
  port (
     eoc          : in  std_logic; -- High on muart has finished collecting data
     eot          : in  std_logic; -- High on muart has finished transmitting
     ready        : in  std_logic; -- High if the muart is ready for new transfer
     inst_req     : in  std_logic; -- Low to start a transfer
     write        : out std_logic; -- Pulled high to start muart writing data
     inst_or_data : out std_logic; -- Ouput headers inst or data bit
     inst_ack     : out std_logic; -- Idles high, pulled low when data ready
     muart_input  : out muart_input_state; -- Signal connected to muart input
     muart_output : out muart_output_state; -- Signal connected to muart output
     clk          : in  std_logic
  );
end inst_control_unit;

architecture inst_control_unit_arch of inst_control_unit is
  type m_state_type is (
    idle,
    send_header,   send_add_high, send_add_low,
    get_header,    get_add_high,  get_add_low,
    get_data_high, get_data_low,  finished
  );
  type state_type          is (idle, get_data, wait_clear);
  type read_state_type     is (idle, wait_data, read_data, pause, finished);
  type transmit_state_type is (idle, set_data, trans_data, pause, finished);

  signal state,              next_state              : state_type          := idle;
  signal get_state,          next_get_state          : m_state_type        := idle;
  signal reader_state,       next_reader_state       : read_state_type     := idle;
  signal transmitter_state, next_transmitter_state : transmit_state_type := idle;
begin
  inst_fsm : process(state, inst_req, clk) begin
    case state is
      when idle =>
        if (inst_req = '0') then
          next_state <= get_data;
        end if;

      when get_data =>
        if (get_state = finished) then
          next_state <= wait_clear;
        end if;

      when wait_clear =>
        if (inst_req = '1') then
          next_state <= idle;
        end if;

      when others =>
        NULL;
    end case;
  end process inst_fsm;

  get_inst_fsm : process(state, clk) begin
    if state = get_data then
      case get_state is
        when idle =>
          next_get_state <= send_header;

        when send_header =>
          if transmitter_state = finished then
            next_get_state <= send_add_low;
          end if;

        when send_add_low =>
          if transmitter_state = finished then
            next_get_state <= send_add_high;
          end if;

        when send_add_high =>
          if transmitter_state = finished then
            next_get_state <= get_header;
          end if;

        when get_header =>
          if reader_state = finished then
            next_get_state <= get_add_low;
          end if;

        when get_add_low =>
          if reader_state = finished then
            next_get_state <= get_add_high;
          end if;

        when get_add_high =>
          if reader_state = finished then
            next_get_state <= get_data_low;
          end if;

        when get_data_low =>
          if reader_state = finished then
            next_get_state <= get_data_high;
          end if;

        when get_data_high =>
          if reader_state = finished then
            next_get_state <= finished;
          end if;

        when finished =>
          next_get_state <= idle;

        when others =>
          NULL;
      end case;
    end if;
  end process get_inst_fsm;

  transmit_fsm : process(clk, get_state, transmitter_state, eot) begin
    if ((get_state = send_header) or
        (get_state = send_add_low) or
        (get_state = send_add_high)) then
      case transmitter_state is
        when idle =>
          if ready = '1' and eot = '0' then
            next_transmitter_state <= set_data;
          end if;

        when set_data =>
          next_transmitter_state <= trans_data;

        when trans_data =>
          next_transmitter_state <= pause;

        when pause =>
          if eot = '1' then
            next_transmitter_state <= finished;
          end if;

        when finished =>
          next_transmitter_state <= idle;

        when others =>
          NULL;
      end case;
    end if;
  end process transmit_fsm;

  read_fsm : process(clk, get_state, reader_state, eoc) begin
    if ((get_state = get_header) or
        (get_state = get_add_low) or
        (get_state = get_add_high) or
        (get_state = get_data_low) or
        (get_state = get_data_high)) then
      case reader_state is
        when idle =>
          next_reader_state <= wait_data;

        when wait_data =>
          if eoc = '1' then
            next_reader_state <= read_data;
          end if;

        when read_data =>
          next_reader_state <= pause;

        when pause =>
          if eoc = '0' then
            next_reader_state <= finished;
          end if;

        when finished =>
          next_reader_state <= idle;
        when others =>
          NULL;
      end case;
    end if;
  end process read_fsm;

  switch_states : process(
    clk,
    next_state,
    next_get_state,
    next_reader_state,
    next_transmitter_state) begin
    if rising_edge(clk) then
      state <= next_state;
      get_state <= next_get_state;
      reader_state <= next_reader_state;
      transmitter_state <= next_transmitter_state;
    end if;
  end process switch_states;

  -- Outputs
  with state select
    inst_ack <= '1' when wait_clear,
                '0' when others;

  with state select
    inst_or_data <= '0' when idle,
                    '1' when others;

  with transmitter_state select
    write <= '1' when trans_data,
             '0' when others;


  muart_input <= idle          when transmitter_state /= set_data      else
                 idle          when transmitter_state /= trans_data    else
                 header        when get_state          = send_header   else
                 inst_add_high when get_state          = send_add_high else
                 inst_add_low  when get_state          = send_add_low  else
                 idle;

  muart_output <= idle           when reader_state /= read_data     else
                  header         when get_state     = get_header    else
                  inst_data_high when get_state     = get_data_high else
                  inst_data_low  when get_state     = get_data_low  else
                  idle;
end inst_control_unit_arch;
```

Listing 20: mmu/mmu.vhd

```
-- Authors:
--       Wim Looman, Forrest McKerchar

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;
use work.mmu_types.all;
use work.mmu_control_unit;
use work.header_builder;
use work.header_decoder;
use work.reg8;

use work.minimal_uart_core;

  entity mmu_main is
    port (
      -- instruction bus
      inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
      inst_data : out std_logic_vector(15 downto 0); -- Data lines.
      inst_req  : in  std_logic;                     -- Pulled low to request bus
          usage.
      inst_ack  : out std_logic;                     -- Pulled high to inform of
          request completion.
      -- data bus
      data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
      data_line : inout std_logic_vector(7 downto 0);  -- Data lines.
      data_read : in    std_logic;                     -- High for a read request,
          low for a write request.
      data_req  : in    std_logic;                     -- Pulled low to request bus
          usage.
      data_ack  : inout std_logic;                     -- Pulled high to inform of
          request completion.
      -- extras
      clk           : in  std_logic;
      receive_pin   : in  std_logic;
      transfer_pin  : out std_logic
    );
  end mmu_main;

  architecture mmu_arch of mmu_main is
    component mmu_control_unit is
      port (
        eoc               : in  std_logic; -- High on muart has finished collecting
            data
        eot               : in  std_logic; -- High on muart has finished transmitting
        ready             : in  std_logic; -- High if the muart is ready for new
            transfer
        data_read         : in  std_logic; -- High if the cpu requests a read, else
            write
        data_req          : in  std_logic; -- Low to start a transfer
        data_add_0        : in  std_logic; -- High for memory address, else IO
        inst_req          : in  std_logic; -- Low to start a transfer
        fr                : in  std_logic; -- Input headers fetch request bit
        inst_or_data_in   : in  std_logic; -- Input headers inst or data bit
        rw                : in  std_logic; -- Input headers read/!write bit
        write             : out std_logic; -- Pulled high to start muart writing data
        inst_or_data_out  : out std_logic; -- Ouput headers inst or data bit
        inst_ack          : out std_logic; -- Idles high, pulled low when data ready
        data_ack          : inout std_logic; -- Idles 'Z', high when data not ready,
            pulled low when data ready
        muart_input       : out muart_input_state; -- Signal connected to muart input
        muart_output      : out muart_output_state; -- Signal connected to muart
            output
        clk               : in  std_logic
      );
    end component;

    component header_builder is
      port (
        read_write : in  std_logic; -- 1 = read, 0 = write
        inst_data  : in  std_logic; -- 1 = inst, 0 = data
        header     : out std_logic_vector(7 downto 0)
      );
    end component;

    component header_decoder is
      port (
        read_write    : out  std_logic; -- 1 = read, 0 = write
        fetch_request : out std_logic;
        inst_data     : out  std_logic; -- 1 = inst, 0 = data
        header        : in std_logic_vector(7 downto 0)
      );
    end component;

    component minimal_uart_core is
      port(
        clock : in    std_logic;
        eoc   : out   std_logic;
        outp  : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
        rxd   : in    std_logic;
        txd   : out   std_logic;
        eot   : out   std_logic;
        inp   : in    std_logic_vector(7 downto 0);
        ready : out   std_logic;
        wr    : in    std_logic
      );
    end component;

    component reg8 IS
      port(
        I      : in  std_logic_vector(7 downto 0);
        clock  : in  std_logic;
        enable : in  std_logic;
        reset  : in  std_logic;
        Q      : out std_logic_vector(7 downto 0)
      );
    end component;


    signal eoc           : std_logic;
    signal eot           : std_logic;
    signal ready         : std_logic;
    signal fr            : std_logic;
    signal inst_or_data_in : std_logic;
    signal rw            : std_logic;

    signal write         : std_logic;
    signal inst_or_data_out : std_logic;
    signal muart_input   : muart_input_state;
    signal muart_output  : muart_output_state;

    signal muart_out : std_logic_vector(7 downto 0);
    signal muart_in  : std_logic_vector(7 downto 0);
    signal header_in   : std_logic_vector(7 downto 0);
    signal header_out  : std_logic_vector(7 downto 0);
    signal inst_data_high_enable : std_logic;
    signal inst_data_low_enable : std_logic;
    signal data_data_enable : std_logic;
    signal data_line_tri : std_logic_vector(7 downto 0);

    begin
    muart : minimal_uart_core port map (
      clk,
      eoc,
      muart_out,
      receive_pin,
      transfer_pin,
      eot,
      muart_in,
      ready,
      write
    );
    cu  : mmu_control_unit port map (
      eoc,
      eot,
      ready,
      data_read,
      data_req,
      data_add(0),
      inst_req,
      fr,
      inst_or_data_in,
      rw,
      write,
      inst_or_data_out,
      inst_ack,
      data_ack,
      muart_input,
      muart_output,
      clk
    );
    hb  : header_builder port map (
      data_read,
      inst_or_data_out,
      header_out
    );
    hd  : header_decoder port map (
      rw,
      fr,
      inst_or_data_in,
      header_in
    );
    idh : reg8 port map (
      muart_out,
      clk,
      inst_data_high_enable,
      '0',
      inst_data(15 downto 8)
    );
    idl : reg8 port map (
      muart_out,
      clk,
      inst_data_low_enable,
      '0',
      inst_data(7  downto 0)
    );
    dd  : reg8 port map (
      muart_out,
      clk,
      data_data_enable,
      '0',
      data_line_tri
    );

    with muart_input select
      muart_in <=  header_out                         when header,
                   "0000" & inst_add(11 downto 8) when inst_add_high,
                   inst_add(7  downto 0)              when inst_add_low,
                   '0' & data_add(15 downto 9)        when data_add_high,
                   data_add(8  downto 1)              when data_add_low,
                   data_line                          when data_data,
                   (others => '0')                    when others;

    route_output : process(muart_output, muart_out) begin
      header_in <= (others => '0');
      inst_data_high_enable <= '0';
      inst_data_low_enable <= '0';
      data_data_enable <= '0';
      case muart_output is
        when header =>
          header_in <= muart_out;

        when inst_data_high =>
          inst_data_high_enable <= '1';

        when inst_data_low =>
          inst_data_low_enable <= '1';

        when data_data =>
          data_data_enable <= '1';
        when others =>
          NULL;
      end case;
    end process;

    data_line <= data_line_tri when data_add(0) = '1' and data_read = '1' else
                 (others => 'Z');

  end mmu_arch;
```

Listing 21: mmu/mmu_types.vhd

```
-- Authors:
--      Wim Looman, Forrest McKerchar

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

```
package mmu_types is
   type muart_input_state is (header, inst_add_high, inst_add_low, data_add_high,
       data_add_low, data_data, idle);
   type muart_output_state is (header, inst_data_high, inst_data_low, data_data,
       clear_data, idle);
end mmu_types;
```

Listing 22: mmu/muart/BRG.vhd

```
--**************************************************************************
--*  Minimal UART ip core                                                  *
--* Author: Arao Hayashida Filho        arao@medinovacao.com.br            *
--*                                                                        *
--**************************************************************************
--*                                                                        *
--* Copyright (C) 2009 Arao Hayashida Filho                                *
--*                                                                        *
--* This source file may be used and distributed without                   *
--* restriction provided that this copyright statement is not              *
--* removed from the file and that any derivative work contains            *
--* the original copyright notice and the associated disclaimer.           *
--*                                                                        *
--* This source file is free software; you can redistribute it             *
--* and/or modify it under the terms of the GNU Lesser General             *
--* Public License as published by the Free Software Foundation;           *
--* either version 2.1 of the License, or (at your option) any             *
--* later version.                                                         *
--*                                                                        *
--* This source is distributed in the hope that it will be                 *
--* useful, but WITHOUT ANY WARRANTY; without even the implied             *
--* warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR                *
--* PURPOSE.  See the GNU Lesser General Public License for more           *
--* details.                                                               *
--*                                                                        *
--* You should have received a copy of the GNU Lesser General              *
--* Public License along with this source; if not, download it             *
--* from http://www.opencores.org/lgpl.shtml                               *
--*                                                                        *
--**************************************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned."+";

entity br_generator is
  generic (divider_width: integer := 16);
  port (
    clock      : in  std_logic;
    rx_enable  : in  std_logic;
    clk_txd    : out std_logic;
    tx_enable  : in  std_logic;
    clk_serial : out std_logic
  );
end br_generator;

architecture principal of br_generator is
  -- change the following constant to your desired baud rate
  -- one hz equal to one bit per second
  signal   count_brg     : std_logic_vector(divider_width - 1 downto 0) := (others =>
      '0');
  signal   count_brg_txd : std_logic_vector(divider_width - 1 downto 0) := (others =>
      '0');
  constant brdvd         : std_logic_vector(divider_width - 1 downto 0) := x"0516";
      -- 38400 bps @ 50MHz

  begin
    txd : process (clock)
    begin
      if (rising_edge(clock)) then
        if (count_brg_txd = brdvd) then
```

```
          clk_txd       <= '1';
          count_brg_txd <= (others => '0');
        elsif (tx_enable = '1') then
          clk_txd       <= '0';
          count_brg_txd <= count_brg_txd + 1;
        else
          clk_txd       <= '0';
          count_brg_txd <= (others => '0');
        end if;
      end if;
    end process txd;

    rxd : process (clock)
    begin
      if (rising_edge(clock)) then
        if (count_brg=brdvd) then
          count_brg   <= (others => '0');
          clk_serial  <= '1';
        elsif (rx_enable = '1') then
          count_brg   <= count_brg+1;
          clk_serial  <= '0';
        else
          count_brg   <= '0' & brdvd(divider_width - 1 downto 1);
          clk_serial  <= '0';
        end if;
      end if;
    end process rxd;
end principal;
```

Listing 23: mmu/muart/serial.vhd

```
--*************************************************************************
--*  Minimal UART ip core                                                 *
--* Author: Arao Hayashida Filho        arao@medinovacao.com.br           *
--*                                                                       *
--*************************************************************************
--*                                                                       *
--* Copyright (C) 2009 Arao Hayashida Filho                               *
--*                                                                       *
--* This source file may be used and distributed without                 *
--* restriction provided that this copyright statement is not             *
--* removed from the file and that any derivative work contains           *
--* the original copyright notice and the associated disclaimer.          *
--*                                                                       *
--* This source file is free software; you can redistribute it            *
--* and/or modify it under the terms of the GNU Lesser General            *
--* Public License as published by the Free Software Foundation;          *
--* either version 2.1 of the License, or (at your option) any            *
--* later version.                                                        *
--*                                                                       *
--* This source is distributed in the hope that it will be                *
--* useful, but WITHout ANY WARRANTY; without even the implied            *
--* warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR               *
--* PURPOSE.  See the GNU Lesser General Public License for more          *
--* details.                                                              *
--*                                                                       *
--* You should have received a copy of the GNU Lesser General             *
--* Public License along with this source; if not, download it            *
--* from http://www.opencores.org/lgpl.shtml                              *
--*                                                                       *
--*************************************************************************

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity minimal_uart_core is
  port(
    clock : in    std_logic;
    eoc   : out   std_logic;
    outp  : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
    rxd   : in    std_logic;
    txd   : out   std_logic;
    eot   : out   std_logic;
    inp   : in    std_logic_vector(7 downto 0);
    ready : out   std_logic;
    wr    : in    std_logic
  );
end minimal_uart_core;

architecture principal of minimal_uart_core  is
  type state is (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9);
  signal clk_serial        : std_logic := '0';
  signal start             : std_logic := '0';
  signal eocs, eoc1, eoc2  : std_logic := '0';
  signal rx_ck_enable      : std_logic := '0';
  signal receiving         : std_logic := '0';
  signal transmitting      : std_logic := '0';
  signal clk_txd           : std_logic := '0';
  signal txds              : std_logic := '1';
  signal eots              : std_logic := '0';
  signal inpl              : std_logic_vector(7 downto 0) := x"00";
  signal data              : std_logic_vector(7 downto 0) := x"00";
  signal atual_state, next_state, atual_state_txd, next_state_txd: state := s0;
  signal tx_enable         : std_logic := '0';
  signal tx_ck_enable      : std_logic := '0';

  component br_generator
    port (
      clock      : in  std_logic;
      rx_enable  : in  std_logic;
      clk_txd    : out std_logic;
      tx_enable  : in  std_logic;
      clk_serial : out std_logic
    );
  end component;

  begin
    ready <= not(tx_enable);
    brg : br_generator port map (clock, rx_ck_enable, clk_txd, tx_ck_enable,
        clk_serial);
    rx_ck_enable <= start or receiving;
    tx_ck_enable <= tx_enable or transmitting;

    start_detect : process(rxd, eocs)
    begin
      if (eocs = '1') then
        start <= '0';
      elsif (falling_edge(rxd)) then
        start <= '1';
      end if;
    end process start_detect;

    rxd_states : process (clk_serial)
    begin
      if (rising_edge(clk_serial)) then
        atual_state <= next_state;
      end if;
    end process rxd_states;

    rxd_state_machine : process(start, atual_state)
    begin
      if (start = '1' or receiving = '1') then
        case atual_state is
          when s0 =>
            eocs <= '0';
            if (start = '1') then
              next_state <= s1;
              receiving  <= '1';
            else
              next_state <= s0;
              receiving  <= '0';
            end if;

          when s1 =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s2;

          when s2   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s3;

          when s3   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s4;

          when s4   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s5;

          when s5   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s6;

          when s6   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s7;

          when s7   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s8;

          when s8   =>
            receiving  <= '1';
            eocs       <= '0';
            next_state <= s9;

          when s9   =>
            receiving  <= '1';
            eocs       <= '1';
            next_state <= s0;

          when others =>
            null;

        end case;
      end if;
    end process rxd_state_machine;

    rxd_shift : process(clk_serial)
    begin
      if (rising_edge(clk_serial)) then
        if (eocs = '0') then
          data <= rxd & data(7 downto 1);
        end if;
      end if;
    end process rxd_shift;

    process (clock)
    begin
      if (rising_edge(clock)) then
        eoc <= eocs;
      end if;
    end process;

    process(atual_state)
    begin
      if (atual_state=s9) then
        outp <= data;
      end if;
    end process;
```

```
txd_states : process(clk_txd)
begin
  if (rising_edge(clk_txd)) then
    atual_state_txd <= next_state_txd;
  end if;
end process txd_states;

txd_state_machine : process(atual_state_txd , tx_enable)
begin
  case atual_state_txd is
    when s0 =>
      inpl <= inp;
      eots <= '0';
      if (tx_enable = '1') then
        txds           <= '0';
        transmitting   <= '1';
        next_state_txd <= s1;
      else
        txds           <= '1';
        transmitting   <= '0';
        next_state_txd <= s0;
      end if;

    when s1 =>
      txds           <= inpl(0);
      eots           <= '0';
      transmitting   <= '1';
      next_state_txd <= s2;

    when s2 =>
      txds           <= inpl(1);
      eots           <= '0';
      transmitting   <= '1';
      next_state_txd <= s3;

    when s3 =>
      txds           <= inpl(2);
      eots           <= '0';
      transmitting   <= '1';
      next_state_txd <= s4;

    when s4 =>
      txds           <= inpl(3);
      eots           <= '0';
      transmitting   <= '1';
      next_state_txd <= s5;

    when s5 =>
      txds           <= inpl(4);
      eots           <= '0';
      transmitting   <= '1';
      next_state_txd <= s6;

    when s6 =>
      txds           <= inpl(5);
      eots           <= '0';
      transmitting   <= '1';
      next_state_txd <= s7;

    when s7 =>
      txds           <= inpl(6);
      eots           <= '0';
      transmitting   <= '1';
      next_state_txd <= s8;

    when s8 =>
      txds           <= inpl(7);
      eots           <= '0';
      transmitting   <= '1';
      next_state_txd <= s9;

    when s9 =>
      txds           <= '1';
```

```
          eots            <= '1';
          transmitting    <= '1';
          next_state_txd  <= s0;

        when others =>
          null;

      end case;
    end process txd_state_machine;

    tx_start:process (clock, wr,  eots)
    begin
      if (eots = '1') then
        tx_enable <= '0';
      elsif (falling_edge(clock)) then
        if (wr = '1') then
          tx_enable <= '1';
        end if;
      end if;
    end process tx_start;

    eot<=eots;

    process (clock)
    begin
      if (rising_edge(clock)) then
        txd <= txds;
      end if;
    end process;

end principal ;
```

Test Benches

Listing 24: processor/alu_tb.vhd

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

--  A testbench has no ports.
entity alu_tb is
  end alu_tb;

architecture behav of alu_tb is
  --  Declaration of the component that will be instantiated.
  component alu
   Port (f    : in   STD_LOGIC_VECTOR (3 downto 0);  -- Function (opcode)
         rx   : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input x (Rx)
         ry   : in   STD_LOGIC_VECTOR (7 downto 0);  -- Input y (Ry)
         ro   : out  STD_LOGIC_VECTOR (7 downto 0);  -- Output Normaly (Ry)
         Cin  : in   STD_LOGIC;                      -- Carry in
         sr   : out  STD_LOGIC_VECTOR (2 downto 0)); -- Status register out Z(0),
             C(1), N(2)
  end component;
  --  Specifies which entity is bound with the component.
  for alu_0: alu use entity work.alu;
    signal f            : STD_LOGIC_VECTOR (3 downto 0);
    signal rx, ry, ro : STD_LOGIC_VECTOR (7 downto 0);
    signal Cin          : STD_LOGIC;
    signal sr           : STD_LOGIC_VECTOR (2 downto 0);
  begin
    --  Component instantiation.
    alu_0: alu port map (f => f, rx => rx, ry => ry, ro => ro, Cin => Cin, sr => sr);

    --  This process does the real job.
    process
    type pattern_type is record
      f          : STD_LOGIC_VECTOR (3 downto 0);
      rx, ry     : STD_LOGIC_VECTOR (7 downto 0);
      ro         : STD_LOGIC_VECTOR (7 downto 0);
      Cin        : STD_LOGIC;
      sr         : STD_LOGIC_VECTOR (2 downto 0);
    end record;
  --  The patterns to apply.
  type pattern_array is array (natural range <>) of pattern_type;
  constant patterns : pattern_array :=
  -- f       rx          ry          ro          Cin   sr
  (("0001", "00000000", "00000000", "00000000", '0', "001"), --AND tests - 1ns
   ("0001", "00000001", "00000001", "00000001", '0', "000"), --AND tests
   ("0001", "00000000", "00000001", "00000000", '0', "001"), --AND tests
   ("0001", "10101010", "10101010", "10101010", '0', "100"), --AND tests
   ("0001", "01010101", "01010101", "01010101", '0', "000"), --AND tests - 5ns
   ("0001", "11111111", "00000000", "00000000", '0', "001"), --AND tests
   ("0001", "11111111", "11111111", "11111111", '0', "100"), --AND tests
   ("0001", "00000000", "01010101", "00000000", '0', "001"), --AND tests
   ("0001", "00000000", "10101010", "00000000", '0', "001"), --AND tests
   ("0001", "11111111", "01010101", "01010101", '0', "000"), --AND tests - 10 ns
   ("0001", "11111111", "10101010", "10101010", '0', "100"), --AND tests
   ("0001", "10000011", "10110010", "10000010", '0', "100"), --AND tests
   ("0001", "00000011", "00110010", "00000010", '0', "000"), --AND tests

   ("0011", "00000000", "00000000", "00000000", '0', "001"), --OR tests - 14 ns
   ("0011", "00000001", "00000001", "00000001", '0', "000"), --OR tests
   ("0011", "00000000", "00000001", "00000001", '0', "000"), --OR tests
   ("0011", "10101010", "10101010", "10101010", '0', "100"), --OR tests
   ("0011", "01010101", "01010101", "01010101", '0', "000"), --OR tests
   ("0011", "11111111", "00000000", "11111111", '0', "100"), --OR tests
   ("0011", "11111111", "11111111", "11111111", '0', "100"), --OR tests - 20 ns
   ("0011", "00000000", "01010101", "01010101", '0', "000"), --OR tests
   ("0011", "00000000", "10101010", "10101010", '0', "100"), --OR tests
   ("0011", "11111111", "01010101", "11111111", '0', "100"), --OR tests
   ("0011", "11111111", "10101010", "11111111", '0', "100"), --OR tests
   ("0011", "10000011", "10110010", "10110011", '0', "100"), --OR tests - 25 ns
   ("0011", "00000011", "00110010", "00110011", '0', "000"), --OR tests
```

```
    ("0101", "00000000", "00000000", "11111111", '0', "100"), --NOT tests - ry should
        not matter
    ("0101", "00000001", "00000001", "11111110", '0', "100"), --NOT tests
    ("0101", "00000000", "00000001", "11111111", '0', "100"), --NOT tests
    ("0101", "10101010", "10101010", "01010101", '0', "000"), --NOT tests - 30 ns
    ("0101", "01010101", "01010101", "10101010", '0', "100"), --NOT tests
    ("0101", "11111111", "00000000", "00000000", '0', "001"), --NOT tests
    ("0101", "11111111", "11111111", "00000000", '0', "001"), --NOT tests
    ("0101", "00000000", "01010101", "11111111", '0', "100"), --NOT tests
    ("0101", "00000000", "10101010", "11111111", '0', "100"), --NOT tests - 35 ns
    ("0101", "11111111", "01010101", "00000000", '0', "001"), --NOT tests
    ("0101", "11111111", "10101010", "00000000", '0', "001"), --NOT tests
    ("0101", "10000011", "10110010", "01111100", '0', "000"), --NOT tests
    ("0101", "00000011", "00110010", "11111100", '0', "100"), --NOT tests - 39 ns

    ("0111", "00000000", "00000000", "00000000", '0', "001"), --XOR tests - 40 ns
    ("0111", "00000001", "00000001", "00000000", '0', "001"), --XOR tests
    ("0111", "00000000", "00000001", "00000001", '0', "000"), --XOR tests
    ("0111", "10101010", "10101010", "00000000", '0', "001"), --XOR tests
    ("0111", "01010101", "01010101", "00000000", '0', "001"), --XOR tests
    ("0111", "11111111", "00000000", "11111111", '0', "100"), --XOR tests - 45 ns
    ("0111", "11111111", "11111111", "00000000", '0', "001"), --XOR tests
    ("0111", "00000000", "01010101", "01010101", '0', "000"), --XOR tests
    ("0111", "00000000", "10101010", "10101010", '0', "100"), --XOR tests
    ("0111", "11111111", "01010101", "10101010", '0', "100"), --XOR tests
    ("0111", "11111111", "10101010", "01010101", '0', "000"), --XOR tests - 50 ns
    ("0111", "10000011", "10110010", "00110001", '0', "000"), --XOR tests
    ("0111", "00000011", "00110010", "00110001", '0', "000")  --XOR tests
  );
begin
  --  Check each pattern.
  for i in patterns'range loop
    --  Set the inputs.
    Cin <= patterns(i).Cin;
    f <= patterns(i).f;
    rx <= patterns(i).rx;
    ry <= patterns(i).ry;
    --  Wait for the results.
    wait for 1 ns;
    --  Check the outputs.
    assert ro = patterns(i).ro
    report "bad output register value" severity error;
    assert sr = patterns(i).sr
    report "bad status register value" severity error;
    assert sr(0) = patterns(i).sr(0)
    report " *Zero is incorrect" severity error;
    assert sr(1) = patterns(i).sr(1)
    report " *Carry is incorrect" severity error;
    assert sr(2) = patterns(i).sr(2)
    report " *Negitive is incorrect" severity error;
  end loop;
  assert false report "end of test" severity note;
  --  Wait forever; this will finish the simulation.
  wait;
end process;
end behav;
```

Listing 25: processor/fulladder_tb.vhd

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

--  A testbench has no ports.
entity fulladder8_tb is
  end fulladder8_tb;

architecture behav of fulladder8_tb is
  --  Declaration of the component that will be instantiated.
  component fulladder8
  Port (A    : in   STD_LOGIC_VECTOR( 7 downto 0);
        B    : in   STD_LOGIC_VECTOR( 7 downto 0);
        Cin   : in    STD_LOGIC;
        Sum   : out   STD_LOGIC_VECTOR( 7 downto 0);
        Cout  : out   STD_LOGIC
        );
  end component;
  --  Specifies which entity is bound with the component.
  for fulladder8_0: fulladder8 use entity work.fulladder8;
    signal A,B,Sum     : STD_LOGIC_VECTOR (7 downto 0);
    signal Cin,Cout    : STD_LOGIC;
  begin
    --  Component instantiation.
    fulladder8_0: fulladder8 port map (A => A, B => B, Cin => Cin, Sum => Sum, Cout
        => Cout);

    --  This process does the real job.
    process
    type pattern_type is record
      A     :  STD_LOGIC_VECTOR( 7 downto 0);
      B     :  STD_LOGIC_VECTOR( 7 downto 0);
      Cin   :  STD_LOGIC;
      Sum   :  STD_LOGIC_VECTOR( 7 downto 0);
      Cout  :  STD_LOGIC;
    end record;
  --  The patterns to apply.
  type pattern_array is array (natural range <>) of pattern_type;
  constant patterns : pattern_array :=
  -- A            B           Cin    Sum      Cout
  (("00000000", "00000000", '0', "00000000", '0'), --AND tests - 1ns
   ("11111111", "11111111", '1', "11111111", '1'), --AND tests
   ("00000000", "00000000", '1', "00000001", '0'), --AND tests
   ("00000000", "11111111", '0', "11111111", '0'), --AND tests
   ("11111111", "00000000", '0', "11111111", '0'), --AND tests
   ("11111111", "00000000", '1', "00000000", '1'), --AND tests
   ("10101010", "01010101", '0', "11111111", '0'), --AND tests
   ("10101010", "01010101", '1', "00000000", '1'), --AND tests
   ("11111111", "11111111", '0', "11111110", '1')  --XOR tests
  );
begin
  --  Check each pattern.
  for i in patterns'range loop
    --  Set the inputs.
    A    <= patterns(i).A;
    B    <= patterns(i).B;
    Cin  <= patterns(i).Cin;
    --  Wait for the results.
    wait for 1 ns;
    --  Check the outputs.
    assert Sum = patterns(i).Sum
    report "The sum check failed" severity error;
    assert Cout = patterns(i).Cout
    report "The carry out is wrong" severity error;
  end loop;
  assert false report "end of test" severity note;
  --  Wait forever; this will finish the simulation.
  wait;
end process;
end behav;
```

Listing 26: processor/spr_tb.vhd

```
-----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
--use ieee.std_logic_unsigned.all;
--use ieee.std_logic_arith.all;

entity spr_TB is       -- entity declaration
  end spr_TB;

architecture TB of spr_TB is

  component sr
  Port (clk       : in   STD_LOGIC;
        enable    : in   STD_LOGIC;                        -- Enable write
        reset     : in   STD_LOGIC;                        -- Reset the register
        Ri        : in   STD_LOGIC_VECTOR (15 downto 0);   -- The input to the SPR
        Ro        : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output from SPR
  end component;

  signal sr_enable : std_logic;
  signal sr_reset  : std_logic;
  signal sr_Ri     : std_logic_vector(15 downto 0);
  signal sr_Ro     : std_logic_vector(15 downto 0);

  component pc
  Port (clk       : in   STD_LOGIC;
        enable    : in   STD_LOGIC;                        -- Enable write
        reset     : in   STD_LOGIC;                        -- Reset the register
        Ri        : in   STD_LOGIC_VECTOR (15 downto 0);   -- The input to the SPR
        Ro        : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output from SPR
  end component;

  signal pc_enable : std_logic;
  signal pc_reset  : std_logic;
  signal pc_Ri     : std_logic_vector(15 downto 0);
  signal pc_Ro     : std_logic_vector(15 downto 0);

  signal T_clk  : std_logic;

begin

  U_sr: sr port map (clk => T_clk, enable => sr_enable, reset => sr_reset, Ri =>
      sr_Ri, Ro => sr_Ro);
  U_pc: pc port map (clk => T_clk, enable => pc_enable, reset => pc_reset, Ri =>
      pc_Ri, Ro => pc_Ro);

    -- concurrent process to offer the clk signal
  process
  begin
    T_clk <= '0';
    wait for 5 ns;
    T_clk <= '1';
    wait for 5 ns;
  end process;

  process

    variable err_cnt: integer :=0;

  begin

    -- Write
    sr_enable <= '1';
    sr_reset  <= '0';
    sr_Ri     <= "0100011001011001";
    pc_enable <= '1';
    pc_reset  <= '0';
    pc_Ri     <= "0101011010110100";
    wait for 20 ns;

    -- Read
    assert (sr_Ro="0100011001011001") report "Read sr #1 failed" severity error;
    assert (pc_Ro="0101011010110100") report "Read pc #1 failed" severity error;

    -- Change Ri
    sr_Ri <= "1001100101110100";
    pc_Ri <= "0001010001110000";
    wait for 20 ns;
    assert (sr_Ro = "1001100101110100") report "Read sr #2 failed" severity error;
    assert (pc_Ro = "0001010001110000") report "Read pc #2 failed" severity error;

    -- Disable sr, pc still enabled
    sr_enable <= '0';
    sr_Ri <= "0101010101010101";
    pc_Ri <= "1010101010101010";
    wait for 20 ns;
    assert (sr_Ro = "1001100101110100") report "Wrote to sr while disabled" severity
        error;
    assert (pc_Ro = "1010101010101010") report "Read pc #3 failed" severity error;

    -- Enable sr
    sr_enable <= '1';
    wait for 20 ns;
    assert (sr_Ro = "0101010101010101") report "Read sr #3 failed" severity error;

    -- Disable pc, sr still enabled
    pc_enable <= '0';
    sr_Ri <= "0000000011111111";
    pc_Ri <= "1111111100000000";
    wait for 20 ns;
    assert (sr_Ro = "0000000011111111") report "Read sr #4 failed" severity error;
    assert (pc_Ro = "1010101010101010") report "Wrote to pc while disabled" severity
        error;

    -- Enable pc
    pc_enable <= '1';
    wait for 20 ns;
    assert (pc_Ro = "1111111100000000") report "Read pc #4 failed" severity error;


    assert false report "End of test" severity note;
    wait; -- wait forever to end the test

  end process;

end TB;

------------------------------------------------------------------
configuration CFG_TB of spr_TB is
  for TB
  end for;
end CFG_TB;
```

Listing 27: processor/gpr_tb.vhd

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

--  A testbench has no ports.
entity gpr_tb is
  end gpr_tb;

architecture behav of gpr_tb is
  --  Declaration of the component that will be instantiated.
  component gpr
  Port(clk                  : IN std_logic;                       -- Clock
       enable               : IN std_logic;                       -- Enable input
           (output is always enabled)
       SelRx, SelRy, SelRi  : IN std_logic_vector(2 DOWNTO 0);  -- Selecti which
           registers to use
       Ri                   : IN std_logic_vector(7 DOWNTO 0);  -- Input
       Rx, Ry               : OUT std_logic_vector(7 DOWNTO 0)); -- Outputs
  end component;
  --  Specifies which entity is bound with the component.
  for gpr_0: gpr use entity work.gpr;
  signal clk, enable           : std_logic;
  signal SelRx, SelRy, SelRi   : std_logic_vector(2 DOWNTO 0);
  signal Ri, Rx, Ry            : std_logic_vector(7 DOWNTO 0);
begin
  --  Component instantiation.
  gpr_0: gpr port map (clk => clk, enable => enable, SelRx => SelRx, SelRy => SelRy,
      SelRi => SelRi, Ri => Ri, Rx => Rx, Ry => Ry);

  -- Does the clock signal
  process
  begin
    clk <= '0';
    wait for 5 ns;
    clk <= '1';
    wait for 5 ns;
  end process;

  --  This process does the real job.
  process
  begin

    -- Write to R0
    SelRi <= "000";
    Ri <= "00010100";
    enable <= '1';
    wait for 20 ns;

    -- Read R0 from Rx
    SelRx <= "000";
    wait for 20 ns;
    assert (Rx = "00010100") report "Read from Rx failed #1" severity error;

    -- Read R0 from Ry
    SelRy <= "000";
    wait for 20 ns;
    assert (Ry = "00010100") report "Read from Ry failed #1" severity error;

    -- Disable write
    enable <= '0';
    wait for 20 ns;

    -- Change Ri (should not write as it is disabled)
    Ri <= "00101010";
    wait for 20 ns;
    assert (Rx = "00010100") report "Wrote to register while disabled #1" severity
        error;
    assert (Ry = "00010100") report "Wrote to register while disabled #2" severity
        error;

    -- Enable write
    enable <= '1';
    wait for 20 ns;
    assert (Rx = "00101010") report "Read from Rx failed #2" severity error;
    assert (Ry = "00101010") report "Read from Ry failed #2" severity error;

    -- Write to R2
    SelRi <= "010";
    Ri <= "01010001";
    wait for 20 ns;

    -- Read R2 from Rx
    SelRx <= "010";
    wait for 20 ns;
    assert (Rx = "01010001") report "Read from Rx failed #3" severity error;

    -- Read R2 from Ry
    SelRy <= "010";
    wait for 20 ns;
    assert (Ry = "01010001") report "Read from Ry failed #3" severity error;

    -- Read R0 from Rx again (should not have changed from previous results)
    SelRx <= "000";
    wait for 20 ns;
    assert (Rx = "00101010") report "Read from Rx failed #4" severity error;

    -- Read R0 from Ry again (should not have changed from previous results)
    SelRy <= "000";
    wait for 20 ns;
    assert (Ry = "00101010") report "Read from Ry failed #4" severity error;

    -- Wait for a long time
    wait for 1 ms;
    assert (Rx = "00101010") report "Read from Rx failed #5" severity error;
    assert (Ry = "00101010") report "Read from Ry failed #5" severity error;

```

```
    -- Read R2 after a long time
    SelRx <= "010";
    SelRy <= "010";
    wait for 1 ms;
    assert (Rx = "01010001") report "Read from Rx failed #6" severity error;
    assert (Ry = "01010001") report "Read from Ry failed #6" severity error;


    assert false report "End of test" severity note;
    wait; -- wait forever to end the test

  end process;
end behav;
```

Listing 28: mmu/mmu_tb.vhd

```
-- Author:
--       Wim Looman

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;
use work.mmu_main;
use work.minimal_uart_core;
use work.txt_util.all;

entity mmu_tb is
end mmu_tb;

architecture tb of mmu_tb is
  component mmu_main is
    port (
      -- instruction bus
      inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
      inst_data : out std_logic_vector(15 downto 0); -- Data lines.
      inst_req  : in  std_logic;                     -- Pulled low to request bus
          usage.
      inst_ack  : out std_logic;                     -- Pulled high to inform of
          request completion.
      -- data bus
      data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
      data_line : inout std_logic_vector(7 downto 0);  -- Data lines.
      data_read : in    std_logic;                     -- High for a read request,
          low for a write request.
      data_req  : in    std_logic;                     -- Pulled low to request bus
          usage.
      data_ack  : inout std_logic;                     -- Pulled high to inform of
          request completion.
      -- extras
      clk          : in  std_logic;
      receive_pin  : in  std_logic;
      transfer_pin : out std_logic
    );
  end component;

  component minimal_uart_core is
    port(
      clock : in    std_logic;
      eoc   : out   std_logic;
      outp  : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
      rxd   : in    std_logic;
      txd   : out   std_logic;
      eot   : out   std_logic;
      inp   : in    std_logic_vector(7 downto 0);
      ready : out   std_logic;
      wr    : in    std_logic
    );
  end component;

  signal inst_add     : std_logic_vector(11 downto 0);
  signal inst_data    : std_logic_vector(15 downto 0);
  signal inst_req     : std_logic := '1';
  signal inst_ack     : std_logic;
  signal data_add     : std_logic_vector(15 downto 0);
  signal data_line    : std_logic_vector(7 downto 0);
  signal data_read    : std_logic;
  signal data_req     : std_logic;
  signal data_ack     : std_logic;
  signal clk          : std_logic;
  signal receive_pin  : std_logic;
  signal transfer_pin : std_logic;

  signal eoc, rxd, txd, eot, ready, wr: std_logic;
  signal outp, inp : std_logic_vector(7 downto 0);

  signal current_recv : std_logic_vector(7 downto 0);
  signal current_send : std_logic_vector(7 downto 0);
begin
  m : mmu_main port map (inst_add, inst_data, inst_req, inst_ack, data_add,
      data_line, data_read, data_req, data_ack, clk, receive_pin, transfer_pin);
  muart : minimal_uart_core port map (clk, eoc, outp, rxd, txd, eot, inp, ready, wr);

  rxd <= transfer_pin;
  receive_pin <= txd;

  clk_gen : process begin
    clk <= '0';
    wait for 10 ns;
    clk <= '1';
    wait for 10 ns;
  end process;

  inst_test : process
    type pattern_type is record
      inst_add      : std_logic_vector(11 downto 0);
      recv_head     : std_logic_vector( 7 downto 0);
      send_head     : std_logic_vector( 7 downto 0);
      inst_data     : std_logic_vector(15 downto 0);
    end record;
    type pattern_array is array (natural range <>) of pattern_type;
    constant patterns : pattern_array :=
--      inst_add       recv_head    send_head    inst_data
    ((x"52E", x"81", x"83", x"83A7"),
     (x"96F", x"81", x"83", x"4F5E"),
     (x"8F1", x"81", x"83", x"5937"),
     (x"65A", x"81", x"83", x"A8F2"));
  begin
    wr <= '0';
    for i in patterns'range loop
      wait for 10000 ns;
      inst_add <= patterns(i).inst_add;
      wait for 20 ns;
      inst_req <= '0';

      wait until eoc'event;
      assert outp = patterns(i).recv_head
        report "Bad header expected '" & str(patterns(i).recv_head) & "' recieved '"
            & str(outp) & "'"
        severity error;
      wait until eoc'event;

      assert false report "passed header" severity note;

      wait until eoc'event;
      assert outp = patterns(i).inst_add(7 downto 0)
        report "Bad address low expected '" & str(patterns(i).inst_add(7 downto 0)) &
            "' recieved '" & str(outp) & "'"
        severity error;
      wait until eoc'event;

      assert false report "passed address low" severity note;

      wait until eoc'event;
      assert outp = "0000" & patterns(i).inst_add(11 downto 8)
```

```
        report "Bad address high expected '" & str(patterns(i).inst_add(11 downto
            8)) & "' recieved '" & str(outp) & "'"
        severity error;
      wait until eoc'event;

      assert false report "passed address high" severity note;

      wait for 100 ns;
      inp <= patterns(i).send_head;
      wait for 20 ns;
      wr <= '1';
      wait for 20 ns;
      wr <= '0';
      wait until eot'event;
      wait until eot'event;


      wait for 100 ns;
      inp <= "0000" & patterns(i).inst_add(11 downto 8);
      wait for 20 ns;
      wr <= '1';
      wait for 20 ns;
      wr <= '0';
      wait until eot'event;
      wait until eot'event;


      wait for 100 ns;
      inp <= patterns(i).inst_add(7 downto 0);
      wait for 20 ns;
      wr <= '1';
      wait for 20 ns;
      wr <= '0';
      wait until eot'event;
      wait until eot'event;


      wait for 100 ns;
      inp <= patterns(i).inst_data(7 downto 0);
      wait for 20 ns;
      wr <= '1';
      wait for 20 ns;
      wr <= '0';
      wait until eot'event;
      wait until eot'event;


      wait for 100 ns;
      inp <= patterns(i).inst_data(15 downto 8);
      wait for 20 ns;
      wr <= '1';
      wait for 20 ns;
      wr <= '0';
      wait until eot'event;
      wait until eot'event;

      assert inst_ack = '1'
        report "receipt not acknowledged"
        severity error;

      assert inst_data = patterns(i).inst_data
        report "Wrong data recieve expected '" & str(patterns(i).inst_data) & "'
            recieved '" & str(inst_data) & "'"
        severity error;

      assert false report "finished transmission" severity note;

      wait for 20 ns;

      inst_req <= '1';
    end loop;
    wait;
  end process;
```

```
end tb;
```

Listing 29: data_tb.vhd

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

library work;
use work.mmu_main;
use work.minimal_uart_core;
use work.txt_util.all;

entity data_tb is
end data_tb;

architecture tb of data_tb is
  component mmu_main is
    port (
      -- instruction bus
      inst_add   : in  std_logic_vector(11 downto 0); -- Address lines.
      inst_data  : out std_logic_vector(15 downto 0); -- Data lines.
      inst_req   : in  std_logic;                     -- Pulled low to request bus
          usage.
      inst_ack   : out std_logic;                     -- Pulled high to inform of
          request completion.
      -- data bus
      data_add   : in    std_logic_vector(15 downto 0); -- Address lines.
      data_line  : inout std_logic_vector(7 downto 0);  -- Data lines.
      data_read  : in    std_logic;                     -- High for a read request,
          low for a write request.
      data_req   : in    std_logic;                     -- Pulled low to request bus
          usage.
      data_ack   : inout std_logic;                     -- Pulled high to inform of
          request completion.
      -- extras
      clk          : in  std_logic;
      receive_pin  : in  std_logic;
      transfer_pin : out std_logic
    );
  end component;

  component IO is
    PORT(
      -- data bus --
      data_add    : IN      std_logic_vector(15 DOWNTO 0);     -- address lines --
      data_data  : INOUT    std_logic_vector(7 DOWNTO 0);    -- data lines --
      data_read  : INOUT    std_logic;                  -- pulled high for read, low
          for write --
      data_req    : INOUT   std_logic;                  -- pulled low to request bus
          usage --
      data_ack    : INOUT   std_logic;                  -- pulled high to inform
          request completion --
      -- io --
      clk         : IN      std_logic;
      sw1         : IN      std_logic;
      sw2         : IN      std_logic
    );
  end component;

  component minimal_uart_core is
    port(
      clock : in    std_logic;
      eoc   : out   std_logic;
      outp  : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
      rxd   : in    std_logic;
      txd   : out   std_logic;
      eot   : out   std_logic;
      inp   : in    std_logic_vector(7 downto 0);
      ready : out   std_logic;
      wr    : in    std_logic
    );
  end component;
```

```vhdl
  signal inst_add     : std_logic_vector(11 downto 0);
  signal inst_data    : std_logic_vector(15 downto 0);
  signal inst_req     : std_logic := '1';
  signal inst_ack     : std_logic;
  signal data_add     : std_logic_vector(15 downto 0);
  signal data_line    : std_logic_vector(7 downto 0);
  signal data_read    : std_logic;
  signal data_req     : std_logic;
  signal data_ack     : std_logic;
  signal clk          : std_logic;
  signal receive_pin  : std_logic;
  signal transfer_pin : std_logic;
  signal sw1, sw2 : std_logic;

  signal eoc, rxd, txd, eot, ready, wr: std_logic;
  signal outp, inp : std_logic_vector(7 downto 0);
begin
  m : mmu_main port map (inst_add, inst_data, inst_req, inst_ack, data_add,
      data_line, data_read, data_req, data_ack, clk, receive_pin, transfer_pin);
  i : IO       port map (data_add, data_line, data_read, data_req, data_ack, clk,
      sw1, sw2);
  muart : minimal_uart_core port map (clk, eoc, outp, rxd, txd, eot, inp, ready, wr);

  rxd <= transfer_pin;
  receive_pin <= txd;

  clk_gen : process begin
    clk <= '0';
    wait for 10 ns;
    clk <= '1';
    wait for 10 ns;
  end process;

  data_test : process
    type pattern_type is record
      data_add      : std_logic_vector(15 downto 0);
      recv_head     : std_logic_vector( 7 downto 0);
      send_head     : std_logic_vector( 7 downto 0);
      data_data     : std_logic_vector( 7 downto 0);
      switch_data   : std_logic_vector( 1 downto 0);
      rw            : std_logic;
    end record;
    type pattern_array is array (natural range <>) of pattern_type;
    constant patterns : pattern_array :=
-- data_add, recv_head, send_head, data_data, switch_data, rw
    ((x"0581", x"80", x"00", x"A7", "00", '1' ),
     (x"0273", x"00", x"00", x"5E", "00", '0' ));
  begin
    wr <= '0';
    data_req <= '1';
    for i in patterns'range loop
      wait for 10000 ns;
      data_add <= patterns(i).data_add;
      data_read <= patterns(i).rw;
      wait for 20 ns;
      if patterns(i).rw = '0' then
        data_line <= patterns(1).data_data;
      else
        data_line <= (others => 'Z');
      end if;
      wait for 20 ns;
      data_req <= '0';

      if patterns(i).data_add(0) = '1' then
        wait until eoc'event;
        assert outp = patterns(i).recv_head
          report "Bad header expected '" & str(patterns(i).recv_head) & "' recieved
              '" & str(outp) & "'"
          severity error;
        wait until eoc'event;

        assert false report "passed header" severity note;

        wait until eoc'event;
        assert outp = patterns(i).data_add(8 downto 1)
          report "Bad address low expected '" & str(patterns(i).data_add(7 downto
              0)) & "' recieved '" & str(outp) & "'"
          severity error;
        wait until eoc'event;

        assert false report "passed address low" severity note;

        wait until eoc'event;
        assert outp = "0" & patterns(i).data_add(15 downto 9)
          report "Bad address high expected '" & str(patterns(i).data_add(11 downto
              8)) & "' recieved '" & str(outp) & "'"
          severity error;
        wait until eoc'event;

        assert false report "passed address high" severity note;
        if patterns(i).rw = '0' then
          wait until eoc'event;
          assert outp = patterns(i).data_data
            report "Bad data expected '" & str(patterns(i).data_data) & "' recieved
                '" & str(outp) & "'"
            severity error;
          wait until eoc'event;

          assert false report "passed data" severity note;
        else
          wait for 100 ns;
          inp <= patterns(i).send_head;
          wait for 20 ns;
          wr <= '1';
          wait for 20 ns;
          wr <= '0';
          wait until eot'event;
          wait until eot'event;


          wait for 100 ns;
          inp <= patterns(i).data_add(8 downto 1);
          wait for 20 ns;
          wr <= '1';
          wait for 20 ns;
          wr <= '0';
          wait until eot'event;
          wait until eot'event;


          wait for 100 ns;
          inp <=  "0" & patterns(i).data_add(15 downto 9);
          wait for 20 ns;
          wr <= '1';
          wait for 20 ns;
          wr <= '0';
          wait until eot'event;
          wait until eot'event;


          wait for 100 ns;
          inp <= patterns(i).data_data;
          wait for 20 ns;
          wr <= '1';
          wait for 20 ns;
          wr <= '0';
          wait until eot'event;
          wait until eot'event;
        end if;
      else

      end if;

      assert data_ack = '1'
        report "receipt not acknowledged"
```

```
          severity error;

        if  patterns(i).rw = '1' then
          assert data_line = patterns(i).data_data
            report "Wrong data recieve expected '" & str(patterns(i).data_data) & "'
                recieved '" & str(data_line) & "'"
            severity error;
        end if;

        assert false report "finished transmission" severity note;

        wait for 20 ns;

        data_req <= '1';
      end loop;
      wait;
    end process;
end tb;
```

Tools

Listing 30: assembler.rb

```ruby
#!/usr/bin/env ruby

# Author:
#       Wim Looman
# Copyright:
#       Copyright (c) 2010 Wim Looman
# License:
#       GNU General Public License (see http://www.gnu.org/licenses/gpl-3.0.txt)

def assert(error=nil)
  raise (error || "Assertion Failed!") unless yield
end

# For 8-bit twos complement
def twos_complement(num)
  return 256 + num
end


def logical_operands(chunks)
      y = chunks[1][1..1].to_i
      assert(chunks[1] + " is not a valid register") {y >= 0 && y < 8}
      x = chunks[2][1..1].to_i
      assert(chunks[2] + " is not a valid register") {x >= 0 && x < 8}
      return (y << 5) + x
end


def immediate(chunk, symbols=nil, move_from=nil)
  v = chunk.to_i
  if v == 0 && chunk != "0" && symbols != nil
    assert(chunk + " is not a valid symbol") {symbols.include?(chunk)}
    move_to = symbols[chunk]
    diff = move_to - move_from
    if diff < 1
      diff = twos_complement(diff)
    end
    return diff
  else
    assert(chunk + " is not a valid immediate") {v >= -127 && v < 128}
    if v < 0
      v = twos_complement(v)
    end
    return v
  end
end


def register(chunk, num_registers)
  x = chunk[1..1].to_i
  assert(chunk + " is not a valid register") {x >=0 && x < num_registers}
  return x
end


def auto(chunk)
  if chunk[-1..-1] == "+"
    return 0x08
  elsif chunk[-1..-1] == "-"
    return 0x10
  else
    return 0x00
  end
end


def convert(lines)
  table = first_pass(lines)
  return second_pass(lines, table)
end


def first_pass(lines)
  instruction = 0
  symbols = {}
  lines.each do |line|
    chunks = line.sub(",", " ").split
    case chunks[0]
      when "LDI", "LD", "STI", "ST", "MV", "AND", "OR", "NOT", "XOR", "ADD", "ADC",
          "SUB", "SBB", "NEG", "CMP", "BEQ", "BNE", "BLT", "BGT", "BC", "BNC", "RJMP",
           "JMP"
        instruction += 1

      when "label:"
        symbols[chunks[1]] = instruction
    end
  end
  return symbols
end


def second_pass(lines, symbols)
  line_no = 0
  output = []
  lines.each do |line|
    label = line.sub(",", " ").split[0]
    case label
      when "LDI", "LD", "STI", "ST", "MV", "AND", "OR", "NOT", "XOR", "ADD", "ADC",
          "SUB", "SBB", "NEG", "CMP", "BEQ", "BNE", "BLT", "BGT", "BC", "BNC", "RJMP",
           "JMP"
        line_no += 1
        output.push(convert_line(line, symbols, line_no))
    end
  end
  return output.flatten
end


def convert_line(line, symbols, line_no)
  chunks = line.sub(",", " ").split#.partition(";")[0].split

  case chunks[0]
    when "LDI"
      instruction = 0x21
      x = register(chunks[1], 4)
      v = immediate(chunks[2])
      operands = (v << 2) + x

    when "LD"
      instruction = 0x01 + auto(chunks[2])
      x = register(chunks[1], 8)
      y = register(chunks[2], 3)
      operands = (y << 5) + x

    when "STI"
      instruction = 0x25
      y = register(chunks[1], 3)
      v = immediate(chunks[2])
      operands = (v << 2) + y

    when "ST"
      instruction = 0x05 + auto(chunks[1])
      y = register(chunks[1], 3)
      x = register(chunks[2], 8)
      operands = (y << 5) + x

    when "MV"
      instruction = 0x04
      if chunks[1][0] == 'r'[0] && chunks[2][0] == 'r'[0]
        y = register(chunks[1], 8)
        x = register(chunks[2], 8)
        operands = (y << 5) + x
      elsif chunks[1][0] == 'a'[0]
        y = register(chunks[1], 3)
        x = register(chunks[2], 8)
        n = chunks[1][-1] == 'H' ? 1 : 0
        operands = (1 << 9) + (n << 8) + (y << 5) + x
      elsif chunks[2][0] == 'a'[0]
        y = register(chunks[1], 8)
        x = register(chunks[2], 3)
        n = chunks[2][-1] == 'H' ? 1 : 0
        operands = (1 << 4) + (n << 3) + (y << 5) + x
      else
        # explode
      end

    when "AND"
      instruction = 0x02
      operands = logical_operands(chunks)

    when "OR"
      instruction = 0x06
      operands = logical_operands(chunks)

    when "NOT"
      instruction = 0x0A
      operands = logical_operands(chunks)

    when "XOR"
      instruction = 0x0E
      operands = logical_operands(chunks)

    when "ADD"
      instruction = 0x12
      operands = logical_operands(chunks)

    when "ADC"
      instruction = 0x16
      operands = logical_operands(chunks)

    when "SUB"
      instruction = 0x1A
      operands = logical_operands(chunks)

    when "SBB"
      instruction = 0x1E
      operands = logical_operands(chunks)

    when "NEG"
      instruction = 0x08
      operands = logical_operands(chunks)

    when "CMP"
      instruction = 0x0C
      operands = logical_operands(chunks)

    when "BEQ"
      instruction = 0x23
      v = immediate(chunks[1], symbols, line_no)
      operands = (v << 2)

    when "BNE"
      instruction = 0x27
      v = immediate(chunks[1], symbols, line_no)
      operands = (v << 2)

    when "BLT"
      instruction = 0x2B
      v = immediate(chunks[1], symbols, line_no)
      operands = (v << 2)

    when "BGT"
      instruction = 0x2F
      v = immediate(chunks[1], symbols, line_no)
```

```
        operands = (v << 2)

      when "BC"
        instruction = 0x33
        v = immediate(chunks[1], symbols, line_no)
        operands = (v << 2)

      when "BNC"
        instruction = 0x37
        v = immediate(chunks[1], symbols, line_no)
        operands = (v << 2)

      when "RJMP"
        instruction = 0x3B
        v = immediate(chunks[1], symbols, line_no)
        operands = (v << 2)

      when "JMP"
        instruction = 0x2F
        y = register(chunks[1], 3)
        operands = (y << 5)
    end
    opcode = (instruction << 10) + operands
    return [(opcode >> 8), (opcode & 0xFF)]
end


if __FILE__ == $0
  if !(1..2).include?(ARGV.length) || !File.exist?(ARGV[0])
    p "Usage: ruby #{$0} <input_file> [<output_file>]"
   exit
  end

  input = IO.readlines(ARGV[0])

  output = convert(input)
  if ARGV.length == 2:
    File.open(ARGV[1], "wb") do |file|
      output.each do |char|
        file.putc(char)
      end
    end
  else
    output.each do |char|
      $stdout.putc(char)
    end
  end
end
```

Listing 31: memory.rb

```
#!/usr/bin/env ruby

# Author:
#       Wim Looman
# Copyright:
#       Copyright (c) 2010 Wim Looman
# License:
#       GNU General Public License (see http://www.gnu.org/licenses/gpl-3.0.txt)

require 'rubygems'
require 'serialport'

def serve(program, data_file, sp)
  while 1
    header = sp.getc
    diagnostic_mode = (header >> 2) & 0x03
    instruction = header & 0x01 == 0x01
    address = sp.getc + (sp.getc << 8)

    case (header >> 7) & 0x01 # read/write bit
      when 0x01               # read
```

```
        header = 0x82
        header += 0x01 if instruction
        sp.putc(header)
        instruction ? sp.write(program[address]) :
                      sp.write(data_file[address])
        p "Sending data for #{instruction ? "Instruction" : "Data"} bus, address:" +
          "#{address}, data: #{instruction ? program[address] : data_file[address]}"
      when 0x00                # write, doesn't support writing to instruction memory
        data = sp.getc
        data_file[address] = data
        p "Writing data, address: " + address + ", data: " + data_file[address]
    end
  end
end

if __FILE__ == $0

  if ARGV.size < 3
    STDERR.print "Usage: ruby #{$0} <device> <baud_rate> <program_file>
        [<data_file>]\n"
    exit
  end

  device = ARGV[0]
  baud_rate = ARGV[1].to_i

  program = Array.new(2**12, 0x00)
  i = 0
  File.open(ARGV[2], 'rb') do |input|
    input.each_byte do |byte|
      program[i] =  byte
      i += 1
    end
  end

  data = Array.new(2**15, 0x00)
  File.open(ARGV[3], 'rb') do |input|
    input.each_byte do |byte|
      data += byte
    end
  end if ARGV.size > 3

  sp = SerialPort.new(device, baud_rate, 8, 1, SerialPort::NONE)

  serve(program, data, sp)
end
```