

Listings

Main

Listing 1: main.vhd

```
1  -- Authors:
   --      Wim Looman, Forrest McKerchar, Henry Jenkins, Joel Koh, Sasha Wang, Tracy
       Jackson
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  library work;
8
9  entity main is
10     port (
11         clk      : in  std_logic;
12         reset    : in  std_logic;
13         tx       : out std_logic;
14         rx       : in  std_logic;
15         sw1      : in  std_logic;
16         sw2      : in  std_logic;
17     );
18     end main;
19
20     architecture main_arch of main is
21         component cpu IS
22             PORT(
23                 -- instruction bus
24                 inst_add  : out std_logic_vector(11 downto 0); -- Address lines.
25                 inst_data : in  std_logic_vector(15 downto 0); -- Data lines.
26                 inst_req  : out std_logic;                      -- Pulled low to request bus
27                                     usage.
28                 inst_ack  : in  std_logic;                      -- Pulled high to inform of
29                                     request completion.
30                 -- data bus
31                 data_add  : out std_logic_vector(15 downto 0); -- Address lines.
32                 data_line : inout std_logic_vector(7 downto 0); -- Data lines.
33                 data_read : out std_logic;                      -- High for a read request,
34                                     low for a write request.
35                 data_req  : out std_logic;                      -- Pulled low to request bus
36                                     usage.
37                 data_ack  : inout std_logic;                    -- Pulled high to inform of
38                                     request completion.
39                 -- extras
40                 clk       : in  std_logic;
41                 reset     : in  std_logic;
42             );
43         end component;
44         component mmu_main is
45             port (
46                 -- instruction bus
47                 inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
48                 inst_data : out std_logic_vector(15 downto 0); -- Data lines.
49                 inst_req  : in  std_logic;                      -- Pulled low to request bus
50                                     usage.
51                 inst_ack  : out std_logic;                      -- Pulled high to inform of
52                                     request completion.
```

```

47     -- data bus
data_add  : in    std_logic_vector(15 downto 0); -- Address lines.
data_line : inout std_logic_vector(7  downto 0); -- Data lines.
49 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.
51 data_ack  : inout std_logic;                  -- Pulled high to inform of
    request completion.
    -- extras
53 clk      : in    std_logic;
receive_pin : in    std_logic;
55 transfer_pin : out std_logic
);
57 END component;
component IO is
59     PORT(
        -- data bus --
61         data_add  : IN    std_logic_vector(15 DOWNT0 0); -- address lines --
        data_data   : INOUT  std_logic_vector(7  DOWNT0 0); -- data lines --
63         data_read  : INOUT  std_logic;                  -- pulled high for
            read, low for write --
        data_req    : INOUT  std_logic;                  -- pulled low to
            request bus usage --
65         data_ack   : INOUT  std_logic;                  -- pulled high to
            inform request completion --
        -- io --
67         clk        : IN    std_logic;
        sw1          : IN    std_logic;
69         sw2        : IN    std_logic);
        --leds       : OUT std_logic_vector(7 DOWNT0 0);
71 END component;
-- instruction bus
73 signal inst_add  : std_logic_vector(11 downto 0); -- Address lines.
signal inst_data : std_logic_vector(15 downto 0); -- Data lines.
75 signal inst_req  : std_logic;                  -- Pulled low to request bus
    usage.
signal inst_ack : std_logic;                  -- Pulled high to inform of
    request completion.
77 -- data bus
signal data_add  : std_logic_vector(15 downto 0); -- Address lines.
79 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.
81 signal data_req  : std_logic;                  -- Pulled low to request bus
    usage.
signal data_ack : std_logic;                  -- Pulled high to inform of
    request completion.
83
begin
85     c : cpu port map(
        -- instruction bus
87         inst_add => inst_add, -- Instruction address
        inst_data => inst_data, -- Instruction data
89         inst_req => inst_req, -- Request
        inst_ack => inst_ack, -- Instruction obtained
91         -- data bus
        data_add => data_add, -- Data address
93         data_line => data_line, -- Data
        data_read => data_read, -- 1 for read, 0 for write
95         data_req => data_req, -- Request
        data_ack => data_ack, -- Data written to/ read from
97         -- extras
        clk      => clk,
99         reset   => reset
    );
101    m : mmu_main port map(
        -- instruction bus
103        inst_add  => inst_add, -- Address lines.
        inst_data  => inst_data, -- Data lines.
105        inst_req   => inst_req, -- Pulled low to request bus usage.
        inst_ack    => inst_ack, -- Pulled high to inform of request completion.
107        -- data bus

```

```

109     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.
111    data_req       => data_req,  -- Pulled low to request bus usage.
    data_ack       => data_ack,  -- Pulled high to inform of request completion.
113    -- extras
    clk            => clk,
115    receive_pin   => rx,
    transfer_pin   => tx
117 );
i : io  port map(
119     clk            => clk,
    data_add       => data_add,
121     data_data     => data_line,
    data_read      => data_read,
123     data_req      => data_req,
    data_ack       => data_ack,
125     -- io --
    sw1            => sw1,
127     sw2           => sw2
);
129 end architecture main_arch;

```

IO

Listing 2: IO/debounce.vhd

```
1  -----
2  -- Module Name:      debounce
3  -- Description: Entity to debounce a mechanical switch/button
4  -- Authors: Tracy Jackson
5  --                Sasha Wang
6  --
7  -----
8
9  library IEEE;
10 use IEEE.STD_LOGIC_1164.ALL;
11 use IEEE.STD_LOGIC_ARITH.ALL;
12 use IEEE.STD_LOGIC_UNSIGNED."+";
13
14 library work;
15
16 ENTITY debounce IS
17     PORT(clk : IN STD_LOGIC;
18           switch : IN STD_LOGIC;
19           switch_state : OUT STD_LOGIC);
20 END debounce;
21
22 ARCHITECTURE debounced_switch OF debounce IS
23     SIGNAL count : STD_LOGIC_VECTOR(2 DOWNTO 0);
24 BEGIN
25     -- Debounce the switch using a counter
26     PROCESS(clk, switch)
27     BEGIN
28         IF switch = '0' THEN
29             count <= "000";
30         ELSIF rising_edge(clk) THEN
31             IF count /= "111" THEN
32                 count <= count + 1;
33             END IF;
34         END IF;
35         IF count = "111" AND switch = '1' THEN
36             switch_state <= '1';
37         ELSE
38             switch_state <= '0';
39         END IF;
40     END PROCESS;
41 END debounced_switch;
```

Listing 3: IO/IO.vhd

```
1  -----
2  -- Module Name:      IO
3  -- Description: Entity to handle IO
4  -- Authors: Tracy Jackson
5  --                Sasha Wang
6  --
7  -----
8
9  library IEEE;
10 use IEEE.STD_LOGIC_1164.ALL;
11 use IEEE.STD_LOGIC_ARITH.ALL;
12 --use IEEE.STD_LOGIC_UNSIGNED.ALL;
13
14 library work;
15 use work.debounce;
16 use work.switch_reg;
17 use work.led_io;
18
19
20 ---- Uncomment the following library declaration if instantiating
21 ---- any Xilinx primitives in this code.
22 --library UNISIM;
23 --use UNISIM.VComponents.all;
```

```

24 entity IO is
25     PORT(
26         -- data bus --
27         data_add      : IN          std_logic_vector(15 DOWNTO 0);
28         -- address lines --
29         data_data      : INOUT       std_logic_vector(7  DOWNTO 0);  -- data
30         lines --
31         data_read      : INOUT       std_logic;
32         -- pulled high for read, low for write --
33         data_req        : INOUT       std_logic;
34         -- pulled low to request bus usage --
35         data_ack        : INOUT       std_logic;
36         -- pulled high to inform request completion --
37         -- io --
38         clk            : IN          std_logic;
39         sw1            : IN          std_logic;
40         sw2            : IN          std_logic);
41         --leds          : OUT std_logic_vector(7  DOWNTO 0);
42 end IO;
43
44 architecture io of IO is
45
46     COMPONENT led_io
47     PORT(
48         data_add      : IN          std_logic_vector(15 DOWNTO 0);  --
49         address lines --
50         data_data      : INOUT       std_logic_vector(7  DOWNTO 0);  -- data lines
51         --
52         data_read      : INOUT       std_logic;                      --
53         pulled high for read, low for write --
54         data_req        : INOUT       std_logic;                      --
55         pulled low to request bus usage --
56         data_ack        : INOUT       std_logic;                      --
57         pulled high to inform request completion --
58         clock          : IN          std_logic
59     );
60 END COMPONENT;
61
62 COMPONENT switch_io IS
63     PORT ( data_add      : IN          std_logic_vector(15 DOWNTO 0);
64           data_data      : INOUT       std_logic_vector(7  DOWNTO 0);
65           data_read      : INOUT       std_logic;
66           data_req        : INOUT       std_logic;
67           data_ack        : INOUT       std_logic;
68           clk            : IN          std_logic;
69           sw1            : IN          std_logic;
70           sw2            : IN          std_logic
71     );
72 END COMPONENT;
73
74 BEGIN
75     led: led_io PORT MAP(data_add, data_data, data_read, data_req, data_ack, clk);
76     switch: switch_io PORT MAP(data_add, data_data, data_read, data_req, data_ack,
77                               clk,sw1,sw2);
78
79     -----
80 END io;

```

Listing 4: IO/leds.vhd

```

-----
2 -- Module Name:      led_io
3 -- Description: Entity control output LEDs
4 -- Authors: Tracy Jackson
5 --                Sasha Wang

```

```

6  --
  -----
8  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
10 use IEEE.STD_LOGIC_ARITH.ALL;

12 library work;

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

26 ARCHITECTURE led_arch OF led_io IS
   Signal led_enable    : std_logic;
28   Signal led_state     : std_logic_vector(7 DOWNTO 0);
   BEGIN
30
   -- Determine if it is the LEDs being accessed
32   PROCESS(clock, data_req, data_add, data_read)
   BEGIN
34       IF data_req = '0' AND data_add = "0000000000001110" AND data_read = '0'
           THEN
36               led_enable <= '1';
           ELSE
38               led_enable <= '0';
           END IF;
40   END PROCESS;

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

52

54 END led_arch;

```

Listing 5: IO/switch_register.vhd

```

1  -----
   -- Module Name:      switch_reg
3  -- Description: Entity to store switch state (can be extended to more than one)
   -- Authors: Tracy Jackson
5  --          Sasha Wang
   --
7  -----
   library IEEE;
9  use IEEE.STD_LOGIC_1164.ALL;
   use IEEE.STD_LOGIC_ARITH.ALL;
11 use IEEE.STD_LOGIC_UNSIGNED.ALL;

13 library work;

```

```

15 ENTITY switch_reg IS
    PORT( D           : IN STD_LOGIC;
17         clk,enable  : IN STD_LOGIC;
           Q           : OUT STD_LOGIC);
19 END switch_reg;

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

```

Listing 6: IO/switches.vhd

```

1  -----
  -- Module Name:      switch_io
3  -- Description: Entity to control input from switches
  -- Authors: Tracy Jackson
  --               Sasha Wang
  --
7  -----
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use IEEE.STD_LOGIC_ARITH.ALL;

11   library work;
13   use work.debounce;
15   use work.switch_reg;

    entity switch_io is
17       PORT(
19           -- data bus --
           data_add      : IN          std_logic_vector(15 DOWNT0 0);
           -- address lines --
           data_data      : INOUT      std_logic_vector(7 DOWNT0 0);  -- data
           lines --
21           data_read     : INOUT      std_logic;
           -- pulled high for read, low for write --
           data_req       : INOUT      std_logic;
           -- pulled low to request bus usage --
23           data_ack      : INOUT      std_logic;
           -- pulled high to inform request completion --
           -- io --
25           clk           : IN          std_logic;
           sw1            : IN          std_logic;
27           sw2            : IN          std_logic);
    end switch_io;

29   architecture Behavioral of switch_io is
31
33   signal enable1          : std_logic;
35   signal switch1_connection : std_logic;
37   signal enable2          : std_logic;
39   signal switch2_connection : std_logic;
41   signal switch2_output    : std_logic;

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

47   COMPONENT switch_reg

```

```

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

53

55 BEGIN

57 sw1_debouncer: debounce PORT MAP(clk, sw1, switch1_connection);
    sw1_status: switch_reg PORT MAP(switch1_connection,clk, enable1, switch1_output);
59

61 sw2_debouncer: debounce PORT MAP(clk, sw2,switch2_connection);
    sw2_status: switch_reg PORT MAP(switch2_connection,clk, enable2, switch2_output);
63

65 PROCESS(clk,switch1_output,switch2_output, data_ack)
    BEGIN
67 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
69         enable1 <= '0';

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

77     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';
79

        --ELSIF data_ack = '0' AND data_add = "0000000000001100" THEN -- when the data is
            sent to the CPU, enable the switch_reg again
81     -- enable2 <= '1';
    ELSE
83         enable2 <= '1';
    END IF;
85

87 END IF;
    END PROCESS;
89

91 PROCESS(clk, data_add, data_read)
    BEGIN
93     IF rising_edge(clk) THEN
        IF data_req = '0' AND data_read = '1' THEN
95         IF data_add = "0000000000001110" THEN -- switch1 address
            IF switch1_output = '1' THEN
97                 data_data <= "00000001";
            ELSE
99                 data_data <= "00000000";
            END IF;
101         data_ack <= '0';
        END IF;
103         IF data_add = "0000000000001100" THEN -- switch2 address
            IF switch2_output = '1' THEN
105                 data_data <= "00000001";
            ELSE
107                 data_data <= "00000000";
            END IF;
109         data_ack <= '0';
        END IF;
111     ELSIF data_req = '1' AND data_ack = '0' THEN
        data_ack <= 'Z';
113     END IF;
    END IF;
115 END PROCESS;

```


117

119

END Behavioral;

CPU

Listing 7: processor/alu.vhd

```
-- Authors:
2 --      Henry Jenkins, Joel Koh

4 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
6 use IEEE.NUMERIC_STD.ALL;
use ieee.std_logic_arith.all;
8 --use ieee.std_logic_unsigned.all;

10
library work;
12 use work.fulladder8;
--use work.cpu.ALL;
14
-- Uncomment the following library declaration if using
16 -- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
18
-- Uncomment the following library declaration if instantiating
20 -- any Xilinx primitives in this code.
--library UNISIM;
22 --use UNISIM.VComponents.all;

24 entity alu is
    Port (f      : in    STD_LOGIC_VECTOR (3 downto 0); -- Function (opcode)
26         rx     : in    STD_LOGIC_VECTOR (7 downto 0); -- Input x (Rx)
         ry     : in    STD_LOGIC_VECTOR (7 downto 0); -- Input y (Ry)
28         ro     : out   STD_LOGIC_VECTOR (7 downto 0); -- Output Normaly (Ry)
         Cin    : in    STD_LOGIC;                    -- Carry in
30         sr     : out   STD_LOGIC_VECTOR (15 downto 0)); -- Status register out Z(0),
                C(1), N(2)
    end alu;
32

34 architecture alu_arch of alu is
    component fulladder8 IS
36     Port (A      : in    STD_LOGIC_VECTOR( 7 downto 0);
         B      : in    STD_LOGIC_VECTOR( 7 downto 0);
38         Cin    : in    STD_LOGIC;
         Sum    : out   STD_LOGIC_VECTOR( 7 downto 0);
40         Cout   : out   STD_LOGIC
        );
42    end component;
    signal A      : std_logic_vector(7 downto 0);
44    signal B      : std_logic_vector(7 downto 0);
    signal AdderCin : std_logic;
46    signal Sum    : std_logic_vector(7 downto 0);
    signal AdderCout : std_logic;
48    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
50 BEGIN
    Adder: fulladder8 port map(A, B, AdderCin, Sum, AdderCout);
52    process(f, rx, ry, Cin, Sum, AdderCout)
        --signal Z,C,N : std_logic; -- Make the code easier to read
54    BEGIN
        -- use case statement to achieve
56        -- different operations of ALU

58        AdderCin <= '0';
        A <= (others => '0');
60        B <= (others => '0');
        output <= (others => '0');
62        C <= '0';
        N <= '0';
64        IF f = "0001" THEN -- Do AND operation
            output <= ry and rx;
66        ELSIF f = "0011" THEN -- Do OR operation
            output <= ry or rx;
```

```

68     ELSIF f = "0101" THEN
        output <= not rx;
70     ELSIF f = "0111" THEN -- Do XOR operation
        output <= ry xor rx;
72     ELSIF f = "1001" THEN -- Do ADD operation
        AdderCin <= '0';
74         A <= ry;
        B <= rx;
76         output <= Sum;
        ELSIF f = "1011" THEN -- Do ADC operation
78         AdderCin <= Cin;
        A <= ry;
80         B <= rx;
        output <= Sum;
82     ELSIF f = "1101" THEN -- Do SUB operation
        AdderCin <= '1';
84         A <= ry;
        B <= (not rx);
86         output <= Sum;
        ELSIF f = "1111" THEN -- Do SBB operation
88         AdderCin <= (not Cin);
        A <= ry;
90         B <= (not rx);
        output <= Sum;
92     ELSIF f = "0100" THEN -- Do NEG operation ( two's complement )
        AdderCin <= '1';
94         A <= (others => '0');
        B <= (not rx);
96         output <= Sum;
        C <= AdderCout;
98         N <= output(7);
        ELSIF f = "0110" THEN -- Do CMP operation
100         AdderCin <= '1';
        A <= rx;
102         B <= (not ry);
        output <= Sum;
104         C <= AdderCout;
        N <= output(7);
106     ELSE
        AdderCin <= '0';
108         A <= (others => '0');
        B <= (others => '0');
110         output <= (others => '0');
        C <= '0';
112         N <= '0';
        END IF;
114 --     if (output = "00000000") then -- Set the Zero in status register
--         sr(0) <= '1';
116 --     ELSE
--         sr(0) <= '0';
118 --     end if;

120     C <= AdderCout; -- Carry is always 0
    N <= output(7); -- This might need to be changed to '0'
122     ro <= output;
end process;
124     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));
126     sr(0) <= Z; --Z(0)
    sr(1) <= C; --C(1)
128     sr(2) <= N; --N(2)
    sr(15 downto 3) <= (others => '0');
130
end alu_arch;

```

Listing 8: processor/ar.vhd

```

1 -- Authors:
--     Henry Jenkins, Joel Koh
3
library IEEE;
5 use IEEE.STD_LOGIC_1164.ALL;

```

```

7  library work;
   use work.reg16;
9
   entity ar is
11  Port (clk           : in   STD_LOGIC;
        enable        : in   STD_LOGIC;
13        Sel8Bit       : in   STD_LOGIC;
        SelHighByte    : in   STD_LOGIC;
15        ByteInput     : in   STD_LOGIC_VECTOR (7 downto 0);
        SelRi          : in   STD_LOGIC_VECTOR (1 downto 0);    -- Select the address
                                register
17        SelRo         : in   STD_LOGIC_VECTOR (1 downto 0);    -- Select the address
                                register
        Ri             : in   STD_LOGIC_VECTOR (15 downto 0);    -- The input
19        Ro            : out  STD_LOGIC_VECTOR (15 downto 0));    -- The output
   end ar;
21
   architecture Behavioral of ar is
23     component reg16 IS
        port(I       : in   std_logic_vector(15 downto 0);
25              clock  : in   std_logic;
              enable  : in   std_logic;
27              reset  : in   std_logic;
              Q       : out  std_logic_vector(15 downto 0)
29            );
     end component;
31
     signal ROE      : std_logic; -- Enable signals
33     signal R1E      : std_logic;
     signal R2E      : std_logic;
35     signal input    : std_logic_VECTOR (15 downto 0);
     signal Q0       : std_logic_VECTOR (15 downto 0);
37     signal Q1       : std_logic_VECTOR (15 downto 0);
     signal Q2       : std_logic_VECTOR (15 downto 0);
39 BEGIN
     reg_0 : reg16 port map(input, clk, ROE, '0', Q0);
41     reg_1 : reg16 port map(input, clk, R1E, '0', Q1);
     reg_2 : reg16 port map(input, clk, R2E, '0', Q2);
43
     SetInput: process(clk, enable, SelRi, Ri)
45     BEGIN
         ROE <= '0';
47         R1E <= '0';
         R2E <= '0';
49         IF enable = '1' THEN
             case SelRi IS
51                 WHEN "00" =>
                     ROE <= '1';
53                 WHEN "01" =>
                     R1E <= '1';
55                 WHEN "10" =>
                     R2E <= '1';
57                 WHEN others =>
                     NULL; -- None of them are enabled
59             END CASE;
         END IF;
61     end process;

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

```

```

77         input(7 downto 0) <= Q2(7 downto 0);
       END CASE;
79     else
       input(7 downto 0) <= ByteInput;
81     case SelRi IS
       WHEN "00" =>
83         input(15 downto 8) <= Q0(15 downto 8);
       WHEN "01" =>
85         input(15 downto 8) <= Q1(15 downto 8);
       WHEN others =>
87         input(15 downto 8) <= Q2(15 downto 8);
       END CASE;
89     END IF;
    END IF;
91 end process;

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

```

Listing 9: processor/cpu.vhd

```

1  -- Authors:
  --      Henry Jenkins, Joel Koh
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
  --use IEEE.STD_LOGIC_ARITH.ALL;
6  --use IEEE.STD_LOGIC_UNSIGNED.ALL;

9  library work;
  use work.alu;
11 use work.cu;
  use work.ar;
13 use work.gpr;
  use work.sr;
15 use work.pc;

17
  ---- Uncomment the following library declaration if instantiating
19  ---- any Xilinx primitives in this code.
  --library UNISIM;
21  --use UNISIM.VComponents.all;

23 entity cpu is
25     PORT(
        -- instruction bus
27         inst_add  : out std_logic_vector(11 downto 0); -- Address lines.
        inst_data  : in  std_logic_vector(15 downto 0); -- Data lines.
29         inst_req  : out std_logic;                    -- Pulled low to request bus
            usage.
        inst_ack   : in  std_logic;                    -- Pulled high to inform of
            request completion.
31         -- data bus
        data_add   : out std_logic_vector(15 downto 0); -- Address lines.
33         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.
35         data_req  : out std_logic;                    -- Pulled low to request bus
            usage.
        data_ack   : inout std_logic;                    -- Pulled high to inform of
            request completion.
37         -- extras
        clk        : in  std_logic;
39         reset     : in  std_logic
    );
41 end cpu;

```

```

43 architecture cpu_arch of cpu is
44     component alu IS
45         Port (f      : in   STD_LOGIC_VECTOR (3 downto 0); -- Function (opcode)
46              rx      : in   STD_LOGIC_VECTOR (7 downto 0); -- Input x (Rx)
47              ry      : in   STD_LOGIC_VECTOR (7 downto 0); -- Input y (Ry)
48              ro      : out  STD_LOGIC_VECTOR (7 downto 0); -- Output Normally (Ry)
49              Cin      : in   STD_LOGIC; -- Carry in
50              sr      : out  STD_LOGIC_VECTOR (15 downto 0)); -- Status register out Z(0),
                    C(1), N(2)
51     END component;
52     component ar is
53         Port (clk      : in   STD_LOGIC;
54              enable    : in   STD_LOGIC;
55              Sel8Bit    : in   STD_LOGIC;
56              SelHighByte : in   STD_LOGIC;
57              ByteInput  : in   STD_LOGIC_VECTOR (7 downto 0);
58              SelRi      : in   STD_LOGIC_VECTOR (1 downto 0); -- Select the address
                    register
59              SelRo      : in   STD_LOGIC_VECTOR (1 downto 0); -- Select the address
                    register
60              Ri         : in   STD_LOGIC_VECTOR (15 downto 0); -- The input
61              Ro         : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output
62     END component;
63     component cu IS
64         Port (reset      : in   STD_LOGIC; -- '0' for reset
65              clock       : in   STD_LOGIC; -- clock
66
67              alu_f       : out  STD_LOGIC_VECTOR (3 downto 0); -- Function
68              alu_Cin     : out  STD_LOGIC; -- Carry in to ALU
69
70              -- General Purpose Registers
71              gpr_InSel   : out  STD_LOGIC; -- select the input path (0
                    - cu, 1 - ALU)
72              gpr_en     : out  STD_LOGIC; -- enable write to GPR
73              gpr_SelRx   : out  STD_LOGIC_VECTOR (2 downto 0); -- select GPR output x
74              gpr_SelRy   : out  STD_LOGIC_VECTOR (2 downto 0); -- select GPR output y
75              gpr_SelRi   : out  STD_LOGIC_VECTOR (2 downto 0); -- select GPR input
76              gpr_Ri      : out  STD_LOGIC_VECTOR (7 downto 0); -- input to GPR
77              gpr_Rx      : in   STD_LOGIC_VECTOR (7 downto 0); -- output Rx from GPR
78              gpr_Ry      : in   STD_LOGIC_VECTOR (7 downto 0); -- output Ry from GPR ,
                    not used
79
80              -- Status Register
81              sr_en       : out  STD_LOGIC; -- enable write to SR
82              sr_reset    : out  STD_LOGIC; -- reset SR
83              sr_Ro       : in   STD_LOGIC_VECTOR (15 downto 0); -- output from SR
84              -- control unit doesnt write to SR, the ALU does
85
86              -- Program Counter
87              pc_en       : out  STD_LOGIC; -- enable write to PC
88              pc_reset    : out  STD_LOGIC; -- reset PC
89              pc_Ri       : out  STD_LOGIC_VECTOR (15 downto 0); -- input to PC
90              pc_Ro       : in   STD_LOGIC_VECTOR (15 downto 0); -- output from PC
91
92              -- Address Registers
93              ar_en       : out  STD_LOGIC; -- enable write to AR
94              ar_SelRi    : out  STD_LOGIC_VECTOR (1 downto 0); -- select AR in
95              ar_SelRo    : out  STD_LOGIC_VECTOR (1 downto 0); -- select AR out
96              ar_Ri       : out  STD_LOGIC_VECTOR (15 downto 0); -- input to AR
97              ar_Ro       : in   STD_LOGIC_VECTOR (15 downto 0); -- output from AR
98              ar_sel8Bit  : out  STD_LOGIC; -- only write half the AR
99              ar_selHByte : out  STD_LOGIC; -- high or low half of the
                    AR to write
100             ar_ByteIn    : out  STD_LOGIC_VECTOR (7 downto 0); -- 8 bit input to write
                    half of AR
101
102             -- Instruction memory
103             inst_add      : out  STD_LOGIC_VECTOR (11 downto 0); -- Instruction address
104             inst_data     : in   STD_LOGIC_VECTOR (15 downto 0); -- Instruction data
105             inst_req      : out  STD_LOGIC; -- Request
106             inst_ack      : in   STD_LOGIC; -- Instruction obtained

```

```

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

115  );
  END component;
117  component gpr is
      Port (clk       : in   STD_LOGIC;
119           enable    : in   STD_LOGIC;
           SelRx      : in   STD_LOGIC_VECTOR (2 downto 0); -- The Rx output selection
              value
121           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
123           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
125           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
127           Ry         : out  STD_LOGIC_VECTOR (7 downto 0); -- The Ry output
  END component;
129  component sr is
      Port (clk       : in   STD_LOGIC;
131           enable    : in   STD_LOGIC;
           reset      : in   STD_LOGIC;
133           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
135  END component;
  component pc is
137      Port (clk       : in   STD_LOGIC;
           enable    : in   STD_LOGIC;
139           reset      : in   STD_LOGIC;
           Ri        : in   STD_LOGIC_VECTOR (15 downto 0); -- The input to the SR
141           Ro         : out  STD_LOGIC_VECTOR (15 downto 0)); -- The output from SR
  END component;
143  signal alu_Cin      : std_logic;
  signal alu_f         : std_logic_vector(3 downto 0);
145  signal alu_rx       : std_logic_vector(7 downto 0);
  signal alu_ry       : std_logic_vector(7 downto 0);
147
  signal sr_reset      : std_logic;
149  signal sr_enable    : std_logic;
  signal sr_Ro         : std_logic_vector(15 downto 0);
151  signal sr_input     : std_logic_vector(15 downto 0);

153  signal ar_enable    : STD_LOGIC; -- enable write to AR
  signal ar_SelRi      : STD_LOGIC_VECTOR (1 downto 0); -- select AR in
155  signal ar_SelRo     : STD_LOGIC_VECTOR (1 downto 0); -- select AR out
  signal ar_Ri         : STD_LOGIC_VECTOR (15 downto 0); -- input to AR
157  signal ar_Ro        : STD_LOGIC_VECTOR (15 downto 0); -- output from AR
  signal ar_sel8Bit    : STD_LOGIC; -- only write half the AR
159  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

161
  signal pc_reset      : std_logic;
163  signal pc_enable    : std_logic;
  signal pc_Ri         : std_logic_vector(15 downto 0);
165  signal pc_Ro        : std_logic_vector(15 downto 0);

167  signal gpr_InSel    : std_logic;
  signal gpr_enable    : std_logic;
169  signal gpr_SelRx     : std_logic_vector(2 downto 0);
  signal gpr_SelRy     : std_logic_vector(2 downto 0);
171  signal gpr_SelRi     : std_logic_vector(2 downto 0);
  signal gpr_RiCU      : std_logic_vector(7 downto 0);

```

```

173 signal gpr_RiCDB : std_logic_vector(7 downto 0);
begin
175
176     a: alu port map(
177         f      => alu_f,
178         rx     => alu_rx,
179         ry     => alu_ry,
180         ro     => gpr_RiCDB,
181         Cin    => alu_Cin,
182         sr     => sr_input
183     );
184     c: cu port map(
185
186         reset    => reset, -- '0' for reset
187         clock    => clk,  -- clock
188
189         alu_f    => alu_f, -- Function
190         alu_Cin  => alu_Cin, -- Carry into the ALU
191
192         -- General Purpose Registers
193         gpr_InSel => gpr_InSel, -- select the input path (0 - cu, 1 - ALU)
194         gpr_en    => gpr_enable, -- enable write to GPR
195         gpr_SelRx => gpr_SelRx, -- select GPR output x
196         gpr_SelRy => gpr_SelRy, -- select GPR output y
197         gpr_SelRi => gpr_SelRi, -- select GPR input
198         gpr_Ri    => gpr_RiCU, -- input to GPR
199         gpr_Rx    => alu_rx, -- Rx from GPR
200         --gpr_Ry   => alu_ry, -- Ry from GPR
201
202         -- Status Register
203         sr_en     => sr_enable, -- enable write to SR
204         sr_reset  => sr_reset, -- reset SR
205         sr_Ro     => sr_Ro, -- output from SR
206         -- control unit doesnt write to SR, the ALU does
207
208         -- Program Counter
209         pc_en     => pc_enable, -- enable write to PC
210         pc_reset  => pc_reset, -- reset PC
211         pc_Ri     => pc_Ri, -- input to PC
212         pc_Ro     => pc_Ro, -- output from PC
213
214         -- Address Registers
215         ar_en     => ar_enable, -- enable write to AR
216         ar_SelRi  => ar_SelRi, -- select AR in
217         ar_SelRo  => ar_SelRo, -- select AR out
218         ar_sel8Bit => ar_sel8Bit,
219         ar_selHByte => ar_selHByte,
220         ar_ByteIn => ar_ByteIn,
221         ar_Ri     => ar_Ri, -- input to AR
222         ar_Ro     => ar_Ro, -- output from AR
223
224         -- Instruction memory
225         inst_add  => inst_add, -- Instruction address
226         inst_data => inst_data, -- Instruction data
227         inst_req  => inst_req, -- Request
228         inst_ack  => inst_ack, -- Instruction obtained
229
230         data_add  => data_add, -- Data address
231         data_data => data_line, -- Data
232         data_read => data_read, -- 1 for read, 0 for write
233         data_req  => data_req, -- Request
234         data_ack  => data_ack, -- Data written to/ read from
235     );
236     address : ar port map(
237         clk      => clk,
238         enable   => ar_enable,
239         Sel8Bit  => ar_Sel8Bit,
240         SelHighByte => ar_selHByte,
241         ByteInput => ar_ByteIn,
242         SelRi    => ar_SelRi,
243         SelRo    => ar_SelRo,
244         Ri       => ar_Ri,
245         Ro       => ar_Ro

```



```

    );
247 g : gpr port map(
        clk      => clk,
249         enable => gpr_enable,
        SelRx    => gpr_SelRx,
251         SelRy  => gpr_SelRy,
        SelRi    => gpr_SelRi,
253         SelIn  => gpr_InSel,
        RiCU     => gpr_RiCU,
255         RiCDB  => gpr_RiCDB,
        Rx       => alu_rx,
257         Ry     => alu_ry
    );
259 s : sr port map(
        clk      => clk,
261         enable => sr_enable,
        reset    => sr_reset,
263         Ri     => sr_input,
        Ro       => sr_Ro
265     );
    programcounter: pc port map(
267         clk      => clk,
        enable    => pc_enable,
269         reset    => pc_reset,
        Ri        => pc_Ri,
271         Ro       => pc_Ro
    );
273 end cpu_arch;

```

Listing 10: processor/cu.vhd

```

1  -- Authors:
   --      Henry Jenkins, Joel Koh
3
   library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
   use IEEE.NUMERIC_STD.ALL;
7  use ieee.std_logic_arith.all;
   --use ieee.std_logic_unsigned.all;
9
11 library work;
   --use work.fulladder;
13 --use work.cpu.ALL;
15 -- Uncomment the following library declaration if using
   -- arithmetic functions with Signed or Unsigned values
17 --use IEEE.NUMERIC_STD.ALL;
19 -- Uncomment the following library declaration if instantiating
   -- any Xilinx primitives in this code.
21 --library UNISIM;
   --use UNISIM.VComponents.all;
23
   entity cu is
25     Port (reset      : in STD_LOGIC;           -- '0' for reset
           clock       : in STD_LOGIC;           -- clock
27
           alu_f       : out STD_LOGIC_VECTOR (3 downto 0); -- Function
29           alu_Cin    : out STD_LOGIC;           -- Carry in to ALU
31
           -- General Purpose Registers
           gpr_InSel   : out STD_LOGIC;           -- select the input path (0
               -- cu, 1 - ALU)
33           gpr_en     : out STD_LOGIC;           -- enable write to GPR
           gpr_SelRx   : out STD_LOGIC_VECTOR (2 downto 0); -- select GPR output x
35           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
37           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
39           --gpr_Ry    : in STD_LOGIC_VECTOR (7 downto 0);  -- output Ry from GPR ,
               not used

```

```

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

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

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

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

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

75      );
end cu;
77

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

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

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

95    -- Decoded data
    signal rx : std_logic_vector(2 downto 0);
97    signal ry : std_logic_vector(2 downto 0);
    signal ay : std_logic_vector(1 downto 0);
99
    -- Indicates what needs to be executed
101    signal write_gpr      : std_logic;
    signal write_sr        : std_logic;
103    signal write_pc       : std_logic;
    signal write_ar        : std_logic;
105    signal write_memory    : std_logic;

107    -- full adders
    signal A16      : std_logic_vector(15 downto 0);
109    signal B16      : std_logic_vector(15 downto 0);

```

```

111 signal    AdderCin16    : std_logic;
signal    Sum16           : std_logic_vector(15 downto 0);
signal    AdderCout16    : std_logic;

113
115 signal v      : STD_LOGIC_VECTOR(7 downto 0);  -- 8-bit immediate

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

119
    -- Process instruction
    -- Assumes all instructions are valid
121 process(clock, state, opcode, gpr_Rx, sr_Ro, pc_Ro, ar_Ro, inst_data, inst_ack,
        data_data, data_ack,
123         rx, ry, ay, v, write_gpr, write_sr, write_pc, write_ar, write_memory)
    BEGIN
125         if rising_edge(clock) then
            case state is
127                 when reset_state =>
                    sr_reset <= '0';
129                     pc_reset <= '0';
                    next_state <= fetch;

131
                    when fetch =>
133                         sr_reset <= '1';
                        pc_reset <= '1';

135
                        gpr_en <= '0';
137                         sr_en <= '0';
                        pc_en <= '0';
139                         ar_en <= '0';

141
                        write_gpr <= '0';
                        write_sr <= '0';
143                         write_pc <= '0';
                        write_ar <= '0';
145                         write_memory <= '0';

147
                        inst_add <= pc_Ro(11 downto 0);
                        if inst_ack = '0' then
149                             inst_req <= '1';
                        else
151                             opcode <= inst_data;
                            inst_req <= '0';

153
                            -- increment program counter
155                             AdderCin16 <= '1';
                            A16 <= PC_Ro;
157                             B16 <= "0000000000000000";
                            pc_Ri <= Sum16;
159                             pc_en <= '1';

161
                            next_state <= decode;
                        end if;
163                     when decode =>
                            pc_en <= '0';

165
                            -- ALU
167                             if opcode(15) = '0' and opcode(10) = '0' and not opcode(14 downto 11) =
                                "0010" then

169
                                    ry <= opcode(7 downto 5);
                                    rx <= opcode(2 downto 0);

171
                                    gpr_SelRy <= ry;
                                    gpr_SelRx <= rx;
                                    gpr_SelRi <= ry;
173                                     gpr_InSel <= '1';
                                    alu_f <= opcode(14 downto 11);
175                                     alu_Cin <= sr_Ro(1);  -- Carry

177
                                    if not opcode(14 downto 11) = "0110" then -- CMP doesnt write to gpr, all
                                        others do

```

```

181         write_gpr <= '1';
    end if;

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

185
-- Branching
187     elsif opcode(11 downto 10) = "11" then

189         v <= "00000000"; -- initialise v

191         if opcode(15) = '1' then
            case opcode(14 downto 12) is
193                 when "000" => -- BEQ
                    if sr_Ro(0) = '1' then -- Z=1
195                        v <= opcode(9 downto 2);
                    end if;
197                 when "001" => -- BNE
                    if sr_Ro(0) = '0' then -- Z=0
199                        v <= opcode(9 downto 2);
                    end if;
201                 when "010" => -- BLT
                    if sr_Ro(0) = '0' and sr_Ro(2) = '1' then -- Z=0 and N=1
203                        v <= opcode(9 downto 2);
                    end if;
205                 when "011" => -- BGT
                    if sr_Ro(0) = '0' and sr_Ro(2) = '0' then -- Z=0 and N=0
207                        v <= opcode(9 downto 2);
                    end if;
209                 when "100" => -- BC
                    if sr_Ro(1) = '1' then -- C=1
211                        v <= opcode(9 downto 2);
                    end if;
213                 when "101" => -- BNC
                    if sr_Ro(1) = '0' then -- C=0
215                        v <= opcode(9 downto 2);
                    end if;
217                 when "110" => -- RJMP
                    v <= opcode(9 downto 2);

219
                    when others =>
221                        v <= "00000000";
            end case;

223
            -- PC <- PC + v
225            AdderCin16 <= '0';
            A16 <= PC_Ro;
227            B16 <= "00000000" & v;
            pc_Ri <= Sum16;

229
        elsif opcode(15 downto 12) = "0111" then -- JMP
231            ay <= opcode(6 downto 5);

233
            -- PC <- ay
            ar_SelRo <= ay;
235            pc_Ri <= ar_Ro;
        else
237            -- should not reach here
            pc_Ri <= pc_Ro; -- no change
239        end if;

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

243
-- Addressing
245     else
247         gpr_Insel <= '0';

249         case opcode(12 downto 10) is

251             when "001" => -- Load
                if opcode(15) = '1' then -- immediate

```

```

253         rx <= '0' & opcode(1 downto 0);
254         v <= opcode(9 downto 2);
255
256         -- rx <- v
257         gpr_SelRi <= rx;
258         gpr_Ri <= v;
259         write_gpr <= '1';
260         next_state <= execute;
261     else -- direct
262         rx <= opcode(2 downto 0);
263         ay <= opcode(6 downto 5);
264
265         -- rx <- [ay]
266         gpr_SelRi <= rx;
267         ar_selRo <= ay;
268         data_add <= ar_Ro;
269         data_read <= '1';
270         if data_ack = '0' then -- request data
271             data_req <= '1';
272         else -- data obtained
273             gpr_Ri <= data_data;
274             data_req <= '0';
275             write_gpr <= '1';
276
277         case opcode(14 downto 13) is
278             when "01" => -- auto increment
279                 AdderCin16 <= '1';
280                 A16 <= ar_Ro;
281                 B16 <= "0000000000000000";
282                 ar_selRi <= ay;
283                 ar_sel8bit <= '0';
284                 ar_Ri <= Sum16;
285                 write_ar <= '1';
286             when "10" => -- auto decrement
287                 AdderCin16 <= '0';
288                 A16 <= ar_Ro;
289                 B16 <= "1111111111111111";
290                 ar_selRi <= ay;
291                 ar_sel8bit <= '0';
292                 ar_Ri <= Sum16;
293                 write_ar <= '1';
294             when others =>
295                 -- do nothing
296         end case;
297         next_state <= execute;
298     end if;
299 end if;
300
301 when "101" => -- Store
302     if opcode(15) = '1' then -- immediate
303         ay <= opcode(1 downto 0);
304         v <= opcode(9 downto 2);
305
306         -- [ay] <- v
307         ar_selRo <= ay;
308         data_add <= ar_Ro;
309         data_read <= '0';
310         data_data <= v;
311         write_memory <= '1';
312         next_state <= execute;
313     else -- direct
314         rx <= opcode(2 downto 0);
315         ay <= opcode(6 downto 5);
316
317         -- [ay] <- rx
318         gpr_selRx <= rx;
319         ar_selRo <= ay;
320         data_add <= ar_Ro;
321         data_read <= '0';
322         data_data <= gpr_Rx;
323         write_memory <= '1';
324
325         case opcode(14 downto 13) is

```

```

327         when "01" => -- auto increment
            AdderCin16 <= '1';
            A16 <= ar_Ro;
329            B16 <= "0000000000000000";
            ar_selRi <= ay;
331            ar_sel8bit <= '0';
            ar_Ri <= Sum16;
333            write_ar <= '1';
            when "10" => -- auto decrement
335                AdderCin16 <= '0';
                A16 <= ar_Ro;
337                B16 <= "1111111111111111";
                ar_selRi <= ay;
339                ar_sel8bit <= '0';
                ar_Ri <= Sum16;
341                write_ar <= '1';
            when others =>
343                -- do nothing
            end case;
345            next_state <= execute;
        end if;
347
    when "100" => -- Move
349        if opcode(9) = '1' then
            rx <= opcode(2 downto 0);
351            ay <= opcode(6 downto 5);

353            -- ayn <- rx
            gpr_selRx <= rx;
355            ar_selRi <= ay;
            ar_sel8bit <= '1';
357            ar_ByteIn <= gpr_Rx;

359            if opcode(8) = '1' then -- high
                ar_selHByte <= '1';
361            else -- low
                ar_selHByte <= '0';
363            end if;

365            write_ar <= '1';
            next_state <= execute;
367
        elsif opcode(4) = '1' then
369            rx <= opcode(7 downto 5);
            ay <= opcode(1 downto 0);
371

            -- rx <- ayn
            gpr_selRi <= rx;
373            ar_selRo <= ay;
375

            if opcode(3) = '1' then -- high
377                gpr_Ri <= ar_Ro(15 downto 8);
            else -- low
379                gpr_Ri <= ar_Ro(7 downto 0);
            end if;
381

            write_gpr <= '1';
            next_state <= execute;
383

        else
385            rx <= opcode(2 downto 0);
            ry <= opcode(7 downto 5);
387

            -- ry <- rx
            gpr_SelRx <= rx;
391            gpr_SelRi <= ry;
            gpr_Ri <= gpr_Rx;
393

            write_gpr <= '1';
            next_state <= execute;
395

        end if;
397

```

```

399         when others =>
401             -- should not reach here

403     end case;

405 end if;

407 when execute =>
408     if write_memory = '1' then
409         if data_ack = '0' then -- request write
410             data_req <= '1';
411         else -- data written
412             data_req <= '0';
413             gpr_en <= write_gpr;
414             sr_en <= write_sr;
415             pc_en <= write_pc;
416             ar_en <= write_ar;
417             next_state <= fetch;
418         end if;
419     else
420         gpr_en <= write_gpr;
421         sr_en <= write_sr;
422         pc_en <= write_pc;
423         ar_en <= write_ar;
424         next_state <= fetch;
425     end if;

427 when others =>
428     -- shouldnt reach here
429     next_state <= reset_state;
430 end case;
431 end if;
432 end process;

433 process(clock, reset, next_state)
434 BEGIN
435     if reset = '0' then
436         state <= reset_state;
437     elsif rising_edge(clock) then
438         state <= next_state;
439     end if;
440 end process;

441 end Behavioral;

```

Listing 11: processor/fulladder.vhd

```

-- Authors:
2 --      Henry Jenkins, Joel Koh

4 library IEEE;
5 use IEEE.STD_LOGIC_1164.ALL;

6
7 entity fulladder is
8     Port (Ax   : in   STD_LOGIC;
9           Bx   : in   STD_LOGIC;
10          Ci   : in   STD_LOGIC;
11          Sx   : out  STD_LOGIC;
12          Co   : out  STD_LOGIC
13        );
14 end fulladder;

16
17 architecture arch_fulladder of fulladder is
18 BEGIN
19     process(Ax, Bx, Ci)
20     BEGIN
21         Sx <= (Ax XOR Bx) XOR Ci;
22         Co <= (Ax and Bx) or (Ax and Ci) OR (Bx AND Ci);
23     end process;
24 end arch_fulladder;

```

```

26 -----
28 library IEEE;
29 use IEEE.STD_LOGIC_1164.ALL;
30
31 entity fulladder8 is
32     Port (A      : in    STD_LOGIC_VECTOR( 7 downto 0);
33           B      : in    STD_LOGIC_VECTOR( 7 downto 0);
34           Cin     : in    STD_LOGIC;
35           Sum     : out   STD_LOGIC_VECTOR( 7 downto 0);
36           Cout    : out   STD_LOGIC
37           );
38 end fulladder8;
39
40 architecture arch_fulladder8 of fulladder8 is
41     component fulladder IS
42     Port (Ax      : in    STD_LOGIC;
43           Bx      : in    STD_LOGIC;
44           Ci      : in    STD_LOGIC;
45           Sx      : out   STD_LOGIC;
46           Co      : out   STD_LOGIC
47           );
48     end component;
49     signal Carry   : std_logic_vector(8 downto 0);
50 BEGIN
51     Carry(0) <= Cin;
52
53     FA0: fulladder PORT MAP(A(0), B(0), Carry(0), Sum(0), Carry(1));
54     FA1: fulladder PORT MAP(A(1), B(1), Carry(1), Sum(1), Carry(2));
55     FA2: fulladder PORT MAP(A(2), B(2), Carry(2), Sum(2), Carry(3));
56     FA3: fulladder PORT MAP(A(3), B(3), Carry(3), Sum(3), Carry(4));
57     FA4: fulladder PORT MAP(A(4), B(4), Carry(4), Sum(4), Carry(5));
58     FA5: fulladder PORT MAP(A(5), B(5), Carry(5), Sum(5), Carry(6));
59     FA6: fulladder PORT MAP(A(6), B(6), Carry(6), Sum(6), Carry(7));
60     FA7: fulladder PORT MAP(A(7), B(7), Carry(7), Sum(7), Carry(8));
61
62     Cout <= Carry(8);
63 end arch_fulladder8;
64 -----
65
66 library IEEE;
67 use IEEE.STD_LOGIC_1164.ALL;
68
69 entity fulladder16 is
70     Port (A      : in    STD_LOGIC_VECTOR( 15 downto 0);
71           B      : in    STD_LOGIC_VECTOR( 15 downto 0);
72           Cin     : in    STD_LOGIC;
73           Sum     : out   STD_LOGIC_VECTOR( 15 downto 0);
74           Cout    : out   STD_LOGIC
75           );
76 end fulladder16;
77
78 architecture arch_fulladder16 of fulladder16 is
79     component fulladder IS
80     Port (Ax      : in    STD_LOGIC;
81           Bx      : in    STD_LOGIC;
82           Ci      : in    STD_LOGIC;
83           Sx      : out   STD_LOGIC;
84           Co      : out   STD_LOGIC
85           );
86     end component;
87     signal Carry   : std_logic_vector(16 downto 0);
88 BEGIN
89     Carry(0) <= Cin;
90
91     FA0: fulladder PORT MAP(A(0), B(0), Carry(0), Sum(0), Carry(1));
92     FA1: fulladder PORT MAP(A(1), B(1), Carry(1), Sum(1), Carry(2));
93     FA2: fulladder PORT MAP(A(2), B(2), Carry(2), Sum(2), Carry(3));
94     FA3: fulladder PORT MAP(A(3), B(3), Carry(3), Sum(3), Carry(4));
95     FA4: fulladder PORT MAP(A(4), B(4), Carry(4), Sum(4), Carry(5));
96     FA5: fulladder PORT MAP(A(5), B(5), Carry(5), Sum(5), Carry(6));

```



```

98     FA6:    fulladder PORT MAP(A(6), B(6), Carry(6), Sum(6), Carry(7));
99     FA7:    fulladder PORT MAP(A(7), B(7), Carry(7), Sum(7), Carry(8));
100    FA8:    fulladder PORT MAP(A(8), B(8), Carry(8), Sum(8), Carry(9));
101    FA9:    fulladder PORT MAP(A(9), B(9), Carry(9), Sum(9), Carry(10));
102    FA10:   fulladder PORT MAP(A(10), B(10), Carry(10), Sum(10), Carry(11));
103    FA11:   fulladder PORT MAP(A(11), B(11), Carry(11), Sum(11), Carry(12));
104    FA12:   fulladder PORT MAP(A(12), B(12), Carry(12), Sum(12), Carry(13));
105    FA13:   fulladder PORT MAP(A(13), B(13), Carry(13), Sum(13), Carry(14));
106    FA14:   fulladder PORT MAP(A(14), B(14), Carry(14), Sum(14), Carry(15));
107    FA15:   fulladder PORT MAP(A(15), B(15), Carry(15), Sum(15), Carry(16));
108
109    Cout <= Carry(16);
110 end arch_fulladder16;

```

Listing 12: processor/gpr.vhd

```

-- Authors:
2 --      Henry Jenkins, Joel Koh

4 library IEEE;
5 use IEEE.STD_LOGIC_1164.ALL;
6
7 library work;
8 use work.reg8;
9
10 entity gpr is
11     Port (clk      : in    STD_LOGIC;
12          enable   : in    STD_LOGIC;
13          SelRx    : in    STD_LOGIC_VECTOR (2 downto 0); -- The Rx output selection
14              value
15          SelRy    : in    STD_LOGIC_VECTOR (2 downto 0); -- The Ry output selection
16              value
17          SelRi    : in    STD_LOGIC_VECTOR (2 downto 0); -- The Ri input selection
18              value
19          SelIn    : in    STD_LOGIC; -- Select where the input should be from the CU
20              or CDB
21          RiCU     : in    STD_LOGIC_VECTOR (7 downto 0); -- Input from the Control
22              Unit
23          RiCDB    : in    STD_LOGIC_VECTOR (7 downto 0); -- Input from the Common Data
24              Bus
25          Rx       : out   STD_LOGIC_VECTOR (7 downto 0); -- The Rx output
26          Ry       : out   STD_LOGIC_VECTOR (7 downto 0)); -- The Ry output
27 end gpr;
28
29
30 architecture gpr_arch of gpr is
31     component reg8 IS
32         port(I      : in    std_logic_vector(7 downto 0);
33              clock   : in    std_logic;
34              enable  : in    std_logic;
35              reset   : in    std_logic;
36              Q       : out   std_logic_vector(7 downto 0)
37         );
38     end component;
39
40     signal reset: std_logic := '0';
41     signal input: std_logic_VECTOR (7 downto 0);
42     signal R0E : std_logic; -- Enable signals
43     signal R1E : std_logic;
44     signal R2E : std_logic;
45     signal R3E : std_logic;
46     signal R4E : std_logic;
47     signal R5E : std_logic;
48     signal R6E : std_logic;
49     signal R7E : std_logic;
50     signal Q0  : std_logic_VECTOR (7 downto 0);
51     signal Q1  : std_logic_VECTOR (7 downto 0);
52     signal Q2  : std_logic_VECTOR (7 downto 0);
53     signal Q3  : std_logic_VECTOR (7 downto 0);
54     signal Q4  : std_logic_VECTOR (7 downto 0);
55     signal Q5  : std_logic_VECTOR (7 downto 0);
56     signal Q6  : std_logic_VECTOR (7 downto 0);

```

```

    signal Q7 : std_logic_VECTOR (7 downto 0);
52 BEGIN
    reg_0 : reg8 port map(input, clk, R0E, reset, Q0);
54    reg_1 : reg8 port map(input, clk, R1E, reset, Q1);
    reg_2 : reg8 port map(input, clk, R2E, reset, Q2);
56    reg_3 : reg8 port map(input, clk, R3E, reset, Q3);
    reg_4 : reg8 port map(input, clk, R4E, reset, Q4);
58    reg_5 : reg8 port map(input, clk, R5E, reset, Q5);
    reg_6 : reg8 port map(input, clk, R6E, reset, Q6);
60    reg_7 : reg8 port map(input, clk, R7E, reset, Q7);

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

72 -- Set Ri the input
    SetInput: process(clk, enable, SelRi)
74 BEGIN
    R0E <= '0';
76    R1E <= '0';
    R2E <= '0';
78    R3E <= '0';
    R4E <= '0';
80    R5E <= '0';
    R6E <= '0';
82    R7E <= '0';
    IF enable = '1' THEN
84         case SelRi IS
            WHEN "000" =>
86                 R0E <= '1';
            WHEN "001" =>
88                 R1E <= '1';
            WHEN "010" =>
90                 R2E <= '1';
            WHEN "011" =>
92                 R3E <= '1';
            WHEN "100" =>
94                 R4E <= '1';
            WHEN "101" =>
96                 R5E <= '1';
            WHEN "110" =>
98                 R6E <= '1';
            WHEN "111" =>
100                R7E <= '1';
            WHEN others =>
102                NULL; -- None of them are enabled
        end case;
104    END IF;
    end process;

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

118 -- Set the Ry output
    WITH SelRy SELECT
120    Ry <= Q0 WHEN "000",
         Q1 WHEN "001",
122         Q2 WHEN "010",
         Q3 WHEN "011",

```

```

124         Q4 WHEN "100",
           Q5 WHEN "101",
126         Q6 WHEN "110",
           Q7 WHEN others;
128     end gpr_arch;

```

Listing 13: processor/reg.vhd

```

-- Authors:
2  --      Henry Jenkins, Joel Koh

4  library ieee;
   use ieee.std_logic_1164.all;
6
   entity reg8 is
8  port(I      : in  std_logic_vector(7 downto 0);
        clock   : in  std_logic;
10     enable  : in  std_logic;
        reset   : in  STD_LOGIC;
12     Q       : out std_logic_vector(7 downto 0)
        );
14 end reg8;

16 architecture behv of reg8 is
   begin
18     process(I, clock, enable, reset)
19     begin
20         IF reset = '1' THEN
22             Q <= (others => '0');
           ELSIF rising_edge(clock) then
24             if enable = '1' then
26                 Q <= I;
27             end if;
28         end if;
29     end process;
30 end behv;
32
-----
34
35 library ieee;
36 use ieee.std_logic_1164.all;
37
38 entity reg16 is
39 port(I      : in  std_logic_vector(15 downto 0);
40     clock   : in  std_logic;
41     enable  : in  std_logic;
42     reset   : in  STD_LOGIC;
43     Q       : out std_logic_vector(15 downto 0)
44     );
45 end reg16;
46
47 architecture behv of reg16 is
   begin
48     process(I, clock, enable, reset)
49     begin
50         IF reset = '1' THEN
52             Q <= (others => '0');
           ELSIF rising_edge(clock) then
54             if enable = '1' then
56                 Q <= I;
57             end if;
58         end if;
59     end process;
60 end behv;
62

```

Listing 14: processor/spr.vhd

```

-- Authors:
2  --      Henry Jenkins, Joel Koh

4  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;

6
   library work;
8  use work.reg16;

10 entity sr is
    Port (clk      : in  STD_LOGIC;
12         enable   : in  STD_LOGIC;
         reset     : in  STD_LOGIC;
14         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
16 end sr;

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

30
-----
32
   library IEEE;
34 use IEEE.STD_LOGIC_1164.ALL;

36 entity pc is
    Port (clk      : in  STD_LOGIC;
38         enable   : in  STD_LOGIC;
         reset     : in  STD_LOGIC;
40         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
42 end pc;

44
   architecture pc_arch of pc is
46     component reg16 IS
        port(I      : in  std_logic_vector(15 downto 0);
48         clock    : in  std_logic;
         enable    : in  std_logic;
50         reset    : in  STD_LOGIC;
         Q         : out std_logic_vector(15 downto 0)
52     );
        end component;
54 BEGIN
        reg_pc : reg16 port map(Ri, clk, enable, reset, Ro);
56 end pc_arch;

```

MMU

Listing 15: mmu/control_unit.vhd

```
1  -- Authors:
2  --      Wim Looman, Forrest McKerchar
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  library work;
8  use work.mmu_types.all;
9
10 entity mmu_control_unit is
11     port (
12         eoc          : in  std_logic; -- High on muart has finished collecting data
13         eot          : in  std_logic; -- High on muart has finished transmitting
14         ready        : in  std_logic; -- High if the muart is ready for new transfer
15         data_read     : in  std_logic; -- High if the cpu requests a read, else write
16         data_req      : in  std_logic; -- Low to start a transfer
17         data_add_0    : in  std_logic; -- High for memory address, else IO
18         inst_req      : in  std_logic; -- Low to start a transfer
19         fr           : in  std_logic; -- Input headers fetch request bit
20         inst_or_data_in : in  std_logic; -- Input headers inst or data bit
21         rw           : in  std_logic; -- Input headers read/write bit
22         write        : out std_logic; -- Pulled high to start muart writing data
23         inst_or_data_out : out std_logic; -- Output headers inst or data bit
24         inst_ack      : out std_logic; -- Idles high, pulled low when data ready
25         data_ack      : inout std_logic; -- Idles 'Z', high when data not ready,
26             pulled low when data ready
27         muart_input   : out muart_input_state; -- Signal connected to muart input
28         muart_output  : out muart_output_state; -- Signal connected to muart output
29         clk          : in  std_logic
30     );
31 end mmu_control_unit;
32
33 architecture mmu_control_unit_arch of mmu_control_unit is
34     component data_control_unit is
35         port (
36             eoc          : in  std_logic; -- High on muart has finished collecting data
37             eot          : in  std_logic; -- High on muart has finished transmitting
38             ready        : in  std_logic; -- High if the muart is ready for new transfer
39             data_read     : in  std_logic; -- High if the cpu requests a read, else write
40             data_req      : in  std_logic; -- Low to start a transfer
41             data_add_0    : in  std_logic; -- High for memory address, else IO
42             write        : out std_logic; -- Pulled high to start muart writing data.
43             data_ack      : inout std_logic; -- Idles 'Z', high when data not ready, pulled
44                 low when data ready
45             muart_input   : out muart_input_state; -- Signal connected to muart input
46             muart_output  : out muart_output_state; -- Signal connected to muart output
47             clk          : in  std_logic
48         );
49     end component;
50
51     component inst_control_unit is
52         port (
53             eoc          : in  std_logic; -- High on muart has finished collecting data
54             eot          : in  std_logic; -- High on muart has finished transmitting
55             ready        : in  std_logic; -- High if the muart is ready for new transfer
56             inst_req      : in  std_logic; -- Low to start a transfer
57             write        : out std_logic; -- Pulled high to start muart writing data
58             inst_or_data_in : out std_logic; -- Output headers inst or data bit
59             inst_ack      : out std_logic; -- Idles high, pulled low when data ready
60             muart_input   : out muart_input_state; -- Signal connected to muart input
61             muart_output  : out muart_output_state; -- Signal connected to muart output
62             clk          : in  std_logic
63         );
64     end component;
65
66     signal data_write, inst_write, inst_inst_or_data_out : std_logic;
67     signal data_muart_input, inst_muart_input : muart_input_state;
68     signal data_muart_output, inst_muart_output : muart_output_state;
```

```

67 begin
    data_cu : data_control_unit port map (
69         eoc,
            eot,
71         ready,
            data_read,
73         data_req,
            data_add_0,
75         data_write,
            data_ack,
77         data_muart_input,
            data_muart_output,
79         clk
    );
81 inst_cu : inst_control_unit port map (
    eoc,
83     eot,
        ready,
85     inst_req,
        inst_write,
87     inst_inst_or_data_out,
        inst_ack,
89     inst_muart_input,
        inst_muart_output,
91     clk
    );
93
    inst_or_data_out <= inst_inst_or_data_out ;
95 write             <= inst_write or data_write;
    uart_input      <= inst_muart_input when inst_inst_or_data_out = '1' else
97         data_muart_input;
    uart_output     <= inst_muart_output when inst_inst_or_data_out = '1' else
99         data_muart_output;
end mmu_control_unit_arch;

```

Listing 16: mmu/data_control_unit.vhd

```

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

4 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

6
library work;
8 use work.mmu_types.all;

10 entity data_control_unit is
    port (
12         eoc           : in  std_logic; -- High on uart has finished collecting data
            eot           : in  std_logic; -- High on uart has finished transmitting
14         ready        : in  std_logic; -- High if the uart is ready for new transfer
            data_read     : in  std_logic; -- High if the cpu requests a read, else write
16         data_req     : in  std_logic; -- Low to start a transfer
            data_add_0    : in  std_logic; -- High for memory address, else IO
18         write        : out std_logic; -- Pulled high to start uart writing data.
            data_ack      : inout std_logic; -- Idles 'Z', high when data not ready, pulled
                low when data ready
20         uart_input   : out uart_input_state; -- Signal connected to uart input
            uart_output  : out uart_output_state; -- Signal connected to uart output
22         clk          : in  std_logic
    );
24 end data_control_unit;

26 architecture data_control_unit_arch of data_control_unit is
    type m_state_type is (
28         idle,
            send_header, send_add_high, send_add_low, send_data,
30         get_header,  get_add_high,  get_add_low,  get_data,
            finished
32     );
    type state_type      is (idle, get_data, wait_clear);
34    type read_state_type is (idle, wait_data, read_data, pause, finished);

```

```

type transmit_state_type is (idle, set_data, trans_data, pause, finished);
36
signal state,                next_state                : state_type                := idle;
38 signal get_state,          next_get_state            : m_state_type                := idle;
signal reader_state,         next_reader_state         : read_state_type            := idle;
40 signal transmitter_state,  next_transmitter_state    : transmit_state_type        := idle;
begin
42 data_fsm : process(state, data_req, clk) begin
    if (rising_edge(clk)) then
44         case state is
            when idle =>
46             if (data_req = '0' and data_add_0 = '1') then
                 next_state <= get_data;
48             end if;

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

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

60                 when others =>
                     NULL;
62             end case;
        end if;
64    end process data_fsm;

66 get_data_fsm : process(state, clk) begin
    if rising_edge(clk) then
68         if state = get_data then
            case get_state is
70                 when idle =>
                     next_get_state <= send_header;
72

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

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

82                 when send_add_high =>
84                     if transmitter_state = finished then
                         if data_read = '1' then
86                             next_get_state <= get_header;
                         else
88                             next_get_state <= send_data;
                         end if;
90                     end if;

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

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

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

106                 when get_add_high =>

```

```

108         if reader_state = finished then
109             next_get_state <= get_data;
110         end if;

112     when get_data =>
113         if reader_state = finished then
114             next_get_state <= finished;
115         end if;

116     when finished =>
117         next_get_state <= idle;

120     when others =>
121         NULL;
122     end case;
123 end if;
124 end if;
125 end process get_data_fsm;

126 transmit_fsm : process(clk, get_state, transmitter_state, eot) begin
127     if rising_edge(clk) then
128         if ((get_state = send_header) or
129             (get_state = send_add_low) or
130             (get_state = send_add_high) or
131             (get_state = send_data)) then
132             case transmitter_state is
133             when idle =>
134                 if ready = '1' and eot = '0' then
135                     next_transmitter_state <= set_data;
136                 end if;

138             when set_data =>
139                 next_transmitter_state <= trans_data;

142             when trans_data =>
143                 next_transmitter_state <= pause;

144             when pause =>
145                 if eot = '1' then
146                     next_transmitter_state <= finished;
147                 end if;

150             when finished =>
151                 next_transmitter_state <= idle;

152             when others =>
153                 NULL;
154             end case;
155         end if;
156     end if;
157 end process transmit_fsm;

158 read_fsm : process(clk, get_state, reader_state, eoc) begin
159     if rising_edge(clk) then
160         if ((get_state = get_header) or
161             (get_state = get_add_low) or
162             (get_state = get_add_high) or
163             (get_state = get_data)) then
164             case reader_state is
165             when idle =>
166                 next_reader_state <= wait_data;

170             when wait_data =>
171                 if eoc = '1' then
172                     next_reader_state <= read_data;
173                 end if;

174             when read_data =>
175                 next_reader_state <= pause;

176             when pause =>
177                 if eoc = '0' then
178                     next_reader_state <= finished;

```



```

        end if;
182
        when finished =>
184            next_reader_state <= idle;

186            when others =>
                NULL;
188            end case;
        end if;
190    end process read_fsm;

192    switch_states : process(
194        clk,
        next_state,
196        next_get_state,
        next_reader_state,
198        next_transmitter_state
    ) begin
200        if rising_edge(clk) then
            state <= next_state;
202            get_state <= next_get_state;
            reader_state <= next_reader_state;
204            transmitter_state <= next_transmitter_state;
        end if;
206    end process switch_states;

208    -- Outputs
    with state select
210        data_ack <= '1' when wait_clear,
                                'Z' when idle,
212                                '0' when others;

214    with transmitter_state select
        write <= '1' when trans_data,
216                '0' when others;

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

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

```

Listing 17: mmu/header_builder.vhd

```

1  -- Author:
   --      Forrest McKerchar
3
   -- builds a header to feed into the RS-232 link
5
   library IEEE;
7  use IEEE.STD_LOGIC_1164.ALL;

9  library work;

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

```

```

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

```

Listing 18: mmu/header_decoder.vhd

```

-- Author:
2 --      Forrest McKerchar

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

6 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

8
library work;
10
entity header_decoder is
12   port (
       read_write      : out  std_logic; -- 1 = read, 0 = write
14       fetch_request : out  std_logic;
       inst_data       : out  std_logic; -- 1 = inst, 0 = data
16       header        : in  std_logic_vector(7 downto 0)
   );
18 end header_decoder;

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

```

Listing 19: mmu/inst_control_unit.vhd

```

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

4 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

6
library work;
8 use work.mmu_types.all;

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

24
architecture inst_control_unit_arch of inst_control_unit is
26   type m_state_type is (

```

```

idle,
28   send_header,   send_add_high, send_add_low,
   get_header,    get_add_high,  get_add_low,
30   get_data_high, get_data_low,  finished
);
32 type state_type           is (idle, get_data, wait_clear);
type read_state_type       is (idle, wait_data, read_data, pause, finished);
34 type transmit_state_type is (idle, set_data, trans_data, pause, finished);

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

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

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

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

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

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

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

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

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

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

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

98     when get_data_low =>

```

```

100         if reader_state = finished then
101             next_get_state <= get_data_high;
102         end if;

104     when get_data_high =>
105         if reader_state = finished then
106             next_get_state <= finished;
107         end if;

108     when finished =>
109         next_get_state <= idle;

112     when others =>
113         NULL;
114     end case;
115 end if;
116 end process get_inst_fsm;

118 transmit_fsm : process(clk, get_state, transmitter_state, eot) begin
119     if ((get_state = send_header) or
120         (get_state = send_add_low) or
121         (get_state = send_add_high)) then
122     case transmitter_state is
123         when idle =>
124             if ready = '1' and eot = '0' then
125                 next_transmitter_state <= set_data;
126             end if;

128         when set_data =>
129             next_transmitter_state <= trans_data;

130         when trans_data =>
131             next_transmitter_state <= pause;

132         when pause =>
133             if eot = '1' then
134                 next_transmitter_state <= finished;
135             end if;

136         when finished =>
137             next_transmitter_state <= idle;

142         when others =>
143             NULL;
144         end case;
145     end if;
146 end process transmit_fsm;

148 read_fsm : process(clk, get_state, reader_state, eoc) begin
149     if ((get_state = get_header) or
150         (get_state = get_add_low) or
151         (get_state = get_add_high) or
152         (get_state = get_data_low) or
153         (get_state = get_data_high)) then
154     case reader_state is
155         when idle =>
156             next_reader_state <= wait_data;

158         when wait_data =>
159             if eoc = '1' then
160                 next_reader_state <= read_data;
161             end if;

162         when read_data =>
163             next_reader_state <= pause;

164         when pause =>
165             if eoc = '0' then
166                 next_reader_state <= finished;
167             end if;

168         when finished =>
169             next_reader_state <= idle;

```

```

174         when others =>
175             NULL;
176         end case;
177     end if;
178 end process read_fsm;

180 switch_states : process(
181     clk,
182     next_state,
183     next_get_state,
184     next_reader_state,
185     next_transmitter_state) begin
186     if rising_edge(clk) then
187         state <= next_state;
188         get_state <= next_get_state;
189         reader_state <= next_reader_state;
190         transmitter_state <= next_transmitter_state;
191     end if;
192 end process switch_states;

194 -- Outputs
195 with state select
196     inst_ack <= '1' when wait_clear,
197               '0' when others;

198 with state select
199     inst_or_data <= '0' when idle,
200                  '1' when others;

202 with transmitter_state select
203     write <= '1' when trans_data,
204            '0' when others;

206

208 uart_input <= idle          when transmitter_state /= set_data      else
209                  idle        when transmitter_state /= trans_data   else
210                  header       when get_state        = send_header   else
211                  inst_add_high when get_state        = send_add_high else
212                  inst_add_low  when get_state        = send_add_low  else
213                  idle;

214 uart_output <= idle          when reader_state /= read_data      else
215                  header       when get_state    = get_header     else
216                  inst_data_high when get_state    = get_data_high  else
217                  inst_data_low  when get_state    = get_data_low   else
218                  idle;

220 end inst_control_unit_arch;

```

Listing 20: mmu/mmu.vhd

```

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

4 library IEEE;
5 use IEEE.STD_LOGIC_1164.ALL;

6
7 library work;
8 use work.mmu_types.all;
9 use work.mmu_control_unit;
10 use work.header_builder;
11 use work.header_decoder;
12 use work.reg8;

14 use work.minimal_uart_core;

16 entity mmu_main is
17     port (
18         -- instruction bus
19         inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
20         inst_data  : out std_logic_vector(15 downto 0); -- Data lines.

```

```

    inst_req : in std_logic; -- Pulled low to request bus
    usage.
22    inst_ack : out std_logic; -- Pulled high to inform of
        request completion.
    -- data bus
24    data_add : in std_logic_vector(15 downto 0); -- Address lines.
    data_line : inout std_logic_vector(7 downto 0); -- Data lines.
26    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.
28    data_ack : inout std_logic; -- Pulled high to inform of
        request completion.
    -- extras
30    clk : in std_logic;
    receive_pin : in std_logic;
32    transfer_pin : out std_logic
);
34 end mmu_main;

36 architecture mmu_arch of mmu_main is
    component mmu_control_unit is
38    port (
        eoc : in std_logic; -- High on muart has finished collecting
        data
40        eot : in std_logic; -- High on muart has finished transmitting
        ready : in std_logic; -- High if the muart is ready for new
        transfer
42        data_read : in std_logic; -- High if the cpu requests a read, else
        write
        data_req : in std_logic; -- Low to start a transfer
44        data_add_0 : in std_logic; -- High for memory address, else IO
        inst_req : in std_logic; -- Low to start a transfer
46        fr : in std_logic; -- Input headers fetch request bit
        inst_or_data_in : in std_logic; -- Input headers inst or data bit
48        rw : in std_logic; -- Input headers read!/write bit
        write : out std_logic; -- Pulled high to start muart writing data
50        inst_or_data_out : out std_logic; -- Output headers inst or data bit
        inst_ack : out std_logic; -- Idles high, pulled low when data ready
52        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
54        muart_output : out muart_output_state; -- Signal connected to muart
        output
        clk : in std_logic
    );
56 end component;

58 component header_builder is
60    port (
        read_write : in std_logic; -- 1 = read, 0 = write
62        inst_data : in std_logic; -- 1 = inst, 0 = data
        header : out std_logic_vector(7 downto 0)
64    );
    end component;

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

76 component minimal_uart_core is
    port(
78        clock : in std_logic;
        eoc : out std_logic;
80        outp : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
        rxd : in std_logic;
82        txd : out std_logic;
        eot : out std_logic;

```

```

84     inp  : in    std_logic_vector(7 downto 0);
      ready : out   std_logic;
86     wr   : in    std_logic
    );
88 end component;

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

100
102 signal eoc      : std_logic;
104 signal eot      : std_logic;
106 signal ready    : std_logic;
108 signal fr       : std_logic;
110 signal inst_or_data_in : std_logic;
112 signal rw       : std_logic;

114 signal write     : std_logic;
116 signal inst_or_data_out : std_logic;
118 signal muart_input  : muart_input_state;
120 signal muart_output : muart_output_state;

122 signal muart_out : std_logic_vector(7 downto 0);
124 signal muart_in  : std_logic_vector(7 downto 0);
126 signal header_in : std_logic_vector(7 downto 0);
128 signal header_out : std_logic_vector(7 downto 0);
130 signal inst_data_high_enable : std_logic;
132 signal inst_data_low_enable : std_logic;
134 signal data_data_enable : std_logic;
136 signal data_line_tri : std_logic_vector(7 downto 0);

138 begin
140 muart : minimal_uart_core port map (
142     clk,
144     eoc,
146     muart_out,
148     receive_pin,
150     transfer_pin,
152     eot,
154     muart_in,
156     ready,
158     write
    );
160 cu : mmu_control_unit port map (
162     eoc,
164     eot,
166     ready,
168     data_read,
170     data_req,
172     data_add(0),
174     inst_req,
176     fr,
178     inst_or_data_in,
180     rw,
182     write,
184     inst_or_data_out,
186     inst_ack,
188     data_ack,
190     muart_input,
192     muart_output,
194     clk
    );
196 hb : header_builder port map (
198     data_read,
200     inst_or_data_out,
202     header_out

```

```

);
158  hd : header_decoder port map (
      rw,
160      fr,
      inst_or_data_in,
162      header_in
    );
164  idh : reg8 port map (
      muart_out,
166      clk,
      inst_data_high_enable,
168      '0',
      inst_data(15 downto 8)
170  );
172  idl : reg8 port map (
      muart_out,
      clk,
174      inst_data_low_enable,
      '0',
176      inst_data(7 downto 0)
    );
178  dd : reg8 port map (
      muart_out,
180      clk,
      data_data_enable,
182      '0',
      data_line_tri
184  );

186  with muart_input select
      muart_in <= header_out          when header,
188                  "0000" & inst_add(11 downto 8) when inst_add_high,
                  inst_add(7 downto 0)         when inst_add_low,
190                  '0' & data_add(15 downto 9)   when data_add_high,
                  data_add(8 downto 1)         when data_add_low,
192                  data_line                 when data_data,
                  (others => '0')             when others;

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

204          when inst_data_high =>
              inst_data_high_enable <= '1';

206          when inst_data_low =>
              inst_data_low_enable <= '1';

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

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

220  end mmu_arch;

```

Listing 21: mmu/mmu_types.vhd

```

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

4 library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;

```



```

6
package mmu_types is
8   type uart_input_state is (header, inst_add_high, inst_add_low, data_add_high,
      data_add_low, data_data, idle);
      type uart_output_state is (header, inst_data_high, inst_data_low, data_data,
      clear_data, idle);
10 end mmu_types;

```

Listing 22: mmu/muart/BRG.vhd

```

--*****
2  --* Minimal UART ip core *
--* Author: Arao Hayashida Filho      arao@medinovacao.com.br *
4  --* *
--*****
6  --* *
--* Copyright (C) 2009 Arao Hayashida Filho *
8  --* *
--* This source file may be used and distributed without *
12 --* restriction provided that this copyright statement is not *
--* removed from the file and that any derivative work contains *
12 --* the original copyright notice and the associated disclaimer. *
--* *
14 --* This source file is free software; you can redistribute it *
--* and/or modify it under the terms of the GNU Lesser General *
16 --* Public License as published by the Free Software Foundation; *
--* either version 2.1 of the License, or (at your option) any *
18 --* later version. *
--* *
20 --* This source is distributed in the hope that it will be *
--* useful, but WITHOUT ANY WARRANTY; without even the implied *
22 --* warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR *
--* PURPOSE. See the GNU Lesser General Public License for more *
24 --* details. *
--* *
26 --* You should have received a copy of the GNU Lesser General *
--* Public License along with this source; if not, download it *
28 --* from http://www.opencores.org/lgpl.shtml *
--* *
30 --*****

32 library ieee;
use ieee.std_logic_1164.all;
34 use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned."+";
36
entity br_generator is
38   generic (divider_width: integer := 16);
   port (
40     clock      : in  std_logic;
      rx_enable  : in  std_logic;
42     clk_txd    : out std_logic;
      tx_enable  : in  std_logic;
44     clk_serial : out std_logic
   );
46 end br_generator;

48 architecture principal of br_generator is
   -- change the following constant to your desired baud rate
50   -- one hz equal to one bit per second
   signal count_brg      : std_logic_vector(divider_width - 1 downto 0) := (others =>
      '0');
52   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

54   begin
56     txd : process (clock)
       begin
58       if (rising_edge(clock)) then
         if (count_brg_txd = brdvd) then

```

```

60         clk_txd         <= '1';
        count_brg_txd <= (others => '0');
62     elsif (tx_enable = '1') then
        clk_txd         <= '0';
        count_brg_txd <= count_brg_txd + 1;
        else
66         clk_txd         <= '0';
        count_brg_txd <= (others => '0');
68     end if;
    end if;
70 end process txd;

72 rxd : process (clock)
begin
74     if (rising_edge(clock)) then
        if (count_brg=brdvd) then
76         count_brg <= (others => '0');
        clk_serial <= '1';
78     elsif (rx_enable = '1') then
        count_brg <= count_brg+1;
80         clk_serial <= '0';
        else
82         count_brg <= '0' & brdvd(divider_width - 1 downto 1);
        clk_serial <= '0';
84     end if;
    end if;
86 end process rxd;
end principal;

```

Listing 23: mmu/muart/serial.vhd

```

1  --*****
2  --* Minimal UART ip core *
3  --* Author: Arao Hayashida Filho          arao@medinovacao.com.br *
4  --* * *
5  --*****
6  --* *
7  --* Copyright (C) 2009 Arao Hayashida Filho *
8  --* *
9  --* This source file may be used and distributed without *
10 --* restriction provided that this copyright statement is not *
11 --* removed from the file and that any derivative work contains *
12 --* the original copyright notice and the associated disclaimer. *
13 --* *
14 --* This source file is free software; you can redistribute it *
15 --* and/or modify it under the terms of the GNU Lesser General *
16 --* Public License as published by the Free Software Foundation; *
17 --* either version 2.1 of the License, or (at your option) any *
18 --* later version. *
19 --* *
20 --* This source is distributed in the hope that it will be *
21 --* useful, but WITHOUT ANY WARRANTY; without even the implied *
22 --* warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR *
23 --* PURPOSE. See the GNU Lesser General Public License for more *
24 --* details. *
25 --* *
26 --* You should have received a copy of the GNU Lesser General *
27 --* Public License along with this source; if not, download it *
28 --* from http://www.opencores.org/lgpl.shtml *
29 --* *
30 --*****
31
32 library IEEE;
33 use IEEE.STD_LOGIC_1164.ALL;

34
35 entity minimal_uart_core is
    port(
36     clock : in    std_logic;
37     eoc   : out   std_logic;
38     outp  : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
39     rxd   : in    std_logic;
40     txd   : out   std_logic;

```

```

    eot      : out      std_logic;
43    inp      : in       std_logic_vector(7 downto 0);
    ready    : out      std_logic;
45    wr       : in       std_logic
);
47 end minimal_uart_core;

49 architecture principal of minimal_uart_core is
    type state is (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9);
51    signal clk_serial      : std_logic := '0';
    signal start            : std_logic := '0';
53    signal eocs, eoc1, eoc2 : std_logic := '0';
    signal rx_ck_enable     : std_logic := '0';
55    signal receiving       : std_logic := '0';
    signal transmitting     : std_logic := '0';
57    signal clk_txd         : std_logic := '0';
    signal txds             : std_logic := '1';
59    signal eots            : std_logic := '0';
    signal inpl             : std_logic_vector(7 downto 0) := x"00";
61    signal data            : std_logic_vector(7 downto 0) := x"00";
    signal atual_state, next_state, atual_state_txd, next_state_txd : state := s0;
63    signal tx_enable       : std_logic := '0';
    signal tx_ck_enable     : std_logic := '0';
65
    component br_generator
67    port (
        clock      : in  std_logic;
69        rx_enable  : in  std_logic;
        clk_txd    : out std_logic;
71        tx_enable  : in  std_logic;
        clk_serial : out std_logic
    );
73 end component;
75
begin
77    ready <= not(tx_enable);
    brg : br_generator port map (clock, rx_ck_enable, clk_txd, tx_ck_enable,
        clk_serial);
79    rx_ck_enable <= start or receiving;
    tx_ck_enable <= tx_enable or transmitting;
81
    start_detect : process(rxd, eocs)
83    begin
        if (eocs = '1') then
85            start <= '0';
        elsif (falling_edge(rxd)) then
87            start <= '1';
        end if;
89    end process start_detect;

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

97    rxd_state_machine : process(start, atual_state)
99    begin
        if (start = '1' or receiving = '1') then
101            case atual_state is
                when s0 =>
103                eocs <= '0';
                if (start = '1') then
105                    next_state <= s1;
                    receiving <= '1';
107                else
                    next_state <= s0;
                    receiving <= '0';
109                end if;
            end if;
111
            when s1 =>
113                receiving <= '1';

```

```

115         eocs          <= '0';
        next_state <= s2;

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

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

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

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

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

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

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

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

149     when others =>
        null;

151     end case;

153     end if;
155     end process rxd_state_machine;

157     rxd_shift : process(clk_serial)
159     begin
        if (rising_edge(clk_serial)) then
161             if (eocs = '0') then
163                 data <= rxd & data(7 downto 1);
165             end if;
167         end if;
169     end process rxd_shift;

171     process (clock)
173     begin
        if (rising_edge(clock)) then
175             eoc <= eocs;
177         end if;
        end process;

179     process(atual_state)
181     begin
        if (atual_state=s9) then
183             outp <= data;
        end if;
185     end process;

```

```

187   txd_states : process(clk_txd)
188   begin
189       if (rising_edge(clk_txd)) then
190           atual_state_txd <= next_state_txd;
191       end if;
192   end process txd_states;
193
194   txd_state_machine : process(atual_state_txd, tx_enable)
195   begin
196       case atual_state_txd is
197           when s0 =>
198               inpl <= inp;
199               eots <= '0';
200               if (tx_enable = '1') then
201                   txds <= '0';
202                   transmitting <= '1';
203                   next_state_txd <= s1;
204               else
205                   txds <= '1';
206                   transmitting <= '0';
207                   next_state_txd <= s0;
208               end if;
209
210           when s1 =>
211               txds <= inpl(0);
212               eots <= '0';
213               transmitting <= '1';
214               next_state_txd <= s2;
215
216           when s2 =>
217               txds <= inpl(1);
218               eots <= '0';
219               transmitting <= '1';
220               next_state_txd <= s3;
221
222           when s3 =>
223               txds <= inpl(2);
224               eots <= '0';
225               transmitting <= '1';
226               next_state_txd <= s4;
227
228           when s4 =>
229               txds <= inpl(3);
230               eots <= '0';
231               transmitting <= '1';
232               next_state_txd <= s5;
233
234           when s5 =>
235               txds <= inpl(4);
236               eots <= '0';
237               transmitting <= '1';
238               next_state_txd <= s6;
239
240           when s6 =>
241               txds <= inpl(5);
242               eots <= '0';
243               transmitting <= '1';
244               next_state_txd <= s7;
245
246           when s7 =>
247               txds <= inpl(6);
248               eots <= '0';
249               transmitting <= '1';
250               next_state_txd <= s8;
251
252           when s8 =>
253               txds <= inpl(7);
254               eots <= '0';
255               transmitting <= '1';
256               next_state_txd <= s9;
257
258           when s9 =>
259               txds <= '1';

```

```

261         eots          <= '1';
           transmitting <= '1';
           next_state_txd <= s0;
263
           when others =>
265               null;
267
           end case;
       end process txd_state_machine;
269
       tx_start:process (clock, wr, eots)
271       begin
           if (eots = '1') then
273               tx_enable <= '0';
           elsif (falling_edge(clock)) then
275               if (wr = '1') then
                   tx_enable <= '1';
277               end if;
           end if;
279       end process tx_start;
281
       eot<=eots;
283
       process (clock)
       begin
285           if (rising_edge(clock)) then
                   txd <= txds;
287           end if;
       end process;
289
       end principal ;

```

Test Benchs

Listing 24: processor/alu_tb.vhd

```
-- Authors:
2 --      Henry Jenkins, Joel Koh

4 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

6
-- A testbench has no ports.
8 entity alu_tb is
    end alu_tb;

10
architecture behav of alu_tb is
12 -- Declaration of the component that will be instantiated.
    component alu
14     Port (f      : in      STD_LOGIC_VECTOR (3 downto 0); -- Function (opcode)
           rx      : in      STD_LOGIC_VECTOR (7 downto 0); -- Input x (Rx)
16     ry      : in      STD_LOGIC_VECTOR (7 downto 0); -- Input y (Ry)
           ro      : out     STD_LOGIC_VECTOR (7 downto 0); -- Output Normaly (Ry)
18     Cin      : in      STD_LOGIC; -- Carry in
           sr      : out     STD_LOGIC_VECTOR (15 downto 0)); -- Status register out Z(0),
                           C(1), N(2)
20 end component;
-- Specifies which entity is bound with the component.
22 for alu_0: alu use entity work.alu;
    signal f      : STD_LOGIC_VECTOR (3 downto 0);
24    signal rx, ry, ro : STD_LOGIC_VECTOR (7 downto 0);
    signal Cin      : STD_LOGIC;
26    signal sr      : STD_LOGIC_VECTOR (15 downto 0);
begin
28 -- Component instantiation.
    alu_0: alu port map (f => f, rx => rx, ry => ry, ro => ro, Cin => Cin, sr => sr);
30
-- This process does the real job.
32 process
    type pattern_type is record
34         f      : STD_LOGIC_VECTOR (3 downto 0);
         rx, ry      : STD_LOGIC_VECTOR (7 downto 0);
36         ro      : STD_LOGIC_VECTOR (7 downto 0);
         Cin      : STD_LOGIC;
38         sr      : STD_LOGIC_VECTOR (15 downto 0);
    end record;
40 -- The patterns to apply.
    type pattern_array is array (natural range <>) of pattern_type;
42 constant patterns : pattern_array :=
-- f      rx      ry      ro      Cin      sr
44 (( "0001", "00000000", "00000000", "00000000", '0', "0000000000000001"), --AND tests
    - 1ns
    ( "0001", "00000001", "00000001", "00000001", '0', "0000000000000000"), --AND tests
46 ( "0001", "00000000", "00000001", "00000000", '0', "0000000000000001"), --AND tests
    ( "0001", "10101010", "10101010", "10101010", '0', "0000000000000100"), --AND tests
48 ( "0001", "01010101", "01010101", "01010101", '0', "0000000000000000"), --AND tests
    - 5ns
    ( "0001", "11111111", "00000000", "00000000", '0', "0000000000000001"), --AND tests
50 ( "0001", "11111111", "11111111", "11111111", '0', "0000000000000100"), --AND tests
    ( "0001", "00000000", "01010101", "00000000", '0', "0000000000000001"), --AND tests
52 ( "0001", "00000000", "10101010", "00000000", '0', "0000000000000001"), --AND tests
    ( "0001", "11111111", "01010101", "01010101", '0', "0000000000000000"), --AND tests
    - 10 ns
54 ( "0001", "11111111", "10101010", "10101010", '0', "0000000000000100"), --AND tests
    ( "0001", "10000011", "10110010", "10000010", '0', "0000000000000100"), --AND tests
56 ( "0001", "00000011", "00110010", "00000010", '0', "0000000000000000"), --AND tests
    - 14 ns
58 ( "0011", "00000000", "00000000", "00000000", '0', "0000000000000001"), --OR tests
    ( "0011", "00000001", "00000001", "00000001", '0', "0000000000000000"), --OR tests
60 ( "0011", "00000000", "00000001", "00000001", '0', "0000000000000000"), --OR tests
    ( "0011", "10101010", "10101010", "10101010", '0', "0000000000000100"), --OR tests
62 ( "0011", "01010101", "01010101", "01010101", '0', "0000000000000000"), --OR tests
    ( "0011", "11111111", "00000000", "11111111", '0', "0000000000000100"), --OR tests
```

```

64  ("0011", "11111111", "11111111", "11111111", '0', "000000000000100"), --OR tests
    - 20 ns
    ("0011", "00000000", "01010101", "01010101", '0', "0000000000000000"), --OR tests
66  ("0011", "00000000", "10101010", "10101010", '0', "000000000000100"), --OR tests
    ("0011", "11111111", "01010101", "11111111", '0', "000000000000100"), --OR tests
68  ("0011", "11111111", "10101010", "11111111", '0', "000000000000100"), --OR tests
    ("0011", "10000011", "10110010", "10110011", '0', "000000000000100"), --OR tests
    - 25 ns
70  ("0011", "00000011", "00110010", "00110011", '0', "0000000000000000"), --OR tests

72  ("0101", "00000000", "00000000", "11111111", '0', "000000000000100"), --NOT tests
    - ry should not matter
    ("0101", "00000001", "00000001", "11111110", '0', "000000000000100"), --NOT tests
74  ("0101", "00000000", "00000001", "11111111", '0', "000000000000100"), --NOT tests
    ("0101", "10101010", "10101010", "01010101", '0', "0000000000000000"), --NOT tests
    - 30 ns
76  ("0101", "01010101", "01010101", "10101010", '0', "000000000000100"), --NOT tests
    ("0101", "11111111", "00000000", "00000000", '0', "0000000000000001"), --NOT tests
78  ("0101", "11111111", "11111111", "00000000", '0', "0000000000000001"), --NOT tests
    ("0101", "00000000", "01010101", "11111111", '0', "000000000000100"), --NOT tests
80  ("0101", "00000000", "10101010", "11111111", '0', "000000000000100"), --NOT tests
    - 35 ns
    ("0101", "11111111", "01010101", "00000000", '0', "0000000000000001"), --NOT tests
82  ("0101", "11111111", "10101010", "00000000", '0', "0000000000000001"), --NOT tests
    ("0101", "10000011", "10110010", "01111100", '0', "0000000000000000"), --NOT tests
84  ("0101", "00000011", "00110010", "11111100", '0', "000000000000100"), --NOT tests
    - 39 ns

86  ("0111", "00000000", "00000000", "00000000", '0', "0000000000000001"), --XOR tests
    - 40 ns
    ("0111", "00000001", "00000001", "00000000", '0', "0000000000000001"), --XOR tests
88  ("0111", "00000000", "00000001", "00000001", '0', "0000000000000000"), --XOR tests
    ("0111", "10101010", "10101010", "00000000", '0', "0000000000000001"), --XOR tests
90  ("0111", "01010101", "01010101", "00000000", '0', "0000000000000001"), --XOR tests
    ("0111", "11111111", "00000000", "11111111", '0', "000000000000100"), --XOR tests
    - 45 ns
92  ("0111", "11111111", "11111111", "00000000", '0', "0000000000000001"), --XOR tests
    ("0111", "00000000", "01010101", "01010101", '0', "0000000000000000"), --XOR tests
94  ("0111", "00000000", "10101010", "10101010", '0', "000000000000100"), --XOR tests
    ("0111", "11111111", "01010101", "10101010", '0', "000000000000100"), --XOR tests
96  ("0111", "11111111", "10101010", "01010101", '0', "0000000000000000"), --XOR tests
    - 50 ns
    ("0111", "10000011", "10110010", "00110001", '0', "0000000000000000"), --XOR tests
98  ("0111", "00000011", "00110010", "00110001", '0', "0000000000000000"), --XOR tests
);
100 begin
    -- Check each pattern.
102 for i in patterns'range loop
    -- Set the inputs.
104  Cin <= patterns(i).Cin;
    f <= patterns(i).f;
106  rx <= patterns(i).rx;
    ry <= patterns(i).ry;
108  -- Wait for the results.
    wait for 1 ns;
110  -- Check the outputs.
    assert ro = patterns(i).ro
112  report "bad output register value" severity error;
    assert sr = patterns(i).sr
114  report "bad status register value" severity error;
    assert sr(0) = patterns(i).sr(0)
116  report " *Zero is incorrect" severity error;
    assert sr(1) = patterns(i).sr(1)
118  report " *Carry is incorrect" severity error;
    assert sr(2) = patterns(i).sr(2)
120  report " *Negative is incorrect" severity error;
end loop;
122 assert false report "end of test" severity note;
    -- Wait forever; this will finish the simulation.
124 wait;
end process;
126 end behav;

```


Listing 25: processor/fulladder_tb.vhd

```

1  -- Authors:
   --       Henry Jenkins, Joel Koh
3
   library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;

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

11 architecture behav of fulladder8_tb is
   -- Declaration of the component that will be instantiated.
13     component fulladder8
       Port (A      : in    STD_LOGIC_VECTOR( 7 downto 0);
15           B      : in    STD_LOGIC_VECTOR( 7 downto 0);
           Cin     : in    STD_LOGIC;
17           Sum     : out   STD_LOGIC_VECTOR( 7 downto 0);
           Cout    : out   STD_LOGIC
19       );
       end component;
   -- Specifies which entity is bound with the component.
21   for fulladder8_0: fulladder8 use entity work.fulladder8;
23       signal A,B,Sum      : STD_LOGIC_VECTOR (7 downto 0);
       signal Cin,Cout     : STD_LOGIC;
25   begin
       -- Component instantiation.
27       fulladder8_0: fulladder8 port map (A => A, B => B, Cin => Cin, Sum => Sum, Cout
           => Cout);

29       -- This process does the real job.
       process
31         type pattern_type is record
           A      : STD_LOGIC_VECTOR( 7 downto 0);
33           B      : STD_LOGIC_VECTOR( 7 downto 0);
           Cin     : STD_LOGIC;
35           Sum     : STD_LOGIC_VECTOR( 7 downto 0);
           Cout    : STD_LOGIC;
37         end record;
       -- The patterns to apply.
39       type pattern_array is array (natural range <>) of pattern_type;
       constant patterns : pattern_array :=
41         -- A      B      Cin    Sum    Cout
         (( "00000000", "00000000", '0', "00000000", '0'), --AND tests - 1ns
43          ("11111111", "11111111", '1', "11111111", '1'), --AND tests
           ("00000000", "00000000", '1', "00000001", '0'), --AND tests
45          ("00000000", "11111111", '0', "11111111", '0'), --AND tests
           ("11111111", "00000000", '0', "11111111", '0'), --AND tests
47          ("11111111", "00000000", '1', "00000000", '1'), --AND tests
           ("10101010", "01010101", '0', "11111111", '0'), --AND tests
49          ("10101010", "01010101", '1', "00000000", '1'), --AND tests
           ("11111111", "11111111", '0', "11111110", '1') --XOR tests
51       );
       begin
53         -- Check each pattern.
         for i in patterns'range loop
55             -- Set the inputs.
             A <= patterns(i).A;
57             B <= patterns(i).B;
             Cin <= patterns(i).Cin;
59             -- Wait for the results.
             wait for 1 ns;
61             -- Check the outputs.
             assert Sum = patterns(i).Sum
63             report "The sum check failed" severity error;
             assert Cout = patterns(i).Cout
65             report "The carry out is wrong" severity error;
         end loop;
67         assert false report "end of test" severity note;
       -- Wait forever; this will finish the simulation.
69         wait;
       end process;

```

71 end behav;

Listing 26: processor/spr_tb.vhd

```
-- Authors:
2 --      Henry Jenkins, Joel Koh

4 library ieee;
use ieee.std_logic_1164.all;
6 --use ieee.std_logic_unsigned.all;
--use ieee.std_logic_arith.all;
8
entity spr_TB is          -- entity declaration
10 end spr_TB;

12 architecture TB of spr_TB is

14     component sr
Port (clk      : in    STD_LOGIC;
16         enable : in    STD_LOGIC;          -- Enable write
        reset   : in    STD_LOGIC;          -- Reset the register
18         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
20 end component;

22     signal sr_enable : std_logic;
signal sr_reset  : std_logic;
24     signal sr_Ri     : std_logic_vector(15 downto 0);
signal sr_Ro     : std_logic_vector(15 downto 0);
26
    component pc
28     Port (clk      : in    STD_LOGIC;
        enable : in    STD_LOGIC;          -- Enable write
30         reset   : in    STD_LOGIC;          -- Reset the register
        Ri     : in    STD_LOGIC_VECTOR (15 downto 0); -- The input to the SPR
32         Ro      : out   STD_LOGIC_VECTOR (15 downto 0)); -- The output from SPR
    end component;
34
    signal pc_enable : std_logic;
signal pc_reset  : std_logic;
36     signal pc_Ri     : std_logic_vector(15 downto 0);
signal pc_Ro     : std_logic_vector(15 downto 0);
38
40     signal T_clk : std_logic;

42 begin

44     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);
46
    -- concurrent process to offer the clk signal
48 process
begin
50     T_clk <= '0';
    wait for 5 ns;
52     T_clk <= '1';
    wait for 5 ns;
54 end process;

56 process

58     variable err_cnt: integer :=0;

60 begin

62     -- Write
    sr_enable <= '1';
64     sr_reset <= '0';
    sr_Ri     <= "0100011001011001";
66     pc_enable <= '1';
```

```

68     pc_reset    <= '0';
    pc_Ri        <= "0101011010110100";
    wait for 20 ns;

70
    -- Read
72     assert (sr_Ro="0100011001011001") report "Read sr #1 failed" severity error;
    assert (pc_Ro="0101011010110100") report "Read pc #1 failed" severity error;
74
    -- Change Ri
76     sr_Ri <= "1001100101110100";
    pc_Ri <= "0001010001110000";
78     wait for 20 ns;
    assert (sr_Ro = "1001100101110100") report "Read sr #2 failed" severity error;
80     assert (pc_Ro = "0001010001110000") report "Read pc #2 failed" severity error;

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

90     -- Enable sr
    sr_enable <= '1';
92     wait for 20 ns;
    assert (sr_Ro = "0101010101010101") report "Read sr #3 failed" severity error;
94
    -- Disable pc, sr still enabled
96     pc_enable <= '0';
    sr_Ri <= "0000000011111111";
98     pc_Ri <= "1111111100000000";
    wait for 20 ns;
100    assert (sr_Ro = "0000000011111111") report "Read sr #4 failed" severity error;
    assert (pc_Ro = "1010101010101010") report "Wrote to pc while disabled" severity
        error;
102
    -- Enable pc
104    pc_enable <= '1';
    wait for 20 ns;
106    assert (pc_Ro = "1111111100000000") report "Read pc #4 failed" severity error;

108
    assert false report "End of test" severity note;
110    wait; -- wait forever to end the test

112 end process;

114 end TB;

116 -----
117 configuration CFG_TB of spr_TB is
118     for TB
119         end for;
120 end CFG_TB;

```

Listing 27: processor/gpr_tb.vhd

```

-- Authors:
2 --     Henry Jenkins, Joel Koh

4 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

6
-- A testbench has no ports.
8 entity gpr_tb is
    end gpr_tb;

10
architecture behav of gpr_tb is
12     -- Declaration of the component that will be instantiated.
    component gpr

```

```

14  Port(clk                : IN std_logic;                -- Clock
        enable             : IN std_logic;                -- Enable input
        (output is always enabled)
16      SelRx, SelRy, SelRi : IN std_logic_vector(2 DOWNT0 0); -- Selecti which
        registers to use
        Ri                 : IN std_logic_vector(7 DOWNT0 0); -- Input
18      Rx, Ry              : OUT std_logic_vector(7 DOWNT0 0)); -- Outputs
    end component;
20  -- Specifies which entity is bound with the component.
    for gpr_0: gpr use entity work.gpr;
22  signal clk, enable      : std_logic;
    signal SelRx, SelRy, SelRi : std_logic_vector(2 DOWNT0 0);
24  signal Ri, Rx, Ry       : std_logic_vector(7 DOWNT0 0);
begin
26  -- Component instantiation.
    gpr_0: gpr port map (clk => clk, enable => enable, SelRx => SelRx, SelRy => SelRy,
        SelRi => SelRi, Ri => Ri, Rx => Rx, Ry => Ry);
28
    -- Does the clock signal
30  process
    begin
32      clk <= '0';
        wait for 5 ns;
34      clk <= '1';
        wait for 5 ns;
36  end process;

38  -- This process does the real job.
    process
    begin
42      -- Write to R0
        SelRi <= "000";
44      Ri <= "00010100";
        enable <= '1';
46      wait for 20 ns;

48      -- Read R0 from Rx
        SelRx <= "000";
50      wait for 20 ns;
        assert (Rx = "00010100") report "Read from Rx failed #1" severity error;
52
        -- Read R0 from Ry
54      SelRy <= "000";
        wait for 20 ns;
56      assert (Ry = "00010100") report "Read from Ry failed #1" severity error;

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

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

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

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

78      -- Read R2 from Rx
80      SelRx <= "010";
        wait for 20 ns;

```

```

82     assert (Rx = "01010001") report "Read from Rx failed #3" severity error;

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

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

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

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

104    -- Read R2 after a long time
    SelRx <= "010";
106    SelRy <= "010";
    wait for 1 ms;
108    assert (Rx = "01010001") report "Read from Rx failed #6" severity error;
    assert (Ry = "01010001") report "Read from Ry failed #6" severity error;
110

112    assert false report "End of test" severity note;
    wait; -- wait forever to end the test
114
    end process;
116 end behav;

```

Listing 28: mmu/mmu_tb.vhd

```

1  -- Author:
   --      Wim Looman
3
   library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;

7  library work;
   use work.mmu_main;
9  use work.minimal_uart_core;
   use work.txt_util.all;
11
   entity mmu_tb is
13 end mmu_tb;

15 architecture tb of mmu_tb is
   component mmu_main is
17     port (
       -- instruction bus
19     inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
       inst_data : out std_logic_vector(15 downto 0); -- Data lines.
21     inst_req  : in  std_logic;                      -- Pulled low to request bus
       usage;
       inst_ack  : out std_logic;                      -- Pulled high to inform of
       request completion.
23     -- data bus
       data_add  : in  std_logic_vector(15 downto 0); -- Address lines.
25     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.
27     data_req  : in  std_logic;                      -- Pulled low to request bus
       usage;
       data_ack  : inout std_logic;                    -- Pulled high to inform of
       request completion.
29     -- extras

```

```

31         clk          : in  std_logic;
        receive_pin    : in  std_logic;
        transfer_pin   : out std_logic
33     );
    end component;

35     component minimal_uart_core is
37     port(
        clock : in    std_logic;
39         eoc      : out std_logic;
        outp      : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
41         rxd      : in    std_logic;
        txd      : out    std_logic;
43         eot      : out    std_logic;
        inp      : in    std_logic_vector(7 downto 0);
45         ready    : out    std_logic;
        wr      : in    std_logic
47     );
    end component;

49     signal inst_add      : std_logic_vector(11 downto 0);
51     signal inst_data     : std_logic_vector(15 downto 0);
    signal inst_req       : std_logic := '1';
53     signal inst_ack      : std_logic;
    signal data_add       : std_logic_vector(15 downto 0);
55     signal data_line     : std_logic_vector(7 downto 0);
    signal data_read      : std_logic;
57     signal data_req      : std_logic;
    signal data_ack       : std_logic;
59     signal clk           : std_logic;
    signal receive_pin    : std_logic;
61     signal transfer_pin  : std_logic;

63     signal eoc, rxd, txd, eot, ready, wr: std_logic;
    signal outp, inp : std_logic_vector(7 downto 0);
65
    signal current_rcv : std_logic_vector(7 downto 0);
67     signal current_send : std_logic_vector(7 downto 0);
begin
69     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);
71
    rxd <= transfer_pin;
73     receive_pin <= txd;

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

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

```

```

103     inst_req <= '0';

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

112     assert false report "passed header" severity note;

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

121     assert false report "passed address low" severity note;

123     wait until eoc'event;
124     assert outp = "0000" & patterns(i).inst_add(11 downto 8)
125         report "Bad address high expected '" & str(patterns(i).inst_add(11 downto
126             8)) & "' recieved '" & str(outp) & "'"
127         severity error;
128     wait until eoc'event;

130     assert false report "passed address high" severity note;

132     wait for 100 ns;
133     inp <= patterns(i).send_head;
134     wait for 20 ns;
135     wr <= '1';
136     wait for 20 ns;
137     wr <= '0';
138     wait until eot'event;
139     wait until eot'event;

141     wait for 100 ns;
142     inp <= "0000" & patterns(i).inst_add(11 downto 8);
143     wait for 20 ns;
144     wr <= '1';
145     wait for 20 ns;
146     wr <= '0';
147     wait until eot'event;
148     wait until eot'event;

150     wait for 100 ns;
151     inp <= patterns(i).inst_add(7 downto 0);
152     wait for 20 ns;
153     wr <= '1';
154     wait for 20 ns;
155     wr <= '0';
156     wait until eot'event;
157     wait until eot'event;

159     wait for 100 ns;
160     inp <= patterns(i).inst_data(7 downto 0);
161     wait for 20 ns;
162     wr <= '1';
163     wait for 20 ns;
164     wr <= '0';
165     wait until eot'event;
166     wait until eot'event;

168     wait for 100 ns;
169     inp <= patterns(i).inst_data(15 downto 8);
170     wait for 20 ns;
171     wr <= '1';

```

```

    wait for 20 ns;
173    wr <= '0';
    wait until eot'event;
175    wait until eot'event;

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

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

185    assert false report "finished transmission" severity note;

187    wait for 20 ns;

189    inst_req <= '1';
    end loop;
191    wait;
    end process;
193 end tb;

```

Listing 29: data_tb.vhd

```

1  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;

3
   library work;
5  use work.mmu_main;
   use work.minimal_uart_core;
7  use work.txt_util.all;

9  entity data_tb is
   end data_tb;

11
   architecture tb of data_tb is
13     component mmu_main is
        port (
15         -- instruction bus
            inst_add  : in  std_logic_vector(11 downto 0); -- Address lines.
17         inst_data  : out std_logic_vector(15 downto 0); -- Data lines.
            inst_req   : in  std_logic;                    -- Pulled low to request bus
                usage.
19         inst_ack   : out std_logic;                    -- Pulled high to inform of
                request completion.
            -- data bus
21         data_add   : in  std_logic_vector(15 downto 0); -- Address lines.
            data_line  : inout std_logic_vector(7 downto 0); -- Data lines.
23         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.
25         data_ack   : inout std_logic;                    -- Pulled high to inform of
                request completion.
            -- extras
27         clk        : in  std_logic;
            receive_pin : in  std_logic;
29         transfer_pin : out std_logic
        );
31     end component;

33     component IO is
        PORT(
35         -- data bus --
            data_add   : IN      std_logic_vector(15 DOWNT0 0);    -- address lines --
37         data_data   : INOUT    std_logic_vector(7 DOWNT0 0);    -- data lines --
            data_read  : INOUT    std_logic;                        -- pulled high for read, low
                for write --
39         data_req    : INOUT    std_logic;                        -- pulled low to request bus
                usage --

```



```

        data_ack      : INOUT    std_logic;                                -- pulled high to inform
        request completion --
41    -- io --
        clk           : IN       std_logic;
43    sw1             : IN       std_logic;
        sw2           : IN       std_logic
45    );
end component;

47    component minimal_uart_core is
49    port(
        clock : in      std_logic;
51    eoc      : out     std_logic;
        outp   : inout   std_logic_vector(7 downto 0) := "ZZZZZZZZ";
53    rxd      : in      std_logic;
        txd    : out     std_logic;
55    eot      : out     std_logic;
        inp    : in      std_logic_vector(7 downto 0);
57    ready    : out     std_logic;
        wr     : in      std_logic
59    );
end component;

61
63    signal inst_add      : std_logic_vector(11 downto 0);
    signal inst_data      : std_logic_vector(15 downto 0);
    signal inst_req       : std_logic := '1';
65    signal inst_ack      : std_logic;
    signal data_add       : std_logic_vector(15 downto 0);
67    signal data_line     : std_logic_vector(7 downto 0);
    signal data_read      : std_logic;
69    signal data_req      : std_logic;
    signal data_ack       : std_logic;
71    signal clk           : std_logic;
    signal receive_pin    : std_logic;
73    signal transfer_pin  : std_logic;
    signal sw1, sw2 : std_logic;
75
    signal eoc, rxd, txd, eot, ready, wr : std_logic;
77    signal outp, inp : std_logic_vector(7 downto 0);
begin
79    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);
81    muart : minimal_uart_core port map (clk, eoc, outp, rxd, txd, eot, inp, ready, wr);

83    rxd <= transfer_pin;
    receive_pin <= txd;
85
    clk_gen : process begin
87        clk <= '0';
        wait for 10 ns;
89        clk <= '1';
        wait for 10 ns;
91    end process;

93    data_test : process
        type pattern_type is record
95        data_add      : std_logic_vector(15 downto 0);
        recv_head      : std_logic_vector( 7 downto 0);
97        send_head     : std_logic_vector( 7 downto 0);
        data_data      : std_logic_vector( 7 downto 0);
99        switch_data   : std_logic_vector( 1 downto 0);
        rw             : std_logic;
101    end record;
        type pattern_array is array (natural range <>) of pattern_type;
103    constant patterns : pattern_array :=
-- data_add, recv_head, send_head, data_data, switch_data, rw
105    ((x"0581", x"80", x"00", x"A7", "00", '1' ),
        (x"0273", x"00", x"00", x"5E", "00", '0' ));
107    begin
        wr <= '0';
109    data_req <= '1';

```

```

111     for i in patterns'range loop
112         wait for 10000 ns;
113         data_add <= patterns(i).data_add;
114         data_read <= patterns(i).rw;
115         wait for 20 ns;
116         if patterns(i).rw = '0' then
117             data_line <= patterns(1).data_data;
118         else
119             data_line <= (others => 'Z');
120         end if;
121         wait for 20 ns;
122         data_req <= '0';
123
124     if patterns(i).data_add(0) = '1' then
125         wait until eoc'event;
126         assert outp = patterns(i).recv_head
127             report "Bad header expected '" & str(patterns(i).recv_head) & "' recieved
128                 '" & str(outp) & '"
129             severity error;
130         wait until eoc'event;
131
132         assert false report "passed header" severity note;
133
134         wait until eoc'event;
135         assert outp = patterns(i).data_add(8 downto 1)
136             report "Bad address low expected '" & str(patterns(i).data_add(7 downto
137                 0)) & "' recieved '" & str(outp) & '"
138             severity error;
139         wait until eoc'event;
140
141         assert false report "passed address low" severity note;
142
143         wait until eoc'event;
144         assert outp = "0" & patterns(i).data_add(15 downto 9)
145             report "Bad address high expected '" & str(patterns(i).data_add(11 downto
146                 8)) & "' recieved '" & str(outp) & '"
147             severity error;
148         wait until eoc'event;
149
150         assert false report "passed address high" severity note;
151         if patterns(i).rw = '0' then
152             wait until eoc'event;
153             assert outp = patterns(i).data_data
154                 report "Bad data expected '" & str(patterns(i).data_data) & "' recieved
155                     '" & str(outp) & '"
156             severity error;
157             wait until eoc'event;
158
159             assert false report "passed data" severity note;
160         else
161             wait for 100 ns;
162             inp <= patterns(i).send_head;
163             wait for 20 ns;
164             wr <= '1';
165             wait for 20 ns;
166             wr <= '0';
167             wait until eot'event;
168             wait until eot'event;
169
170             wait for 100 ns;
171             inp <= patterns(i).data_add(8 downto 1);
172             wait for 20 ns;
173             wr <= '1';
174             wait for 20 ns;
175             wr <= '0';
176             wait until eot'event;
177             wait until eot'event;
178
179             wait for 100 ns;
180             inp <= "0" & patterns(i).data_add(15 downto 9);
181             wait for 20 ns;

```

```

179         wr <= '1';
180         wait for 20 ns;
181         wr <= '0';
182         wait until eot'event;
183         wait until eot'event;

185
186         wait for 100 ns;
187         inp <= patterns(i).data_data;
188         wait for 20 ns;
189         wr <= '1';
190         wait for 20 ns;
191         wr <= '0';
192         wait until eot'event;
193         wait until eot'event;
194     end if;
195 else
196
197 end if;

199 assert data_ack = '1'
200     report "receipt not acknowledged"
201     severity error;

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

209 assert false report "finished transmission" severity note;

211 wait for 20 ns;

213 data_req <= '1';
214 end loop;
215 wait;
216 end process;
217 end tb;

```

Tools

Listing 30: assembler.rb

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

3 # Author:
4 #     Wim Looman
5 # Copyright:
6 #     Copyright (c) 2010 Wim Looman
7 # License:
8 #     GNU General Public License (see http://www.gnu.org/licenses/gpl-3.0.txt)
9
10 def assert(error=nil)
11   raise (error || "Assertion Failed!") unless yield
12 end
13
14 # For 8-bit twos complement
15 def twos_complement(num)
16   return 256 + num
17 end
18
19
20 def logical_operands(chunks)
21   y = chunks[1][1..1].to_i
22   assert(chunks[1] + " is not a valid register") {y >= 0 && y < 8}
23   x = chunks[2][1..1].to_i
24   assert(chunks[2] + " is not a valid register") {x >= 0 && x < 8}
25   return (y << 5) + x
26 end
27
28
29 def immediate(chunk, symbols=nil, move_from=nil)
30   v = chunk.to_i
31   if v == 0 && chunk != "0" && symbols != nil
32     assert(chunk + " is not a valid symbol") {symbols.include?(chunk)}
33     move_to = symbols[chunk]
34     diff = move_to - move_from
35     if diff < 1
36       diff = twos_complement(diff)
37     end
38     return diff
39   else
40     assert(chunk + " is not a valid immediate") {v >= -127 && v < 128}
41     if v < 0
42       v = twos_complement(v)
43     end
44     return v
45   end
46 end
47
48
49 def register(chunk, num_registers)
50   x = chunk[1..1].to_i
51   assert(chunk + " is not a valid register") {x >= 0 && x < num_registers}
52   return x
53 end
54
55
56 def auto(chunk)
57   if chunk[-1..-1] == "+"
58     return 0x08
59   elsif chunk[-1..-1] == "-"
60     return 0x10
61   else
62     return 0x00
63   end
64 end
65
66
67 def convert(lines)
68   table = first_pass(lines)
```

```

69  return second_pass(lines, table)
end
71

73  def first_pass(lines)
    instruction = 0
75    symbols = {}
    lines.each do |line|
77      chunks = line.sub(" ", " ").split
      case chunks[0]
79        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

81          when "label:"
83            symbols[chunks[1]] = instruction
            end
85      end
      return symbols
87  end

89  def second_pass(lines, symbols)
91    line_no = 0
    output = []
93    lines.each do |line|
      label = line.sub(" ", " ").split[0]
95      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"
97          line_no += 1
            output.push(convert_line(line, symbols, line_no))
99        end
      end
101    return output.flatten
    end
103

105  def convert_line(line, symbols, line_no)
    chunks = line.sub(" ", " ").split#.partition(";")[0].split
107
    case chunks[0]
109      when "LDI"
        instruction = 0x21
111        x = register(chunks[1], 4)
        v = immediate(chunks[2])
113        operands = (v << 2) + x

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

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

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

133      when "MV"
        instruction = 0x04
135        if chunks[1][0] == 'r'[0] && chunks[2][0] == 'r'[0]
            y = register(chunks[1], 8)
137            x = register(chunks[2], 8)

```

```

139         operands = (y << 5) + x
140     elif chunks[1][0] == 'a'[0]
141         y = register(chunks[1], 3)
142         x = register(chunks[2], 8)
143         n = chunks[1][-1] == 'H' ? 1 : 0
144         operands = (1 << 9) + (n << 8) + (y << 5) + x
145     elif chunks[2][0] == 'a'[0]
146         y = register(chunks[1], 8)
147         x = register(chunks[2], 3)
148         n = chunks[2][-1] == 'H' ? 1 : 0
149         operands = (1 << 4) + (n << 3) + (y << 5) + x
150     else
151         # explode
152     end
153
154 when "AND"
155     instruction = 0x02
156     operands = logical_operands(chunks)
157
158 when "OR"
159     instruction = 0x06
160     operands = logical_operands(chunks)
161
162 when "NOT"
163     instruction = 0x0A
164     operands = logical_operands(chunks)
165
166 when "XOR"
167     instruction = 0x0E
168     operands = logical_operands(chunks)
169
170 when "ADD"
171     instruction = 0x12
172     operands = logical_operands(chunks)
173
174 when "ADC"
175     instruction = 0x16
176     operands = logical_operands(chunks)
177
178 when "SUB"
179     instruction = 0x1A
180     operands = logical_operands(chunks)
181
182 when "SBB"
183     instruction = 0x1E
184     operands = logical_operands(chunks)
185
186 when "NEG"
187     instruction = 0x08
188     operands = logical_operands(chunks)
189
190 when "CMP"
191     instruction = 0x0C
192     operands = logical_operands(chunks)
193
194 when "BEQ"
195     instruction = 0x23
196     v = immediate(chunks[1], symbols, line_no)
197     operands = (v << 2)
198
199 when "BNE"
200     instruction = 0x27
201     v = immediate(chunks[1], symbols, line_no)
202     operands = (v << 2)
203
204 when "BLT"
205     instruction = 0x2B
206     v = immediate(chunks[1], symbols, line_no)
207     operands = (v << 2)
208
209 when "BGT"
210     instruction = 0x2F
211     v = immediate(chunks[1], symbols, line_no)

```

```

211     operands = (v << 2)

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

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

221     when "RJMPP"
222         instruction = 0x3B
223         v = immediate(chunks[1], symbols, line_no)
224         operands = (v << 2)

225     when "JMP"
226         instruction = 0x2F
227         y = register(chunks[1], 3)
228         operands = (y << 5)
229     end
230     opcode = (instruction << 10) + operands
231     return [(opcode >> 8), (opcode & 0xFF)]
232 end

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

238     input = IO.readlines(ARGV[0])

239     output = convert(input)
240     if ARGV.length == 2:
241         File.open(ARGV[1], "wb") do |file|
242             output.each do |char|
243                 file.putc(char)
244             end
245         end
246     else
247         output.each do |char|
248             $stdout.putc(char)
249         end
250     end
251 end

```

Listing 31: memory.rb

```

#!/usr/bin/env ruby
2
# Author:
4 #     Wim Looman
# Copyright:
6 #     Copyright (c) 2010 Wim Looman
# License:
8 #     GNU General Public License (see http://www.gnu.org/licenses/gpl-3.0.txt)

10 require 'rubygems'
11 require 'serialport'
12
13 def serve(program, data_file, sp)
14     while 1
15         header = sp.getc
16         diagnostic_mode = (header >> 2) & 0x03
17         instruction = header & 0x01 == 0x01
18         address = sp.getc + (sp.getc << 8)

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

```

```

22     header = 0x82
23     header += 0x01 if instruction
24     sp.putc(header)
25     instruction ? sp.write(program[address]) :
26         sp.write(data_file[address])
27     p "Sending data for #{instruction ? "Instruction" : "Data"} bus, address:" +
28         "#{address}, data: #{instruction ? program[address] : data_file[address]}"
29     when 0x00 # write, doesn't support writing to instruction memory
30         data = sp.getc
31         data_file[address] = data
32     p "Writing data, address: " + address + ", data: " + data_file[address]
33     end
34 end
35 end
36
37 if __FILE__ == $0
38
39     if ARGV.size < 3
40         STDERR.print "Usage: ruby #{$0} <device> <baud_rate> <program_file>
41             [<data_file>]\n"
42         exit
43     end
44
45     device = ARGV[0]
46     baud_rate = ARGV[1].to_i
47
48     program = Array.new(2**12, 0x00)
49     i = 0
50     File.open(ARGV[2], 'rb') do |input|
51         input.each_byte do |byte|
52             program[i] = byte
53             i += 1
54         end
55     end
56
57     data = Array.new(2**15, 0x00)
58     File.open(ARGV[3], 'rb') do |input|
59         input.each_byte do |byte|
60             data += byte
61         end
62     end if ARGV.size > 3
63
64     sp = SerialPort.new(device, baud_rate, 8, 1, SerialPort::NONE)
65     serve(program, data, sp)
66 end

```
