SPECIFICATION AND IMPLEMENTATION OF A MICROCOMPUTER

- BASIC COMPONENTS OF A COMPUTER SYSTEM
- INFORMAL AND $\mu$VHDL-BASED DESCRIPTION
  - ARCHITECTURE
  - IMPLEMENTATION
- OPERATION OF SIMPLE MICROCOMPUTER SYSTEM:
  XMC: eXample MicroComputer
- ITS CYCLE TIME
BASIC COMPONENTS OF A COMPUTER

- PROCESSOR;
- MEMORY SUBSYSTEM;
- INPUT/OUTPUT (I/O) SUBSYSTEM

Figure 15.1: COMPUTER SYSTEM.
Virtual memory
(Disk) 2 Gbyte
Main memory
(Dynamic devices) 16 Mbyte
Cache memory
(Static devices) 64 Kbyte
Processor

Figure 15.2: MEMORY HIERARCHY.
Figure 15.3: STRUCTURE OF XMC.
LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE comp_pkg IS
  SUBTYPE WordT IS STD_LOGIC_VECTOR(31 DOWNTO 0);
  SUBTYPE MAddrT IS STD_LOGIC_VECTOR(23 DOWNTO 0);
  SUBTYPE IOAddrT IS STD_LOGIC_VECTOR(10 DOWNTO 0);
  SUBTYPE ByteT IS STD_LOGIC_VECTOR( 7 DOWNTO 0);
  TYPE StatusT IS (undef, p_reset, fetch, execute, memop, ioop);

  FUNCTION get_carry(RA_Data,RB_Data,Imm,Opcode:STD_LOGIC_VECTOR)
  RETURN STD_LOGIC;
  FUNCTION get_ovf(RA_Data,RB_Data,Imm,Opcode:STD_LOGIC_VECTOR)
  RETURN STD_LOGIC;
  FUNCTION get_cc(RA_Data,RB_Data,Opcode:STD_LOGIC_VECTOR)
  RETURN STD_LOGIC_VECTOR;
END comp_pkg;

PACKAGE BODY comp_pkg IS
  FUNCTION get_carry(RA_Data,RB_Data,Imm,Opcode:STD_LOGIC_VECTOR)
  RETURN STD_LOGIC
  END BODY;
IS VARIABLE cy: STD_LOGIC:= '0';

BEGIN
   -- description of carry generation included here
   RETURN(cy);
END get_carry;

FUNCTION get_ovf (RA_Data,RB_Data,Imm,Opcode: STD_LOGIC_VECTOR)
   RETURN STD_LOGIC
IS VARIABLE ovf: STD_LOGIC:= '0';
BEGIN
   -- description of overflow generation included here
   RETURN(ovf);
END get_ovf;

FUNCTION get_cc (RA_Data,RB_Data,Opcode: STD_LOGIC_VECTOR)
   RETURN STD_LOGIC_VECTOR
IS VARIABLE cc: STD_LOGIC_VECTOR(3 DOWNTO 0):= "0000";
BEGIN
   -- description of cc generation included here
   RETURN(cc);
END get_cc;
END comp_pkg;
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.ALL, WORK.comp_pkg.ALL;

ENTITY Computer IS
  PORT (Reset, Clk : IN STD_LOGIC);
END Computer;

ARCHITECTURE structural OF Computer IS
  SIGNAL MemAddr : MAddrT;  -- memory address bus
  SIGNAL MemLength, MemRd : STD_LOGIC;  -- memory control signals
  SIGNAL MemWr, MemEnable : STD_LOGIC;
  SIGNAL MemRdy : STD_LOGIC;  -- memory status signal
  SIGNAL MemData : WordT;  -- memory data bus

  SIGNAL IOAddr : IOAddrT;  -- I/O address bus
  SIGNAL IOLength, IORd : STD_LOGIC;  -- I/O control signals
  SIGNAL IOWr, IOEnable : STD_LOGIC;
  SIGNAL IORdy : STD_LOGIC;  -- I/O status signal
  SIGNAL IOData : WordT;  -- I/O data bus

  SIGNAL Status : StatusT;
BEGIN

U1: ENTITY Memory
   PORT MAP (MemAddr, MemLength, MemRd, MemWr, MemEnable,
               MemRdy, MemData);

U2: ENTITY IO
   PORT MAP (IOAddr, IOLength, IORd, IOWr, IOEnable,
             IORdy, IOData);

U3: ENTITY Processor
   PORT MAP (MemAddr, MemData, MemLength, MemRd, MemWr,
             MemEnable, MemRdy,
             IOAddr, IOData, IOLength, IORd, IOWr,
             IOEnable, IORdy,
             Status, Reset, Clk);

END structural;
Figure 15.4: MEMORY SUBSYSTEM. (a) EXTERNAL SIGNALS. (b) INTERNAL ORGANIZATION. (c) TIMING DIAGRAM
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE WORK.comp_pkg.ALL;
ENTITY Memory IS
    PORT (Addr : IN MAddrT ; -- memory address bus
          Length : IN STD_LOGIC; -- byte/word operand
          Rd, Wr : IN STD_LOGIC; -- access control signals
          Enable : IN STD_LOGIC; -- enable signal
          Rdy   : OUT STD_LOGIC; -- access completion signal
          Data  : INOUT WordT ); -- memory data bus
END Memory;
LIBRARY ieee;
USE ieee.std_logic_unsigned.ALL;

ARCHITECTURE behavioral OF Memory IS
  CONSTANT Tmem : TIME := 8 ns; -- nanoseconds
  CONSTANT Td   : TIME := 200 ps; -- picoseconds
  CONSTANT Tsu  : TIME := 200 ps; -- picoseconds
BEGIN
  PROCESS (Rd, Wr, Enable)
    CONSTANT byte_l: STD_LOGIC:= '0'; -- constant declarations
    CONSTANT word_l: STD_LOGIC:= '1';
     -- memory declaration
    CONSTANT MaxMem : NATURAL := 16#FFFFFF#; -- 2**24 bytes
  TYPE MemArrayT IS ARRAY(0 TO MaxMem-1) OF ByteT;
  VARIABLE Mem : MemArrayT;
   -- working variables
  VARIABLE tAddr : NATURAL;
  VARIABLE tData : WordT ;
  VARIABLE tCtrls: STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
  tCtrls:= Rd & Wr & Enable; -- group signals for simpler decoding
  CASE tCtrls IS
    -- output to tri-state
    WHEN "000" => Data <= (OTHERS =>'Z') AFTER Td;

    WHEN "011" =>
      -- write access;
      -- indicate module busy
      Rdy <= '0' AFTER Td, '1' AFTER Tmem;
    IF (Length = byte_1) THEN
      -- read address
      tAddr:= CONV_INTEGER(Addr); -- bit-vector to integer
      -- from pkg std_logic_unsigned
    ELSE
      tAddr:= CONV_INTEGER(Addr(23 DOWNTO 2) & "00");
    END IF;
  END CASE;

  CASE Length IS
    WHEN byte_1 => Mem(tAddr) := (Data( 7 DOWNTO 0));
    WHEN word_1 => Mem(tAddr) := (Data( 7 DOWNTO 0));
      Mem(tAddr+1) := (Data(15 DOWNTO 8));
      Mem(tAddr+2) := (Data(23 DOWNTO 16));
      Mem(tAddr+3) := (Data(31 DOWNTO 24));
    WHEN OTHERS => NULL;
  END CASE;

Introduction to Digital Systems
END CASE;

WHEN "101" => -- read access
   -- indicate module busy
   Rdy <= '0' AFTER Td, '1' AFTER Tmem;
   IF (Length = byte_1) THEN -- read address
      tAddr:= CONV_INTEGER(Addr); -- bit-vector to integer
   ELSE
      tAddr:= CONV_INTEGER(Addr(23 DOWNTO 2) & "00");
   END IF;
   CASE Length IS
      WHEN byte_1 => tData( 7 DOWNTO 0):= (Mem(tAddr));
      WHEN word_1 => tData( 7 DOWNTO 0):= (Mem(tAddr));
      tData(15 DOWNTO 8):= (Mem(tAddr+1));
      tData(23 DOWNTO 16):= (Mem(tAddr+2));
      tData(31 DOWNTO 24):= (Mem(tAddr+3));
      WHEN OTHERS => NULL;
   END CASE;
   Data <= tData AFTER Tmem; -- deliver data

   WHEN OTHERS => NULL; -- memory not enabled
END CASE;
END PROCESS;
-- timing verifications

ASSERT NOT (Rd’EVENT AND Rd=’1’ AND NOT Addr’STABLE(Tsu))
  REPORT "Read address setup time violation";

ASSERT NOT (Rd’EVENT AND Rd=’1’ AND NOT Enable’STABLE(Tsu))
  REPORT "Read enable setup time violation";

ASSERT NOT (Wr’EVENT AND Wr=’1’ AND NOT Addr’STABLE(Tsu))
  REPORT "Write address setup time violation";

ASSERT NOT (Wr’EVENT AND Wr=’1’ AND NOT Enable’STABLE(Tsu))
  REPORT "Write enable setup time violation";

END behavioral;
Figure 15.5: INPUT/OUTPUT SUBSYSTEM.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE WORK.comp_pkg.ALL;
ENTITY IO IS
  PORT (Addr : IN  IOAddrT ; -- I/O address bus
        Length : IN  STD_LOGIC; -- byte/word control
        Rd, Wr : IN  STD_LOGIC; -- I/O access control
        Enable : IN  STD_LOGIC; -- I/O enable control
        Rdy : OUT  STD_LOGIC; -- I/O completion signal
        Data : INOUT WordT ); -- I/O data bus
END IO;
Processor state

- 32 general-purpose registers (32-bits wide), called R0, R1, ..., R31;

- a 24-bit Program Counter register (PC);

- a 4-bit Condition Register (CR); and

- a 32-bit Instruction Register (IR).

Figure 15.6: PROCESOR STATE.
Figure 15.7: BEHAVIOR OF THE PROCESSOR. (a) INSTRUCTION LOOP. (b) MEMORY BUS BEHAVIOR FOR REGISTER OPERATION. (c) MEMORY BUS BEHAVIOR FOR LOAD OPERATION.
Figure 15.8: BEHAVIOR OF INSTRUCTIONS. (a) ADD instruction. (b) UNCONDITIONAL BRANCH INSTRUCTION. (c) CONDITIONAL BRANCH INSTRUCTION.
INSTRUCTION SEQUENCING

• SEQUENTIAL UNLESS

  1. UNCONDITIONAL BRANCH
  2. CONDITIONAL BRANCH
### Figure 15.9: INSTRUCTION FORMATS.

<table>
<thead>
<tr>
<th>RT:= op(RA)</th>
<th>31 25 20 15 10 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>RT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RT:= RA op RB</th>
<th>Opcode</th>
<th>RT</th>
<th>RA</th>
<th>RB</th>
<th>--</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>RT:= RA op SI</th>
<th>Opcode</th>
<th>RT</th>
<th>RA</th>
<th>SI</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>RT:= RA op UI</th>
<th>Opcode</th>
<th>RT</th>
<th>RA</th>
<th>UI</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>RT:= M[RA+D]</th>
<th>Opcode</th>
<th>RT</th>
<th>RA</th>
<th>D</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>M[RA+D]:= RS</th>
<th>Opcode</th>
<th>RS</th>
<th>RA</th>
<th>D</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>RT:= IO[P]N</th>
<th>Opcode</th>
<th>RT</th>
<th>RA</th>
<th>--</th>
<th>PN</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>IO[P]N:= RS</th>
<th>Opcode</th>
<th>RS</th>
<th>RA</th>
<th>--</th>
<th>PN</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PC:= PC + 4 + D</th>
<th>Opcode</th>
<th>--</th>
<th>D</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PC:= RA</th>
<th>Opcode</th>
<th>--</th>
<th>RA</th>
<th>--</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Opcode</td>
<td>Function</td>
<td>CR</td>
<td>Assembly Language</td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>------------------</td>
<td>----</td>
<td>-------------------</td>
</tr>
<tr>
<td>No-op</td>
<td>000000</td>
<td>no operation</td>
<td></td>
<td>nop</td>
</tr>
<tr>
<td>NOT</td>
<td>000010</td>
<td>RT:= not(RA)</td>
<td>Y</td>
<td>not RT,RA</td>
</tr>
<tr>
<td>Left shift</td>
<td>000100</td>
<td>RT:= lshift(RA)</td>
<td>Y</td>
<td>lsh RT,RA</td>
</tr>
<tr>
<td>Right shift</td>
<td>000110</td>
<td>RT:= rshift(RA)</td>
<td>Y</td>
<td>rsh RT,RA</td>
</tr>
<tr>
<td>Left rotate</td>
<td>001000</td>
<td>RT:= lrot(RA)</td>
<td>Y</td>
<td>lrt RT,RA</td>
</tr>
<tr>
<td>Right rot.</td>
<td>001010</td>
<td>RT:= rrot(RA)</td>
<td>Y</td>
<td>rrt RT,RA</td>
</tr>
<tr>
<td>Add</td>
<td>010000</td>
<td>RT:= RA + RB</td>
<td>Y</td>
<td>add RT,RA,RB</td>
</tr>
<tr>
<td>Add immed.</td>
<td>010001</td>
<td>RT:= RA + SI</td>
<td>Y</td>
<td>adi RT,RA,SI</td>
</tr>
<tr>
<td>Subtract</td>
<td>010010</td>
<td>RT:= RA - RB</td>
<td>Y</td>
<td>sub RT,RA,RB</td>
</tr>
<tr>
<td>Sub. immed.</td>
<td>010011</td>
<td>RT:= RA - SI</td>
<td>Y</td>
<td>sbi RT,RA,SI</td>
</tr>
<tr>
<td>AND</td>
<td>010100</td>
<td>RT:= RA and RB</td>
<td>Y</td>
<td>and RT,RA,RB</td>
</tr>
<tr>
<td>AND immed.</td>
<td>010101</td>
<td>RT:= RA and UI</td>
<td>Y</td>
<td>ani RT,RA,UI</td>
</tr>
<tr>
<td>OR</td>
<td>010110</td>
<td>RT:= RA or RB</td>
<td>Y</td>
<td>or RT,RA,RB</td>
</tr>
<tr>
<td>OR immed.</td>
<td>010111</td>
<td>RT:= RA or UI</td>
<td>Y</td>
<td>ori RT,RA,UI</td>
</tr>
<tr>
<td>XOR</td>
<td>011000</td>
<td>RT:= RA xor RB</td>
<td>Y</td>
<td>xor RT,RA,RB</td>
</tr>
<tr>
<td>XOR immed.</td>
<td>011001</td>
<td>RT:= RA xor UI</td>
<td>Y</td>
<td>xri RT,RA,UI</td>
</tr>
<tr>
<td>Name</td>
<td>Opcode</td>
<td>Function</td>
<td>CR</td>
<td>Assembly Language</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------</td>
<td>---------------------------</td>
<td>----</td>
<td>------------------</td>
</tr>
<tr>
<td>Load byte</td>
<td>100000</td>
<td>RT( 7 to 0):= Mem(RA+D,1)</td>
<td>ldb RT,D(RA)</td>
<td></td>
</tr>
<tr>
<td>Load word</td>
<td>100001</td>
<td>RT(31 to 0):= Mem(RA+D,4)</td>
<td>ldw RT,D(RA)</td>
<td></td>
</tr>
<tr>
<td>Store byte</td>
<td>100010</td>
<td>Mem(RA+D,1):= RS( 7 to 0)</td>
<td>stb RS,D(RA)</td>
<td></td>
</tr>
<tr>
<td>Store word</td>
<td>100011</td>
<td>Mem(RA+D,4):= RS(31 to 0)</td>
<td>stw RS,D(RA)</td>
<td></td>
</tr>
<tr>
<td>I/O Rd byte</td>
<td>100100</td>
<td>RT( 7 to 0):= IO(PN,1)</td>
<td>irb RT,PN</td>
<td></td>
</tr>
<tr>
<td>I/O Rd word</td>
<td>100101</td>
<td>RT(31 to 0):= IO(PN,4)</td>
<td>irw RT,PN</td>
<td></td>
</tr>
<tr>
<td>I/O Wr byte</td>
<td>100110</td>
<td>IO(PN,1):= RS( 7 to 0)</td>
<td>iwb RS,PN</td>
<td></td>
</tr>
<tr>
<td>I/O Wr word</td>
<td>100111</td>
<td>IO(PN,4):= RS(31 to 0)</td>
<td>iww RS,PN</td>
<td></td>
</tr>
</tbody>
</table>
Table 15.4: INSTRUCTION SET (cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Opcode</th>
<th>Function</th>
<th>CR</th>
<th>Assembly Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch</td>
<td>111000</td>
<td>PC:= PC + 4 + D</td>
<td>br</td>
<td>D</td>
</tr>
<tr>
<td>Branch indirect</td>
<td>111001</td>
<td>PC:= RA</td>
<td>bri</td>
<td>RA</td>
</tr>
<tr>
<td>Branch if N=0</td>
<td>110000</td>
<td>If N=0 then PC:= PC+4+D</td>
<td>brp</td>
<td>D</td>
</tr>
<tr>
<td>Branch if N=1</td>
<td>110001</td>
<td>If N=1 then PC:= PC+4+D</td>
<td>brn</td>
<td>D</td>
</tr>
<tr>
<td>Branch if Z=0</td>
<td>110010</td>
<td>If Z=0 then PC:= PC+4+D</td>
<td>bnz</td>
<td>D</td>
</tr>
<tr>
<td>Branch if Z=1</td>
<td>110011</td>
<td>If Z=1 then PC:= PC+4+D</td>
<td>brz</td>
<td>D</td>
</tr>
<tr>
<td>Branch if C=0</td>
<td>110100</td>
<td>If C=0 then PC:= PC+4+D</td>
<td>bnc</td>
<td>D</td>
</tr>
<tr>
<td>Branch if C=1</td>
<td>110101</td>
<td>If C=1 then PC:= PC+4+D</td>
<td>brc</td>
<td>D</td>
</tr>
<tr>
<td>Branch if V=0</td>
<td>110110</td>
<td>If V=0 then PC:= PC+4+D</td>
<td>bnv</td>
<td>D</td>
</tr>
<tr>
<td>Branch if V=1</td>
<td>110111</td>
<td>If V=1 then PC:= PC+4+D</td>
<td>brv</td>
<td>D</td>
</tr>
</tbody>
</table>
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE WORK.comp_pkg.ALL;

ENTITY processor IS
    PORT (MemAddr : OUT MAddrT ; -- memory address bus
           MemData : INOUT WordT ; -- data bus to/from memory
           MemLength: OUT STD_LOGIC; -- memory operand length
           MemRd   : OUT STD_LOGIC; -- memory read control signal
           MemWr   : OUT STD_LOGIC; -- memory write control signal
           MemEnable: OUT STD_LOGIC; -- memory enable signal
           MemRdy  : IN STD_LOGIC; -- memory completion signal
           IOAddr  : OUT IOAddrT ; -- I/O address bus
           IOData  : INOUT WordT ; -- data bus to/from I/O
           IOLength : OUT STD_LOGIC; -- I/O operand length
           IORd    : OUT STD_LOGIC; -- I/O read control signal
           IOWr    : OUT STD_LOGIC; -- I/O write control signal
           IOEnable: OUT STD_LOGIC; -- memory enable signal
           IORdy   : IN STD_LOGIC; -- I/O completion signal
           Status  : OUT StatusT ; -- processor status signal
           Reset   : IN STD_LOGIC; -- reset signal
           Clk     : IN STD_LOGIC); -- clock signal

END processor;
LIBRARY ieee;
USE ieee.std_logic_arith.all; -- use definitions and operations
USE ieee.std_logic_signed.all; -- on signed values

ARCHITECTURE behavioral OF processor IS
  -- registers (processor state)
  TYPE RegFileT IS ARRAY(0 to 31) OF WordT;
  SIGNAL GPR: RegFileT; -- general registers
  SIGNAL PC : MAddrT; -- Program Counter register
  SIGNAL CR : STD_LOGIC_VECTOR( 3 DOWNTO 0); -- Condition Register
  SIGNAL IR : STD_LOGIC_VECTOR(31 DOWNTO 0); -- Instruction register

  -- signals used by output function
  SIGNAL Phase: StatusT ; -- instr. cycle phase
  SIGNAL tMemAddr: WordT ; -- memory address
  SIGNAL tData : WordT ; -- memory/io data

  ALIAS Z : STD_LOGIC IS CR(0) ; -- Condition code Zero
  ALIAS N : STD_LOGIC IS CR(1) ; -- Condition code Negative
  ALIAS C : STD_LOGIC IS CR(2) ; -- Condition code Carry
  ALIAS O : STD_LOGIC IS CR(3) ; -- Condition code Overflow
ALIAS Opcode : STD_LOGIC_VECTOR(5 DOWNTO 0) IS IR(31 DOWNTO 26);
ALIAS RT : STD_LOGIC_VECTOR(4 DOWNTO 0) IS IR(25 DOWNTO 21);
ALIAS RA : STD_LOGIC_VECTOR(4 DOWNTO 0) IS IR(20 DOWNTO 16);
ALIAS RB : STD_LOGIC_VECTOR(4 DOWNTO 0) IS IR(15 DOWNTO 11);
ALIAS RS : STD_LOGIC_VECTOR(4 DOWNTO 0) IS IR(15 DOWNTO 11);
ALIAS Imm : STD_LOGIC_VECTOR(15 DOWNTO 0) IS IR(15 DOWNTO 0);
ALIAS D : STD_LOGIC_VECTOR(15 DOWNTO 0) IS IR(15 DOWNTO 0);
ALIAS PN : STD_LOGIC_VECTOR(10 DOWNTO 0) IS IR(10 DOWNTO 0);
ALIAS dlength: STD_LOGIC IS IR(26) ;

-- other declarations
CONSTANT delay : TIME := 200 ps; -- register delay
CONSTANT Reset_delay: TIME := 5 ns;
CONSTANT Exec_delay : TIME := 10 ns; -- Execute delay
CONSTANT Mdelay : TIME := 600 ps; -- MemEnable signal delay
CONSTANT Pulse_Width: TIME := 2.6 ns; -- memory signals width
CONSTANT Fetch_delay: TIME := 3 ns; -- disable memory after
                                 -- access completed
BEGIN

PROCESS -- transition function

-- working variables
VARIABLE RS_data, RA_data, RB_data : WordT;
VARIABLE RT_addr, RA_addr, RB_addr, RS_addr : Natural;

BEGIN

WAIT ON Clk, Reset;

IF (Reset'Event AND Reset = '1') THEN -- reset function
    PC <= (OTHERS => '0'); CR <= "0000"; IR <= (OTHERS => '0');
    
    FOR i IN 0 TO 31 LOOP
        GPR(i) <= (OTHERS => '0');
    END LOOP;
    
    Phase <= p_reset;
    Status <= p_reset;
    
    WAIT UNTIL (Reset = '0') AND (Clk = '1');
END IF;


IF (Clk'Event AND Clk='1') THEN
    -- Instruction cycle
    Status <= Fetch AFTER delay;
    Phase <= Fetch AFTER delay;
    -- instruction fetch
    PC <= PC + 4 AFTER Exec_delay;
    WAIT UNTIL MemRdy='1'; -- wait instr. fetch completed
    IR <= MemData;
    WAIT FOR Fetch_delay;
    -- instruction execution
    Status <= Execute;
    Phase <= Execute;
    RA_addr := CONV_INTEGER('0' & RA); RB_addr := CONV_INTEGER('0' & RB);
    -- '0' to force bit-vector to positive value
    RA_data := GPR(RA_addr); RB_data := GPR(RB_addr);
    RT_addr := CONV_INTEGER('0' & RT);
    RS_addr := CONV_INTEGER('0' & RS); -- source reg. for store
    RS_data := GPR(RS_addr); -- or I/O write
    WAIT FOR Exec_delay;
CASE Opcode IS
WHEN "000000" => null; -- nop
WHEN "000010" => GPR(RT_Addr)<= not(RA_data); -- not
WHEN "000100" => GPR(RT_Addr)<= RA_data(30 DOWNTO 0) & '0'; -- lshift
WHEN "000110" => GPR(RT_Addr)<= '0' & RA_data(31 DOWNTO 1); -- rshift
WHEN "001000" => GPR(RT_Addr)<= RA_data(30 DOWNTO 0) & RA_data(31);
WHEN "001010" => GPR(RT_Addr)<= RA_DATA(0) & RA_data(31 DOWNTO 1);
WHEN "010000" => GPR(RT_Addr)<= RA_data + RB_data ; -- add
WHEN "010001" => GPR(RT_Addr)<= RA_data + Imm;
WHEN "010010" => GPR(RT_Addr)<= RA_data - RB_data ; -- sub
WHEN "010011" => GPR(RT_Addr)<= RA_data - Imm;
WHEN "010100" => GPR(RT_Addr)<= RA_data and RB_data ; -- and
WHEN "010101" => GPR(RT_Addr)<= RA_data and ext(Imm,RA_data’LENGTH);
WHEN "010110" => GPR(RT_Addr)<= RA_data or RB_data ; -- or
WHEN "010111" => GPR(RT_Addr)<= RA_data or ext(Imm,RA_data’LENGTH);
WHEN "011000" => GPR(RT_addr) <= RA_data xor RB_data ; -- xor
WHEN "011001" => GPR(RT_addr) <= RA_data xor ext(Imm, RA_data'LENGTH);

WHEN "100000" | "100001" => -- ldb, ldw
    Phase <= MemOp;
    Status <= MemOp;
    tMemAddr <= RA_data + D; -- mem. addr.
    WAIT until MemRdy = '1';

WHEN "100010" | "100011" => -- stb, stw
    Phase <= MemOp;
    Status <= MemOp;
    tMemAddr <= RA_data + D; -- mem. addr.
    tData <= RS_data; -- mem. data
    WAIT until MemRdy = '1';

WHEN "100100" | "100101" => -- irb, irw
    Phase <= IOOp;
    Status <= IOOp;
    WAIT until IORdy = '1';
WHEN "100110" | "100111" => -- iwb, iww
  Phase <= IOOp;
  Status <= IOOp;
  tData <= RS_data; -- io data
  WAIT until IORdy = '1' ;

WHEN "111000" => PC <= PC + D; -- branch
WHEN "111001" => PC <= RA_data(23 DOWNT0 0); -- br.ind.
WHEN "110000" | "110001"
  => IF (N = Opcode(0)) THEN -- br on N
     PC <= PC + D;
     END IF;
WHEN "110010" | "110011"
  => IF (Z = Opcode(0)) THEN -- br on Z
     PC <= PC + D;
     END IF;
WHEN "110100" | "110101"
  => IF (C = Opcode(0)) THEN -- br on C
     PC <= PC + D;
     END IF;
WHEN "110110" | "110111"
  => IF (O = Opcode(0)) THEN -- br on V
     PC <= PC + D;

Introduction to Digital Systems
END IF;

WHEN others => null;
END CASE;

IF ((Opcode(5 DOWNTO 4) = 0) or (Opcode(5 DOWNTO 4) = 1))
and (Opcode /= 0) THEN
  -- set condition register
  IF (GPR(RT_Addr) = 0) THEN CR(0) <= '1'; -- zero result
  ELSE
    CR(0) <= '0';
  END IF;
  IF (GPR(RT_Addr)(31) = '1') THEN CR(1) <= '1'; -- negative result
  ELSE
    CR(1) <= '0';
  END IF;
  -- check if operation Opcode generates carry out
  CR(2) <= get_carry(RA_Data,RB_Data,Imm,Opcode);
  -- check if operation Opcode generates overflow
  CR(3) <= get_ovf(RA_Data,RB_Data,Imm,Opcode);
END IF;

WAIT FOR 0 ns; -- force signals to be updated
IF (Phase = MemOp) THEN
  IF (dlength = '1') THEN
    -- ldw
    GPR(RT_addr) <= MemData;
  ELSE
    -- ldb
    GPR(RT_addr)(7 DOWNTO 0) <= MemData(7 DOWNTO 0);
    GPR(RT_addr)(31 DOWNTO 8) <= (OTHERS => '0');
  END IF;
  WAIT FOR Fetch_delay;
END IF;
ENDIF;

IF (Phase = IOOp) THEN
  IF (dlength = '1') THEN
    -- irw
    GPR(RT_addr) <= IOData;
  ELSE
    -- irb
    GPR(RT_addr)(7 DOWNTO 0) <= IOData(7 DOWNTO 0);
    GPR(RT_addr)(31 DOWNTO 8) <= (OTHERS => '0');
  END IF;
  WAIT FOR Fetch_delay;
END IF;
ENDIF;
END IF;
END PROCESS;

PROCESS -- output function
BEGIN
-- Instruction cycle
WAIT ON Phase;
IF (Phase = p_reset) THEN -- reset
    MemRd <= '0'; MemWr <= '0'; MemEnable <= '0'; MemLength <= '0';
    MemData <= (OTHERS => 'Z');
    IORd <= '0'; IOWr <= '0'; IOEnable <= '0'; IOLength <= '0';
    IOData <= (OTHERS => 'Z');
ELSIF (Phase = Fetch) THEN -- instruction fetch
    MemAddr <= PC AFTER delay;
    MemEnable <= '1' AFTER delay;
    MemRd <= '1' AFTER Mdelay, '0' AFTER Pulse_Width;
    MemLength <= '1' AFTER delay;
    WAIT UNTIL MemRdy='1'; -- wait instr. fetch completed
    MemEnable <= '0' AFTER Fetch_delay;

ELSIF (Phase = Execute) THEN NULL; -- instruction execution
-- no output signals

ELSIF (Phase = MemOp) THEN
    MemAddr <= tMemAddr(23 DOWNTO 0) AFTER delay;
    MemEnable <= '1' AFTER delay;
MemLength <= dlength AFTER delay;
IF ((To_Bitvector(Opcode) = "100000") OR
    (To_Bitvector(Opcode) = "100001")) THEN -- ldb, ldw
    MemRd <= '1' AFTER Mdelay, '0' AFTER Pulse_Width;
    WAIT until MemRdy = '1';
    MemEnable <= '0' AFTER Fetch_delay;
    WAIT FOR Fetch_delay;
ENDIF;
IF ((To_Bitvector(Opcode) = "100010") OR
    (To_Bitvector(Opcode) = "100011")) THEN -- stb, stw
    MemWr <= '1' AFTER Mdelay, '0' AFTER Pulse_Width;
    IF (dlength = '1') THEN -- stw
        MemData <= tData AFTER delay;
    ELSE -- stb
        MemData(7 DOWNTO 0) <= tData(7 DOWNTO 0) AFTER delay;
    END IF;
    WAIT until MemRdy = '1';
    MemEnable <= '0' AFTER delay;
    MemData <= (OTHERS => 'Z') AFTER delay;
    WAIT FOR delay;
ENDIF;

ELSIF (Phase = IOOp) THEN
IOAddr <= PN AFTER delay;
IOEnable <= '1' AFTER delay;
IOLength <= dlength AFTER delay;
IF ((To_Bitvector(Opcode) = "100100") OR
    (To_Bitvector(Opcode) = "100101") ) THEN -- irb, irw
    IORd <= '1' AFTER Mdelay, '0' AFTER Pulse_Width;
    WAIT until IORdy = '1';
    IOEnable <= '0' AFTER Fetch_delay;
    WAIT FOR Fetch_delay;
END IF;
IF ((To_Bitvector(Opcode) = "100110") OR
    (To_Bitvector(Opcode) = "100111") ) THEN -- iwb, iww
    IF (dlength = '1') THEN -- iww
        IOData <= tData AFTER delay;
    ELSE
        IOData(7 DOWNTO 0) <= tData(7 DOWNTO 0) AFTER delay;
    END IF;
    IOWr <= '1' AFTER Mdelay, '0' AFTER Pulse_Width;
    WAIT until IORdy = '1';
    IOEnable <= '0' AFTER delay;
    IOData <= (OTHERS => 'Z') AFTER delay;
    WAIT FOR delay;
END IF;
END IF;
END PROCESS;
END behavioral;
\[\text{\mu}VHDL\] SPECIFICATION OF MEMORY CONTENTS

-- memory declaration

CONSTANT MaxMem: NATURAL := 16#FFF#; -- 4Kbytes

TYPE MemArrayT IS ARRAY(0 to MaxMem-1) OF ByteT;

VARIABLE Mem : MemArrayT :=

(-- program
  3 => "01100000", 2 => "00000000", 1 => "00000000", 0 => "00000000",
  7 => "01000100", 6 => "00100000", 5 => "00000000", 4 => "00110010",
  11 => "10000110", 10 => "10000001", 9 => "00000000", 8 => "00000000",
  15 => "10000110", 14 => "10100001", 13 => "00000000", 12 => "00000100",
  19 => "01000100", 18 => "01000000", 17 => "00000000", 16 => "01111111",
  23 => "10001000", 22 => "01000001", 21 => "00000000", 20 => "00000000",

(-- data
  51 => "00110011", 50 => "00001111", 49 => "11110000", 48 => "11001100",
  55 => "00110011", 54 => "00001111", 53 => "11110000", 52 => "11001100",
  OTHERS => "00000000")

where the memory contents corresponds to the following instructions:

- 0x000000: xor R0,R0,R0  ; R0 = 0
- 0x000004: adi R1,R0,50  ; R1 = 50
- 0x000008: ldw R20,0(R1)  ; R20 = Mem(50,4) = Mem(48,4)
- 0x00000C: ldw R21,4(R1)  ; R21 = Mem(54,4) = Mem(52,4)
- 0x000010: adi R2,R0,63  ; R2 = 63
- 0x000014: stb R2,0(R1)   ; Mem(50,1) = 63

- 0x000048: 0x330FF0CC
- 0x000052: 0x330FF0CC
IMPLEMENTATION OF XMC

- MEMORY SUBSYSTEM
- PROCESSOR
  1. DATA SUBSYSTEM
  2. CONTROL SUBSYSTEM
Figure 15.10: IMPLEMENTATION OF THE MEMORY SUBSYSTEM.
Figure 15.11: IMPLEMENTATION OF THE PROCESSOR.
Figure 15.12: IMPLEMENTATION OF DATA SUBSYSTEM (I/O signals not shown).
### ALU

<table>
<thead>
<tr>
<th>ALUop</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Zero_32</td>
</tr>
<tr>
<td>0001</td>
<td>A + B</td>
</tr>
<tr>
<td>0010</td>
<td>A - B</td>
</tr>
<tr>
<td>0011</td>
<td>-B</td>
</tr>
<tr>
<td>0100</td>
<td>A and B</td>
</tr>
<tr>
<td>0101</td>
<td>A or B</td>
</tr>
<tr>
<td>0110</td>
<td>A xor B</td>
</tr>
<tr>
<td>0111</td>
<td>not(B)</td>
</tr>
<tr>
<td>1000</td>
<td>unused</td>
</tr>
<tr>
<td>1001</td>
<td>B</td>
</tr>
<tr>
<td>1010</td>
<td>shiftl(A)</td>
</tr>
<tr>
<td>1011</td>
<td>shiftr(A)</td>
</tr>
<tr>
<td>1100</td>
<td>rotl(A)</td>
</tr>
<tr>
<td>1101</td>
<td>rotr(A)</td>
</tr>
<tr>
<td>1110</td>
<td>A + 4</td>
</tr>
<tr>
<td>1111</td>
<td>unused</td>
</tr>
</tbody>
</table>
Figure 15.13: SEQUENCE OF EVENTS IN DATA SUBSYSTEM FOR ALU INSTRUCTION.
Figure 15.14: TIMING DIAGRAM FOR ALU INSTRUCTION IN DATA SUBSYSTEM.
Figure 15.15: CONTROL SUBSYSTEM.
Figure 15.16: STATE DIAGRAM AND TIMING FOR CONTROL SUBSYSTEM OF THE PROCESSOR.
OPERATION OF THE COMPUTER AND CYCLE TIME

Figure 15.17: DEPENDENCIES FOR STATE FETCH.
Figure 15.18: DEPENDENCIES FOR STATE EXECUTE.
Figure 15.19: DEPENDENCIES FOR STATE Memop.
EXAMPLE 15.1: OBTAIN MIN CYCLE PERIOD

<table>
<thead>
<tr>
<th>Component</th>
<th>Delay Type</th>
<th>Delay Time</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Reg_delay</td>
<td>$t_R$</td>
<td>2 ns (setup and propagation delay)</td>
</tr>
<tr>
<td>Register file</td>
<td>RF_delay</td>
<td>$t_{RF}$</td>
<td>4 ns</td>
</tr>
<tr>
<td>ALU</td>
<td>ALU_delay</td>
<td>$t_{ALU}$</td>
<td>6 ns</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>Mux_delay</td>
<td>$t_{mux}$</td>
<td>0.5 ns</td>
</tr>
<tr>
<td>Zero/sign ext.</td>
<td>Ext_delay</td>
<td>$t_{ZSE}$</td>
<td>0.5 ns</td>
</tr>
<tr>
<td>Switch</td>
<td>Switch_delay</td>
<td>$t_{sw}$</td>
<td>0.5 ns</td>
</tr>
<tr>
<td>Control delay</td>
<td>Ctrl_delay</td>
<td>$t_{ctl}$</td>
<td>0.5 ns</td>
</tr>
<tr>
<td>Decode delay</td>
<td>Dec_delay</td>
<td>$t_{dec}$</td>
<td>3 ns</td>
</tr>
<tr>
<td>Memory</td>
<td>Mem_delay</td>
<td>$t_{mem}$</td>
<td>8 ns (static memory)</td>
</tr>
</tbody>
</table>

CRITICAL PATHS ARE OBTAINED:

\[
t_{fetch} = t_R + t_{ctl} + t_{mux} + t_{mem} + t_{sw} = 2 + 0.5 + 1 + 8 + 0.5 = 12 \text{ ns}
\]

\[
t_{exec} = t_R + t_{ctl} + t_{RF} + t_{mux} + t_{ALU} + t_{mux} + t_{RF} = 2 + 0.5 + 4 + 0.5 + 6 + 0.5 + 4 = 17.5 \text{ ns}
\]

\[
t_{memop} = t_R + t_{ctl} + t_{mem} + t_{sw} + t_{mux} + t_{RF} = 2 + 0.5 + 8 + 0.5 + 0.5 + 4 = 15 \text{ ns}
\]
LIBRARY ieee; USE ieee.std_logic_1164.ALL; USE WORK.comp_pkg.ALL, WORK.ALL;

ARCHITECTURE structural OF Processor IS
SIGNAL Instr                  : WordT;
SIGNAL ZE, NG, CY, OV        : STD_LOGIC;
SIGNAL AddrA, AddrB, AddrC   : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL ALUOp                 : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL WrC, WrPC, WrCR, WrIR : STD_LOGIC;
SIGNAL Mem_ALU, PC_RA, IR_RB : STD_LOGIC;
SIGNAL ALU_PC, ZE_SE, SinSout: STD_LOGIC;

BEGIN
  P1: ENTITY Data_Subsystem
    PORT MAP (MemAddr, MemData, IOAddr, IOData,
               Instr, ZE, NG, CY, OV, AddrA, AddrB, AddrC, ALUOp,
               WrC, WrPC, WrCR, WrIR, Mem_ALU, PC_RA, IR_RB, ALU_PC,
               ZE_SE, SinSout, Clk, Reset);

  P2: ENTITY Ctrl_Subsystem
    PORT MAP (Instr, ZE, NG, CY, OV, AddrA, AddrB, AddrC, ALUOp,
               WrC, WrPC, WrCR, WrIR, Mem_ALU, PC_RA, IR_RB, ALU_PC,
               ZE_SE, SinSout, MemRd, MemWr, MemLength, MemEnable,
               MemRdy, IORd, IOWr, IOLength, IOEnable, IORdy, Status,
               Clk, Reset);

END structural;
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.comp_pkg.ALL, WORK.ALL;
ENTITY Data_Subsystem IS
  PORT(MemAddr : OUT MAddrT ;
       MemData : INOUT WordT ;
       IOAddr  : OUT IOAddrT ;
       IOData  : INOUT WordT ;
       Instr   : OUT WordT ;
       ZE, NG, CY, OV : OUT STD_LOGIC ;
       AddrA, AddrB, AddrC : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
       ALUOp : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
       WrC, WrPC, WrCR, WrIR : IN STD_LOGIC ;
       Mem_ALU, PC_RA, IR_RB : IN STD_LOGIC ;
       ALU_PC, ZE_SE, Sin_Sout: IN STD_LOGIC ;
       Clk, Reset : IN STD_LOGIC);
END Data_Subsystem;
ARCHITECTURE structural OF Data_Subsystem IS
  SIGNAL DataA, DataB, DataC : WordT ;
  SIGNAL Ain, Bin : WordT ;
  SIGNAL ALUdata, IRdata : WordT ;
  SIGNAL tMemdata : WordT ;
  SIGNAL Cond, CRout : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL IRreg, IRext : WordT ;
  SIGNAL PCout : WordT := (OTHERS => '0');
BEGIN
  ALU1: ENTITY ALU
    PORT MAP(Ain,Bin,ALUop,ALUdata,Cond);
  GPR: ENTITY Reg_File
    PORT MAP(AddrA,AddrB,AddrC,DataA,DataB,DataC,
    WrC,Reset,Clk);
  PC: ENTITY Reg
    PORT MAP(ALUdata(23 DOWNTO 0),PCout(23 DOWNTO 0),
    WrPC,Reset,Clk);
  CR: ENTITY Reg
    PORT MAP(Cond,CRout,WrCR,Reset,Clk);
  ZE <= CRout(0);  CY <= CRout(1);
  NG <= CRout(2);  OV <= CRout(3);
IR: ENTITY Reg
    PORT MAP(tMemData,IRReg,WrIR,Reset,Clk);

    Instr <= IRReg;

MX1: ENTITY Mux
    PORT MAP(tMemData,ALUdata,Mem_ALU,DataC);

MX2: ENTITY Mux
    PORT MAP(PCout,DataA,PC_RA,Ain);

ZSE: ENTITY Extender
    PORT MAP(IRreg,ZE_SE,IRext);

MX3: ENTITY Mux
    PORT MAP(IRext,DataB,IR_RB,Bin);

MX4: ENTITY Mux
    PORT MAP(ALUdata(23 DOWNTO 0),PCout(23 DOWNTO 0),
              ALU_PC,MemAddr);

SL : ENTITY Switch
    PORT MAP(MemData,tMemData,DataB,Sin_Sout);
END structural;
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE WORK.comp_pkg.ALL;

ENTITY Reg_File IS
  PORT(AddrA, AddrB, AddrC : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
       DataA, DataB : OUT WordT;
       DataC       : IN WordT;
       WrC         : IN STD_LOGIC;
       Reset, Clk  : IN STD_LOGIC);
END Reg_File;

ARCHITECTURE behavioral OF Reg_File IS
  TYPE RegFileT IS ARRAY(0 to 31) OF WordT;
  SIGNAL GPR : RegFileT;
BEGIN
  PROCESS(AddrA, AddrB) -- output function
  CONSTANT RF_delay : TIME := 4 ns;
  BEGIN
    DataA <= GPR(CONV_INTEGER(AddrA)) AFTER RF_delay;
    DataB <= GPR(CONV_INTEGER(AddrB)) AFTER RF_delay;
  END PROCESS;

μVHDL DESCRIPTION OF REGISTER FILE
PROCESS(Reset, Clk) -- transition function
BEGIN
  IF (Reset'EVENT and (Reset = '1')) THEN
    FOR i IN 0 TO 31 LOOP
      GPR(i) <= (OTHERS => '0');
    END LOOP;
  END IF;

  IF (Clk'EVENT AND Clk = '1' AND WrC = '1') THEN
    GPR(CONV_INTEGER(AddrC)) <= DataC;
  END IF;
END PROCESS;
END behavioral;


**μVHDL DESCRIPTION OF ALU**

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;
USE WORK.comp_pkg.ALL;

ENTITY ALU IS
  PORT(A, B: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
       Op : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
       C : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
       Cond: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END ALU;

ARCHITECTURE behavioral OF ALU IS
BEGIN
  PROCESS(A, B, Op)
  CONSTANT ALU_delay : TIME := 6 ns;
  BEGIN
    CASE Op IS
      WHEN "0000" => C<=(OTHERS=>'0') AFTER ALU_delay;
      WHEN "0001" => C<=A+B AFTER ALU_delay;
      WHEN "0010" => C<=A-B AFTER ALU_delay;
      WHEN "0011" => C<=(OTHERS=>'0') AFTER ALU_delay;
      WHEN "0100" => C<=A and B AFTER ALU_delay;
      WHEN "0101" => C<=A or B AFTER ALU_delay;
      WHEN "0110" => C<=A xor B AFTER ALU_delay;
      WHEN "0111" => C<=(OTHERS=>'0') AFTER ALU_delay;
    END CASE;
  END PROCESS;
END behavioral;
```

---

*Introduction to Digital Systems* 15 – Specification and Implementation of a Microcomputer
WHEN "1000" => C <= A AFTER ALU_delay;
WHEN "1001" => C <= B AFTER ALU_delay;
WHEN "1010" => C <= A(30 DOWNTO 0) & '0' AFTER ALU_delay;
WHEN "1011" => C <= '0' & A(31 DOWNTO 1) AFTER ALU_delay;
WHEN "1100" => C <= A(30 DOWNTO 0) & A(31) AFTER ALU_delay;
WHEN "1101" => C <= A(0) & A(31 DOWNTO 1) AFTER ALU_delay;
WHEN "1110" => C <= A + 4 AFTER ALU_delay;
WHEN "1111" => C <= not(A) AFTER ALU_delay;
WHEN OTHERS => NULL;
END CASE;
Cond <= get_cc(A,B,Op) AFTER ALU_delay;
END PROCESS;
END behavioral;
LIBRARY ieee; USE ieee.std_logic_1164.ALL;

ENTITY Reg IS
  PORT(Data_in: IN STD_LOGIC_VECTOR;
       Data_out: OUT STD_LOGIC_VECTOR;
       Wr         : IN STD_LOGIC;
       Reset      : IN STD_LOGIC;
       Clk        : IN STD_LOGIC);
END Reg;

ARCHITECTURE behavioral OF Reg IS
BEGIN
  PROCESS(Wr,Reset,Clk)
  BEGIN
    CONSTANT Reg_delay: TIME := 2 ns;
    VARIABLE BVZero: STD_LOGIC_VECTOR(Data_in'RANGE) := (OTHERS => '0');
    BEGIN
      IF (Reset = '1') THEN
        Data_out <= BVZero AFTER Reg_delay;
      END IF;

      IF (Clk'EVENT AND Clk = '1' AND Wr = '1') THEN
        Data_out <= Data_in AFTER Reg_delay;
      END IF;
    END PROCESS;
  END behavioral;
END Reg;
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Mux IS
  PORT(A_in,B_in: IN STD_LOGIC_VECTOR;
       Sel  : IN STD_LOGIC
         ;
       Data_out : OUT STD_LOGIC_VECTOR);
END Mux;

ARCHITECTURE behavioral OF Mux IS
BEGIN
  PROCESS(A_in, B_in, Sel)
  BEGIN
    CONSTANT Mux_delay: TIME := 500 ps;
    BEGIN
      IF (Sel = '0') THEN
        Data_out <= A_in AFTER Mux_delay;
      ELSE
        Data_out <= B_in AFTER Mux_delay;
      END IF;
    END PROCESS;
  END behavioral;
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Extender IS
  PORT(X_in : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
       ZE_SE : IN STD_LOGIC ;
       X_out : OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
END Extender;

ARCHITECTURE behavioral OF Extender IS
BEGIN
  PROCESS(X_in, ZE_SE)
  BEGIN
    CONSTANT Ext_delay: TIME := 500 ps;
    BEGIN
      IF (ZE_SE = '0') THEN
        X_out(31 DOWNTO 16) <= (OTHERS => '0') AFTER Ext_delay;
        X_out(15 DOWNTO 0) <= X_in(15 DOWNTO 0) AFTER Ext_delay;
      ELSE
        X_out(31 DOWNTO 16) <= (OTHERS => X_in(15)) AFTER Ext_delay;
        X_out(15 DOWNTO 0) <= X_in(15 DOWNTO 0) AFTER Ext_delay;
      END IF;
    END PROCESS;
  END behavioral;
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Switch IS
  PORT(A : INOUT STD_LOGIC_VECTOR;
       B_out: OUT STD_LOGIC_VECTOR;
       C_in : IN STD_LOGIC_VECTOR;
       Sel : IN STD_LOGIC);
END Switch;

ARCHITECTURE behavioral OF Switch IS
BEGIN
  PROCESS(A, C_in, Sel)
  CONSTANT Switch_delay: TIME := 500 ps;
  CONSTANT dataZ: STD_LOGIC_VECTOR(A’RANGE):=(OTHERS => ’Z’);
  BEGIN
    IF (Sel = ’0’) THEN
      B_out <= A AFTER Switch_delay;
      A <= dataZ;
    ELSE
      A <= C_in AFTER Switch_delay;
    END IF;
  END PROCESS;
END behavioral;
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.comp_pkg.ALL;

ENTITY Ctrl_Subsystem IS
  PORT(Instr : IN WordT;
       ZE, NG, CY, OV : IN STD_LOGIC;
       AddrA, AddrB, AddrC : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
       ALUOp : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       WrC, WrPC, WrCR, WrIR : OUT STD_LOGIC;
       Mem_ALU, PC_RA, IR_RB : OUT STD_LOGIC;
       ALU_PC, ZE_SE, Sin_Sout : OUT STD_LOGIC;
       MemRd, MemWr, MemLength, MemEnable : OUT STD_LOGIC;
       MemRdy : IN STD_LOGIC;
       IORd, IOWr, IOLength, IOEnable : OUT STD_LOGIC;
       IORdy : IN STD_LOGIC;
       Status : OUT StatusT;
       Clk, Reset : IN STD_LOGIC);
END Ctrl_Subsystem;
LIBRARY ieee;
USE ieee.std_logic_signed.ALL;

ARCHITECTURE behavioral OF Ctrl_Subsystem IS
  SIGNAL State: StatusT;
BEGIN
  PROCESS -- transition function
    ALIAS Opcode: STD_LOGIC_VECTOR(5 DOWNTO 0) IS Instr(31 DOWNTO 26);
    CONSTANT Reset_delay: TIME := 500 ps ;
    CONSTANT Ctrl_delay : TIME := 500 ps ;
  BEGIN
    WAIT ON Clk,Reset;
    IF (Reset'EVENT AND Reset = '1') THEN
      State <= p_reset AFTER Reset_delay;
      Status <= p_reset AFTER Reset_delay;
      WAIT UNTIL Clk = '1';
    END IF;

    IF (Clk’EVENT) AND (Clk = ’1’) THEN
      CASE State IS
        WHEN p_reset => Status <= fetch AFTER Ctrl_delay;
        State <= fetch AFTER Ctrl_delay;
        WHEN fetch => Status <= execute AFTER Ctrl_delay;
        State <= execute AFTER Ctrl_delay;
      END CASE;
  END PROCESS;
END;
WHEN execute => CASE Opcode IS
  WHEN "100000" | "100001" => State <= memop AFTER Ctrl_delay;
                      Status <= memop AFTER Ctrl_delay;
  WHEN "100010" | "100011" => State <= memop AFTER Ctrl_delay;
                      Status <= memop AFTER Ctrl_delay;
  WHEN OTHERS => State <= fetch AFTER Ctrl_delay;
                Status <= fetch AFTER Ctrl_delay;
END CASE;
WHEN memop | ioop => Status <= fetch AFTER Ctrl_delay;
          State <= fetch AFTER Ctrl_delay;
WHEN undef => NULL;
END CASE;
END IF;
END PROCESS;

PROCESS(State, Instr, MemRdy) -- output function
ALIAS Opcode : STD_LOGIC_VECTOR(5 DOWNTO 0) IS Instr(31 DOWNTO 26);
ALIAS Imm : STD_LOGIC_VECTOR(15 DOWNTO 0) IS Instr(15 DOWNTO 0);
ALIAS D : STD_LOGIC_VECTOR(15 DOWNTO 0) IS Instr(15 DOWNTO 0);
ALIAS PN : STD_LOGIC_VECTOR(10 DOWNTO 0) IS Instr(10 DOWNTO 0);

CONSTANT Dec_delay : TIME:= 3 ns;
CONSTANT Ctrl_delay : TIME:= 500 ps;
CONSTANT MemRd_delay: TIME:= 2500 ps;
CONSTANT MemRd_pulse: TIME:= MemRd_delay + 3 ns ;
CONSTANT MemWr_delay: TIME:= 2500 ps;
CONSTANT MemWr_pulse: TIME:= MemWr_delay + 3 ns ;
TYPE Ctrl_LineT IS
RECORD
    MemOp, WrMem : STD_LOGIC;
    RS_RB, IR_RB : STD_LOGIC;
    WrC, WrPC, WrCR : STD_LOGIC;
    ZE_SE : STD_LOGIC;
    ALUop : STD_LOGIC_VECTOR(3 DOWNTO 0);
END RECORD;

VARIABLE Ctrl_Line : Ctrl_LineT;

TYPE Ctrl_TableT IS ARRAY(NATURAL RANGE 0 TO 63) OF Ctrl_LineT;

CONSTANT Ctrl_Table: Ctrl_TableT:=
    (0 => ('0', '0', '1', '0', '0', '0', '0', '0', "0000"), -- nop
     2 => ('0', '0', '1', '1', '1', '0', '1', '0', "1111"), -- not
     4 => ('0', '0', '1', '1', '1', '0', '1', '0', "1010"), -- lsh
     6 => ('0', '0', '1', '1', '1', '0', '1', '0', "1011"), -- rsh
     8 => ('0', '0', '1', '1', '1', '0', '1', '0', "1100"), -- lrt
     10=> ('0', '0', '1', '1', '1', '0', '1', '0', "1101"), -- rrt
     16=> ('0', '0', '1', '1', '1', '0', '1', '0', "0001"), -- add
     17=> ('0', '0', '1', '1', '1', '0', '1', '0', "0011"), -- adi
     18=> ('0', '0', '1', '1', '1', '0', '1', '0', "0010"), -- sub
     19=> ('0', '0', '1', '1', '1', '0', '1', '0', "0010"), -- sbi
     20=> ('0', '0', '1', '1', '1', '0', '1', '0', "0100"), -- and
     21=> ('0', '0', '1', '1', '1', '0', '1', '0', "0100"), -- ani
\begin{verbatim}
22=> ('0', '0', '1', '1', '1', '0', '1', '0', "0101"), -- or
23=> ('0', '0', '1', '0', '1', '0', '1', '0', "0110"), -- xor
24=> ('0', '0', '1', '1', '1', '0', '1', '0', "0110"), -- xri
25=> ('0', '0', '1', '0', '1', '0', '1', '0', "0110"), -- xri

32=> ('1', '0', '0', '0', '1', '0', '0', '1', "0001"), -- ldb
33=> ('1', '0', '0', '0', '1', '0', '0', '1', "0001"), -- ldw
34=> ('1', '1', '0', '0', '0', '0', '0', '1', "0001"), -- stb
35=> ('1', '1', '0', '0', '0', '0', '0', '1', "0001"), -- stw
36=> ('1', '0', '0', '1', '1', '0', '0', '0', "1001"), -- irb
37=> ('1', '0', '0', '1', '1', '0', '0', '0', "1001"), -- irw
38=> ('1', '1', '0', '1', '0', '0', '0', '0', "1001"), -- iwb
39=> ('1', '1', '0', '1', '0', '0', '0', '0', "1001"), -- iww

56=> ('0', '0', '1', '0', '0', '1', '0', '1', "0001"), -- br
57=> ('0', '0', '1', '0', '0', '1', '0', '1', "1000"), -- bri
48=> ('0', '0', '1', '0', '0', '1', '0', '1', "0001"), -- brp
49=> ('0', '0', '1', '0', '0', '1', '0', '1', "0001"), -- brn
50=> ('0', '0', '1', '0', '0', '1', '0', '1', "0001"), -- bnz
51=> ('0', '0', '1', '0', '0', '1', '0', '1', "0001"), -- brz
52=> ('0', '0', '1', '0', '0', '1', '0', '1', "0001"), -- bnc
53=> ('0', '0', '1', '0', '0', '1', '0', '1', "0001"), -- brc
54=> ('0', '0', '1', '0', '0', '1', '0', '1', "0001"), -- bnv
55=> ('0', '0', '1', '0', '0', '1', '0', '1', "0001"), -- brv

OTHERS => ('0', '0', '1', '1', '0', '0', '0', '1', "0000")
\end{verbatim}
BEGIN
IF (State’EVENT) THEN
  CASE State IS
    WHEN undef => NULL;
    WHEN p_reset => ALUOp <= "0000";
    MemRd <= '0'; MemWr <= '0';
    MemEnable <= '0'; MemLength <= '0';
    IORd <= '0'; IOWr <= '0';
    IOEnable <= '0'; IOLength <= '0';
    WHEN fetch =>
      -- disable write signals from previous cycle
      WrCR <= '0' AFTER Ctrl_delay;
      WrC <= '0' AFTER Ctrl_delay;
      -- fetch instruction
      ALU_PC <= '1' AFTER Ctrl_delay;
      MemLength <= '1' AFTER Ctrl_delay;
      MemEnable <= '1' AFTER Ctrl_delay;
      MemRd <= '1' AFTER MemRd_delay, '0' AFTER MemRd_pulse;
      Sin_Sout <= '0' AFTER Ctrl_delay; -- switch in
      -- increment PC
      PC_RA <= '0' AFTER Ctrl_delay;
      ALUop <= "1110" AFTER Ctrl_delay; -- PC + 4
      WrIR <= '1' AFTER Ctrl_delay;
      WrPC <= '1' AFTER Ctrl_delay;
  END CASE;
END IF;
WHEN execute =>
    -- disable signals from fetch cycle
    WrIR <= '0' AFTER Ctrl_delay;
    WrPC <= '0' AFTER Ctrl_delay;
    MemEnable<= '0' AFTER Ctrl_delay;

    -- other actions done by Instr'EVENT

WHEN memop | ioop =>
    -- initiate memory access
    ALU_PC <= '0' AFTER Ctrl_delay;        -- address to memory
    MemEnable<= '1' AFTER Ctrl_delay;
    MemLength<= Opcode(0) AFTER Ctrl_delay;   -- operand length
    WrC <= Ctrl_Line.WrC AFTER Ctrl_delay;
    IF (Ctrl_Line.WrMem = '0') THEN
        MemRd <= '1' AFTER MemRd_delay, '0' AFTER MemRd_pulse;
        Mem_ALU <= '0' AFTER Ctrl_delay;
    ELSE
        MemWr <= '1' AFTER MemWr_delay, '0' AFTER MemWr_pulse;
        Sin_Sout <= '1' AFTER Ctrl_delay;
    END IF;
END CASE;
ENDIF;
IF (Instr’EVENT) THEN
   -- decode opcode
   Ctrl_Line := Ctrl_Table(CONV_INTEGER(’0’ & Opcode));

   -- decode registers
   AddrA   <= Instr(20 DOWNTO 16) AFTER Dec_delay;
   IF (Ctrl_Line.RS_RB = ’0’) THEN
      AddrB <= Instr(25 DOWNTO 21) AFTER Dec_delay;
   ELSE
      AddrB <= Instr(15 DOWNTO 11) AFTER Dec_delay;
   END IF;
   AddrC   <= Instr(25 DOWNTO 21) AFTER Dec_delay;

   -- decode control signals
   PC_RA   <= not(Ctrl_Line.WrPC) AFTER Ctrl_delay;
   ZE_SE   <= Ctrl_Line.ZE_SE AFTER Ctrl_delay;
   IR_RB   <= Ctrl_Line.IR_RB AFTER Ctrl_delay;
   ALUOp   <= Ctrl_Line.ALUop AFTER Ctrl_delay;
   WrPC    <= Ctrl_Line.WrPC AFTER Ctrl_delay;
   WrCR    <= Ctrl_Line.WrCR AFTER Ctrl_delay;

   IF (Ctrl_Line.MemOp = ’0’) THEN
      WrC    <= Ctrl_Line.WrC AFTER Ctrl_delay;
      Mem_ALU <= ’1’ AFTER Ctrl_delay;
   END IF;
END IF;
IF (MemRdy’EVENT AND MemRdy='1') THEN
    CASE State IS
        WHEN memop => IF (Ctrl_Line.WrMem = '1') THEN
            -- deactivate data bus
            Sin_Sout <= '0' AFTER Ctrl_delay;
            END IF;
        WHEN OTHERS => NULL;
        END CASE;
    END IF;
END PROCESS;
END behavioral;