Solution to Assignment 3
RTL Design

Question 1

The `cputypes` Package

This package defines the types and constants that are used throughout the design.

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

package cputypes is

  -- memory words
  subtype dword is unsigned (7 downto 0) ;
  subtype iword is std_logic_vector (7 downto 0) ;

  -- addresses (5-bit)
  subtype addr is unsigned (4 downto 0) ;

  -- opcodes and codes (cpu and alu use same opcodes)
  subtype     opcode is std_logic_vector (2 downto 0) ;
  subtype alu_opcode is std_logic_vector (2 downto 0) ;
  subtype  pc_opcode is std_logic_vector (1 downto 0) ;

  constant load  : opcode := "000" ;
  constant store : opcode := "001" ;
  constant loadi : opcode := "010" ;
  constant addop : opcode := "011" ;
  constant notop : opcode := "100" ;
  constant andop : opcode := "101" ;
  constant jz    : opcode := "110" ;
  constant jn    : opcode := "111" ;

  constant hold   : pc_opcode := "00" ;
  constant incr   : pc_opcode := "01" ;
  constant jump   : pc_opcode := "10" ;

end cputypes;
```

Instruction Memory ROM

```
-- ELEC 379 Assignment 3 Solutions
-- Ed Casas, October 9, 1998
-- Instruction Memory ROM
-- size: 32x8
-- contents: sample program

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;

use work.cputypes.all ;

entity rom is
        port (
        address : in addr ;
        instr : out iword
        ) ;
end rom ;

architecture rtl of rom is
begin
        with conv_integer(address) select instr <=
                "01000000" when 0,   --        LOADI   0
                "00100000" when 1,   --        STORE   0
                "01000001" when 2,   --        LOADI   1
                "00100001" when 3,   --        STORE   1
                "01000010" when 4,   --        LOADI   2
                "10000000" when 5,   --        NOT
                "01100001" when 6,   --        ADD     1
                "11100110" when 7,   --        JN      6
                "11001000" when 8,   --        JZ      8
                "00000000" when others ;
end rtl ;
```

The select expression should have been a 5-bit-wide type instead of integer to ensure that the synthesizer did not generate 32-bit logic (which is the normal width for the integer type). Figure 1 shows the simulation results.

Data Memory RAM

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use work.cputypes.all ;

entity ram is
        port (
        din : in dword ;
        a : in addr ;
        write, clk : in std_logic ;
        d_out : out dword
        ) ;
end ram ;

architecture rtl of ram is
        type dataarray is array (31 downto 0) of dword ;
        signal ramarray : dataarray ;
        signal d, nextd : dword ;
begin
        -- output value is the indexed array element
        d <= ramarray (conv_integer(unsigned(a))) ;
```

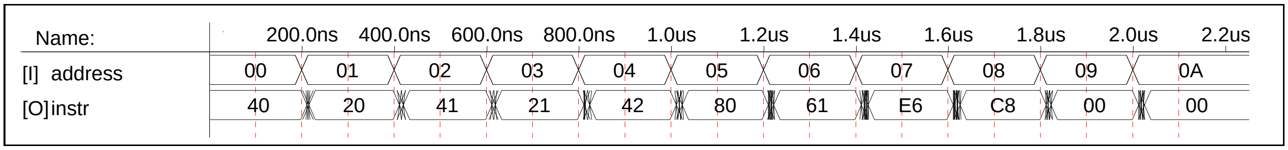

Figure 1: Simulation Results for Instruction ROM.

```
        -- next value of the indexed array element
        nextd <= din when write = '1' else d ;

        -- register the indexed array element
        process(clk)
        begin
                if clk'event and clk = '1' then
                        ramarray(conv_integer(unsigned(a)))
                                 <= nextd ;
                end if ;
        end process ;

        d_out <= d ;
end rtl ;
```

Figure 2 shows the simulation results.

ALU

The accumulator register datapath is often called the ALU (arithmetic and logic unit).

```
-- ELEC 379 Assignment 3 Solutions
-- Ed Casas, October 9, 1998
-- ALU Datapath

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;

use work.cputypes.all ;

entity alu is
        port (
        d : in dword ;          -- addressed RAM data
        ia : in addr ;          -- address field
        op : in alu_opcode ;    -- alu operation
        clk : in std_logic ;    -- clock
        a_out : out dword ;     -- current accumulator
        zero, negative : out std_logic  -- result flags
        ) ;
end alu ;

architecture rtl of alu is
        signal a, nexta : dword ;
begin
        -- ALU operations
        with op select nexta <=
                d                               when load,
                unsigned("000" & ia)            when loadi,
                d + a                           when addop,
                unsigned(not
                        std_logic_vector(a))    when notop,
                unsigned(std_logic_vector(d) and
                        std_logic_vector(a))    when andop,
                a                               when others ;

        -- zero and negative flags from accumulator
        zero <=
                '1' when a = 0 else
                '0' ;

        negative <= a(7) ;

        -- accumulator register
        process(clk)
        begin
                if clk'event and clk = '1' then
                        a <= nexta ;
                end if ;
        end process ;

        a_out <= a ;
end rtl ;
```

Figure 3 shows the simulation results.

Program Counter

```
-- ELEC 379 Assignment 3 Solutions
-- Ed Casas, October 9, 1998
-- PC Datapath
-- implements PC load, hold, increment and reset

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;

use work.cputypes.all ;

entity pc is
        port (
        ia : in addr ;          -- instruction address
        op : in pc_opcode ;     -- opcode
        reset : in std_logic ;
        clk : in std_logic ;
        pc_out : out addr       -- current program counter
        ) ;
end pc ;

architecture rtl of pc is
        signal pc, nextpc, nextaddr : addr ;
begin
        -- next PC value if not reset
        with op select nextaddr <=
                pc + 1  when incr,
                ia      when jump,
```

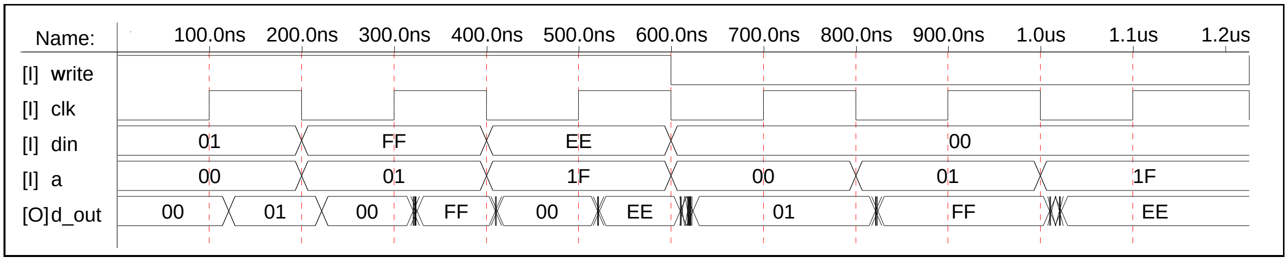

Figure 2: Simulation Results for Data RAM.

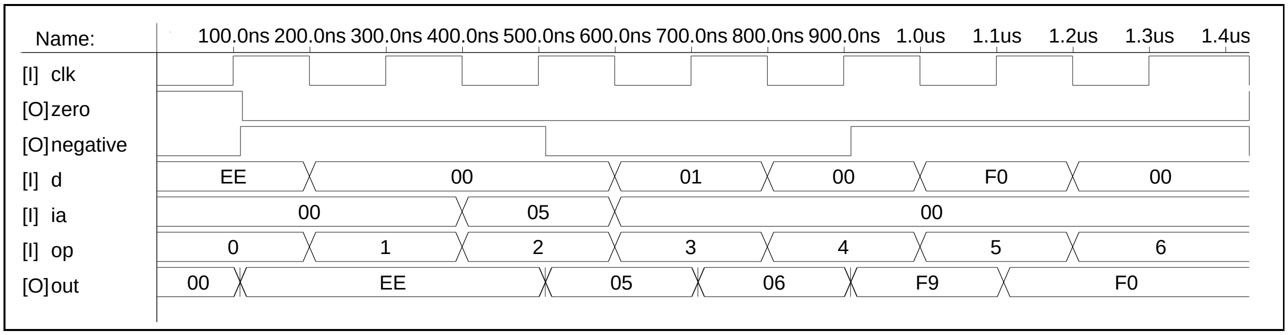

Figure 3: Simulation Results for Accumulator Datapath (ALU).

```
                pc      when others ;

        -- next PC value
        nextpc <=
                conv_unsigned(0,addr'length) when
                        reset = '1' else
                nextaddr ;

        -- register PC
        process(clk)
        begin
                if clk'event and clk = '1' then
                        pc <= nextpc ;
                end if ;
        end process ;

        pc_out <= pc ;
end rtl ;
```

Figure 4 shows the simulation results.

Instruction Decoder

```
-- ELEC 379 Assignment 3 Solutions
-- Ed Casas, October 9, 1998
-- Controller (Instruction Decoder)
-- Executes one instruction per clock cycle

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;

use work.cputypes.all ;

entity decoder is
        port (
        instr : in iword ;                  -- instruction
        zero, negative : in std_logic ; -- zero/neg. flags
        aluop : out opcode ;                -- ALU operation
        pcop : out pc_opcode ;              -- PC operation
        write : out std_logic               -- RAM write
        ) ;
end decoder ;

architecture rtl of decoder is
        signal op : opcode ;
begin

        -- extract opcode field
        op <= instr(7 downto 5) ;

        -- ALU opcode is same as instruction opcode
        aluop <= op ;

        -- with single-cycle instruction execution the
        -- PC opcode is either load (for branch) or
        -- increment
        pcop <=
                jump when ( op = jz and zero = '1' ) else
                jump when ( op = jn and negative = '1' ) else
                incr ;

        -- write only active for 'store'
        write <= '1' when op = store else '0' ;

end rtl ;
```

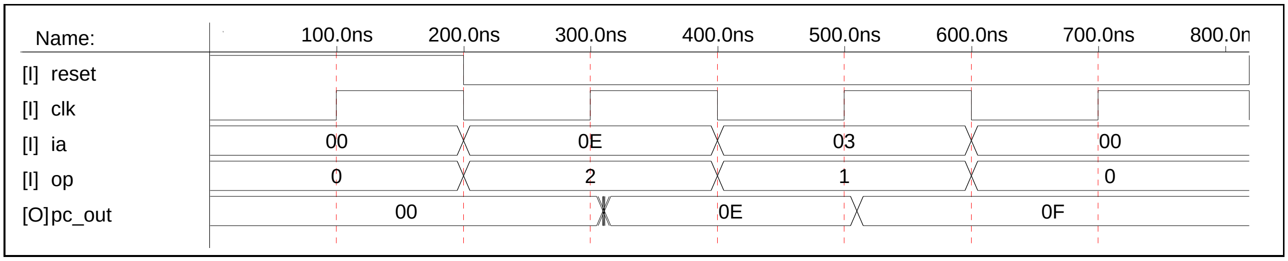

Figure 4: Simulation Results for Program Counter Datapath.

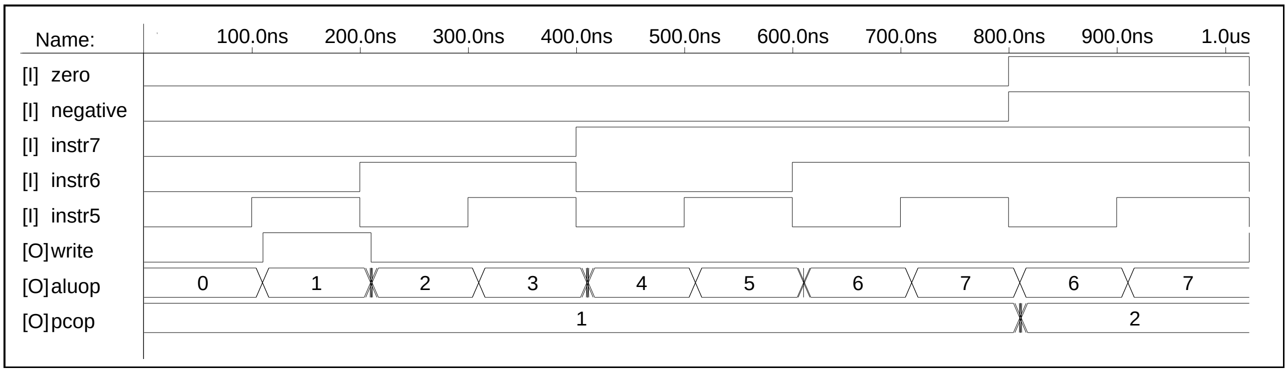

Figure 5: Simulation Results for Instruction Decoder.

Figure 5 shows the simulation results.

The `cpucomponents` Package

This package declares components for the above entities so they can be instantiated in the top-level architecture.

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

use work.cputypes.all ;

package cpucomponents is

-- RAM

component ram
        port (
        din : in dword ;
        a : in addr ;
        write, clk : in std_logic ;
        d_out : out dword
        ) ;
end component ;

-- ROM

component rom
        port (
        address : in addr ;
        instr : out iword
        ) ;
end component ;

-- alu

component alu
        port (
        d : in dword ;          -- addressed RAM data
        ia : in addr ;          -- instruction address
        op : in alu_opcode ;    -- alu operation (=opcode)
        clk : in std_logic ;    -- clock
        a_out : out dword ;     -- current accumulator
        zero, negative : out std_logic  -- flags
        ) ;
end component ;

-- pc

component pc
        port (
        ia : in addr ;          -- instruction address
        op : in pc_opcode ;     -- opcode
        reset : in std_logic ;
        clk : in std_logic ;
        pc_out : out addr       -- current program counter
        ) ;
end component ;

-- decoder
```

```
component decoder
        port (
        instr : in iword ;              -- instruction
        zero, negative : in std_logic ; -- flags
        aluop : out opcode ;            -- ALU operation
        pcop : out pc_opcode ;          -- PC operation
        write : out std_logic           -- RAM write
        ) ;
end component ;

end cpucomponents;
```

Computer

This is the top level of the design. It instantiates the above entities.

```
-- ELEC 379 Assignment 3 Solutions
-- Ed Casas, October 9, 1998
-- Simple Computer Assignment (top-level)

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;

use work.cputypes.all ;
use work.cpucomponents.all ;

entity cpu is
        port (
        reset, clk : in std_logic ;     -- reset / clock
        pc_out : out addr ;             -- PC (for test)
        instr_out : out iword ;         -- instruction (")
        acc_out : out dword             -- accumulator (")
        ) ;
end cpu ;

architecture rtl of cpu is
        signal ip : addr ;              -- from PC
        signal instr : iword ;          -- from ROM
        signal ramout : dword ;         -- from RAM
        signal acc : dword ;            -- from ALU
        signal zero, negative : std_logic ;
        signal aluop : alu_opcode ;     -- from controller
        signal pcop :   pc_opcode ;
        signal write : std_logic ;

        signal add : addr ;             -- address field
begin
        -- extract address field from current instruction
        add <= unsigned(instr(4 downto 0)) ;

        -- computer components
        pc1:  pc     port map ( add, pcop, reset, clk, ip ) ;

        rom1: rom     port map ( ip, instr ) ;

        ram1: ram     port map ( acc, add, write, clk, ramout ) ;

        alu1: alu     port map ( ramout, add, aluop, clk, acc,
                zero, negative) ;

        decoder1:
              decoder port map ( instr, zero, negative,
                aluop, pcop, write ) ;

        -- test outputs
        pc_out <= ip ;
        instr_out <= instr ;
        acc_out <= acc ;
end rtl ;
```

Figure 6 shows the simulation results.

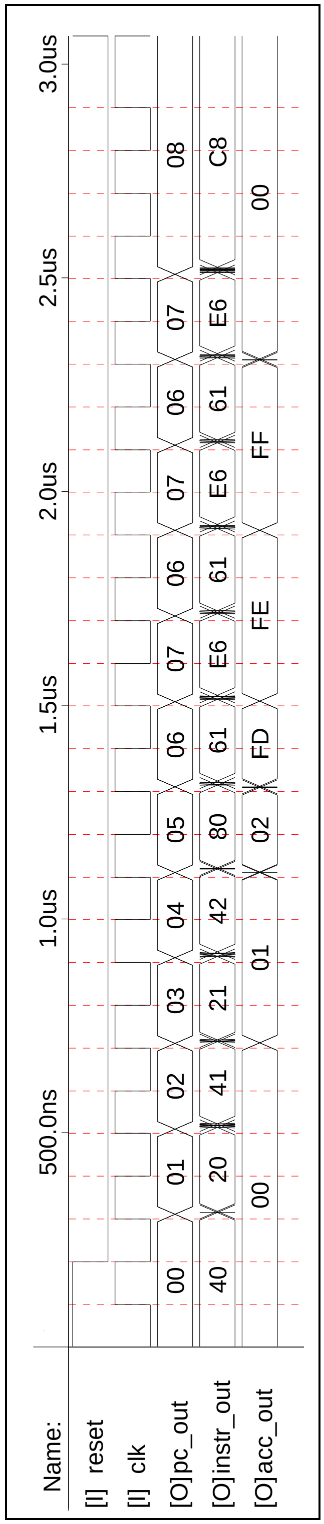

Figure 6: Simulated Execution of Sample Program.