Solution to Assignment 3
RTL Design

Question 1

The `cputypes` Package

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

```
-- ELEC 379 Assignment 3 Solutions
-- Ed Casas, March 2, 1999
-- CPU Types

library ieee ;
use ieee.std_logic_1164.all ;

package cputypes is

  -- memory words
  subtype word is std_logic_vector (7 downto 0) ;

  -- addresses
  subtype address is std_logic_vector (4 downto 0) ;

  -- opcodes
  subtype opcode is std_logic_vector (2 downto 0) ;

  constant load_op  : opcode := "000" ;
  constant store_op : opcode := "001" ;
  constant add_op   : opcode := "010" ;
  constant sub_op   : opcode := "011" ;
  constant jmp_op   : opcode := "100" ;
  constant jnz_op   : opcode := "101" ;
  constant jn_op    : opcode := "110" ;

  -- datapath operations
  type alu_opcode is ( hold, load, add, sub ) ;
  type   i_opcode is ( hold, load ) ;
  type  pc_opcode is ( hold, load, incr, clear) ;

end cputypes;
```

Memory

```
-- ELEC 379 Assignment 3 Solutions
-- Ed Casas, March 2, 1999
-- Memory

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

entity memory is
    port (
    addr : in address ;
    data_out : out word ;
    data_in : in word ;
    write, clk : in std_logic
    ) ;
end memory ;

architecture rtl of memory is
    type dataarray is array (15 downto 0) of word ;
    signal ram_data : dataarray ;
    signal ram_out, next_ram_out, rom_out : word ;
    signal ram_enable : std_logic ;
    signal addr_low : unsigned(address'left-1 downto 0) ;
begin
    -- RAM is upper half
    ram_enable <= addr(address'left) ;

    -- LS bits of address
    addr_low <= unsigned(addr(address'left-1 downto 0)) ;

    -- ROM contains the program
    with addr_low select rom_out <=
        "00000111" when "0000",
        "00110000" when "0001",
        "00000110" when "0010",
        "01110000" when "0011",
        "10100011" when "0100",
        "10000101" when "0101",
        "00000011" when "0110",
        "00000001" when "0111",
        "00000000" when others ;

    -- RAM output value is the indexed array element
    ram_out <=
        ram_data (conv_integer(addr_low)) ;

    -- next value to be stored in RAM
    next_ram_out <=
        data_in when write = '1' and ram_enable = '1' else
        ram_out ;

    -- register the indexed array element
    process(clk)
    begin
        if clk'event and clk = '1' then
           ram_data(conv_integer(addr_low)) <=
                next_ram_out ;
        end if ;
    end process ;

    data_out <=
        ram_out when ram_enable = '1' else
        rom_out ;

end rtl ;
```

Figure 1 shows the simulation results.

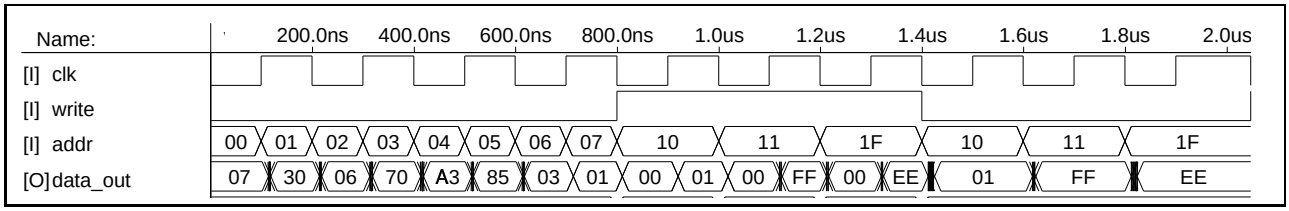

Figure 1: Simulation Results for Memory.

A Datapath

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

```
-- ELEC 379 Assignment 3 Solutions
-- Ed Casas, March 8, 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 (
    a_in : in word ;        -- addressed RAM data
    op : in alu_opcode ;    -- alu operation
    a_out : out word ;      -- current accumulator value
    zero, negative : out std_logic ;  -- flags
    clk : in std_logic      -- clock
    ) ;
end alu ;

architecture rtl of alu is
    signal a, nexta : word ;
begin
    -- ALU operations
    with op select nexta <=
        a_in                            when load,
        unsigned(a) + unsigned(a_in)    when add,
        unsigned(a) - unsigned(a_in)    when sub,
        a                               when others ;

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

    a_out <= a ;

    -- zero and negative flags from current accumulator
    zero <=
        '1' when a = "00000000" else
        '0' ;

    negative <= a(word'left) ;
end rtl ;
```

Figure 2 shows the simulation results.

Instruction Register

```
-- ELEC 379 Assignment 3 Solutions
-- Ed Casas, March 2, 1999
-- Instruction Register Datapath

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

entity iregister is
    port (
    i_in : in word ;
    op : in i_opcode ;
    i_out : out word ;
    clk : in std_logic      ) ;
end iregister ;

architecture rtl of iregister is
    signal i, nexti : word ;
begin
    -- next instruction register value if not reset
    with op select nexti <=
        i_in    when load,
        i       when others ;

    -- register I
    process(clk)
    begin
        if clk'event and clk = '1' then
            i <= nexti ;
        end if ;
    end process ;

    i_out <= i ;
end rtl ;
```

Figure 3 shows the simulation results.

Program Counter

```
-- ELEC 379 Assignment 3 Solutions
-- Ed Casas, March 2, 1999
-- PC Datapath
-- implements PC hold, load, increment and reset

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

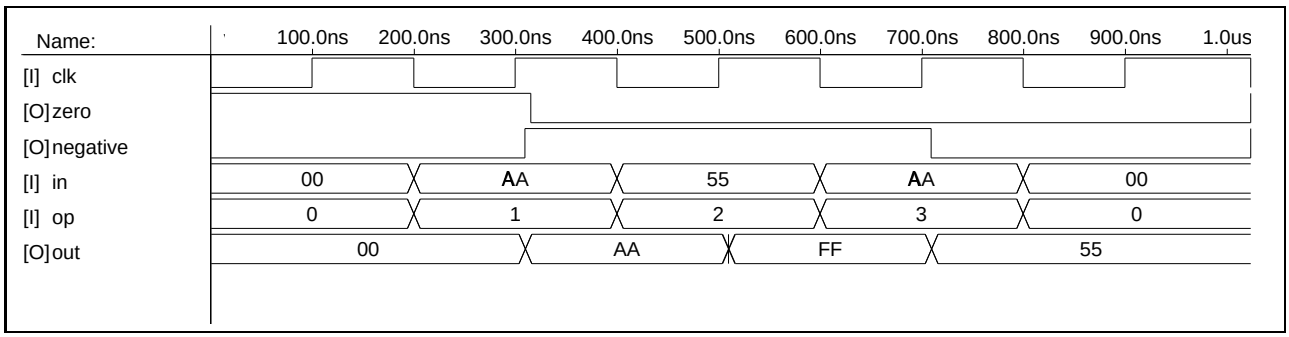

Figure 2: Simulation Results for Accumulator Datapath (ALU).

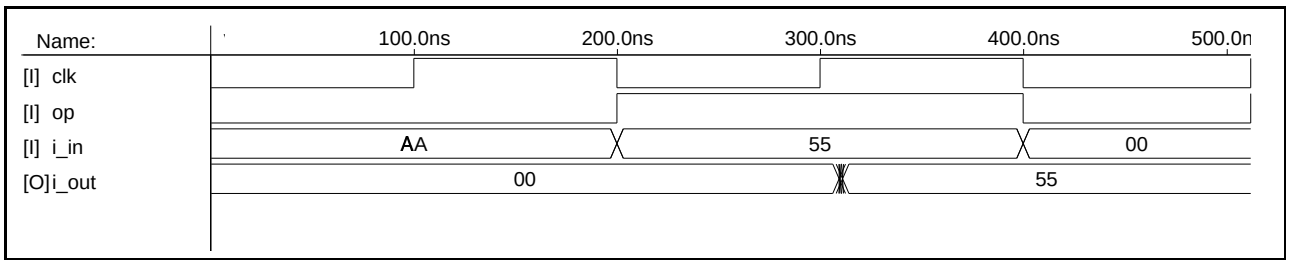

Figure 3: Simulation Results for Instruction Register Datapath.

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

architecture rtl of pc is
    signal pc, nextpc : address ;
begin
    -- next PC value if not reset
    with op select nextpc <=
        pc                      when hold,
        pc_in                   when load,
        unsigned(pc) + 1        when incr,
        "00000"                 when others ;

    -- 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, March 2, 1999
-- Controller (Instruction Decoder)
-- Each instruction takes 2 cycles (fetch, execute)

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

entity decoder is
    port (
    clk, reset : in std_logic ;     -- clock and reset
    instr : in opcode ;             -- current instruction
    zero, negative : in std_logic ; -- flags
    alu_op : out alu_opcode ;       -- A operation
    i_op : out i_opcode ;           -- I operation
    pc_op : out pc_opcode ;         -- PC operation
    fetch_out, write : out std_logic -- control
    ) ;
end decoder ;

architecture rtl of decoder is
    signal fetch, nextfetch : std_logic ;
begin

    -- fetch/execute state machine
    nextfetch <=
        '1' when reset = '1' or fetch = '0' else
        '0' ;

    process(clk)
```

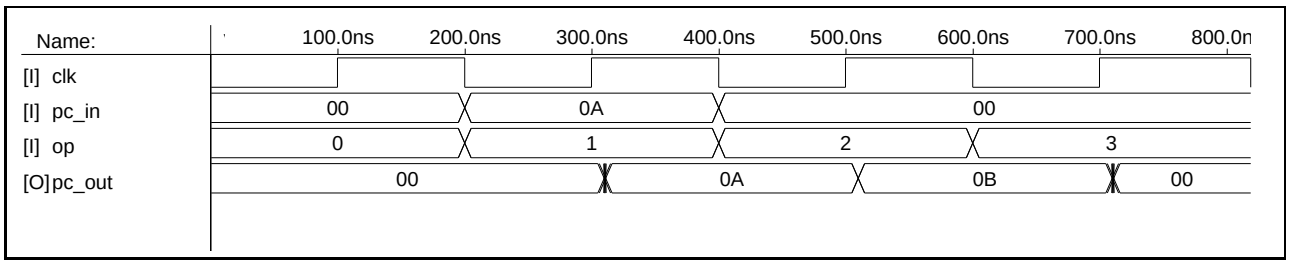

Figure 4: Simulation Results for Program Counter Datapath.

```
    begin
        if clk'event and clk='1' then
            fetch <= nextfetch ;
        end if ;
    end process ;

    fetch_out <= fetch ;

    -- ALU operation
    alu_op <=
        hold when fetch = '1' else
        load when instr = load_op else
        add  when instr = add_op else
        sub  when instr = sub_op else
        hold ;

    -- I operation
    i_op <=
        load when fetch = '1' else
        hold ;

    -- PC operation
    pc_op <=
        clear when reset = '1' else
        hold when fetch = '1' else
        load when
            instr = jmp_op or
            (instr = jnz_op and zero /= '1') or
            (instr = jn_op and negative = '1') else
        incr ;

    -- write strobe
    write <=
        '0' when fetch = '1' else
        '1' when instr = store_op else
        '0' ;

end rtl ;
```

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.

```
-- ELEC 379 Assignment 3 Solutions
-- Ed Casas, March 8, 1999

-- CPU Components

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

package cpucomponents is

-- memory

component memory
    port (
    addr : in address ;
    data_out : out word ;
    data_in : in word ;
    write, clk : in std_logic
    ) ;
end component ;

-- alu

component alu
    port (
    a_in : in word ;        -- addressed RAM data
    op : in alu_opcode ;    -- alu operation
    a_out : out word ;      -- current accumulator value
    zero, negative : out std_logic ;  -- flags
    clk : in std_logic      -- clock
    ) ;
end component ;

-- instruction register

component iregister
    port (
    i_in : in word ;
    op : in i_opcode ;
    i_out : out word ;
    clk : in std_logic ) ;
end component ;

-- pc

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

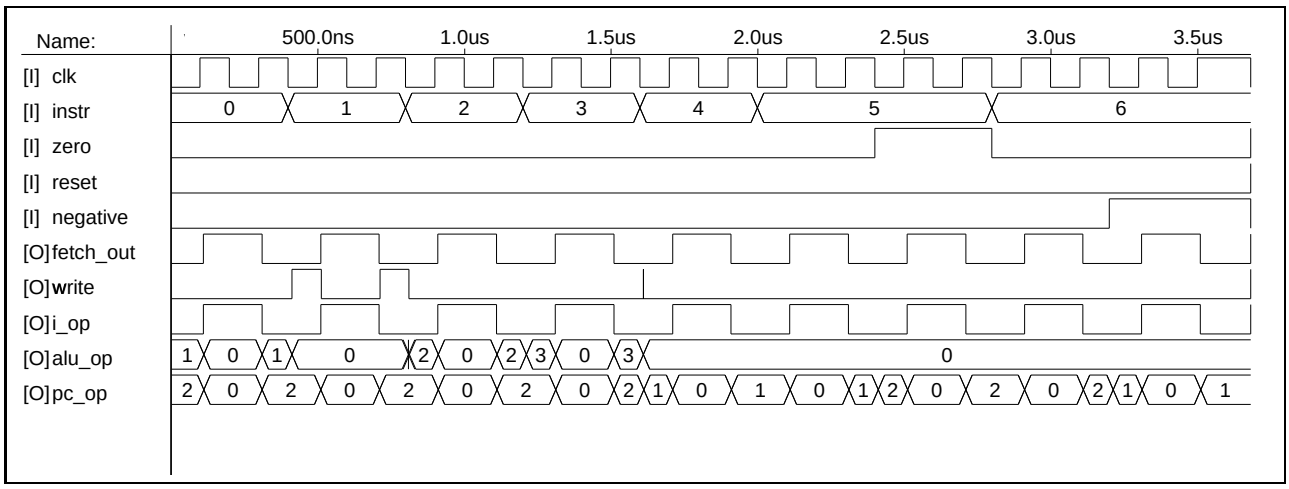

Figure 5: Simulation Results for Instruction Decoder.

```
-- decoder

component decoder
    port (
    clk, reset : in std_logic ;     -- clock and reset
    instr : in opcode ;             -- current instruction
    zero, negative : in std_logic ; -- flags
    alu_op : out alu_opcode ;       -- A operation
    i_op : out i_opcode ;           -- I operation
    pc_op : out pc_opcode ;         -- PC operation
    fetch_out, write : out std_logic -- control
    ) ;
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 an clock
    pc_out_out : out address ;  -- current PC
    mem_out_out : out word ;    -- memory output
    alu_out_out : out word      -- alu output
    ) ;
end cpu ;

architecture rtl of cpu is
    signal addr : address ; -- address into memory

    signal mem_out, alu_out, i_out : word ;
    signal pc_out : address ;
    signal alu_op : alu_opcode ;
    signal i_op : i_opcode ;
    signal pc_op : pc_opcode ;
    signal write, zero, negative, fetch : std_logic ;

    signal i_out_op : std_logic_vector (2 downto 0) ;
    signal i_out_addr : std_logic_vector (4 downto 0) ;
begin
    i_out_op <= i_out (7 downto 5) ;
    i_out_addr <= i_out (4 downto 0) ;

    addr <=
        pc_out when fetch = '1' else
        i_out_addr ;

    m1: memory port map ( addr, mem_out, alu_out,
                write, clk ) ;

    a1: alu port map ( mem_out, alu_op, alu_out,
                zero, negative, clk ) ;

    i1: iregister port map ( mem_out, i_op, i_out, clk ) ;

    p1: pc port map ( i_out_addr, pc_op, pc_out, clk ) ;

    d1: decoder port map ( clk, reset, i_out_op, zero,
        negative, alu_op, i_op, pc_op, fetch, write ) ;

    -- test outputs
    pc_out_out <= pc_out ;
    mem_out_out <= mem_out ;
    alu_out_out <= alu_out ;

end rtl ;
```

Figure 6 shows the simulation results.

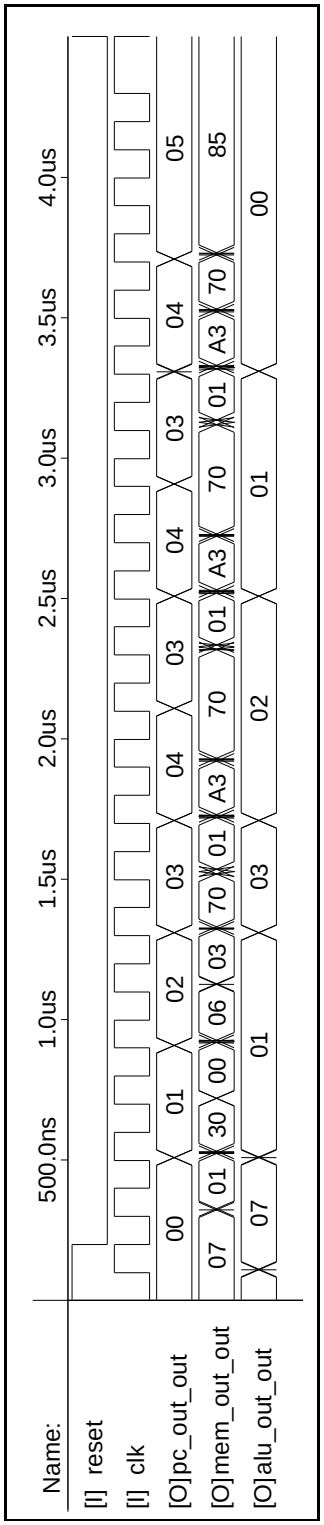

Figure 6: Simulated Execution of Sample Program.