Solution for Assignment 3

cpu_package.vhd

```
-- EECE 379 1999/2000 Term 2 Assignment 3
-- cpu_package.vhd
-- Ed Casas 2000-2-19

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

package cpu_package is

-- CPU word size
   subtype byte is unsigned (7 downto 0) ;

-- stack movement direction
   subtype direction is std_logic_vector (1 downto 0) ;

   constant up   : direction := "00" ;
   constant down : direction := "01" ;
   constant none : direction := "10" ;

-- opcode encodings (depends on student number)
   constant PUSH :   byte := "10000000" ; -- 128
   constant POP :    byte := "10000001" ; -- 129
   constant INP :    byte := "10000010" ; -- 130
   constant OUTP :   byte := "10000011" ; -- 131
   constant DUP :    byte := "10000100" ; -- 132
   constant SWAP :   byte := "10000101" ; -- 133
   constant ADD :    byte := "10000110" ; -- 134
   constant SUBT :   byte := "10000111" ; -- 135
   constant JNZ :    byte := "10001000" ; -- 136
   constant CALL :   byte := "10001001" ; -- 137
   constant RET :    byte := "10001010" ; -- 138

-- register to build LIFO stacks
   component stack_register
      port ( reg_out : out byte ;
         if_down, if_hold, if_up : in byte ;
         dir : in direction ;
         clk : in std_logic ) ;
   end component ;

   -- program ROM
   component rom
      port ( address : in byte ;
         data : out byte ) ;
   end component ;

end cpu_package ;
```

rom.vhd

```
-- EECE 379 1999/2000 Term 2 Assignment 3
-- rom.vhd
-- Ed Casas 2000-2-19

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

entity ROM is
   port ( address : in byte ;
      data : out byte ) ;
end ROM;

architecture rtl of ROM is
begin

   -- program as given in assignment
   with address select data <=
   "00000001"  when "00000000", -- START: PUSH 1
   OUTP        when "00000001",
   INP         when "00000010",
   "00001110"  when "00000011", -- PUSH DELAY
   CALL        when "00000100",
   "00000000"  when "00000101", -- PUSH 0
   OUTP        when "00000110",
   "00000101"  when "00000111", -- PUSH 5
   INP         when "00001000",
   SUBT        when "00001001",
   "00001110"  when "00001010", -- PUSH DELAY
   CALL        when "00001011",
   "00000000"  when "00001100", -- PUSH START
   RET         when "00001101",
   SWAP        when "00001110", -- DELAY:
   "00000001"  when "00001111", -- DELAY1: PUSH 1
   SUBT        when "00010000",
   "00001111"  when "00010001", -- PUSH DELAY1
   JNZ         when "00010010",
   POP         when "00010011",
   RET         when "00010100",
   "00000000"  when others ;
end rtl;
```

stack_register.vhd

```
-- EECE 379 1999/2000 Term 2 Assignment 3
-- stack_register.vhd
-- Ed Casas 2000-2-19

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

entity stack_register is
   port ( reg_out : out byte ;
      if_down, if_hold, if_up : in byte ;
      dir : in direction ;
      clk : in std_logic ) ;
```

```
end stack_register ;

architecture rtl of stack_register is
   signal reg, next_reg : byte ;
begin

   -- load register with input selected by stack
   -- direction
   with dir select next_reg <=
      if_down when down,
      if_up   when up,
      if_hold when others ;

   process(clk)
   begin
      if clk'event and clk = '1' then
         reg <= next_reg ;
      end if ;
   end process ;

   reg_out <= reg ;
end rtl ;
```

cpu.vhd

```
-- EECE 379 1999/2000 Term 2 Assignment 3
-- cpu.vhd
-- Ed Casas 2000-2-19

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

entity CPU is
   port ( I : in byte ;
      O_out, PC_out, X_out : out byte ;
      reset, clk : in std_logic ) ;
end CPU ;

architecture rtl of CPU is
   signal x, next_x : byte ;
   signal y, next_y : byte ;
   signal a, b : byte ;
   signal o, next_o : byte ;
   signal pc, next_pc, pc_plus_1 : byte ;
   signal instruction, rom_out : byte ;
   signal dir : direction ;
begin
   -- program memory and instruction
   r1: rom port map ( pc, rom_out ) ;

   instruction <=
      PUSH when rom_out(7) = '0' else
      rom_out ;

   -- x and y register inputs
   with instruction select next_x <=
      rom_out     when PUSH ,
      y           when POP | OUTP | SWAP | JNZ | RET ,
      i           when INP ,
      y+x         when ADD ,
      y-x         when SUBT ,
      pc_plus_1   when CALL ,
      x           when others ;   --   DUP and no-op

   with instruction select next_y <=
      x     when SWAP ,
      y     when others ;

   -- stack registers
   with instruction select dir <=
      down when PUSH | INP  | DUP ,
      up   when POP  | OUTP | ADD | SUBT | JNZ | RET ,
      none when others ;

   s0: stack_register port map (x, next_x, next_x, next_x, dir, clk) ;
   s1: stack_register port map (y, x,      next_y, a,      dir, clk) ;
   s2: stack_register port map (a, y,      a,      b,      dir, clk) ;
   s3: stack_register port map (b, a,      b,      b,      dir, clk) ;

   -- PC and O[utput] registers
   next_pc <=
      conv_unsigned(0,byte'length) when reset = '1' else
      x when instruction = CALL or instruction = RET else
      x when instruction = JNZ and y /= 0 else
      pc_plus_1 ;

   next_o <=
      x when instruction = OUTP else
      o ;

   pc_plus_1 <= pc + 1 ;

   process(clk)
   begin
    if clk'event and clk='1' then
    pc <= next_pc ;
    o  <= next_o ;
    end if ;
   end process ;

   -- connect output ports
    o_out <= o ;
   pc_out <= pc ;
    x_out <= x ;

end rtl ;
```

The `rom` and `stack_register` simulation results are shown in Figures 1 and 2. The simulation results for the two `cpu` test cases are shown in Figures 3 through 6.

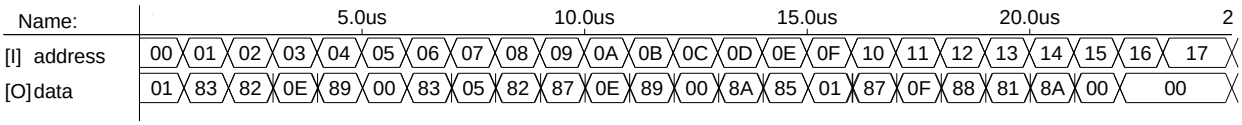

Figure 1: ROM contents.

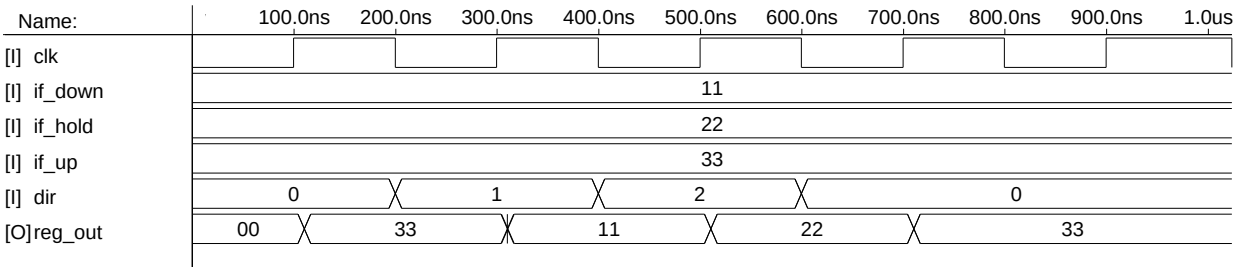

Figure 2: Pop, push and hold.

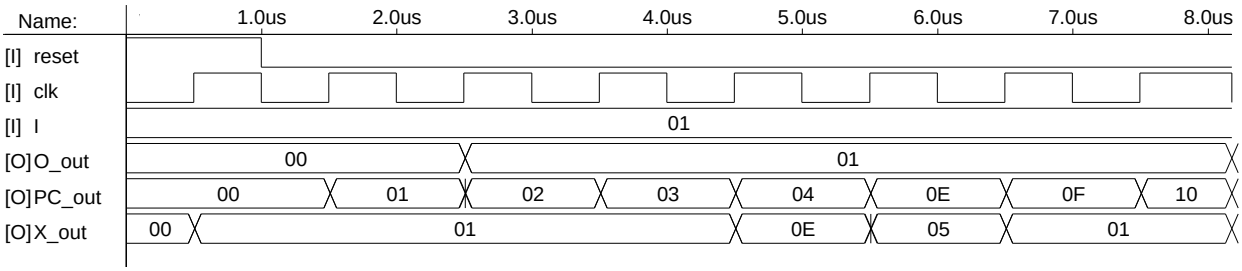

Figure 3: I=1, first 8 clock cycles.

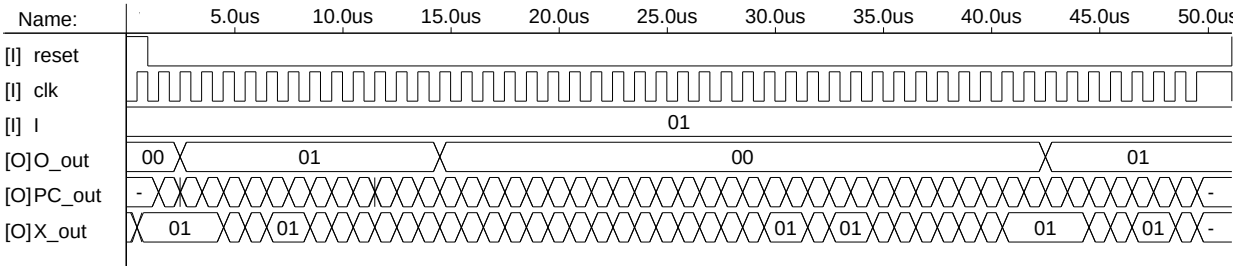

Figure 4: I=1, first 50 clock cycles.

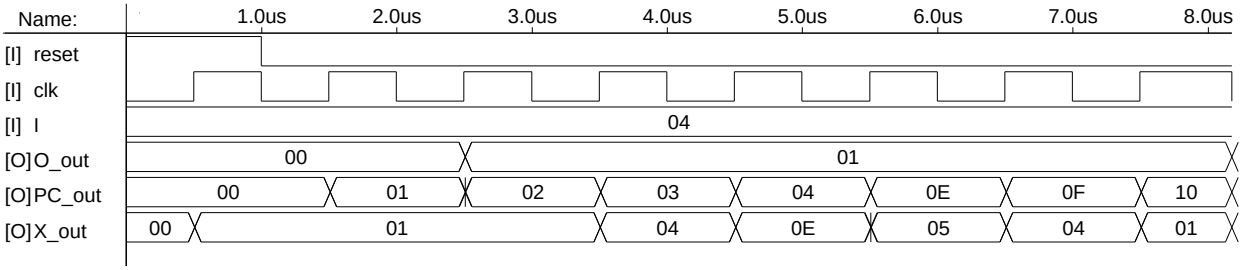

Figure 5: I=4, first 8 clock cycles.

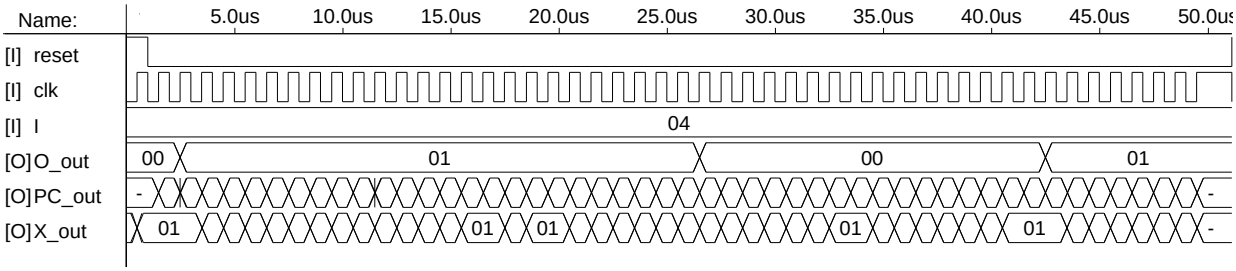

Figure 6: I=4, first 50 clock cycles.