-- cpu.vhd: Simple 8-bit CPU (BrainLove interpreter)

-- Copyright (C) 2021 Brno University of Technology,

-- Faculty of Information Technology

-- Author(s): Tomáš Souček (xsouce15)

--

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

-- ----------------------------------------------------------------------------

-- Entity declaration

-- ----------------------------------------------------------------------------

entity cpu is

port (

CLK : in std\_logic; -- hodinovy signal

RESET : in std\_logic; -- asynchronni reset procesoru

EN : in std\_logic; -- povoleni cinnosti procesoru

-- synchronni pamet ROM

CODE\_ADDR : out std\_logic\_vector(11 downto 0); -- adresa do pameti

CODE\_DATA : in std\_logic\_vector(7 downto 0); -- CODE\_DATA <- rom[CODE\_ADDR] pokud CODE\_EN='1'

CODE\_EN : out std\_logic; -- povoleni cinnosti

-- synchronni pamet RAM

DATA\_ADDR : out std\_logic\_vector(9 downto 0); -- adresa do pameti

DATA\_WDATA : out std\_logic\_vector(7 downto 0); -- ram[DATA\_ADDR] <- DATA\_WDATA pokud DATA\_EN='1'

DATA\_RDATA : in std\_logic\_vector(7 downto 0); -- DATA\_RDATA <- ram[DATA\_ADDR] pokud DATA\_EN='1'

DATA\_WREN : out std\_logic; -- cteni z pameti (DATA\_WREN='0') / zapis do pameti (DATA\_WREN='1')

DATA\_EN : out std\_logic; -- povoleni cinnosti

-- vstupni port

IN\_DATA : in std\_logic\_vector(7 downto 0); -- IN\_DATA obsahuje stisknuty znak klavesnice pokud IN\_VLD='1' a IN\_REQ='1'

IN\_VLD : in std\_logic; -- data platna pokud IN\_VLD='1'

IN\_REQ : out std\_logic; -- pozadavek na vstup dat z klavesnice

-- vystupni port

OUT\_DATA : out std\_logic\_vector(7 downto 0); -- zapisovana data

OUT\_BUSY : in std\_logic; -- pokud OUT\_BUSY='1', LCD je zaneprazdnen, nelze zapisovat, OUT\_WREN musi byt '0'

OUT\_WREN : out std\_logic -- LCD <- OUT\_DATA pokud OUT\_WE='1' a OUT\_BUSY='0'

);

end cpu;

-- ----------------------------------------------------------------------------

-- Architecture declaration

-- ----------------------------------------------------------------------------

architecture behavioral of cpu is

-- PC

signal pc : std\_logic\_vector(11 downto 0);

signal pcInc : std\_logic;

signal pcDec : std\_logic;

-- CNT

signal cnt : std\_logic\_vector(11 downto 0);

signal cntInc : std\_logic;

signal cntDec : std\_logic;

-- PTR

signal ptr : std\_logic\_vector(9 downto 0);

signal ptrInc : std\_logic;

signal ptrDec : std\_logic;

-- MUX

-- 00...IN\_DATA || 01...DATA\_RDATA-1 || 10...DATA\_RDATA+1

signal mux : std\_logic\_vector(7 downto 0);

signal muxSel : std\_logic\_vector(1 downto 0) := "00";

-- FSM stavy

type fsmState is (

sInit,

sFetch,

sDecode,

sPtrInc, -- >

sPtrDec, -- <

sValueInc, sValueInc1, sValueInc2, -- +

sValueDec0, sValueDec1, sValueDec2, -- -

sLoop0, sLoop1, sLoop2, sLoopEn, -- [

sLoopEnd0, sLoopEnd1, sLoopEnd2, sLoopEnd3, sLoopEndEn, -- ]

sWrite0, sWrite1, -- .

sRead0, sRead1, -- ,

sBreak0, sBreak1, sBreakEn, -- ~

sNull -- null

);

signal pState : fsmState := sInit;

signal nState : fsmState;

begin

-- PC

process (CLK, RESET, pcInc, pcDec)

begin

if RESET = '1' then

pc <= (others => '0');

elsif rising\_edge(CLK) then

if pcInc = '1' then

pc <= pc + 1;

elsif pcDec = '1' then

pc <= pc - 1;

end if;

end if;

end process;

CODE\_ADDR <= pc;

-- PTR

process (CLK, RESET, ptrInc, ptrDec)

begin

if RESET = '1' then

ptr <= (others => '0');

elsif rising\_edge(CLK) then

if ptrInc = '1' then

ptr <= ptr + 1;

elsif ptrDec = '1' then

ptr <= ptr - 1;

end if;

end if;

end process;

DATA\_ADDR <= ptr;

-- CNT

process (CLK, RESET, cntInc, cntDec)

begin

if RESET = '1' then

cnt <= (others => '0');

elsif rising\_edge(CLK) then

if cntInc = '1' then

cnt <= cnt + 1;

elsif cntDec = '1' then

cnt <= cnt - 1;

end if;

end if;

end process;

OUT\_DATA <= DATA\_RDATA;

-- MUX

process (CLK, RESET, muxSel)

begin

if RESET = '1' then

mux <= (others => '0');

elsif rising\_edge(CLK) then

case muxSel is

when "00" =>

mux <= IN\_DATA;

when "01" =>

mux <= DATA\_RDATA - 1;

when "10" =>

mux <= DATA\_RDATA + 1;

when others =>

mux <= DATA\_RDATA;

end case;

end if;

end process;

DATA\_WDATA <= mux;

-- FSM

-- aktualni stav

process(RESET, CLK, EN)

begin

if (RESET='1') then

pstate <= SInit;

elsif rising\_edge(CLK) then

if (EN='1') then

pstate <= nstate;

end if;

end if;

end process;

-- nasledujici stav

process(pstate, IN\_VLD, OUT\_BUSY, cnt, CODE\_DATA, DATA\_RDATA)

begin

-- inicializace

OUT\_WREN <= '0';

DATA\_WREN <= '0';

IN\_REQ <= '0';

CODE\_EN <= '0';

DATA\_EN <= '0';

pcInc <= '0';

pcDec <= '0';

cntInc <= '0';

cntDec <= '0';

ptrInc <= '0';

ptrDec <= '0';

muxSel <= "00";

case pState is

when sInit =>

nState <= sFetch;

--------------------------

when sFetch =>

CODE\_EN <= '1';

nState <= sDecode;

--------------------------

when sDecode =>

case CODE\_DATA is

when X"3E" =>

nState <= sPtrInc; -- >

when X"3C" =>

nState <= sPtrDec; -- <

when X"2B" =>

nState <= sValueInc; -- +

when X"2D" =>

nState <= sValueDec0; -- -

when X"5B" =>

nState <= sLoop0; -- [

when X"5D" =>

nState <= sLoopEnd0; -- ]

when X"2E" =>

nState <= sWrite0; -- .

when X"2C" =>

nState <= sRead0; -- ,

when X"7E" =>

nState <= sBreak0; -- ~

when X"00" =>

nState <= sNull; -- null

when others =>

pcInc <= '1';

nState <= sFetch;

end case;

--------------------------

when sPtrInc =>

ptrInc <= '1';

pcInc <= '1';

nState <= sFetch;

--------------------------

when sPtrDec =>

ptrDec <= '1';

pcInc <= '1';

nState <= sFetch;

--------------------------

when sValueInc =>

DATA\_EN <= '1';

DATA\_WREN <= '0';

nState <= sValueInc1;

when sValueInc1 =>

muxSel <= "10";

nState <= sValueInc2;

when sValueInc2 =>

DATA\_EN <= '1';

DATA\_WREN <= '1';

pcInc <= '1';

nState <= sFetch;

--------------------------

when sValueDec0 =>

DATA\_EN <= '1';

DATA\_WREN <= '0';

nState <= sValueDec1;

when sValueDec1 =>

muxSel <= "01";

nState <= sValueDec2;

when sValueDec2 =>

DATA\_EN <= '1';

DATA\_WREN <= '1';

pcInc <= '1';

nState <= sFetch;

--------------------------

when sLoop0 =>

DATA\_EN <= '1'; -- povoleni cinnosti

DATA\_WREN <= '0'; -- nastaveni na cteni

pcInc <= '1';

nstate <= sLoop1;

when sLoop1 =>

if DATA\_RDATA = (DATA\_RDATA'range => '0') then

cntInc <= '1';

if cnt = (cnt'range => '0') then

nState <= sFetch;

else

nState <= sLoopEn;

end if;

else

nState <= sFetch;

end if;

when sLoop2 =>

if cnt = (cnt'range => '0') then

nState <= sFetch;

else

if CODE\_DATA = X"5B" then -- [

cntInc <= '1';

elsif CODE\_DATA = X"5D" then -- ]

cntDec <= '1';

end if;

pcInc <= '1';

nState <= sLoopEn;

end if;

when sLoopEn =>

CODE\_EN <= '1';

nState <= sLoop2;

--------------------------

when sLoopEnd0 =>

DATA\_EN <= '1';

DATA\_WREN <= '0';

nState <= sLoopEnd1;

when sLoopEnd1 =>

if DATA\_RDATA = (DATA\_RDATA'range => '0') then

pcInc <= '1';

nState <= sFetch;

else

cntInc <= '1';

pcDec <= '1';

nState <= sLoopEndEn;

end if;

when sLoopEnd2 =>

if cnt = (cnt'range => '0') then

nState <= sFetch;

else

if CODE\_DATA = X"5D" then -- ]

cntInc <= '1';

elsif CODE\_DATA = X"5B" then -- [

cntDec <= '1';

end if;

nState <= sLoopEnd3;

end if;

when sLoopEnd3 =>

if cnt = (cnt'range => '0') then

pcInc <= '1';

else

pcDec <= '1';

end if;

nState <= sLoopEndEn;

when sLoopEndEn =>

CODE\_EN <= '1';

nState <= sLoopEnd2;

--------------------------

when sWrite0 =>

DATA\_EN <= '1';

DATA\_WREN <= '0';

nState <= sWrite1;

when sWrite1 =>

if OUT\_BUSY = '1' then

DATA\_EN <= '1';

DATA\_WREN <= '0';

nState <= sWrite1;

else

OUT\_WREN <= '1';

pcInc <= '1';

nState <= sFetch;

end if;

--------------------------

when sRead0 =>

IN\_REQ <= '1';

muxSel <= "00";

nState <= sRead1;

when sRead1 =>

if IN\_VLD = '1' then

-- RAM[PTR] = DATA\_WDATA

DATA\_EN <= '1';

DATA\_WREN <= '1';

pcInc <= '1';

nState <= sFetch;

else

IN\_REQ <= '1';

muxSel <= "00"; -- DATA\_WDATA = IN\_DATA

nState <= sRead1;

end if;

--------------------------

when sBreak0 =>

cntInc <= '1';

pcInc <= '1';

nState <= sBreakEn;

when sBreak1 =>

if cnt = (cnt'range => '0') then

nState <= sFetch;

else

if CODE\_DATA = X"5B" then -- pokud je na vstupu [

cntInc <= '1';

elsif CODE\_DATA = X"5D" then -- pokud je na vstupu ]

cntDec <= '1';

end if;

pcInc <= '1';

nState <= sBreakEn;

end if;

when sBreakEn =>

CODE\_EN <= '1';

nState <= sBreak1;

--------------------------

when sNull =>

nState <= sNull;

when others => null;

end case;

end process;

end behavioral;