Kpi-best

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №7**

з дисципліни «Технології проектування   
комп’ютерних систем»

на тему: «Ядро мікропроцесора»

Виконав:

студент 4-го курсу

факультету ІОТ

групи ІО-41

Демчик В. В.

НЗК 4111

Перевірив:

проф. Сергієнко А. М.

Київ 2017

**Тема:** Проектування ядра мікропроцесору.

**Мета та основні завдання** **роботи**: оволодіти знаннями і практичними навичками з проектування таких складних обчислювальних блоків з програмним управлінням, як ядро мікропроцесору (CPU). Лабораторна робота також служить для оволодіння навичками програмування і налагодження опису CPU на мові VHDL.

**Завдання на лабораторну роботу:** розробити RISC-подібне ядро мікропроцесору, використовуючи модифікації компонентів, які були розроблені під час виконання попередніх лабораторних робіт даного курсу.

**Структура розробленого ядра:**

****

**Cтруктура розробленого процесору:**

****

**Код програми:**

library IEEE;

use IEEE.std\_logic\_1164.all,IEEE.std\_logic\_signed.all, work. RISC\_lib.all;

entity ALU is

port(CLK : in std\_logic;

RST : in std\_logic;

ENA : in std\_logic; --разрешение работы

CY : in std\_logic; --флаг переноса

RSTD:in std\_logic; --сброс, синхронизированный по CLK

RD : in WORD; --первый операнд

RS : in WORD; --второй операнд

INSTR : in WORD; --команда

DALU : out WORD; --результат

CNVZ : out NIBBLE ); --вектор условий

end ALU;

architecture ALU\_SYNT of ALU is

signal INSTR\_R:WORD;

signal func :NIBBLE;

signal cop:TRIPLET;

signal c:std\_logic; --перенос -+1

begin

R\_INSTR:process(CLK,RST) --регистр команды

begin

if RST='1' then

INSTR\_R<=NOP;

elsif Rising\_edge(CLK) then

if ENA='1' and RSTD='0' then

INSTR\_R<=INSTR;

end if;

end if;

end process;

ALU\_2:process(RD,RS,func,INSTR\_R, CY,c)

variable y, a,b: std\_logic\_vector(17 downto 0);

variable carry,neg,overf,zero:std\_logic;

begin

func <= INSTR\_R(3 downto 0); --функция АЛУ

if INSTR\_R(12)='0' then --приведение операции ADDI к операции ADD

func(3)<='1'; func(0)<='0';

end if;

if func(1)='0' or INSTR\_R(12)='0' then --коррекция CY с учетом знака сложения

c<=func(0); --функции ADD, SUB

else

c<=CY xor func(0); --функции ADDC,SUBC,ADDI

end if;

b:= RD(15) & RD & c ; --присоединение бита переноса

a:= RS(15) & RS & '1';

case func is --логические функции, сдвиг, вычитание и сложение

when \AND\=> y:= b and a;

when \XOR\=> y:= b xor a;

when \SRL\=> y:= "00" & a (15 downto 0);

when \SRA\=> y:= '0' & a(15) & a (15 downto 0);

when SUB | SUBC=> y:=b - a;

when others=> y:=b + a;

end case;

carry:=y(17); -- флаг переноса

neg:=y(16); -- флаг отрицательного

overf:=y(17) xor y(16) xor RS(15) xor RD(15); -- флаг переполнения

if y(16 downto 1)=X"0000" then

zero:='1'; -- флаг нулевого результата

else

zero:= '0' ;

end if;

CNVZ<=(carry,neg,overf,zero); --вектор флагов условий

DALU<=y(16 downto 1); --результат АЛУ

end process;

end ALU\_SYNT;

library IEEE;

use IEEE.std\_logic\_1164.all, IEEE.std\_logic\_UNSIGNED.all,work. RISC\_lib.all;

entity DRAM is

port(CLK : in std\_logic;

RST : in std\_logic;

RE: in std\_logic; --разрешение чтения

WE: in std\_logic; --разрешение записи

ADDRD : in WORD; --адрес данного

DATAI: in WORD; --входное данное

RDY:out std\_logic; -- готовность памяти

DATAO : out WORD ); --выходное данное

end DRAM;

architecture DRAM\_BEH of DRAM is

signal address : natural;

signal dataoi,dataoii:WORD;

signal rdyi:std\_logic;

begin

RAMD: process

variable DRAM:DMEM\_ARR;

variable addr:natural;

begin

Load\_F(IDATA\_FILE,DRAM); --загрузка памяти из файла

loop

addr:= Conv\_Integer(To\_X01(ADDRD));

if addr>DATA\_ADDRH then -- ограничение диапазона адресов

addr:=0;

end if;

if RST='1' then -- регистр адреса

address<=0 ;

elsif

Rising\_edge(CLK) then

address<=addr;

if ( WE='1' ) then

DRAM(addr):=DATAI;

end if;

end if;

dataoi<=DRAM(address);

if end\_simulation then -- сохранение состояния памяти в конце моделирования

Store\_F(ODATA\_FILE,DRAM);

wait;

end if;

wait on CLK,RST,ADDRD ;

end loop;

end process;

-- выдача прочитанного данного, если адрес входит в диапазон

DATAOii<=dataoi when address<=DATA\_ADDRH else (others=>'Z');

RDY<='H';

end DRAM\_BEH;

library IEEE;

use IEEE.std\_logic\_1164.all, IEEE.std\_logic\_UNSIGNED.all,work. RISC\_lib.all;

--pragma translate\_off

library unisim; --библиотека с моделями ОЗУ, реализуемыми в ПЛИС

--pragma translate\_on

entity PRAM is port( CLK : in std\_logic;

RST : in std\_logic;

ENA:in std\_logic; -- разрешение работы

ADDRI : in WORD; -- адрес команды

INSTR : out WORD);--считанная команда

end PRAM;

architecture PRAM\_BEH of PRAM is --поведенческая модель памяти программ

signal address : natural;

begin

PRAM: process

variable PROM:PMEM\_ARR;

begin

Load\_F( PROG\_FILE,PROM); --загрузка памяти из файла

loop

if Rising\_edge(CLK) and (ENA='1' or ENA='H') then

address<= Conv\_Integer(ADDRI); --запоминание адреса

end if;

INSTR<=PROM(address); -- чтение команды

wait on CLK,RST,ADDRI,address ;

end loop;

end process;

end PRAM\_BEH;

library IEEE;

use IEEE.std\_logic\_1164.all, IEEE.std\_logic\_UNSIGNED.all,IEEE.std\_logic\_arith.all;

use work. RISC\_lib.all;

entity INT\_CNTRL is port( CLK : in std\_logic;

RST : in std\_logic;

INTCALL: in std\_logic; --=1 при вызове ПП прерывания

INTRET:in std\_logic; --=1 при RETI

WE:in std\_logic; --запись в intena

ADDRD : in WORD; --адрес intena

DATAI:in WORD; --шина данных

INTREQ : in std\_logic\_vector(N\_INTSRC downto 0); --запросы на прерывание

INTAFD : out std\_logic; --требование прерывания

INTVEC : out std\_logic\_vector(N\_INTVEC downto 0));-- вектор прерывания

end INT\_CNTRL;

architecture INT\_CNTRL\_SYNT of INT\_CNTRL is

signal intena:WORD; --регистр маски

signal intreqd1,intreqd2,intreq\_edge,intreqr:std\_logic\_vector(N\_INTSRC downto 0);

signal intveci,intnum:std\_logic\_vector(N\_INTVEC downto 0);

signal intreq\_event,intbusy,intbusyd,intafdi:std\_logic;

begin

R\_INTENA:process(CLK,RST) begin --регистр разрешения прерывания

if RST='1' then

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

elsif Rising\_edge(CLK) then

if ADDRD= INTENA\_ADDR and WE='1' then

intena<=DATAI;

end if;

end if;

end process;

RR\_INT :process(CLK,RST) begin --регистры прерывания

if RST='1' then

intreqd1<=(others=>'0'); --первый уровень триггеров запроса прерывания

intreqd2<=(others=>'0'); --второй уровень триггеров запроса прерывания

intreqr<=(others=>'0'); --регистр фиксации запросов прерывания

elsif Rising\_edge(CLK) then

intreqd1<=INTREQ; --запросы прерывания, задержанные на 1 и 2 такта

intreqd2<=intreqd1;

for i in 0 to N\_INTSRC loop

if ( intreqd1(i) and not intreqd2(i)) ='1' then --фронт сигнала запроса

intreqr(i)<='1' ; --фиксация запроса прерывания

elsif i= Conv\_Integer(intveci) and INTRET='1' then

intreqr(i)<='0' ; --сброс запроса прерывания

end if;

end loop;

end if;

end process ;

PRIORITY:process(INTREQr ,intena) --приоритетный шифратор запроса прерывания

variable reqnum:natural;

begin

intreq\_event<='0';

intnum<=(others=>'1');

for i in 0 to N\_INTSRC loop -- i=0 - старший приоритет

if intreqr(i)='1' and intena(i)='1' then

intnum<=Conv\_std\_logic\_vector(i,N\_INTVEC+1);--номер запроса

intreq\_event<='1'; --событие запроса

exit;

end if;

end loop;

end process;

RTT\_INTVEC:process (CLK,RST)

begin

if RST='1' then

intveci<=(others=>'1') ; --вектор прерывания

intafdi<='0'; --триггер требования прерывания

intbusy<='0'; --триггер занятости отработкой прерывания

intbusyd<='0'; --триггер занятости отработкой прерывания, задержан

elsif Rising\_edge(CLK) then

intbusyd<=intbusy; --задержка на такт состояния занятости

if intbusy='0' and intreq\_event='1' and intafdi='0' then

intafdi<='1';

elsif INTCALL='1' then --был вызов ПП прерывания

intafdi<='0';

intbusy<='1';

elsif INTRET='1' then --был возрат из ПП прерывания

intbusy<='0';

end if;

if (intbusy='0' and intreq\_event='1' and INTCALL='0') or

(intbusy='0' and intbusyd='1') then -- задний фронт intbusy

intveci<=intnum; --новый вектор прерывания

end if;

end if;

end process;

INTVEC<=intveci;

INTAFD<=intafdi;

end INT\_CNTRL\_SYNT;

library IEEE;

use IEEE.std\_logic\_1164.all, IEEE.std\_logic\_unsigned.all, IEEE.std\_logic\_arith.all;

use work. RISC\_lib.all;

entity PC is port( CLK : in std\_logic;

RST : in std\_logic;

ENA : in std\_logic; --разрешение работы

INTAFD : in std\_logic; -- требование прерывания

INSTR : in WORD; --шина команд

RS : in WORD; --операнд RS из RRAM

CNVZ : in NIBBLE; --вектор условий из АЛУ

INTVEC : in std\_logic\_vector(N\_INTVEC downto 0); --вектор прерывания

CY : out std\_logic; --флаг переноса в АЛУ

RSTD:buffer std\_logic; --синхронизированный RST

INTCALL : out std\_logic; --флаг вызова ПП отработки прерывания

INTRET: out std\_logic; -- флаг исполнения команды RETI

ADDRI : out WORD ); -- Адрес команды

end PC;

architecture PC\_SYNT of PC is

signal INSTR\_R: WORD; --регистр команды

signal cop:TRIPLET; --код операции

signal conds : TRIPLET; --код условия

signal neg\_cond:std\_logic;--инверсия условия

signal name:NIBBLE; --код операции перехода

signal disp: BYTE; --смещение адреса

signal pc,pci: WORD; --счетчик команд

signal pcplus1:WORD; --счетчик команд + 1

signal int\_vect:WORD; --адрес ПП прерывания

signal jcond,jcondu:std\_logic; -- условие перехода

signal carry,neg,overf,zero:std\_logic; --биты условий

signal stack\_i,stack\_o:std\_logic\_vector(19 downto 0);--вход-выход стека

signal sp,spcall,spret:NIBBLE; --указатель стека

signal wstack:std\_logic; --запись в стек

signal CNVZi:NIBBLE; --вектор условий

signal no\_int:std\_logic; --запрет прерывания в данной команде

begin

T\_RST:process(CLK,RST) --Триггер синхронизации RST

begin

if RST='1' then

RSTD<='1';

elsif Rising\_edge(CLK) then

RSTD<='0';

end if;

end process;

R\_INSTR:process(CLK,RST) --Регистр команды

begin

if RST='1' then

INSTR\_R<=NOP;

elsif Rising\_edge(CLK) then

if ENA='1' and RSTD= '0' then

INSTR\_R<=INSTR;

end if;

end if;

end process;

--Выделение полей команды

no\_int<= INSTR\_R(15); --запрет прерываний

cop<= INSTR\_R(14 downto 12); --код операции

conds<=INSTR\_R(11 downto 9); --код условия перехода

neg\_cond<=INSTR\_R(8); --инверсия условия перехода

name<= INSTR\_R(3 downto 0); -- имя команды перехода

disp<= INSTR\_R(7 downto 0); --смещение адреса

int\_vect<= INT\_NULL & INTVEC & "000"; --на 1 ПП прерывания -до 8 команд

(carry,neg,overf,zero)<=CNVZi; --вектор условий перехода

MX\_COND: with conds select --Мультиплексор условия перехода

jcondu<= '1' when JUMP,

zero when EQ,

carry xor neg when LT,

zero or carry when LE,

overf when OVF ,

carry when others;

jcond<=jcondu xnor neg\_cond; -- бит условия перехода

--Регистр - счетчик команд

R\_PC:process(CLK,RST,RSTD,int\_vect,cop,name,no\_int,intafd,jcond,pc,

disp,RS,stack\_o,pcplus1,pci)

begin

pcplus1<=pc+1; -- инкремент адреса

pci<=pcplus1; --следующий адрес команды при остальных условиях

if INTAFD='1' and no\_int='0' then --переход при прерывании

pci<=int\_vect;

elsif jcond='1' then -- переход по условию перехода - истинно

if cop=BRA then -- если условный переход на РС+смещение

pci<= pc+ SXT(disp,16);

elsif cop=JMP then -- если условный переход на абсолютный адес

if name=LJMP or name=CALL then

pci<= RS; -- если длинный условный переход или вызов ПП

else

pci<= stack\_o(15 downto 0); -- если возврат из ПП

end if;

end if;

elsif RSTD='1' then

pci<=INIT\_ADDR; --адрес после синхронизированного сброса

end if;

if RST='1' then

pc<=INIT\_ADDR; --адрес после сброса

elsif Rising\_edge(CLK) then

if ENA='1' then

pc<= pci; --регистр-счетчик адреса

end if;

end if;

end process;

R\_CNVZ:process(CLK,RST) --Регистр cocтояния флагов

begin

if RST='1' then

CNVZi<="0000";

elsif Rising\_edge(CLK) then

if ENA='1' and RSTD='0' then

if jcond='1' and cop=JMP and ((name = RET)or(name = RETI)) then

CNVZi<=stack\_o(19 downto 16); --загрузка из стека при возврате из ПП

else

CNVZi<=CNVZ; -- запись флагов из АЛУ

end if;

end if;

end if;

end process;

stack\_i<=CNVZ & pcplus1; --состояние в стек

RAM\_STACK:process(CLK,sp) -- ОЗУ стека, синтезируется на RAM16X1S

type ARR is array (0 to 15) of std\_logic\_vector(19 downto 0);

variable RAM:ARR:=(others=>X"00000");

begin

stack\_o<= RAM(Conv\_Integer(sp)); -- чтение стека

if Rising\_edge(CLK) then

if wstack ='1' then

RAM(Conv\_Integer(sp)):= stack\_i; --запись в стек

end if;

end if;

end process;

TTR\_SP:process(CLK,RST) --Указатели стека и триггеры флагов

begin

if RST='1' then

spcall<= "0000"; --указатель стека для вызова ПП

spret<= "1111"; --указатель стека для возврата из ПП, на 1 меньше, чем spcall

INTRET<='0'; --флаг исполнения команды RETI

elsif Rising\_edge(CLK) then

if ENA='1' then

INTRET<='0';

if INTAFD='1' and no\_int='0' then --вызов ПП прерывания

spcall<=spcall+1 ;

spret<=spret+1 ;

elsif jcond='1' and cop=JMP and name = CALL then --вызов ПП

spcall<=spcall+1 ;

spret<=spret+1 ;

elsif jcond='1' and cop=JMP and name = RET then --возврат из ПП

spcall<=spcall-1 ;

spret<=spret-1 ;

elsif jcond='1' and cop=JMP and name = RETI then

spcall<=spcall-1 ; --возврат из ПП прерывания

spret<=spret-1 ;

INTRET <='1';

end if;

end if;

end if;

end process;

MX\_SP:process(INTAFD,no\_int,jcond,cop,name) --мультиплексор указателя стека

begin

if ( INTAFD='1' and no\_int='0')or( jcond='1' and cop=JMP and name = CALL) then

wstack<='1';

sp<=spcall ;

else

wstack<='0';

sp<=spret ;

end if;

end process;

INTCALL<='1' when INTAFD='1' and no\_int='0' else '0'; --флаг вызова ПП отработки прерывания

ADDRI<=pci;

CY<=carry;

end PC\_SYNT;

library IEEE;

use IEEE.std\_logic\_1164.all,work. RISC\_lib.all;

entity RISC\_CPU is port(CLK : in std\_logic; --синхровход

RST : in std\_logic; --сброс

INTREQ: in std\_logic\_vector(3 downto 0); --запросы прерывания

PORT0\_I: in std\_logic\_vector(15 downto 0);--входной порт

PORT0\_O: out std\_logic\_vector(15 downto 0));--выходной порт

end RISC\_CPU;

architecture RISC\_CPU\_SYNT of RISC\_CPU is

component RISC\_ST\_CORE is port(CLK :in std\_logic;

RST : in std\_logic;

ENA : in std\_logic; --готовность данных в DRAM

INSTR : in WORD; --команда из PROM

INTREQ : in std\_logic\_vector( N\_INTSRC downto 0);--запросы на прерывание

DATAI : in WORD; --даннoе из DRAM

ADDRI : out WORD; --адрес команды

RE : out std\_logic; --сигнал чтения из DRAM

WE : out std\_logic; --сигнал записи в DRAM

DATAO : out WORD; --даннoе в DRAM

ADDRD : out WORD ); --адрес данного

end component;

component DRAM is port(CLK :in std\_logic;

RST : in std\_logic;

RE: in std\_logic;

WE: in std\_logic;

ADDRD : in WORD;

DATAI: in WORD;

RDY: out std\_logic;

DATAO : out WORD );

end component ;

component PRAM is port( CLK :in std\_logic;

RST : in std\_logic;

ENA: in std\_logic;

ADDRI : in WORD;

INSTR : out WORD );

end component;

component PORT0 is port( CLK :in std\_logic;

RST : in std\_logic;

DATA\_I : in WORD; --шина данных

ADDRD : in WORD; --Шина адреса

WE: in std\_logic; --разрешение записи

RE: in std\_logic; --разрешение чтения

RDY:out std\_logic; -- готовность порта

PORT0\_I : in WORD; -- Вход порта

DATA\_O : out WORD; --шина данных

PORT0\_O : out WORD ); --Выход порта

end component;

signal rdy,re,we,ena:std\_logic;

signal instr:WORD;

signal datai,datao:WORD;

signal addri:WORD;

signal addrd:WORD;

begin

U\_CORE: RISC\_ST\_CORE port map( CLK =>CLK,

RST=>RST,

ENA =>rdy,

INSTR =>instr,

INTREQ =>INTREQ,

DATAI =>datai,

ADDRI =>addri,

RE =>re,

WE =>we,

DATAO =>datao,

ADDRD =>addrd );

U\_DRAM: DRAM port map(CLK=>CLK,

RST=>RST,

RE=>RE,

WE=>WE,

ADDRD =>addrd,

DATAI=>datao,

RDY=>rdy,

DATAO=>datai );

U\_PRAM: PRAM port map( CLK =>CLK,

RST=>RST,

ENA=>rdy,

ADDRI =>addri,

INSTR =>instr);

U\_PORT0:PORT0 port map( CLK=>CLK,

RST =>RST,

DATA\_I =>datao,

ADDRD =>addrd,

WE=>we,

RE=>re,

RDY=>rdy,

PORT0\_I=>PORT0\_I,

DATA\_O =>datai,

PORT0\_O =>PORT0\_O);

end RISC\_CPU\_SYNT;

library IEEE;

use IEEE.std\_logic\_1164.all, work. RISC\_lib.all;

entity RISC\_ST\_CORE is port(CLK : in std\_logic;

RST : in std\_logic;

ENA : in std\_logic; --готовность данных в DRAM

INSTR : in WORD; --команда из PRAM

INTREQ : in std\_logic\_vector( N\_INTSRC downto 0);--запросы на прерывание

DATAI : in WORD; --даннoе из DRAM

ADDRI : out WORD; --адрес команды

RE : out std\_logic; --сигнал чтения из DRAM

WE : out std\_logic; --сигнал записи в DRAM

DATAO : out WORD; --даннoе в DRAM

ADDRD : out WORD ); --адрес данного

end RISC\_ST\_CORE;

architecture RISC\_SYNT of RISC\_ST\_CORE is

component ALU is port(CLK : in std\_logic;

RST : in std\_logic;

ENA : in std\_logic;

CY:in std\_logic; --флаг переноса

RSTD:in std\_logic;

RD : in WORD; --первый операнд

RS : in WORD; --второй операнд

INSTR : in WORD; --команда

DALU : out WORD; --результат

CNVZ : out NIBBLE ); --вектор состояний

end component;

component PC is port( CLK : in std\_logic;

RST : in std\_logic;

ENA : in std\_logic;

INTAFD : in std\_logic; -- требование прерывания

INSTR : in WORD; --шина команд

RS : in WORD; --операнд RS

CNVZ : in NIBBLE; --вектор условий

INTVEC : in std\_logic\_vector(1 downto 0); --вектор прерывания

RSTD:buffer std\_logic;

CY : out std\_logic; --флаг переноса

INTCALL : out std\_logic; --флаг отработки прерывания

INTRET: out std\_logic; --квитирование INTAFD

ADDRI : out WORD ); --Адрес команды

end component;

component RRAM is port(CLK : in std\_logic;

RST : in std\_logic;

RSTD:in std\_logic;

ENA : in std\_logic;

DALU : in WORD; --данное из ALU

INSTR : in WORD; -- команда

DATAI : in WORD; -- данное из DRAM

RE:out std\_logic; --чтение DRAM

WE:out std\_logic; --запись DRAM

RS : out WORD; --операнд RS

RD : out WORD; --операнд RD

ADDRD : out WORD; --адрес DRAM

DATAO : out WORD); --данное в DRAM

end component;

component INT\_CNTRL is port (CLK : in std\_logic;

RST : in std\_logic;

INTCALL: in std\_logic; --=1 при вызове ПП прерывания

INTRET:in std\_logic; --=1 при RETI

WE:in std\_logic; --запись в intmask

ADDRD : in WORD; --адрес intmask

DATAI:in WORD;

INTREQ : in std\_logic\_vector(N\_INTSRC downto 0); --запросы на прерывание

INTAFD : out std\_logic; --требование прерывания

INTVEC : out std\_logic\_vector(N\_INTVEC downto 0));-- вектор прерывания

end component;

signal cy,rstd: std\_logic;

signal intafd, intcall, intret,wei,ENAi :std\_logic;

signal rs,rd,dataoi:WORD;

signal dalu,addrdi:WORD;

signal cnvz:NIBBLE;

signal intvec:std\_logic\_vector(N\_INTVEC downto 0);

begin

ENAi<=To\_X01(ENA); --приведение 'H','L' к 1,0

U\_RAM: RRAM port map(CLK=>CLK, -- ОЗУ

RST=>RST,

RSTD=>rstd,

ENA =>ENAi,

DALU =>dalu,

INSTR =>INSTR,

DATAI=>DATAI,

RE=>RE,

WE=>WEi,

RS =>rs,

RD =>rd,

ADDRD =>ADDRDi,

DATAO =>DATAOi) ;

U\_PC:PC port map( CLK=>CLK, --блок счетчика команд

RST=>RST,

ENA =>ENAi,

INTAFD=>intafd,

INSTR =>INSTR,

RS =>rs,

RSTD=>rstd,

CNVZ =>cnvz,

INTVEC =>intvec,

CY =>cy,

INTCALL =>intcall,

INTRET=>intret,

ADDRI =>ADDRI );

U\_ALU:ALU port map(CLK =>CLK, -- АЛУ

RST=>rst,

ENA =>ENAi,

CY =>cy,

RSTD=>rstd,

RD => rd,

RS =>rs,

INSTR =>INSTR,

DALU =>dalu,

CNVZ =>cnvz );

U\_INT: INT\_CNTRL port map( CLK=>CLK, --блок управления прерываниями

RST=>RST,

WE => WEi,

ADDRD=>addrdi,

DATAI =>dataoi,

INTCALL=>intcall,

INTRET=>intret,

INTREQ=>INTREQ,

INTAFD=>intafd,

INTVEC=>intvec);

WE<=WEi;

ADDRD<=addrdi;

DATAO<=dataoi;

end RISC\_SYNT;

library IEEE;

use IEEE.std\_logic\_1164.all, work. RISC\_lib.all;

entity RISC\_ST\_CORE is port(CLK : in std\_logic;

RST : in std\_logic;

ENA : in std\_logic; --готовность данных в DRAM

INSTR : in WORD; --команда из PRAM

INTREQ : in std\_logic\_vector( N\_INTSRC downto 0);--запросы на прерывание

DATAI : in WORD; --даннoе из DRAM

ADDRI : out WORD; --адрес команды

RE : out std\_logic; --сигнал чтения из DRAM

WE : out std\_logic; --сигнал записи в DRAM

DATAO : out WORD; --даннoе в DRAM

ADDRD : out WORD ); --адрес данного

end RISC\_ST\_CORE;

architecture RISC\_SYNT of RISC\_ST\_CORE is

component ALU is port(CLK : in std\_logic;

RST : in std\_logic;

ENA : in std\_logic;

CY:in std\_logic; --флаг переноса

RSTD:in std\_logic;

RD : in WORD; --первый операнд

RS : in WORD; --второй операнд

INSTR : in WORD; --команда

DALU : out WORD; --результат

CNVZ : out NIBBLE ); --вектор состояний

end component;

component PC is port( CLK : in std\_logic;

RST : in std\_logic;

ENA : in std\_logic;

INTAFD : in std\_logic; -- требование прерывания

INSTR : in WORD; --шина команд

RS : in WORD; --операнд RS

CNVZ : in NIBBLE; --вектор условий

INTVEC : in std\_logic\_vector(1 downto 0); --вектор прерывания

RSTD:buffer std\_logic;

CY : out std\_logic; --флаг переноса

INTCALL : out std\_logic; --флаг отработки прерывания

INTRET: out std\_logic; --квитирование INTAFD

ADDRI : out WORD ); --Адрес команды

end component;

component RRAM is port(CLK : in std\_logic;

RST : in std\_logic;

RSTD:in std\_logic;

ENA : in std\_logic;

DALU : in WORD; --данное из ALU

INSTR : in WORD; -- команда

DATAI : in WORD; -- данное из DRAM

RE:out std\_logic; --чтение DRAM

WE:out std\_logic; --запись DRAM

RS : out WORD; --операнд RS

RD : out WORD; --операнд RD

ADDRD : out WORD; --адрес DRAM

DATAO : out WORD); --данное в DRAM

end component;

component INT\_CNTRL is port (CLK : in std\_logic;

RST : in std\_logic;

INTCALL: in std\_logic; --=1 при вызове ПП прерывания

INTRET:in std\_logic; --=1 при RETI

WE:in std\_logic; --запись в intmask

ADDRD : in WORD; --адрес intmask

DATAI:in WORD;

INTREQ : in std\_logic\_vector(N\_INTSRC downto 0); --запросы на прерывание

INTAFD : out std\_logic; --требование прерывания

INTVEC : out std\_logic\_vector(N\_INTVEC downto 0));-- вектор прерывания

end component;

signal cy,rstd: std\_logic;

signal intafd, intcall, intret,wei,ENAi :std\_logic;

signal rs,rd,dataoi:WORD;

signal dalu,addrdi:WORD;

signal cnvz:NIBBLE;

signal intvec:std\_logic\_vector(N\_INTVEC downto 0);

begin

ENAi<=To\_X01(ENA); --приведение 'H','L' к 1,0

U\_RAM: RRAM port map(CLK=>CLK, -- ОЗУ

RST=>RST,

RSTD=>rstd,

ENA =>ENAi,

DALU =>dalu,

INSTR =>INSTR,

DATAI=>DATAI,

RE=>RE,

WE=>WEi,

RS =>rs,

RD =>rd,

ADDRD =>ADDRDi,

DATAO =>DATAOi) ;

U\_PC:PC port map( CLK=>CLK, --блок счетчика команд

RST=>RST,

ENA =>ENAi,

INTAFD=>intafd,

INSTR =>INSTR,

RS =>rs,

RSTD=>rstd,

CNVZ =>cnvz,

INTVEC =>intvec,

CY =>cy,

INTCALL =>intcall,

INTRET=>intret,

ADDRI =>ADDRI );

U\_ALU:ALU port map(CLK =>CLK, -- АЛУ

RST=>rst,

ENA =>ENAi,

CY =>cy,

RSTD=>rstd,

RD => rd,

RS =>rs,

INSTR =>INSTR,

DALU =>dalu,

CNVZ =>cnvz );

U\_INT: INT\_CNTRL port map( CLK=>CLK, --блок управления прерываниями

RST=>RST,

WE => WEi,

ADDRD=>addrdi,

DATAI =>dataoi,

INTCALL=>intcall,

INTRET=>intret,

INTREQ=>INTREQ,

INTAFD=>intafd,

INTVEC=>intvec);

WE<=WEi;

ADDRD<=addrdi;

DATAO<=dataoi;

end RISC\_SYNT;