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.

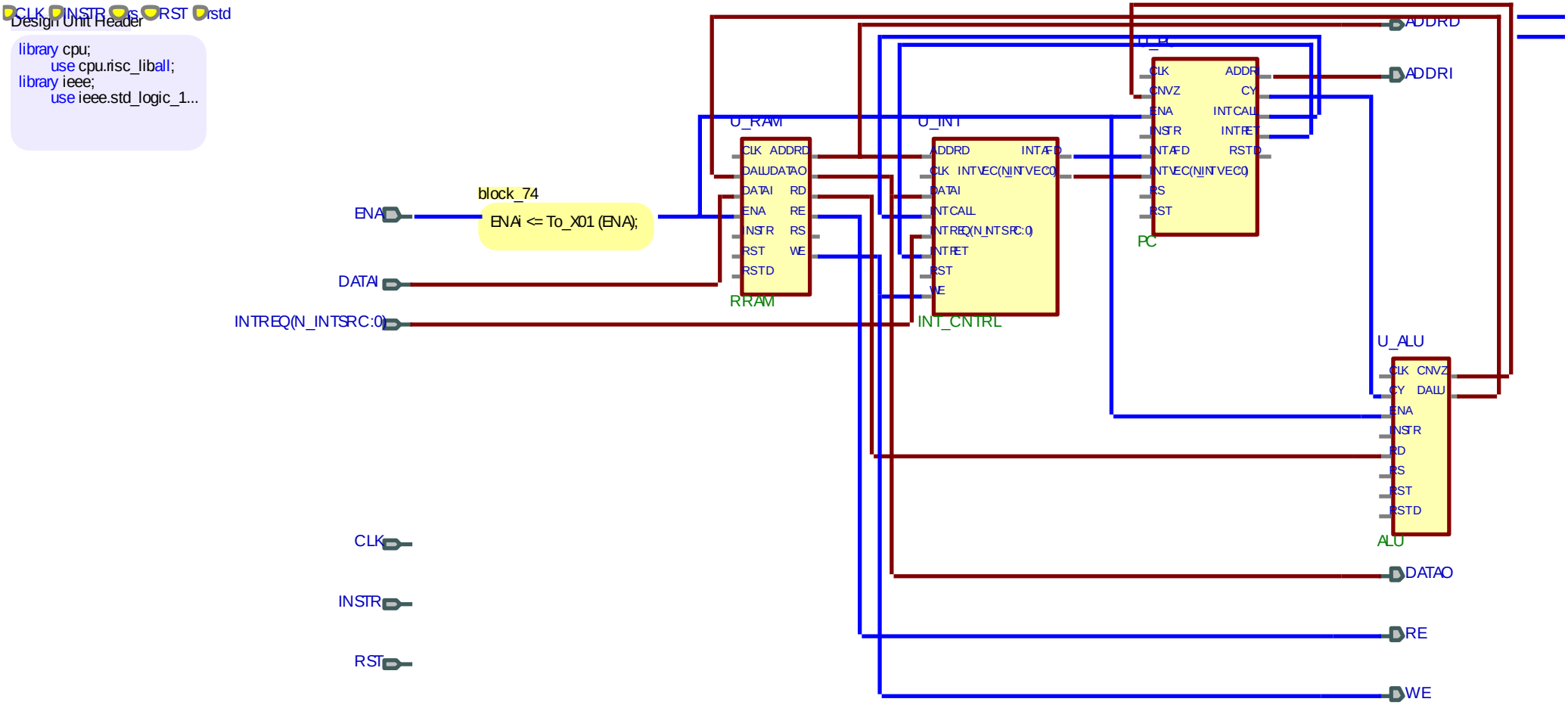
Київ 2017

Тема: Проектування ядра мікропроцесору.

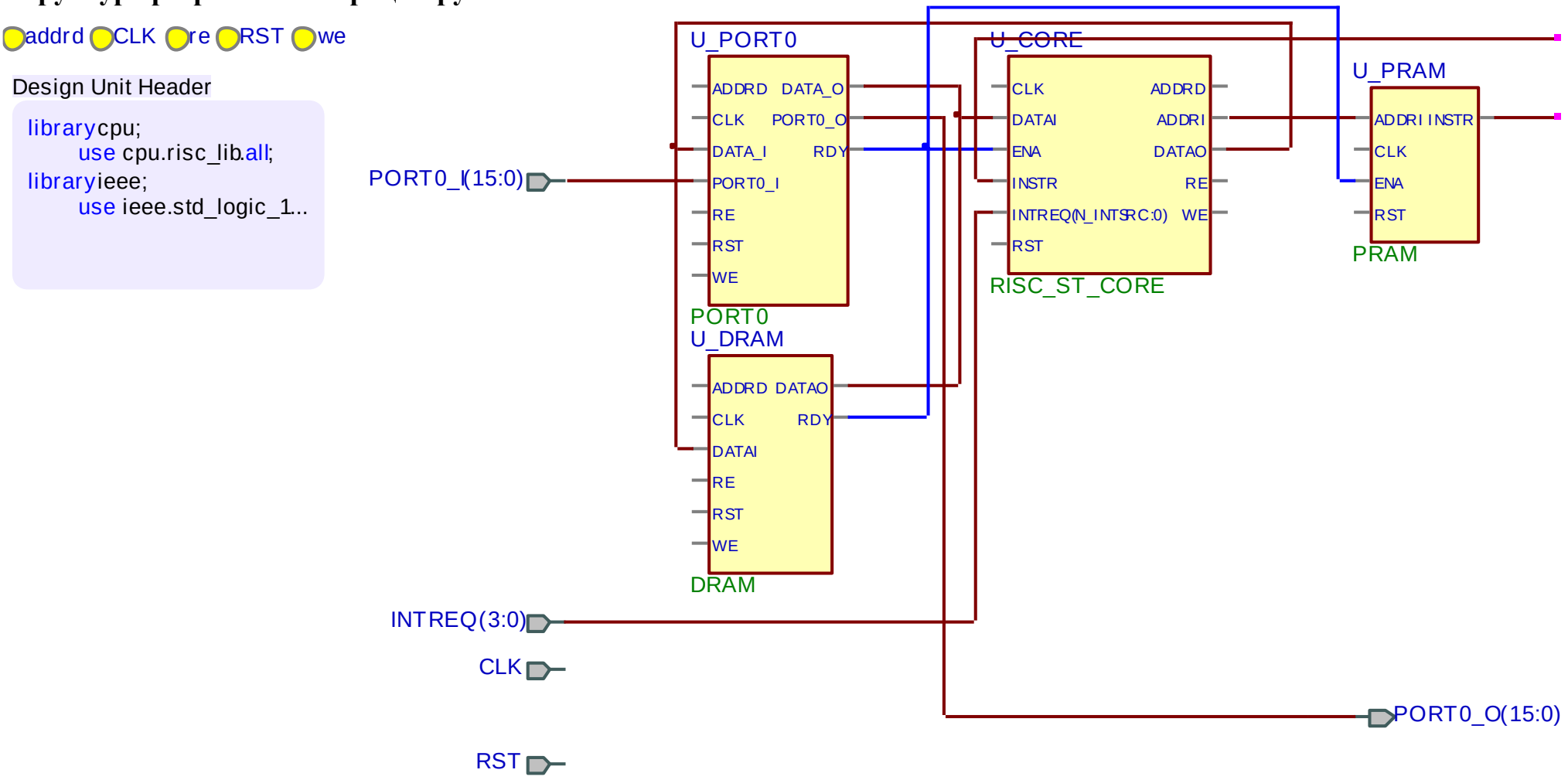
Мета та основні завдання роботи: оволодіти знаннями і практичними навичками з проектування таких складних обчислювальних блоків з програмним управлінням, як ядро мікропроцесору (CPU). Лабораторна робота також служить для оволодіння навичками програмування і налагодження опису CPU на мові VHDL.

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:



Структура 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 PC+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 состояния 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;          --данное 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;     --данное 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;           --данное 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;      --данное 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;          --данное 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;     --данное 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;
```