
微处理器

计科 1603-张家升-201608010606


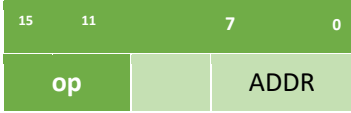
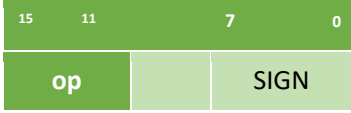




目录

一、指令格式设计	3
二、微操作的定义	4
三、节拍的划分	6
四、处理器详细结构设计框图及功能描述	7
五、各功能模块结构设计框图及功能描述	10
六、VHDL 代码、UCF 文件、测试指令序列	13
七、实验总结，在调试和下载过程中遇到的问题	48

一、指令格式设计

类别	指令	指令格式	操作码	含义												
传送类指令	MVRR R _i , R _j	<table><tr><td>15</td><td>11</td><td>10</td><td>8</td><td>2</td><td>0</td></tr><tr><td colspan="2">op</td><td colspan="2">R_i</td><td colspan="2">R_j</td></tr></table>	15	11	10	8	2	0	op		R _i		R _j		00000	寄存器 i 到寄存器 j
15	11	10	8	2	0											
op		R _i		R _j												

	MVIR Ri , INS		00001	立即数到寄存器 Ri
	MVRM M , Ri		00010	寄存器 Ri 到内存 M
	MVMR Ri , M		00011	内存 M 到寄存器 Ri
算 运 算 指 令	ADCRR Ri , Rj		01000	$Ri + Rj \rightarrow Ri$
	ADCRI Ri , INS		01001	$Ri + INS \rightarrow Ri$
	SBBRR Ri , Rj		01010	$Ri - Rj \rightarrow Ri$
	SBBRI Ri , INS		01011	$Ri - INS \rightarrow Ri$
	ANDRR Ri , Rj		01100	$Ri \text{ and } Rj \rightarrow Ri$
	ANDRI Ri , INS		01101	$Ri \text{ and } INS \rightarrow Ri$
	ORRR Ri , Rj		01110	$Ri \text{ or } Rj \rightarrow Ri$
	ORRI Ri , INS		01111	$Ri \text{ or } INS \rightarrow Ri$
	CLC		10000	$0 \rightarrow cy$

	STC		10001	1-> cy
跳转指令	JMP ADDR		10100	R7 ADDR -> pc
	JZ SIGN		10101	Z = 1 pc + sign + 1 -> pc
	JC SIGN		10110	Cy = 1 pc + sign + 1 -> pc
扩展指令	MOVMRR Ri , Rj		10111	M([Rj]) -> Ri
	MOVMRIX Ri, IXOFFSET		11000	M([R6] + IXOFFSET) -> Ri
额外需要	LR Ri		11001	Ri 高低八位互换

二、微操作的定义

定义 AD(IR)为取 IR 的 7-0 位， AD1(IR)为取 IR 的 10-8 位， AD2(IR) 为取 IR 的 2-0 位， AD3(IR)为取 IR 中的 10-3 位， OP(IR)为取 IR 中的操作码；由于每条指令的取指阶段都一致，故单独列出：

T0: PC -> MAR , 1 -> R
T1: M(MAR) -> MDR
T2: MDR -> IR , pc + 1 -> pc

以下给出每条指令执行阶段的微操作定义：

指令	微操作定义
MVRR	T0 : Reg(Ad2(IR)) -> rtemp T1 : rtemp -> rdata T2 : rdata -> Reg(Ad1(IR))
MVIR	T0 : Ad(IR) -> rtemp

	T1 : rtemp -> rdata T2 : rdata -> Reg(ad1(IR))
MVRM	T0: Reg(7) Ad(IR) -> addr , Reg(Ad2(IR)) -> rtemp T1:rtemp -> MDR , addr -> MAR , 1 -> w T2:MDR -> M(MAR)
MVMR	T0 : Reg(7) Ad(IR) -> addr T1 : 1 -> r , MDR -> rtemp ; T2 : rtemp -> rdata T3 : rdata -> Reg(ad1(IR))
ADCRR	T0 : reg(ad1(ir)) -> ax , reg(ad2(ir)) -> bx T1 : ax + bx + cy -> aluout , aluout -> rtemp T2 : rtemp -> rdata , rdata -> reg(ad1(ir))
ADCRI	T0 : reg(ad1(ir)) -> ax , ad(ir) -> bx T1 : ax + bx + cy -> aluout , aluout -> rtemp T2 : rtemp -> rdata , rdata -> reg(ad1(ir))
SBBRR	T0 : reg(ad1(ir)) -> ax , reg(ad2(ir)) -> bx T1 : ax - bx - cy -> aluout , aluout -> rtemp T2 : rtemp -> rdata , rdata -> reg(ad1(ir))
SBBRI	T0 : reg(ad1(ir)) -> ax , ad(ir) -> bx T1 : ax - bx - cy -> aluout , aluout -> rtemp T2 : rtemp -> rdata , rdata -> reg(ad1(ir))
ANDRR	T0 : reg(ad1(ir)) -> ax , reg(ad2(ir)) -> bx T1 : ax and bx -> aluout , aluout -> rtemp T2 : rtemp -> rdata , rdata -> reg(ad1(ir))
ANDRI	T0 : reg(ad1(ir)) -> ax , ad(ir) -> bx T1 : ax and bx -> aluout , aluout -> rtemp T2 : rtemp -> rdata , rdata -> reg(ad1(ir))
ORRR	T0 : reg(ad1(ir)) -> ax , reg(ad2(ir)) -> bx T1 : ax or bx -> aluout , aluout -> rtemp T2 : rtemp -> rdata , rdata -> reg(ad1(ir))
ORRI	T0 : reg(ad1(ir)) -> ax , ad(ir) -> bx T1 : ax or bx -> aluout , aluout -> rtemp T2 : rtemp -> rdata , rdata -> reg(ad1(ir))
CLC	T0 : 0 -> cy
STC	T0 : 1 -> cy
JMP	T0 : ad(ir) -> aluout , aluout -> rtemp T1 : rtemp -> pcnew T2 : pcnew -> pc
JZ	T0 : ad(ir)+pc -> aluout , aluout -> rtemp T1 : rtemp -> pcnew T2 : if z = 1 then pcnew -> pc

JC	T0 : ad(ir)+pc -> aluout , aluout -> rtemp T1 : rtemp -> pcnew T2 : if cy = 1 then pcnew -> pc
MOVRR	T0 : ad2(ir) -> mar , 1 -> r T1: mdr -> rtemp , rtemp -> rdata T2 : rdata -> reg(ad1(ir))
MOVRIX	T0 : ad(ir) + reg(6) -> mar , 1 -> r T1 : mdr -> rtemp , rtemp -> rdata T2: rdata -> reg(ad1(ir))
LR	T0 : reg(ad1(ir)) -> ax T1: ax(7 downto 0) -> aluout(15 downto 8) , ax(15 downto 8) -> aluout(7 downto 0) T2 : aluout -> rtemp , rtemp -> rdata T3 : rdata -> reg(ad1(ir))

三、 节拍的划分

指令	节拍划分			
	取指	运算	存储管理	回写
MVRR Ri , Rj	Pc -> mar ,1 -> r ; Mdr -> ir ;	Ri -> aluout J -> raddr Pc+1 -> pc	Aluout ->rtemp Rtemp - > rdata	Rdata ->rj
MVIR Ri , INS	Pc -> mar ,1 -> r ; Mdr -> ir ;	INS -> aluout J -> raddr Pc+1 -> pc	Aluout ->rtemp Rtemp - > rdata	Rdata ->rj
MVRM M , Ri	Pc -> mar ,1 -> r ; Mdr -> ir ;	Ri -> aluout R7 M -> addr Pc+1 -> pc	Aluout -> rtemp Rtemp -> mdr 1 -> w	
MVMR Ri , M	Pc -> mar ,1 -> r ; Mdr -> ir ;	I -> raddr R7 M -> addr	1 -> r Mdr -> rtemp Rtemp -> rdata	rdata -> ri
ADCRR Ri , Rj	Pc -> mar ,1 -> r ; Mdr -> ir ;	Ri + Rj + cy -> aluout I -> raddr Pc+1 -> pc	Aluout -> rtemp ; Rtemp -> rdata	Rdata -> ri
ADCRI Ri , INS	Pc -> mar ,1 -> r ; Mdr -> ir ;	Ri + INS+ cy -> aluout I -> raddr Pc + 1 -> pc	Aluout -> rtemp Rtemp -> rdata	Rdata -> ri
SBBRR Ri , Rj	Pc -> mar ,1 -> r ; Mdr -> ir ;	Ri - Rj - cy -> aluout I -> raddr Pc+1 -> pc	Aluout -> rtemp ; Rtemp -> rdata	Rdata -> ri

SBBRI Ri , INS	Pc -> mar ,1 -> r ; Mdr -> ir ;	Ri – INS – cy -> aluout I -> raddr Pc + 1 -> pc	Aluout -> rtemp Rtemp -> rdata	Rdata -> ri
ANDRR Ri , Rj	Pc -> mar ,1 -> r ; Mdr -> ir ;	Ri and Rj -> aluout I -> raddr Pc+1 -> pc	Aluout -> rtemp ; Rtemp -> rdata	Rdata -> ri
ANDRI Ri , INS	Pc -> mar ,1 -> r ; Mdr -> ir ;	Ri and INS -> aluout I -> raddr Pc + 1 -> pc	Aluout -> rtemp Rtemp -> rdata	Rdata -> ri
ORRR Ri , Rj	Pc -> mar ,1 -> r ; Mdr -> ir ;	Ri or Rj -> aluout I -> raddr Pc+1 -> pc	Aluout -> rtemp ; Rtemp -> rdata	Rdata -> ri
ORRI Ri , INS	Pc -> mar ,1 -> r ; Mdr -> ir ;	Ri or INS -> aluout I -> raddr Pc + 1 -> pc	Aluout -> rtemp Rtemp -> rdata	Rdata -> ri
CLC	Pc -> mar ,1 -> r ; Mdr -> ir ;	0 -> cy		
STC	Pc -> mar ,1 -> r ; Mdr -> ir ;	1 -> cy		
JMP ADDR	Pc -> mar ,1 -> r ; Mdr -> ir ;	Addr -> aluout Pc+1 -> pc	Aluout -> rtemp Rtemp -> pcnew	Pcnew -> pc
JZ SIGN	Pc -> mar ,1 -> r ; Mdr -> ir ;	Pc + sign + 1 -> aluout Pc + 1 -> pc	Aluout -> rtemp Rtemp -> pcnew	If z = 1 then Pcnew -> pc
JC SIGN	Pc -> mar ,1 -> r ; Mdr -> ir ;	Pc + sign + 1 -> aluout Pc + 1 -> pc	Aluout -> rtemp Rtemp -> pcnew	If cy = 1 then Pcnew -> pc
MOVMMR Ri , Rj	Pc -> mar ,1 -> r ; Mdr -> ir ;	Rj -> addr I -> raddr Pc+1 -> pc	1 -> r Mdr -> rtemp Rtemp -> rdata	Rdata -> ri
MOVMRi Ri , IXOFFSET	Pc -> mar ,1 -> r ; Mdr -> ir ;	R6 + IXOFFSET -> addr I -> raddr Pc + 1 -> pc	1 -> r Mdr -> rtemp Rtemp -> rdata	Rdata -> ri
LR Ri	Pc -> mar ,1 -> r ; Mdr -> ir ;	Ri(7 downto 0) -> aluout(15 downto 8) Ri(15 downto 8) -> aluout(7 downto 0) I -> raddr Pc + 1 -> pc	Aluout -> rtemp Rtemp -> rdata	Rdata -> Ri

四、 处理器详细结构设计框图及功能描述

功能描述:

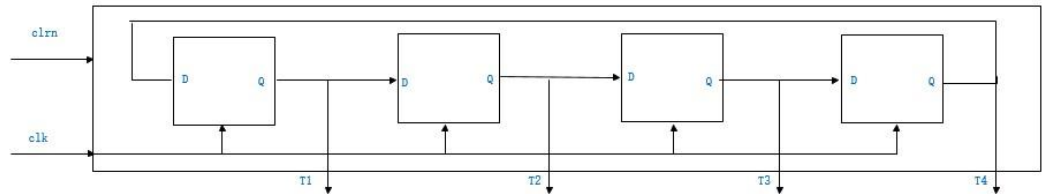
1. 处理器在给定的指令集下构建，设计有 20 条指令。其中传送类指令 4 条，包含寄存器到寄存器，寄存器到内存，内存到寄存器，立即数到寄存器;算逻运算类指令 10 条，包含进位加、进位减，与、或各 2 条指令（寄存器与寄存器，寄存器与立即数），cy 复位、置位指令；跳转指令 3 条；扩展指令 2 条；另加 1 条指令（LR）仅作测试需要。

-
2. 处理器包含时钟管理、取指、运算、存储管理、回写和访存控制 6 个模块，包含的主要寄存器有：1 个 16 位的指令寄存器 IR，1 个 16 位的程序计数器 PC，两个标志寄存器 Cy 和 Z，8 个 16 位的通用寄存器 $R_7 \sim R_0$ 。处理器的指令字长为 16 位，处理器的地址总线宽度是 16 位，数据总线宽度也是 16 位。
 3. 处理器在时钟模块的控制下，一条指令分为 4 个节拍进行，分别进行取指、运算、存储管理、回写操作，取指都存储管理都通过访存控制模块与内存进行数据交互。
 4. $pc+1$ 在运算模块通过回写完成。

五、 各功能模块结构设计框图及功能描述

1. 时钟管理模块

设计框图：

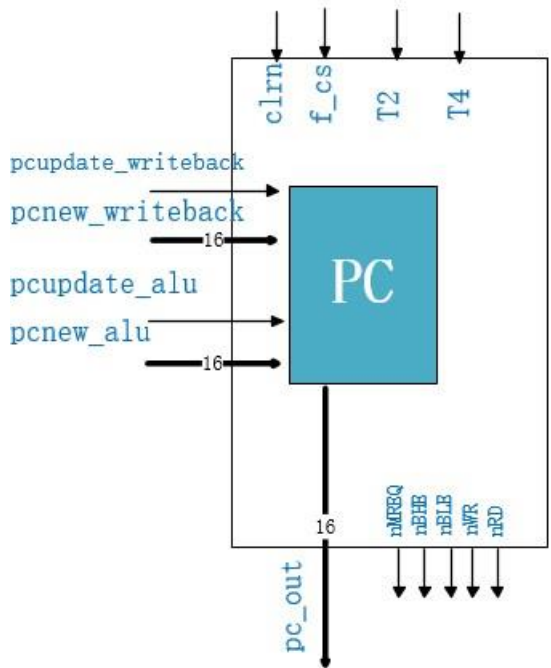


功能描述：四位循环移位，将结果输出作为各个模块的驱动信号。初始化为 1000.

功

2. 取指模块

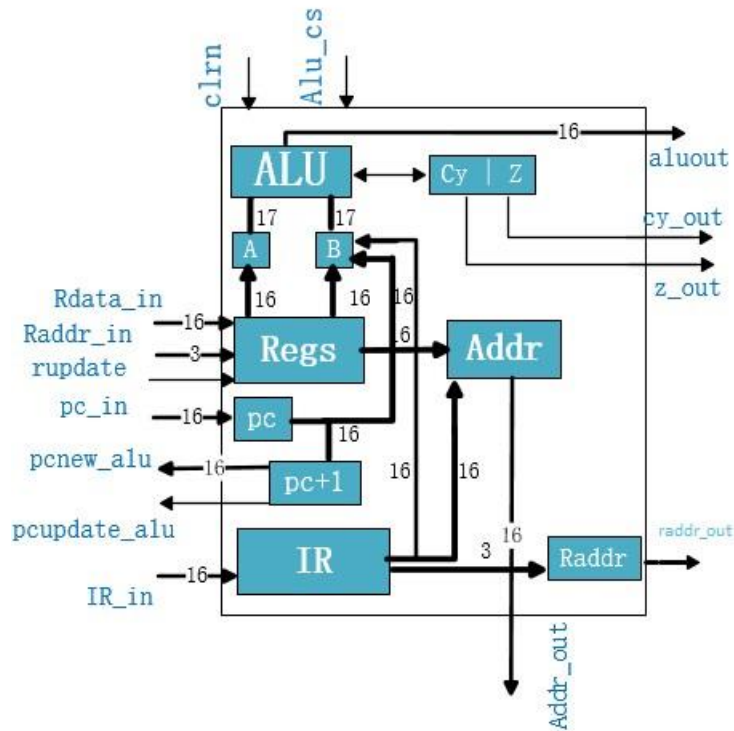
设计框图：



功能描述：将 pc 的值送到访存控制模块，同时送出访存信号。

3. 运算模块

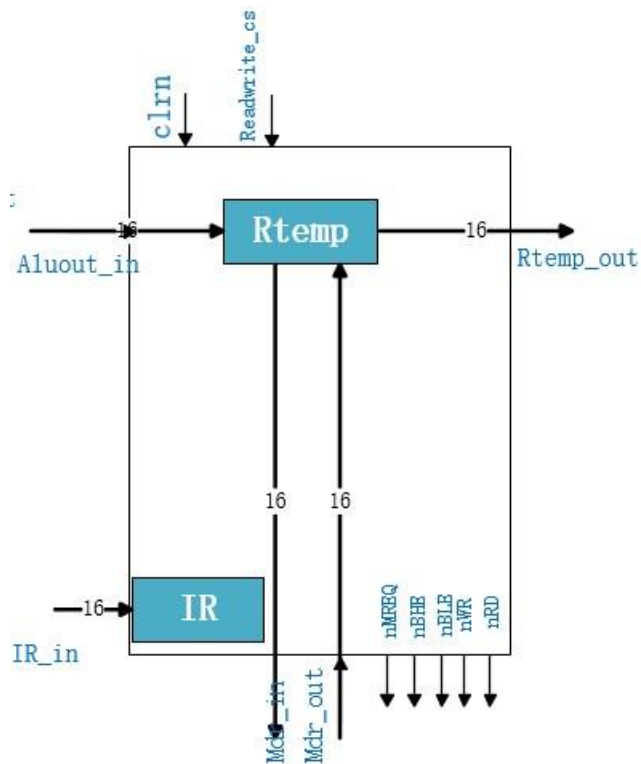
设计框图：



功能描述：完成运算输出（包括新 pc 的值）；完成访存地址的输出
完成回写寄存器地址的输出 完成 pc+1 的回写

4. 存储管理模块

设计框图：

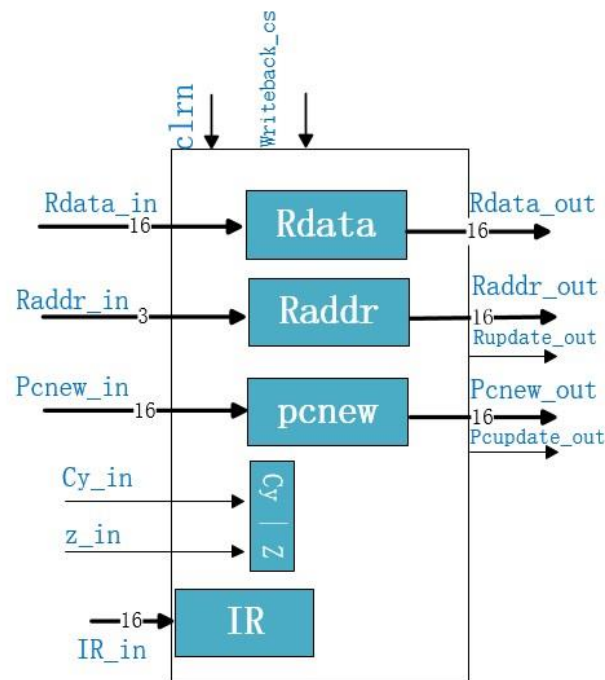


功能描述：

完成读取、存储内存的值传递
或仅作中间变量传递 aluout 的值到回写模块（如 jmp, jc, jz 指令时）

5. 回写模块

设计框图：



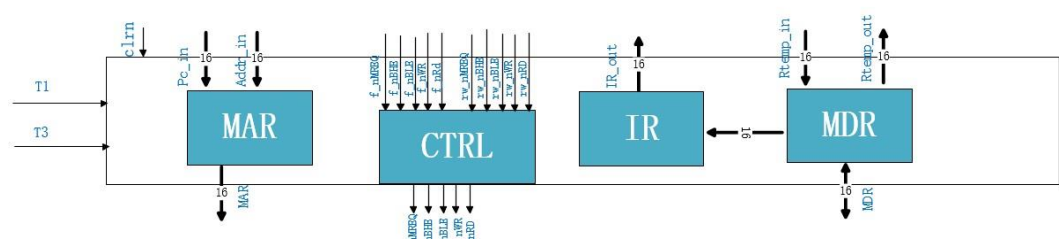
功能描述：

完成寄存器的回写

完成 pc 的回写

6. 访存控制模块

设计框图：



```
IEEE.STD_LOGIC_ARITH.ALL; use
```

```
IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
---- Uncomment the following library declaration if instantiating
```

```
---- any Xilinx primitives in this code.
```

```
--library UNISIM;
```

```
--use UNISIM.VComponents.all;
```

```
entity beat is
```

```
    port(
```

```
        clk,clrn : in std_logic ;
```

```
        T1,T2,T3,T4:out std_logic
```

```
    );
```

```
end beat;
```

```
--高电平有效
```

```
architecture Behavioral of beat is    signal beats :
```

```
std_logic_vector(3 downto 0):="0000" ; begin
```

```
process(clk,clrn)
```

```
    begin  if( clrn = '0' ) then
```

```
        beats <= "0000" ;
```

```
        elsif (clk = '1' and clk'event ) then
```

```
            beats(2)<=beats(3) ;
```

```
            beats(1)<=beats(2) ;
```

```
beats(0)<=beats(1) ;
```

```
if(beats = "0000") then
```

```
beats(3) <= '1' ;
```

```
        else
```

```
            beats(3)<=beats(0) ;
```

```
        end if ;
```

```
        end if ;
```

```
    end process ;
```

```
    T1 <= beats(3) ;
```

```
    T2 <= beats(2) ;
```

```
    T3 <= beats(1) ;
```

```
    T4 <= beats(0) ;
```

```
end Behavioral;
```

b) fetch.vhd

```
-- 取指模块
```

```
--
```

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL; use
```

```
IEEE.STD_LOGIC_ARITH.ALL; use
```

```
IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
---- Uncomment the following library declaration if instantiating
```

---- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity fetch is

```
    port(
        clk : in std_logic ;           t2,T4 : IN STD_LOGIC ;
        f_cs,clrn: in std_logic ;       pc_update_alu: in
std_logic ;   pc_new_alu: in std_logic_vector(15 downto 0) ;
pc_update_writeback: in std_logic ; pc_new_writeback: in
std_logic_vector(15 downto 0) ;

        pc_out : out std_logic_vector(15 downto
0) ; ir_in : in std_logic_vector(15 downto 0) ;
        nrd,nMREQ,nwr : out std_logic ; nble,nbhe :
out std_logic
    );
```

end fetch;

architecture Behavioral of fetch is

```
    signal
pc:std_logic_vector(15 downto 0) ;
    ir:std_logic_vector(15 downto 0) ; begin
    process(clrn,pc_update_alu,t2,t4,pc_new_alu,clk)
    begin
        if(clrn = '0') then
            pc <= "0000000000000000" ;
        elsif(t2 = '1' and pc_update_alu = '1') then
            pc <= pc_new_alu ;
        elsif(t4 = '1' and pc_update_writeback = '1' ) then
            pc <= pc_new_writeback ;
        end if ;
    end process ;
```

```
    process(clrn,f_cs)
    begin
        if(clrn = '0') then
pc_out <= "0000000000000000" ;
        elsif(f_cs = '1') then
            pc_out <= pc ;
        end if ;
    end process ;
```

```
    process(clrn,f_cs)
    begin --存控制信号输出
        if(clrn = '0' or f_cs = '0') then
nrd <= '1' ;
```

```

        nMREQ <= '1' ; nWR <= '1' ;

        nBHE <= '1' ; nBLE <= '1' ;

        elsif(clrn = '1' and f_cs = '1')
        then nrd <= '0' ; nMREQ <= '0' ;

        nWR <= '1' ;

        nBHE <= '0' ;

nBLE <= '0' ; end if ;
end process ; end
Behavioral;

```

c) alu.vhd

```

-- Company:

-- Engineer:

--

-- Create Date: 12:04:58 11/25/2013 --
Design Name:
-- Module Name: alu - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created
-- Additional Comments:

--
-----

-----

library IEEE; use IEEE.STD_LOGIC_1164.ALL; use
IEEE.STD_LOGIC_ARITH.ALL ; use
IEEE.STD_LOGIC_UNSIGNED.ALL ;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity alu is port( --cs alu_cs,clrn:in std_logic ; -- alu_cs '1' is effective , clrn is '0'
effective

--

ir,pc_in : in std_logic_vector(15 downto 0) ;

```

```

    ---write back

    Raddr:in std_logic_vector(2 downto 0) ;

    Rdata:in std_logic_vector(15 downto 0) ;

    Rupdate:in std_logic ;

    ---output

    ALUOUT :out std_logic_vector(15 downto 0) ;


    addr_out:out std_logic_vector(15 downto 0) ;

    Raddr_out:out std_logic_vector(2 downto 0) ;--output the raddr to the write-back moudle

cy_out,z_out:out std_logic ;

    --pcupdate

    pcnew : out std_logic_vector(15 downto 0) ;

pcupdate : out std_logic

    ) ;

end alu;


architecture Behavioral of alu is    type regType is array(7 downto 0) of
std_logic_vector(15 downto 0) ;

    --reg

    signal regs : regType ;

    signal cy,z :std_logic ;

    --cal

begin

    --alu_ready

    process(clrn,alu_cs,ir)

        --ad1:10-8 ,ad2:2-0 ,ad3:7-0 , ad4:10-3

        variable ad1,ad2: integer ;          variable
        ad3,ad4:std_logic_vector(7 downto 0) ;          variable
        alu,ax,bx:std_logic_vector(16 downto 0) ;          variable
        isPositive : std_logic ;

        begin

            if clrn = '0' then

addr_out <="ZZZZZZZZZZZZZZZZ" ;

                ALUOUT <= "1010101010101010" ;

                cy <= '0' ;

                z <= '0' ;

            elsif alu_cs = '1' and alu_cs'event then

                --get index or value ad1 :=
                conv_integer(ir(10 downto 8)) ;

                ad2 := conv_integer(ir(2 downto 0)) ;

                ad3 := ir(7 downto 0) ; ad4 := ir(10
                downto 3) ; -----
                alu_ready-----
                -----

                case ir(15 downto 11) is

                    --mvRR Ri, Rj

                    when "00000" =>

```

```

        ALUOUT <= regs(ad2) ;
        Raddr_out <= ir(10 downto 8) ;

--mvIR
when "00001" =>
        ALUOUT(7 downto 0) <= ad3 ;
        ALUOUT(15 downto 8) <= "00000000" ;
        Raddr_out <= ir(10 downto 8) ;

--mvRM M,Ri                                when
"00010" =>                                ALUOUT <= regs(ad2) ;
        addr_out(7 downto 0) <= ad4 ;
addr_out(15 downto 8) <= regs(7)(7 downto 0) ;

--mvMR R,M

when "00011" =>
        --just for invoke the alu_calculate process,is it ok?
        addr_out(7 downto 0) <= ad3 ;
addr_out(15 downto 8) <= regs(7)(7 downto 0) ;
        --addr_out <= "0101010101010101" ;
        Raddr_out <= ir(10 downto 8) ;

--ADC
        --ADCRR

        when "01000" =>
                ax := '0'&regs(ad1) ;
                bx := '0'&regs(ad2) ;
                alu := ax + bx + cy ;
                ALUOUT <= alu(15 downto 0) ;
                --ALUOUT <= "1111111111111111" ;
                cy <= alu(16) ;

                if(conv_integer(alu) = 0) then
                        z <= '1' ;
                else
                        z <= '0' ;
                end if ;
                Raddr_out <= ir(10 downto 8) ;
                --ADCIR

when "01001" => ax := '0'&regs(ad1) ;
        bx(7 downto 0) := ad3 ; bx(16
downto 8) := "000000000" ; alu :=
        ax + bx + cy ;

        ALUOUT <= alu(15 downto 0) ; cy
        <= alu(16) ; if(conv_integer(alu)
        = 0) then
                z <= '1' ;
        else
                z <= '0' ;
        end if ;
        Raddr_out <= ir(10 downto 8) ;

--SBB

```

```

--SBBRR
when "01010" =>

    ax := '0'&regs(ad1);
    bx := '0'&regs(ad2);
    alu := ax -
    bx - cy;
    ALUOUT <= alu(15
    downto 0);
    cy <= alu(16);
    if(conv_integer(alu) = 0) then
        z <= '1';
    else
        z <= '0';
    end if;
    Raddr_out <= ir(10 downto 8);

--SBBRI
when "01011" =>

    ax := '0'&regs(ad1);
    bx(7
    downto 0) := ad3;
    bx(16 downto 8) :=
    "000000000";
    alu := ax - bx - cy;
    ALUOUT <= alu(15 downto 0);
    cy <= alu(16);
    if(conv_integer(alu) = 0) then
        z <= '1';
    else
        z <= '0';
    end if;
    Raddr_out <= ir(10 downto 8);

--AND
--ANDRR
when "01100" => ax :=
    '0'&regs(ad1); bx :=
    '0'&regs(ad2); alu := ax and
    bx; ALUOUT <= alu(15
    downto 0); cy <= alu(16);
    if(conv_integer(alu) = 0) then
        z <= '1';
    else
        z <= '0';
    end if;
    Raddr_out <= ir(10 downto 8);

--ANDRI
when "01101" =>

    ax := '0'&regs(ad1);
    bx(7
    downto 0) := ad3;
    bx(16 downto 8) :=
    "000000000";
    alu := ax and bx;
    ALUOUT <= alu(15 downto 0);
    cy <= alu(16);
    if(conv_integer(alu) = 0) then
        z <= '1';

```

```

        else
            z <= '0' ;
        end if ;
        Raddr_out <= ir(10 downto 8) ;
--OR
--ORRR
when "01110" =>
    ax := '0'&regs(ad1) ;
    bx(7
downto 0) := ad3 ;
    bx(16 downto 8) :=
"00000000" ;
    alu := ax or bx ;
    ALUOUT <= alu(15 downto 0) ;
    cy <= alu(16) ;
    if(conv_integer(alu) = 0) then
        z <= '1' ;
    else
        z <= '0' ;
    end if ;
    Raddr_out <= ir(10 downto 8) ;
--ORRI
when "01111" =>    ax := '0'&regs(ad1) ;
    bx(7 downto 0) := ad3 ; bx(16
downto 8) := "00000000" ; alu := ax or
bx ; ALUOUT <= alu(15 downto 0) ; cy
<= alu(16) ; if(conv_integer(alu) = 0)
then
    z <= '1' ;
else
    z <= '0' ;
end if ;
    Raddr_out <= ir(10 downto 8) ; --CLC
when "10000" =>
    cy <= '0' ;
--STC
when "10001" =>
    cy <= '1' ;
--JMP
when "10100" =>

    ALUOUT(7 downto 0) <= ad3 ;
    ALUOUT(15 downto 8) <= "00000000" ;
-----JZ,JC ,in alu we just calculate it and ouput it from aluout to pc_new or write-back
-----and decided if write back pc at write-back moudle
--JZ
when "10101" =>
    isPositive := '1' ;
if ad3(7) = '1' then
    ad3 :=

```

```

not ad3 + '1' ;

isPositive := '0' ;

                                end if ;

                                ax(15 downto 0) := pc_in ;

                                bx(7 downto 0) := ad3 ;

                                bx(15 downto 8) := "00000000" ;

if isPositive = '1' then                                alu := ax

+bx + '1' ;

                                else

                                alu:= ax - bx + '1' ;

                                end if ;

                                ALUOUT <= alu(15 downto 0) ;

--JC

when "10110" =>

                                isPositive := '1' ;      if

ad3(7) = '1' then                                ad3 :=

not ad3 + '1' ;

                                isPositive := '0' ;

                                end if ; ax(15 downto 0) :=

pc_in ; bx(7 downto 0) := ad3 ;

                                bx(15 downto 8) :=

"00000000" ; if isPositive = '1'

                                then alu := ax + bx + '1' ; else

                                alu := ax - bx + '1' ; end if ;

                                ALUOUT <= alu(15 downto 0) ;

-----extend

--movMRR

when "10111" =>

                                addr_out <= regs(ad2) ;

                                Raddr_out <= ir(10 downto 8) ;

--movMRIX

when "11000" =>

                                ax := '0'&regs(6) ;                                bx(7 downto

0) := ad3 ;                                bx(16 downto 8) :=

"0000000000" ;

                                alu := ax +bx ;

                                addr_out <= alu(15 downto 0) ;

                                Raddr_out <= ir(10 downto 8) ;

--LR

when "11001" =>

                                ALUOUT(7 downto 0) <= regs(ad1)(15 downto 8) ;

                                ALUOUT(15 downto 8) <= regs(ad1)(7 downto 0) ;

                                Raddr_out <= ir(10 downto 8) ;

when others =>                                end case ;

-----end alu_ready-----

                                end if ;

                                end process ;

```

```

-----out put-----
        cy_out <= cy ;
        z_out <= z ;
-----write_back-----
        process(Rupdate,Raddr,Rdata)
        begin
            if(Rupdate = '1') then
regs(conv_integer(Raddr)) <= Rdata ;
                end if ;
            end process ;
---to update pc
        process(clrn,alu_cs)
        begin if(clrn = '0')
        then

            pcupdate <= '0' ; elsif(alu_cs
= '1') then
                pcnew <= pc_in + "0000000000000001" ;
                pcupdate<= '1' ;
            else
                pcupdate <= '0' ;
        end if ;        end process ;
end Behavioral;

```

d) readwrite.vhd

```

-- Company:
-- Engineer:
--
-- Create Date: 09:04:18 11/27/2013 --
Design Name:
-- Module Name: readwrite - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision: -- Revision 0.01 -
File Created
-- Additional Comments:
--

```

```

library IEEE; use IEEE.STD_LOGIC_1164.ALL;

```

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity readwrite is
```

```
    port( --basic  readwrite_cs , clrn : in
          std_logic ;
          --rtemp in
          aluout_in: in std_logic_vector(15 downto 0) ;
          mdr_in :    in std_logic_vector(15 downto 0) ;
          --rtemp out
          rtemp_out : out std_logic_vector(15 downto 0) ;
          mdr_out : out std_logic_vector(15 downto 0) ;
          --ir
          ir: in std_logic_vector(15 downto 0) ;
          --read write control
          nMREQ,nRD,nWR : out std_logic ;
          nBHE,nBLE : out std_logic --high / low 8 bits is active
    );
end readwrite;
```

```
architecture Behavioral of readwrite is
```

```
    signal rtemp : std_logic_vector(15 downto 0) ; begin
        --read write
        process(clrn,readwrite_cs,ir,aluout_in,mdr_in)
        begin
            if clrn = '0' then
                nMREQ <= '1' ;
                nBHE <= '1' ;
            nBLE <= '1' ;                nRD <=
            '1' ;                nWR <= '1' ;
                --it is not necessary ,but ....
                mdr_out <= "ZZZZZZZZZZZZZZZZ" ;
                rtemp_out <= "ZZZZZZZZZZZZZZZZ" ;                elsif
            readwrite_cs = '1' then                case ir(15 downto
            11) is                --mvRM,write the aluout_in to memory
                when "00010" =>
                    nMREQ <= '0' ;
                    nBHE <= '0' ;                nBLE <= '0' ;
                    nRD <= '1' ;                nWR <= '0' ;
                    mdr_out <= aluout_in ;
```

```

--mvMR,read the memory to the rtemp_out
--i don't know if it will be ok
when "00011" =>
    nMREQ <= '0' ;          nBHE
    <= '0' ;                nBLE <= '0' ;
    nRD <= '0' ;
    nWR <= '1' ;
    rtemp_out <= mdr_in ;
--movMRR,read
when "10111" =>            nMREQ
    <= '0' ;                nBHE <= '0' ;
    nBLE <= '0' ;
    nRD <= '0' ;            nWR <= '1' ;
    rtemp_out <= mdr_in ;
--movMRIX,read
when "11000" =>            nMREQ
    <= '0' ;                nBHE <= '0' ;
    nBLE <= '0' ;
    nRD <= '0' ;            nWR <= '1' ;
    rtemp_out <= mdr_in ;
    when others =>
        nMREQ <= '1' ;
        nBHE <= '1' ;      nBLE <= '1' ;
        nRD <= '1' ;
        nWR <= '1' ;
        rtemp_out <=
aluout_in ;                end case ;      end if ;
    end process ;
end Behavioral;
```

e) writeback.vhd

```

library
ieee ;

use ieee.std_logic_1164.all ;

entity writeback is port(

    --basic clrn,writeback_cs : in std_logic ; --reg
    write-back rdata_out: out std_logic_vector(15
    downto 0) ; raddr_out: out std_logic_vector(2
    downto 0) ; rupdate_out : out std_logic ; --
    high is active
    rdata_in:in std_logic_vector(15 downto 0) ;
    raddr_in:in std_logic_vector(2 downto 0) ;
    --pc write-back
    pcnew_in : in std_logic_vector(15 downto 0) ;
    pcnew_out : out std_logic_vector(15 downto 0) ;
```

```

pcupdate_out :out std_logic ;          cy_in,z_in : in
std_logic ;

--ir
ir : in std_logic_vector(15 downto 0)

);
end writeback ;

```

architecture behavior of writeback is

```

begin
--just line it
rdata_out <= rdata_in ;
raddr_out <= raddr_in ;

process(clrn,writeback_cs)
begin
if clrn = '0' then
pcupdate_out <= '0' ;
rupdate_out <= '0' ;
else
if writeback_cs = '1' then
case ir(15 downto 11) is
--write back reg
when "00000" =>
rupdate_out <= '1' ;
pcupdate_out <= '0' ;
when
"00001" =>
rupdate_out <= '1' ;
pcupdate_out <= '0' ;
when "00010" =>
rupdate_out <= '0' ;
pcupdate_out <= '0' ;
when "00011" =>
rupdate_out <= '1' ;
pcupdate_out <= '0' ;
when "01000" =>
rupdate_out <= '1' ;
pcupdate_out <= '0' ;
when "01001" =>
rupdate_out <= '1' ;
pcupdate_out <= '0' ;
when
"01010" =>
rupdate_out <= '1' ;
pcupdate_out <= '0' ;
when "01011" =>
rupdate_out <= '1' ;
pcupdate_out <= '0' ;
when
"01100" =>
rupdate_out <= '1' ;
pcupdate_out <= '0' ;
when "01101" =>
rupdate_out <= '1' ;

```

```

pcupdate_out <= '0' ;                                when
"01110" =>                                           rupdate_out <= '1' ;
                                           pcupdate_out <= '0' ;

                                when "01111" =>
rupdate_out <= '1' ;
pcupdate_out <= '0' ;                                --write back
pc                                           when "10100" =>
                                rupdate_out <= '0' ;

pcupdate_out <= '1' ;                                when
"10101" =>                                           rupdate_out <= '0' ;
                                if(z_in = '1') then
pcupdate_out <= '1' ;
                                else
pcupdate_out <= '0' ;
                                end if ;
                                when "10110" =>
rupdate_out <= '0' ;                                if cy_in
= '1' then                                           pcupdate_out <= '1' ;
                                else
pcupdate_out <= '0' ;
                                end if ;
                                --write back reg
when "10111" =>                                           rupdate_out <= '1' ;
                                pcupdate_out <= '0' ;
when "11000" =>                                           rupdate_out <= '1' ;
                                pcupdate_out <= '0' ;
                                when "11001" =>
rupdate_out <= '1' ;
pcupdate_out <= '0' ;                                when
others =>                                           rupdate_out
<= '0' ;                                           pcupdate_out
<= '0' ;                                end case ;
else
                                --when the cs has been 0 ,the update signal should be clear
                                rupdate_out <= '0' ;
                                pcupdate_out <= '0' ;
end if ;
end if ;
end process ;

process(clrn,ir,writeback_cs)
begin
    if(clrn = '0') then
pcnew_out <= "1111111111111111" ;
        elsif( clrn = '1' and writeback_cs = '1') then
pcnew_out <= pcnew_in ;
    end if ;
end process ;

```

```
        end process ;
end behavior ;
```

f) memctrl.vhd

```
library ieee ; use
ieee.std_logic_1164.all ;

entity memctrl is port(
    -- clrn : in std_logic ; --from 'fetch'
    moudle f_nBLE,f_nBHE: in std_logic ;
    f_nMREQ,f_nRD,f_nWR : in
    std_logic ; pc_in : in
    std_logic_vector(15 downto 0) ;
    ir_out : out std_logic_vector(15
    downto 0) ;
    --from alu addr_in : in std_logic_vector(15
    downto 0) ; --from 'readwrite' moudle
    rw_nBLE,rw_nBHE: in std_logic ;
    rw_nMREQ,rw_nRD,rw_nWR: in std_logic ;
    rtemp_in : in std_logic_vector(15 downto 0) ;
    rtemp_out : out std_logic_vector(15 downto 0) ;
    --select state
    T0: in std_logic ;
    T2: in std_logic ;
    --out
    nMREQ,nBHE,nBLE,nWR,nRD : out std_logic ;
    MAR : out std_logic_vector(15 downto 0) ;
    MDR : inout std_logic_vector(15 downto 0)
);
end memctrl ;

architecture behavior of memctrl is

begin
    process(clrn,T0,T2,pc_in,f_nMREQ,f_nRD,rw_nRD,rw_nWR)
    begin
        if(clrn = '1' and T0 = '1' and T2 = '0' and f_nRD = '0') then
            nMREQ <= '0' ;
            nRD <= '0' ;
            nWR <= '1' ;          nBHE <=
            '0' ;          nBLE <= '0' ;
            elsif(clrn = '1' and T0 = '0' and T2 = '1' and rw_nWR = '0') then
                nMREQ <= '0' ;
                nRD <= '1' ;
                nWR <= '0' ;          nBHE <=
                '0' ;          nBLE <= '0' ;
                elsif(clrn = '1' and T0 = '0' and T2 = '1' and rw_nRD = '0') then
```

```

        nMREQ <= '0' ;

nRD <= '0' ;

        nWR <= '1' ;

        nBHE <= '0' ;

        nBLE <= '0' ;

    else

        nMREQ <= '1' ;

        nRD <= '1' ;

        nWR <= '1' ;

        nBHE <= '1' ;

        nBLE <= '1' ;

    end if ; end

process ;

process(clrn,T0,T2,pc_in,f_nMREQ,f_nRD,rw_nRD,rw_nWR)

begin

    if(clrn = '1' and T0 = '1' and T2 = '0' and f_nRD = '0') then

        MDR <= "ZZZZZZZZZZZZZZZZ";

    ir_out <= MDR ;

        elsif(clrn = '1' and T0 = '0' and T2 = '1' and rw_nWR = '0') then

            MDR <= rtemp_in ;

        elsif(clrn = '1' and T0 = '0' and T2 = '1' and rw_nRD = '0') then

            MDR <= "ZZZZZZZZZZZZZZZZ" ;

    rtemp_out <= MDR ;

        else

            MDR <= "ZZZZZZZZZZZZZZZZ" ;

    end if ;

    end process ;

process(clrn,T0,T2,pc_in,f_nMREQ,f_nRD,rw_nRD,rw_nWR)

begin

    if(clrn = '1' and T0 = '1' and T2 = '0' and f_nRD = '0') then

        MAR <= pc_in ;

    elsif(clrn = '1' and T0 = '0' and T2 = '1' and rw_nWR = '0') then

        MAR <= addr_in ;

    elsif(clrn = '1' and T0 = '0' and T2 = '1' and rw_nRD = '0') then

        MAR <= addr_in ;

    else

        MAR <= "ZZZZZZZZZZZZZZZZ" ;

    end if ;

    end process ;

end behavior ;

```

2. UCF 文件

#First , line the memory and the cpu_xx NET

"nMREQ" LOC = "P168" ;

NET "nBHE" LOC = "P138" ;

NET "nBLE" LOC = "P137" ;

NET "nWR" LOC = "P152" ;

NET "nRD" LOC = "P139" ;

#ABUS

NET "ABUS<0>" LOC = "p179" ;

NET "ABUS<1>" LOC = "p178" ;

NET "ABUS<2>" LOC = "p177" ;

NET "ABUS<3>" LOC = "p172" ;

#

NET "ABUS<4>" LOC = "p171" ;

NET "ABUS<5>" LOC = "p151" ;

NET "ABUS<6>" LOC = "p150" ;

NET "ABUS<7>" LOC = "p147" ;

#

NET "ABUS<8>" LOC = "p146" ;

NET "ABUS<9>" LOC = "p113" ;

NET "ABUS<10>" LOC = "p115" ;

NET "ABUS<11>" LOC = "p116" ;

#

NET "ABUS<12>" LOC = "p119" ;

NET "ABUS<13>" LOC = "p140" ;

NET "ABUS<14>" LOC = "p144" ;

NET "ABUS<15>" LOC = "p145" ;

#

#DBUS

NET "DBUS<0>" LOC = "p167" ;

NET "DBUS<1>" LOC = "p165" ;

NET "DBUS<2>" LOC = "p164" ;

NET "DBUS<3>" LOC = "p163" ;

NET "DBUS<4>" LOC = "p162" ;

NET "DBUS<5>" LOC = "p161" ;

NET "DBUS<6>" LOC = "p160" ;

NET "DBUS<7>" LOC = "p153" ;

NET "DBUS<8>" LOC = "p120" ;

NET "DBUS<9>" LOC = "p122" ;

NET "DBUS<10>" LOC = "p123" ;

NET "DBUS<11>" LOC = "p128" ;

NET "DBUS<12>" LOC = "p132" ;

NET "DBUS<13>" LOC = "p133" ;

NET "DBUS<14>" LOC = "p134" ;

NET "DBUS<15>" LOC = "p135" ;

#clk

NET "clk" LOC = "p75" ;

#clrn

```
NET "clrn" LOC = "p51" ;

#cy,z

NET "t_cy" LOC = "p187" ;

NET "t_z" LOC = "p186" ;

#beat

NET "t_beat<3>" LOC = "p196" ;

NET "t_beat<2>" LOC = "p193" ;

NET "t_beat<1>" LOC = "p192" ;

NET "t_beat<0>" LOC = "p190" ;

#out MEMCTRL

#NET "nMREQ" LOC = "p185" ;

#NET "nWR" LOC = "p77" ;

#NET "nRD" LOC = "p82" ;

#NET "nBHE" LOC = "p83" ;

#NET "nBLE" LOC = "p98" ;

#OUT PC

NET "t_pc<0>" LOC = "p60" ;

NET "t_pc<1>" LOC = "p61" ;

NET "t_pc<2>" LOC = "p62" ;

NET "t_pc<3>" LOC = "p63" ;

NET "t_pc<4>" LOC = "p2" ;

NET "t_pc<5>" LOC = "p108" ;

NET "t_pc<6>" LOC = "p109" ;

NET "t_pc<7>" LOC = "p112" ;

NET "t_pc<8>" LOC = "p126" ;

NET "t_pc<9>" LOC = "p127" ;

NET "t_pc<10>" LOC = "p129" ;

NET "t_pc<11>" LOC = "p202" ;

NET "t_pc<12>" LOC = "p203" ;

NET "t_pc<13>" LOC = "p205" ;

NET "t_pc<14>" LOC = "p206" ;

NET "t_pc<15>" LOC = "p103" ;

#out ir

NET "t_ir<0>" LOC = "p31" ;

NET "t_ir<1>" LOC = "p33" ;

NET "t_ir<2>" LOC = "p34" ;

NET "t_ir<3>" LOC = "p35" ;

NET "t_ir<4>" LOC = "p36" ;

NET "t_ir<5>" LOC = "p39" ;

NET "t_ir<6>" LOC = "p40" ;

NET "t_ir<7>" LOC = "p41" ;

NET "t_ir<8>" LOC = "p42" ;

NET "t_ir<9>" LOC = "p45" ;

NET "t_ir<10>" LOC = "p47" ;

NET "t_ir<11>" LOC = "p48" ; NET "t_ir<12>" LOC = "p49" ;

NET "t_ir<13>" LOC = "p50" ;

NET "t_ir<14>" LOC = "p55" ;
```

```

NET "t_ir<15>" LOC = "p56" ;

#out mdr

NET "t_DBUS<0>" LOC = "p4" ;
NET "t_DBUS<1>" LOC = "p5" ;
NET "t_DBUS<2>" LOC = "p8" ;
NET "t_DBUS<3>" LOC = "p9" ;
NET "t_DBUS<4>" LOC = "p11" ;
NET "t_DBUS<5>" LOC = "p12" ;
NET "t_DBUS<6>" LOC = "p15" ;
NET "t_DBUS<7>" LOC = "p16" ;
NET "t_DBUS<8>" LOC = "p18" ;
NET "t_DBUS<9>" LOC = "p19" ;
NET "t_DBUS<10>" LOC = "p22" ;
NET "t_DBUS<11>" LOC = "p23" ;
NET "t_DBUS<12>" LOC = "p24" ;
NET "t_DBUS<13>" LOC = "p25" ;
NET "t_DBUS<14>" LOC = "p28" ;
NET "t_DBUS<15>" LOC = "p29" ;

```

3. 测试指令序列

pc 值	指令	16 进制编码	含义	期望结果
00	MOV R0 , \$10101010	08AA	立即数 AA 送寄存器 R0	R0 = 00AA
01	MOV R7 , R0	0700	R0 -> R7	R7 = 00AA
02	MOV [00010000] ,R0	1080	R0 -> 内存地址 AA10	ABUS = AA10 DBUS = 00AA
03	MOV R1 , [00010000]	1910	内存 AA10 到 R0	ABUS = AA10 DBUS = 00AA
04	MOV [00000000],R1	1001	R1 -> 内存地址 AA00	ABUS = AA00 DBUS = 00AA
05	ADC R1 , R0	4100	R1 + R0 + cy -> R1	R1 = 0154 Cy = 0
06	MOV [00000001] , R1	1009	R1 -> M[AA01]	ABUS = AA01 DBUS = 0154
07	SBB R1 , R0	5100	R1 - R0- cy -> R1	R1 = 00AA Cy = 0
08	MOV [00000010] , R1	1011	R1 -> M[AA02]	ABUS = AA02 DBUS = 00AA
09	LR R1	C900	交换 R1 高低 8 位	R1 = AA00
0A	LR R0	C800	交换 R0 高低 8 位	R0 = AA00

0B	ADC R1 , R0	4100	R1 + R0 + cy -> R1	R1 = 5400 Cy = 1
0C	MOV [00000011] , R1	1019	R1 -> M[AA03]	ABUS = AA03 DBUS = 5400
0D	SBB R1 , R0	5100	R1 - R0 - cy -> R1	R1 = A9FF Cy = 1
0E	MOV [00000100] , R1	1021	R1 -> M[AA04]	ABUS = AA04 DBUS = A9FF
0F	MOV R2 , \$00000000	0A00	0 -> R2	R2 = 0
10	OR R1 , R2	7102	R1 or R2 -> R1	R1 = A9FF
11	MOV [00000101] , R1	1029	R1 -> M[AA05]	ABUS = AA05 DBUS = A9FF
12	AND R1 , R2	6102	R1 and R2 -> R1	R1 = 0 Z = 1
13	MOV [00000110] , R1	1031	R1 -> M[AA06]	ABUS = AA06 DBUS = 0000
14	ADC R1 , \$00000001	4901	R1 + 001 + CY -> R1	R1 = 1 Z = 0
15	SBB R1 , \$00000001	5901	R1 - 0001 - CY -> R1	R1 = 0 Z = 1
16	OR R1 , \$11111111	79FF	R1 OR 00FF -> R1	R1 = 00FF Z = 0
17	AND R1 , \$00000000	6900	R1 AND 0000 -> R1	R1 = 0 Z = 1
18	STC	8800	1 -> CY	CY = 1
19	JC \$00000001	B001	PCNEW -> PC	PC = 001B
1A	JMP \$00011101	A01C	PCNEW -> PC	PC = 001C
1B	JZ \$11111110	A8FE	PCNEW -> PC	PC = 001A
1C	CLC	8000	0 -> CY	CY = 0
1D	ADC R1 , \$00000001	4901	R1+0001 + CY -> R1	R1 = 0001 Z = 0
1E	JC \$00000001	B001	PCNEW -> PC	不影响 PC
1F	JZ \$00000001	A801	PCNEW -> PC	不影响 pc
20	MOV [00100000] , R1	1101	R1 -> M[AA20]	ABUS = AA20 DBUS = 0001
21	MOV R3 , R7	0307	R7 -> R3	R3 = 00AA
22	LR R3	CB00	R3 高低八位 互换	R3 = AA00
23	ADC R3 , \$00100000	4B20	R3 + 0030 + CY -> R3	R3 = AA30

24	MOVMRR R6 , R3	BE03	M{[R3]} -> R6	ABUS = AA20 DBUS = 0001 R6 = 0001
25	MOVMRIX R4,\$00101111	C42F	M{[R6]+002F} -> R4	ABUS = 0030 DBUS = AAAA
26	MOV [00100001] , R4	110C	R4 -> M[AA21]	ABUS = AA21 DBUS = AAAA

{注：需要在 0030 地址处写入数据 AAAA }

七、实验总结，在调试和下载过程中遇到的问题

在设计小波形的时候，一直在担心当把地址、控制信号送内存之后，MDR 不能立即给出值，因此在做读内存时单独拿出一个 process (MDR) 来写，导致访存控制模块特别混乱；另一个地方是设计的时候，由取指模块送出 IR，即是，在取指这个节拍，需要做：取指模块送 pc 到访存控制、送控制信号到访存控制，访存控制将 pc 送 MAR，访存控制信号送内存，在 MDR 变化是将 MDR 送取指模块 IR，取指模块送出 IR 到其他 3 个需要 IR 的模块。这在小波形的时候没有问题，但到了合成做大波形，第一条指令就执行不下去，MAR 只能输出第一个 0000，而且控制信号全部为 1，根本没有送出。于是开始改，弄了一下午，去掉了 procee(MDR)，直接就把 MDR 送出了，同时改了控制信号送出的逻辑，不再传递取指模块送出的控制信号，而是根据送出的信号判读是读是写，然后直接送出 0、1，这下每次取指控制信号能输出了，但是指令执行还是不行，IR 依旧是 UUUU，一条指令都不能执行。晚上回来向同学请教，一语道破天机，IR 不对可定不能执行啊！我问了他们 IR 怎么处理的，原来是不在取指模块送出的，而在访存控制模块。第二天改了大波形的元件例化代码，终于没有问题了。第一条指令能执行后，后面的指令测试都没有问题，于是尝试编写 ucf 准备下载。

下载生成时有问题，发现是管脚冲突，改了管脚定义后生成成功，下载到试验台上。当清零信号无效时，按下第一个时钟，pc 变成了 FFFF，额，有点不知所措。尝试又生成下载一次，还是如此，确定是代码问题。再次向同学请教，他们说在 pc 更新的时候加上 clk 信号，即只在 clk 上升沿才更新 pc，照此改了之后，问题没有得到解决。开始怀疑 pc 在取指模块即要送出，又要+1，是否存在问题。更同学交流，发现他们有的是单独增加了一个节拍做 pc+1，有个第一拍就从回写做起，每次都在回写模块回写 pc。于是决定改 pc+1 的方式，但是加节拍改动太大，在回写模块回写 pc 也需要改逻辑，想到老师之前说的在运算模块做 pc 加 1，觉得改动很小，确定在运算模块做 pc+1 并回写。由于在回写模块有回写 pc，最开始时两种回写就共用一组信号，无效时另一模块设为高阻，这样仿真没有问题。当再次下载时，依旧是 FFFF，怀疑在实际中 (if puupdate = 'Z' then) 这样写是无效的。于是再次改写取指模块，设了两组更新 pc 的信号，无效时 pcupdate 就设为 0，不再置高阻。还好，最后这样写是可行的。其余代码页只遇到了一点问题，就是 A + B + Cy 的时候，若 A + B + Cy 在此次加完后有进位，那么结果就会比预期值多 1，即是将新的 Cy 加上了，这个问题，最开始在波形仿真是就有问题，后来限制只在运算节拍才输出 aluout 时波形变得正确，但是在下载后却不行了，仿真和下载真的不一样。我在运算模块，将原来的电平触发改为边沿触发，结果终于正确了，也没有对其他指令造成影响，最后下载成功。