实验报告

智能 1602　　邱勒铭　　201608010702

实验名称

相对简单 CPU 电路设计

实验目标

利用 VHDL 设计相对简单 CPU 的电路并验证

实验要求

采用 VHDL 描述电路及其测试平台

采用时序逻辑设计电路

采用从 1 累加到 4 的程序进行测试

实验内容

相对简单 CPU 的规格说明

地址总线 16 位，数据总线 8 位

有一个 8 位累加寄存器 AC，一个 8 位通用寄存器 R，一个 1 位的零标志

有一个 16 位 AR 寄存器，一个 16 位程序计数器 PC，一个 8 位数据寄存器 DR，一个 8 位指令寄存器 IR，一个 8 位临时寄存器 TR

有 16 条指令，每条指令 1 个或 3 个字节，其中操作码 8 位。3 字节的指令有 16 位的地址

相对简单 CPU 设计方案

1. 指令执行过程分为取指、译码、执行三个阶段
2. 取指包括三个状态，FETCH1，FETCH2，FETCH3
3. 译码体现为从 FETCH3 状态到各指令执行状态序列的第一个状态

4. 执行根据指令的具体操作分为若干状态

5. 执行的最后一个状态转移到 FETCH1 状态

6. 控制器根据每个状态需要完成的操作产生相应的控制信号

CPU 设计内容

增加了 FETCH4 状态一共四个状态。取址包括四个状态，而译码体现为从 FETCH4 状态

CPU 的端口：

```
entity cpu is
    port(
            clk: in std_logic;
            reset: in std_logic;
            addrbus: out std_logic_vector(15 downto 0);
            databus: inout std_logic_vector(7 downto 0);
            read: out std_logic;
            write: out std_logic
        );
end entity;
```

总体包含时钟信号，16 位地址总线和 8 位数据总线，以及命令内存的 read 和 write。

由于 CPU 是时序电路，我们需要知道当前状态和下一状态 CPU 的工作情况，因此要做好状态转移的工作。

首先，CPU 的地址总线和数据总线，以及 ALU 是非时序部分，将其单独写出：

```
addrbus <= ar;
databus <= thebus(7 downto 0) when write1='1' else "ZZZZZZZZ";

--the bus
thebus<=pc                      when pcbus='1'      else    "ZZZZZZZZZZZZZZZZ";
thebus<="00000000"&databus      when membus='1'     else    "ZZZZZZZZZZZZZZZZ";
thebus<="00000000"&r            when rbus='1'       else    "ZZZZZZZZZZZZZZZZ";
thebus<="00000000"&ac           when acbus='1'      else    "ZZZZZZZZZZZZZZZZ";
thebus<=dr&tr                   when(trbus='1' and drbus='1')   else    "ZZZZZZZZZZZZZZZZ";
thebus<="00000000"&dr           when (drbus='1' and trbus/='1')     else    "ZZZZZZZZZZZZZZZZ";

--alu
aluResult<=thebus(7 downto 0)                                       when s="0000"   else
std_logic_vector(unsigned(ac)+unsigned(thebus(7 downto 0)))         when s="0001"   else
std_logic_vector(unsigned(ac)-unsigned(thebus(7 downto 0)))         when s="0010"   else
ac and thebus(7 downto 0)                                           when s="0011"   else
ac or thebus(7 downto 0)                                            when s="0100"   else
ac xor thebus(7 downto 0)                                           when s="0101"   else
not ac                                                              when s="0110"   else
thebus(7 downto 0);
```

之后是寄存器组，包括：程序计数器 pc，地址寄存器 ar，数据寄存器 dr，指令寄存器 ir，通用寄存器 r，算术运算的专用寄存器 ac，它们都是时序的，将其列为一组：

```
update_regs: process(clk)
begin
    if(rising_edge(clk)) then
        --update registers
        if(pcinc='1')   then    pc<=std_logic_vector(unsigned(pc) + 1);
        end if;
        if(acinc='1')   then    ac<=std_logic_vector(unsigned(ac) + 1);
            if(ac="11111111")   then    z<='1';
            else    z<='0';
            end if;
        end if;
        if(arinc='1')   then    ar<=std_logic_vector(unsigned(ar) + 1);
        end if;

        if(arload='1')  then    ar<=thebus;
        end if;
        if(drload='1')  then    dr<=thebus(7 downto 0);
        end if;
        if(irload='1')  then    ir<=dr;
        end if;
        if(acload='1')  then
            ac<=aluResult;
            if(s="0001" or s="0010" or s="0011" or s="0100" or s="0101" or s="0110")    then
                if(aluResult="00000000")    then z<='1';
                else    z<='0';
                end if;
            end if;
        end if;
        if(rload='1')   then    r<=thebus(7 downto 0);
        end if;
        if(trload='1')  then    tr<=dr;
        end if;
        if(pcload='1')  then    pc<=thebus;
        end if;

        if(acreset='1') then
            ac<="00000000";
            z<='0';
        end if;

        if(reset='1') then
            pc <= "0000000000000000";
            state <= fetch1;
        else
            state <= nextstate;
        end if;

    end if;

end process update_regs;
```

最后是状态转移组：

```
for_nextstate: process(state, ir, z)
begin

    if(state=fetch1)    then    nextstate<=fetch2;
    elsif(state=fetch2) then    nextstate<=fetch3;
    elsif(state=fetch3) then    nextstate<=fetch4;
    elsif(state=fetch4) then
```

（部分）

有了这些具体的转移逻辑和单个部件的控制关系，我们还需建立一个控制器来完

成由指令到具体操作的转换：

```
gen_controls: process(state)
begin
    if(state=fetch1)    then
    arload<='1';pcbus<='1';pcinc<='0';drload<='0';membus<='0';irload<='0';acreset<='0';acinc<='0';--ar<=pc
    rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
    read<='0';write1<='0';write<='0';pcload<='0';
    elsif(state=fetch2) then
    arload<='0';pcbus<='0';pcinc<='1';drload<='1';membus<='1';irload<='0';acreset<='0';acinc<='0';-- dr<=m  pc<=pc+1
    rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
    read<='1';write1<='0';write<='0';pcload<='0';
    elsif(state=fetch3) then
    arload<='0';pcbus<='0';pcinc<='0';drload<='0';membus<='0';irload<='1';acreset<='0';acinc<='0';--ir<=dr
    rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
    read<='0';write1<='0';write<='0';pcload<='0';
    elsif(state=fetch4) then
    arload<='1';pcbus<='1';pcinc<='0';drload<='0';membus<='0';irload<='0';acreset<='0';acinc<='0';--ar<=pc
    rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
    read<='0';write1<='0';write<='0';pcload<='0';
```

（局部）

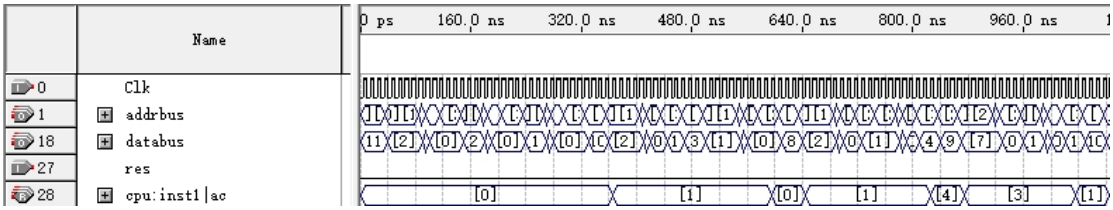
内存设计内容

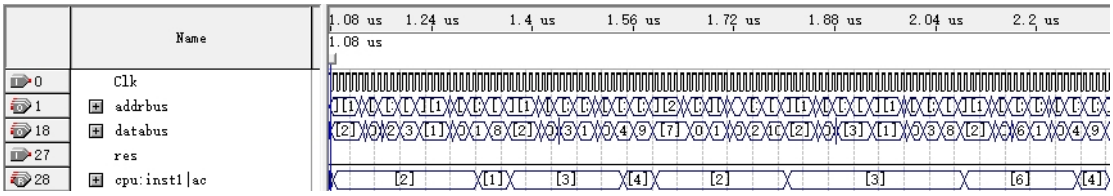
内存中每个地址需要给出具体的指令，以便读取：

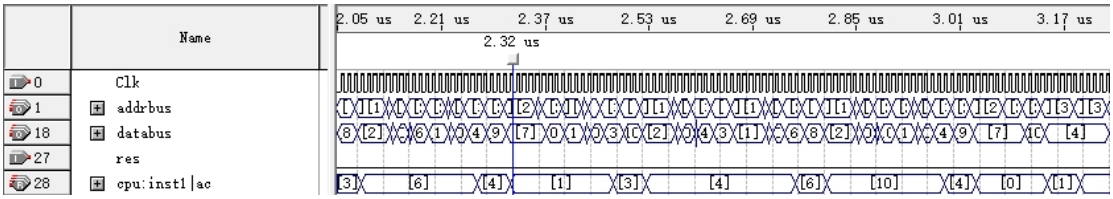
```
signal memdata: memtype(4095 downto 0) := (
0 => RSCLAC,

1 => RSSTAC,--m[total_total]<=ac    total=0
2 => std_logic_vector(to_unsigned(total_addr, 8)),
3 => X"00",

4 => RSSTAC,--m[i_addr]<=ac    i=0
5 => std_logic_vector(to_unsigned(i_addr, 8)),
6 => X"00",

7 => RSLDAC,  -- loop    --ac<=m[i_addr]    ac=i
8 => std_logic_vector(to_unsigned(i_addr, 8)),
9 => X"00",

10 => RSINAC, --ac++

11 => RSSTAC, --i=ac
12 => std_logic_vector(to_unsigned(i_addr, 8)),
13 => X"00",

14 => RSMVAC, --r=ac

15 => RSLDAC, --ac=total
16 => std_logic_vector(to_unsigned(total_addr, 8)),
17 => X"00",

18 => RSADD, --ac=ac+r

19 => RSSTAC, --total=ac
20 => std_logic_vector(to_unsigned(total_addr, 8)),
21 => X"00",

22 => RSLDAC, --ac=n
23 => std_logic_vector(to_unsigned(n_addr, 8)),
24 => X"00",

25 => RSSUB, --ac=ac-r

26 => RSJPNZ, --
27 => std_logic_vector(to_unsigned(loop_addr, 8)),
28 => X"00",

29 => X"00",  -- total
30 => X"00",  -- i
31 => "00000100",  -- n
others => RSNOP
```

仿真结果

可以看到仿真结果是正确的，到了 0x24 就不再变化了也就是 36。







由 ac 可以追踪每次累加和加数，可以看到在 2.85 处结果是 10，即累加结果。