实验报告

实验名称（相对简单 CPU 电路设计）

智能 1602 201608010713 张杨康

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 n 的程序进行测试

实验内容

相对简单 CPU 的设计需求

相对简单 CPU 的设计需求请详见课件，主要特征如下：

- 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

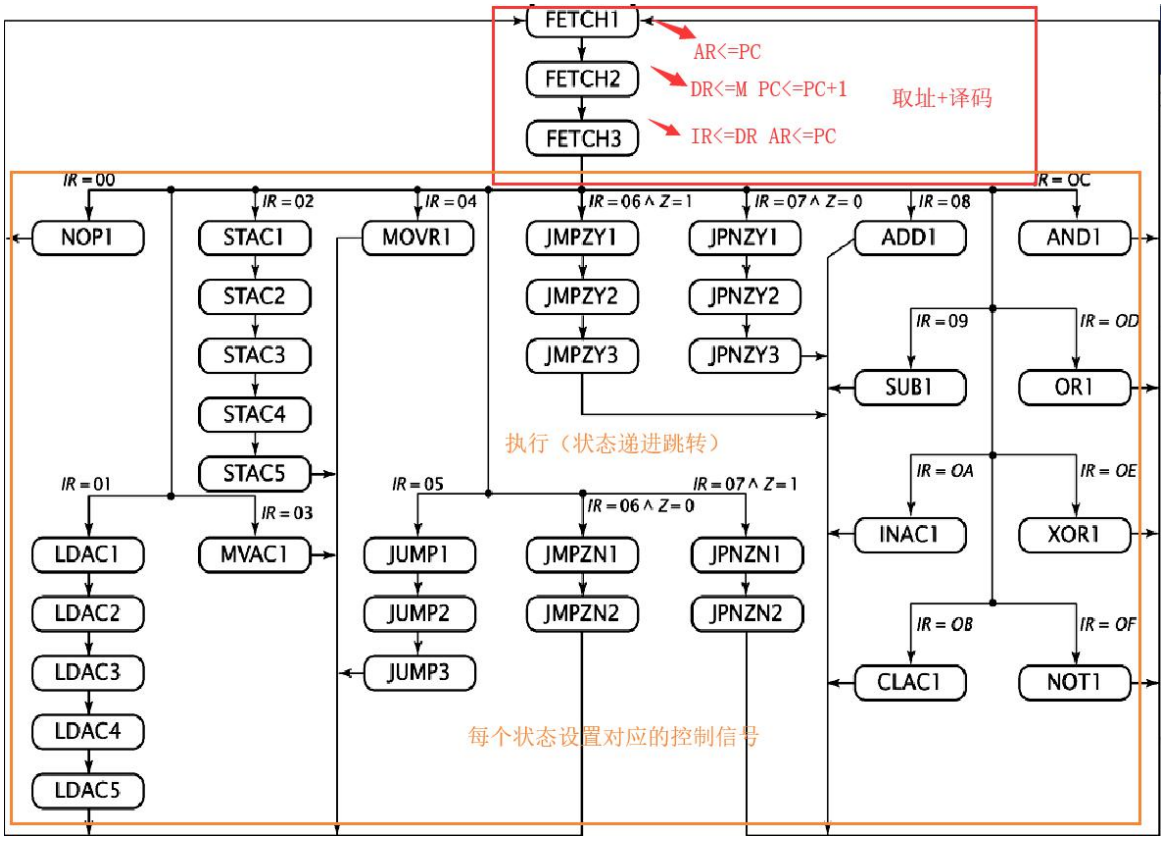
相对简单 CPU 设计方案

相对简单 CPU 的设计方案请详见课件，主要思路如下：

1. 指令执行过程分为取指、译码、执行三个阶段
2. 取指包括四个状态，FETCH1，FETCH2,FETCH3，FETCH4 （这里将取指的最后一个状态分为两个状态，为了防止译码时状态改变）
3. 译码体现为从 FETCH4 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 FETCH4 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

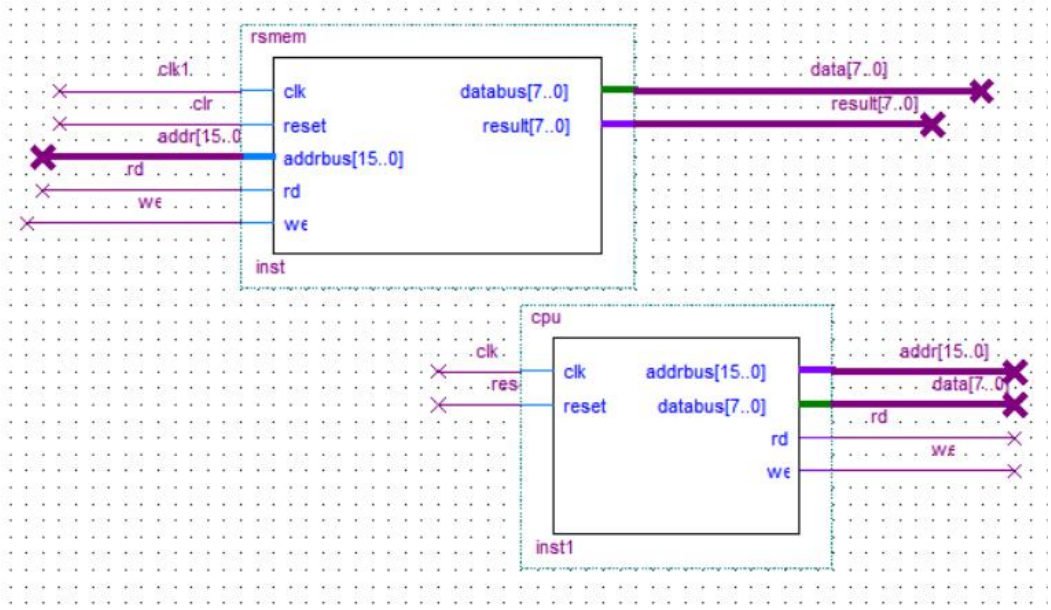
CPU 的状态图：



顶层文件：

```
component rscpu 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);
                    rd: out std_logic;
                    wr: out std_logic
            );
end component;

component rsmem is
        port(
                    clk: in std_logic;
                    reset: in std_logic;
                    addrbus: in std_logic_vector(15 downto 0);
                    databus: inout std_logic_vector(7 downto 0);
                    rd: in std_logic;
                    wr: in std_logic
            );
```

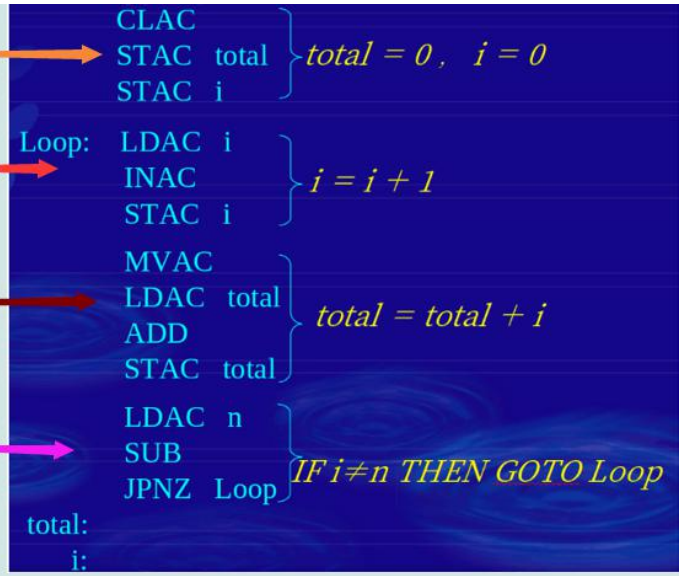


模块设计--存储器模块

1）程序装载

```
0 => RSCLAC,
1 => RSSTAC,--m<=ac    total=0
2 => std_logic_vector(to_unsigned(total_addr, 8)),
3 => X"00",
4 => RSSTAC,--m<=ac    i=0
6 => X"00",
7 => RSLDAC,  --    ac=i
8 => std_logic_vector(to_unsigned(i_addr, 8)),
9 => X"00",
10 => RSINAC, --ac++
11 => RSSTAC, --i=ac
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 => X"06",  -- n
others => RSNOP
```

n=6



2）内存管理（读、写）

```
begin
    -- The process takes addrbus and rd/we signals at first,
    -- then at the next clock does the data transmission.
    for_clk : process(clk)
    begin
        if(falling_edge(clk)) then
            if(reset='1') then
                addr <= (others=>'0');
            else
                addr <= addrbus;
            end if;

            if(we='1') then
                memdata(to_integer(unsigned(addr))) <= databus;
            end if;
        end if;
    end process;

    databus <= memdata(to_integer(unsigned(addr))) when (we='0') else "ZZZZZZZZ";
    result<=memdata(60);
end architecture;
```

写

读

累加结果

CPU 模块（指令处理）

1）设置初始化寄存器、控制信号、各个状态：

寄存器：

```
signal pc: std_logic_vector(15 downto 0);
signal ac: std_logic_vector(7 downto 0);
signal r: std_logic_vector(7 downto 0) ;
signal ar: std_logic_vector(15 downto 0);
signal ir: std_logic_vector(7 downto 0);
signal dr: std_logic_vector(7 downto 0);
signal tr: std_logic_vector(7 downto 0);
signal z: std_logic;
```

控制信号：

```
signal pcload: std_logic;
signal arload: std_logic;
signal drload: std_logic;
signal irload: std_logic;
signal acload: std_logic;
signal rload:  std_logic;
signal trload: std_logic;
```

```
signal pcbus: std_logic;
signal membus: std_logic;
signal rbus: std_logic;
signal acbus: std_logic;
signal trbus: std_logic;
signal drbus: std_logic;
```

状态标号：

```
constant fetch1:
constant fetch2:
constant fetch3:
constant fetch4:
constant clac1:
constant incac1:
constant add1:
constant sub1:
constant and1:
constant or1:
constant xor1:
constant not1:
constant mvac1:
constant movr1:
constant ldac1:
constant ldac2:
constant ldac3:
constant ldac4:
constant ldac5:
constant stac1:
constant stac2:
constant stac3:
constant stac4:
constant stac5:
```

2）设置 ALU 运算：

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

3）控制信号控制数据的传输：

```
if(ar_ld='1')    then    ar<=buss;
end if;
if(dr_ld='1')    then    dr<=buss(7 downto 0);
end if;
if(ir_ld='1')    then    ir<=dr;
end if;
if(r_d='1') then    r<=buss(7 downto 0);
end if;
if(tr_ld='1')    then    tr<=dr;
end if;
if(pc_ld='1')    then    pc<=buss;
end if;
```

4）设置每个状态对应的控制信号：

```
elsif(state=addl)   then
arload<='0';pcbus<='0';pcinc<='0';drload<='0';membus<='0';irload<='0';acreset<='0';acinc<='0';
rbus<='1';s<="0001";acload<='1';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
rd<='0';writel<='0';we<='0';pcload<='0';

 if(state=fetchl)     then     --ar<=pc
 arload<='1';pcbus<='1';pcinc<='0';drload<='0';membus<='0';irload<='0';acreset<='0';acinc<='0';
 rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
```

5）执行时状态间的切换：

单状态指令：

```
elsif(ir=add)          then     nextstate<=addl;
```

多状态指令：

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

elsif(state=jumpl)    then     nextstate<=jump2;
elsif(state=jump2)    then     nextstate<=jump3;
elsif(state=jump3)    then     nextstate<=fetchl;
```

测试

测试平台

相对简单 CPU 电路在如下机器上进行了测试：

部件	配置	备注
CPU	core i7-8500U	
内存	DDR4 8GB	
操作系统	Windows 10	家庭中文版
综合软件	Quartus II 9.0sp1 Web Edition	
仿真软件	Quartus II 9.0sp1 Web Edition 自带仿真器	
波形查看	Quartus II 9.0sp1 Web Edition Simulate Report	

测试输入

我们采用从 1 累加到 n 的程序作为测试输入：（这里 n 设为 9）

```
28 => X"00",
29 => X"00",   -- total
30 => X"00",   -- i
31 => X"09",   -- n
others => RSNOP
```

我们这里直接将 total 的值输出出来，通过在 rsmem 设置一个输出信号：

```
entity rsmem is
    port(
            clk: in std_logic;
            reset: in std_logic;
            addrbus: in std_logic_vector(15 downto 0);
            databus: inout std_logic_vector(7 downto 0);
            result: out std_logic_vector(7 downto 0) ;
            rd: in std_logic;
            we: in std_logic
        );
end entity;
```

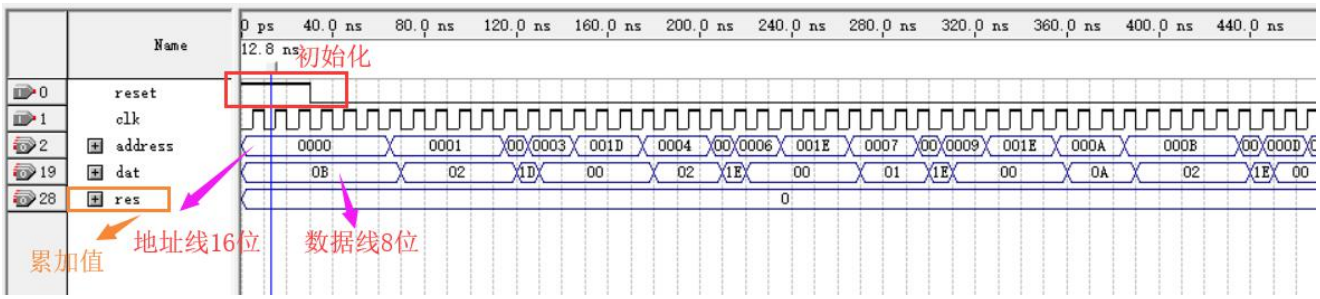
```
result<=memdata(29);
```

测试输入语句：

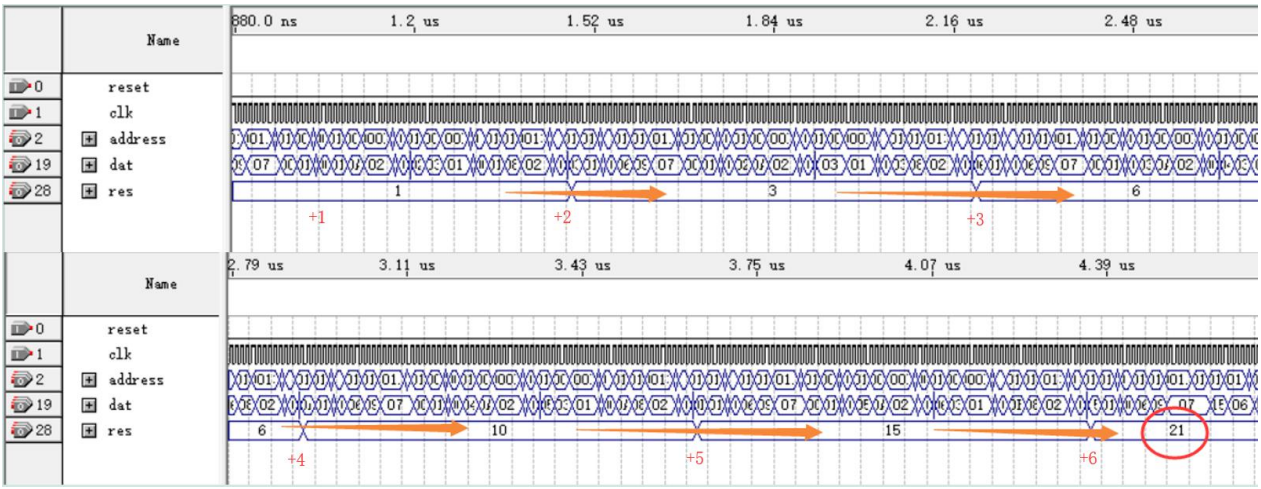
```
        CLAC
        STAC   total      } total = 0 ,   i = 0
        STAC   i
Loop:   LDAC   i
        INAC              } i = i + 1
        STAC   i
        MVAC
        LDAC   total      } total = total + i
        ADD
        STAC   total
        LDAC   n
        SUB               } IF i≠n THEN GOTO Loop
        JPNZ   Loop
total:
    i:
```

测试记录

相对简单 CPU 运行测试程序波形截图如下：



设置的 n=6，所以 total 的值变化到 21 就不再变化：



分析和结论

从测试记录来看，相对简单 CPU 实现了对测试程序指令的读取、译码和执行，得到的运算结果正确。
我们可以看到 total 的值依次为 0、1、3、6、10、15、21，且到 21 后不再发生变化。
根据分析结果，可以认为所设计的相对简单 CPU 实现了所要求的功能，完成了实验目标。