实验报告

实验名称（相对简单 CPU 电路设计）

智能 1602 201608010627 任小禹

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

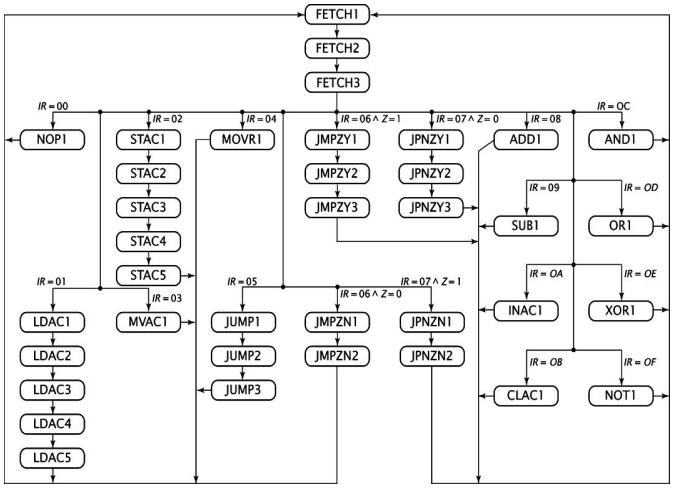
相对简单 CPU 设计方案

相对简单 CPU 的设计方案请详见课件，主要思路如下：

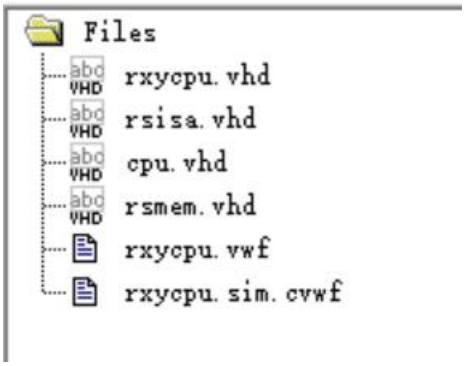
1. 指令执行过程分为取指、译码、执行三个阶段
2. 取指包括四个状态，FETCH1，FETCH2,FETCH3，FETCH4 （这里将取指的最后一个状态分为两个状态，为了防止译码时状态改变）
3. 译码体现为从 FETCH4 状态到各指令执行状态序列的第一个状态
4. 执行根据指令的具体操作分为若干状态

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：



我的设计方案分为四个文件：



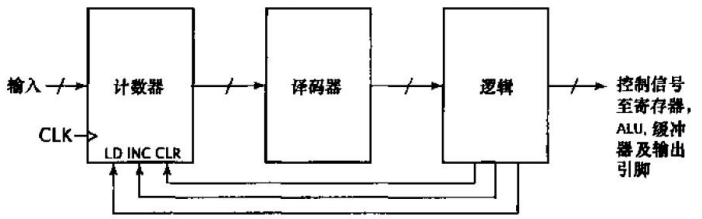
rxycpu.vhd 顶层文件 元件连接 CPU 与内存相连

rsisa.vhd 将指令集打包成库

cpu.vhd CPU 的具体实现

rsmem.vhd 内存的实现，读写功能和输入激励实现

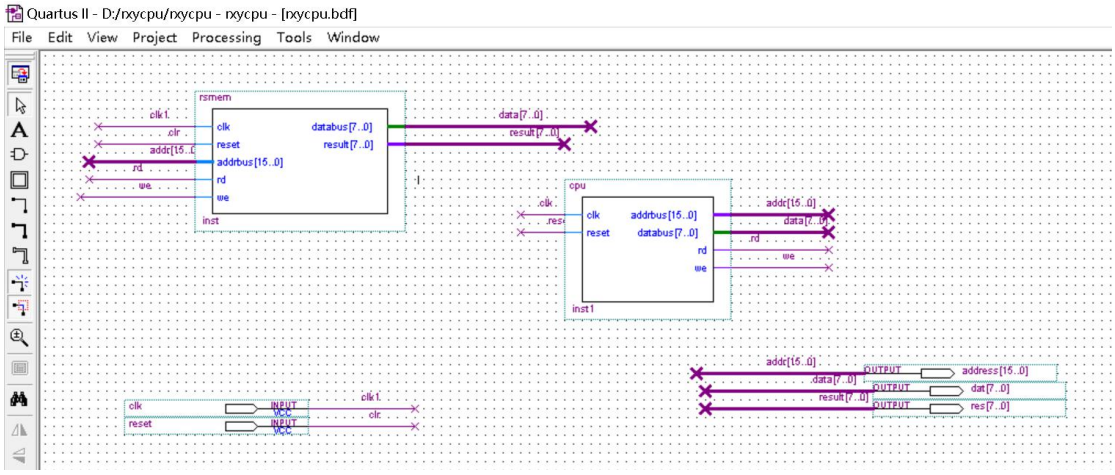
- CPU 的控制单元采用硬布线控制：



cpu.vhd 具体实现
一个进程 产生下一状态
一个进程 根据当前状态产生控制信号
一个进程 跟时序有关的操作，根据控制信号的值进行相应的操作

其余部分 与时序无关的操作，根据控制信号的值进行相应的操作

- 顶层文件也用 BDF 原理图进行了实现：rxycpu.bdf



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);
```

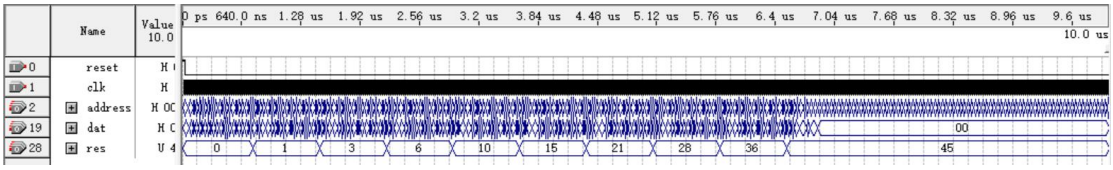
测试输入语句：

```
        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:
```

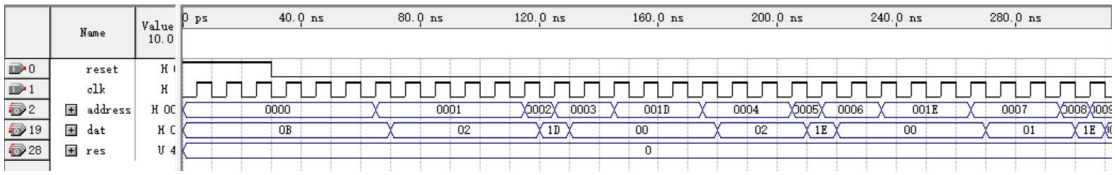
测试记录

相对简单 CPU 运行测试程序波形截图如下：

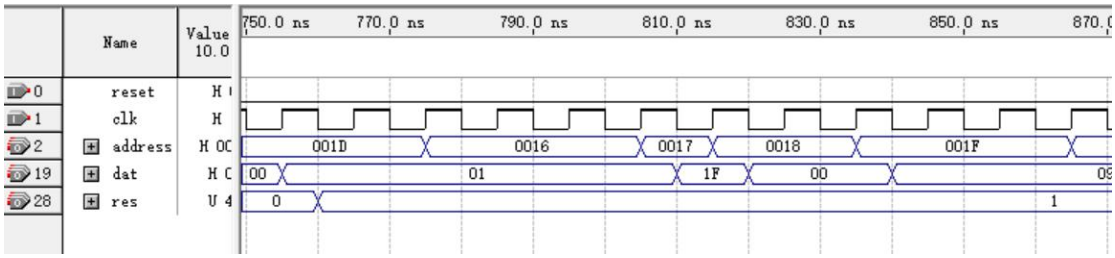
设置的 n 为 9，所以 total 的值变化到 45 就不再变化。



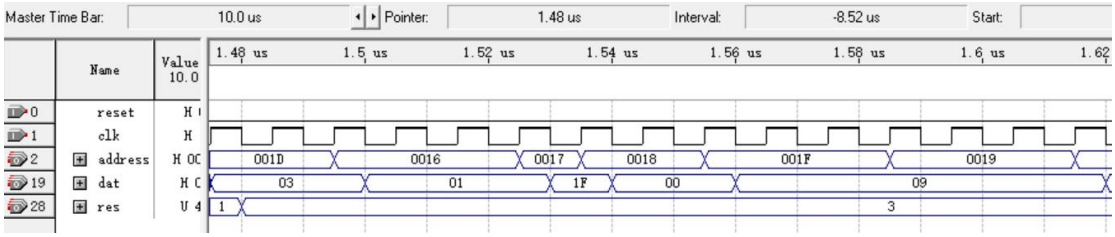
0：


Name
Value 10.0
0 ps
40.0 ns
80.0 ns
120.0 ns
160.0 ns
200.0 ns
240.0 ns
280.0 ns
reset
clk
address
dat
res
0000
0001
0002
0003
001D
0004
0005
0006
001E
0007
0008
0B
02
1D
00
02
1E
00
01
1E
0

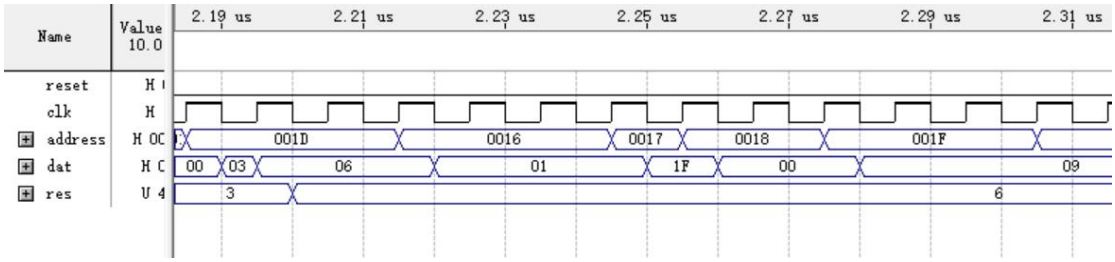
1:


Name
Value 10.0
750.0 ns
770.0 ns
790.0 ns
810.0 ns
830.0 ns
850.0 ns
reset
clk
address
dat
res
001D
0016
0017
0018
001F
00
01
1F
00
0
1

3:

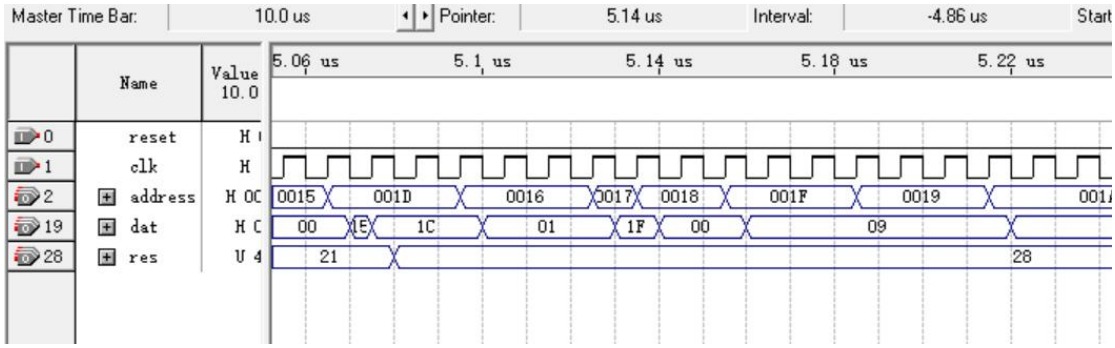

Master Time Bar:
10.0 us
Pointer:
1.48 us
Interval:
-8.52 us
Start:
Name
Value 10.0
1.48 us
1.5 us
1.52 us
1.54 us
1.56 us
1.58 us
1.6 us
reset
clk
address
dat
res
001D
0016
0017
0018
001F
0019
03
01
1F
00
09
1
3

6:

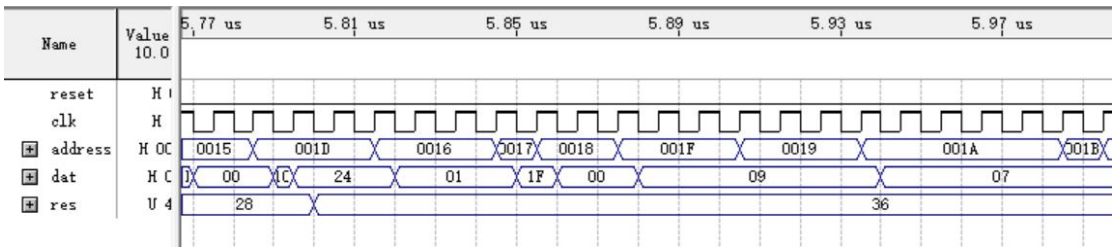

Name
Value 10.0
2.19 us
2.21 us
2.23 us
2.25 us
2.27 us
2.29 us
2.31 us
reset
clk
address
dat
res
001D
0016
0017
0018
001F
00
03
06
01
1F
00
09
3
6

...

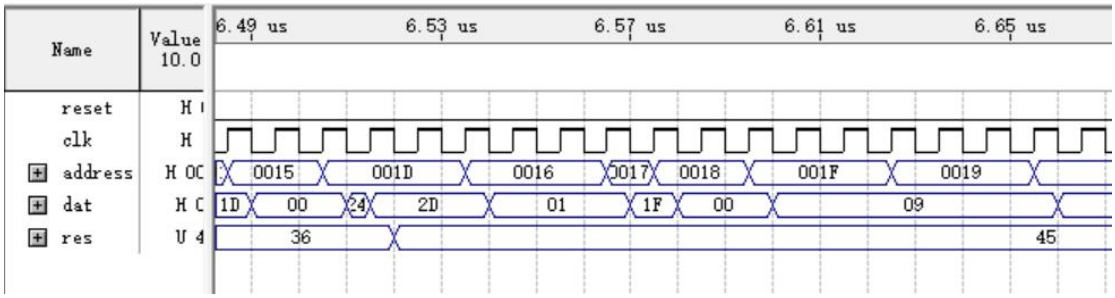
28


Master Time Bar:
10.0 us
Pointer:
5.14 us
Interval:
-4.86 us
Start
Name
Value 10.0
5.06 us
5.1 us
5.14 us
5.18 us
5.22 us
reset
clk
address
dat
res
0015
001D
0016
0017
0018
001F
0019
00
1C
01
1F
00
09
21
28

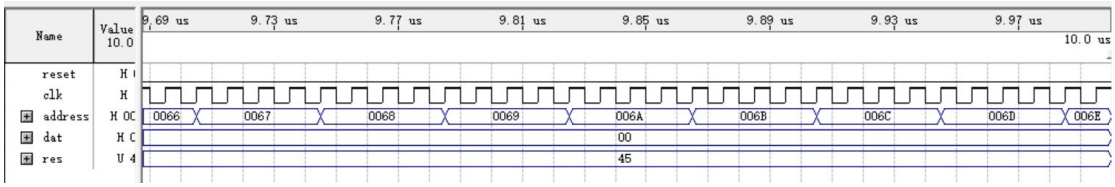
36



45



45 不再发生变化



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、28、36、45，且到 45 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。