电子科技大学通信学院

《数字电路》课程实验
实验 2：CPU 设计
----实验过程与结果分析报告

选课编号：K0166210.02　组长：姓名（周子涵）选课号（53）

2019 年 11 月

一、实验目的

1.掌握简化 CPU 的结构组成、模块划分和工作原理。
2.掌握精简指令集的概念，并设计 CPU 的指令集。
3.用 Verilog 语言设计编写并调试 CPU 内部各模块电路，并在此基础上设计完整 CPU。
4.掌握激励文件的编写方法，学会验证各个模块功能时序的正确性。
5.掌握外围接口电路的设计方法，并学会利用 FPGA 开发板的按键、LED 等外设构造简单的控制系统。
6.编写汇编程序，并依据设计的指令集编译为对应的机器码，控制 CPU 实现特定功能。

二、实验内容

任务1：基础模块（ALU、寄存器阵列、寻址单元）的建模及仿真

任务2：数据通路的设计

任务3：控制通路的设计

任务4：CPU的整合及验证

任务5：XXXXXXXXXXX（如果有做选做内容可以填入）

三、实验设备

硬件： DE1-SOC 实验板，计算机一台
软件： PC 机操作系统 WinXP 或 Win7、Quartus II 13.1、ModelSim-Altera 10.1d

四、实验原理

任务 1

目标: 掌握 CPU 的结构组成、模块划分和工作原理

掌握精简指令集的概念, 并设计 CPU 的指令集

用 Verilog 语言设计编写并调试 CPU 内部各模块电路, 并在此基础上设计完整

CPU

掌握激励文件的编写方法, 学会验证各个模块功能时序的正确性

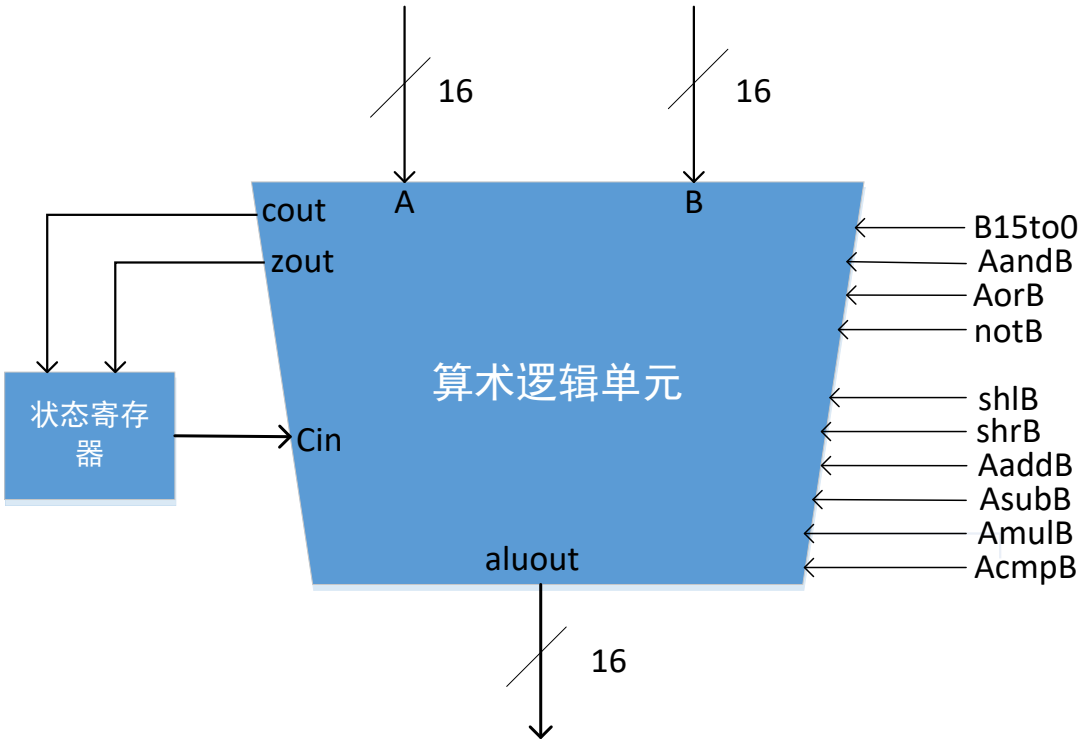
掌握外围接口电路的设计方法，并学会利用 FPGA 开发板的按键、LED 等外设构造简单的控制系统

编写汇编程序，并依据设计的指令集编译为对应的机器码，控制 CPU 实现特定功能

任务计划：课后完成算术逻辑单元，第二次课完成寄存器阵列，课后完成寻址单元。

实验原理：

算术逻辑单元：算术 逻辑运算单元（ALU）的基本功能为加、减、乘、除四则运算，与、或、非、异或 等逻辑操作，以及移位、求补等操作。



pdf

适应页面大小

表 3-4　算术逻辑单元接口信号定义

接口名称	输入/输出	位宽(bits)	功能描述	信号来源/去向
A	input	16	数据输入	通用寄存器阵列
B	input	16	数据输入	通用寄存器阵列
B15to0	input	1	将 B 直接送给输出	控制器
AandB	input	1	将 A 和 B 相与的结果送给输出	控制器
AorB	input	1	将 A 和 B 相或的结果送给输出	控制器
notB	input	1	对 B 取反后的结果送给输出	控制器
shlB	input	1	将 B 的各比特位循环左移一位	控制器
shrB	input	1	将 B 的各比特位循环右移一位	控制器
AaddB	input	1	将 A+B+cin 的结果送给输出	控制器
AsubB	input	1	将 A-B-cin 的结果送给输出	控制器
AmulB	input	1	为防止溢出，将 A[7:0] *B[7:0]的结果送给输出	控制器
AcmpB	input	1	如果 A=B，则 Z=1；如果 A>B，则 cin=1	控制器
cin	input	1	C 标志位输入（数据来自于上一次运算产生的 C 标志位）	状态寄存器
aluout	output	16	数据输出	寄存器阵列或 I/O
zout	output	1	Z 标志位输出	状态寄存器
cout	output	1	C 标志位输出	状态寄存器

◆ 指令寄存器

寄存器阵列：当计算机工作时，需要处理大量的控制信息和数据信息，如对指令信息进行译 码，以便产生相应控制命令；对操作数进行算术或逻辑运算加工，并根据运算结果 决定后续操作等。因此，在处理器中需要设置若干寄存器来暂时存放这些信息。 常用寄存器按所存信息的类型可分为通用寄存器组、暂存器、指令寄存器、地 址寄存器、当前程序状态寄存器、数据缓冲寄存器等。 寄存器组也称为寄存器文件或寄存器堆，是一个寄存器集合。寄存器组是处理 器内部的存储器，用于存放指令和数据，其中的寄存器都可通过制定相应的寄存器序号来进行读写。图 3-11 给出了通用寄存器阵列的结构。

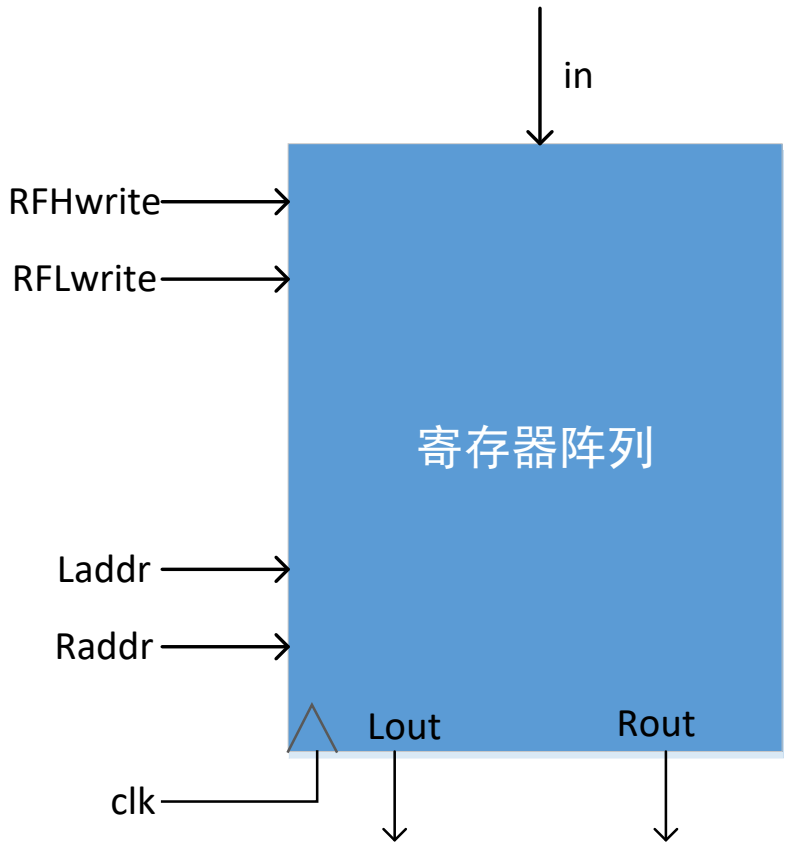


表 3-6 寄存器阵列接口信号定义

接口名称	输入/输出	位宽 (bits)	功能描述	信号来源/去向
in	input	16	数据输入	存储器，数据总线
clk	input	1	时钟输入	分频器
RFLwrite	input	1	低 8 位写有效，Laddr 内存储的数据低 8 位= 输入数据 in 的低 8 位	控制器
RFHwrite	input	1	高 8 位写有效，Laddr 内存储的数据高 8 位= 输入数据 in 的高 8 位	控制器
Laddr	input	2	目的寄存器地址	指令寄存器
Raddr	input	2	源寄存器地址	指令寄存器
Lout	output	16	输出目的寄存器数据	算术逻辑单元
Rout	output	16	输出源寄存器数据	算术逻辑单元

寻址单元: 包括程序计数器和地址逻辑两部分。程序计数器是带使能和复位功 能的简单寄存器，地址逻辑是个小型的算术单元，通过加法和递增来计算程序计数器的值或储存器的地址。

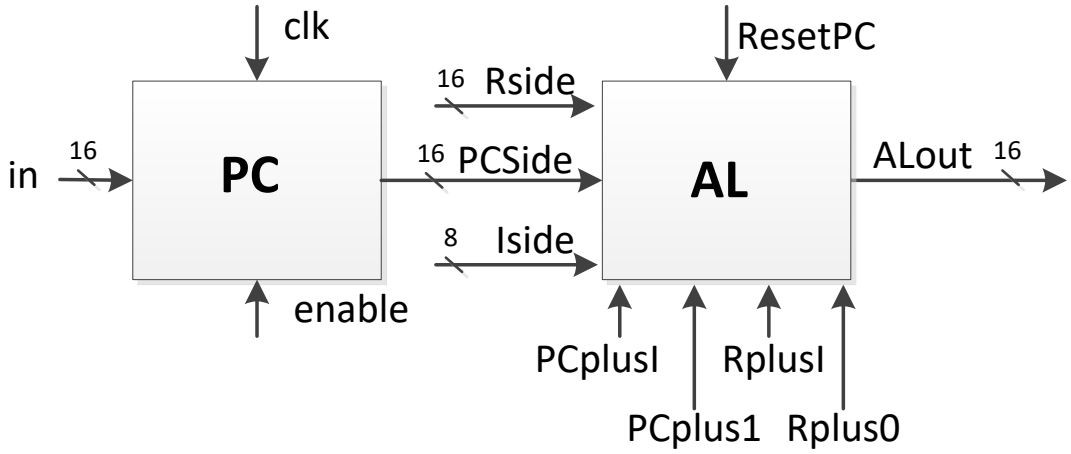


表 3-3 寻址单元接口信号定义

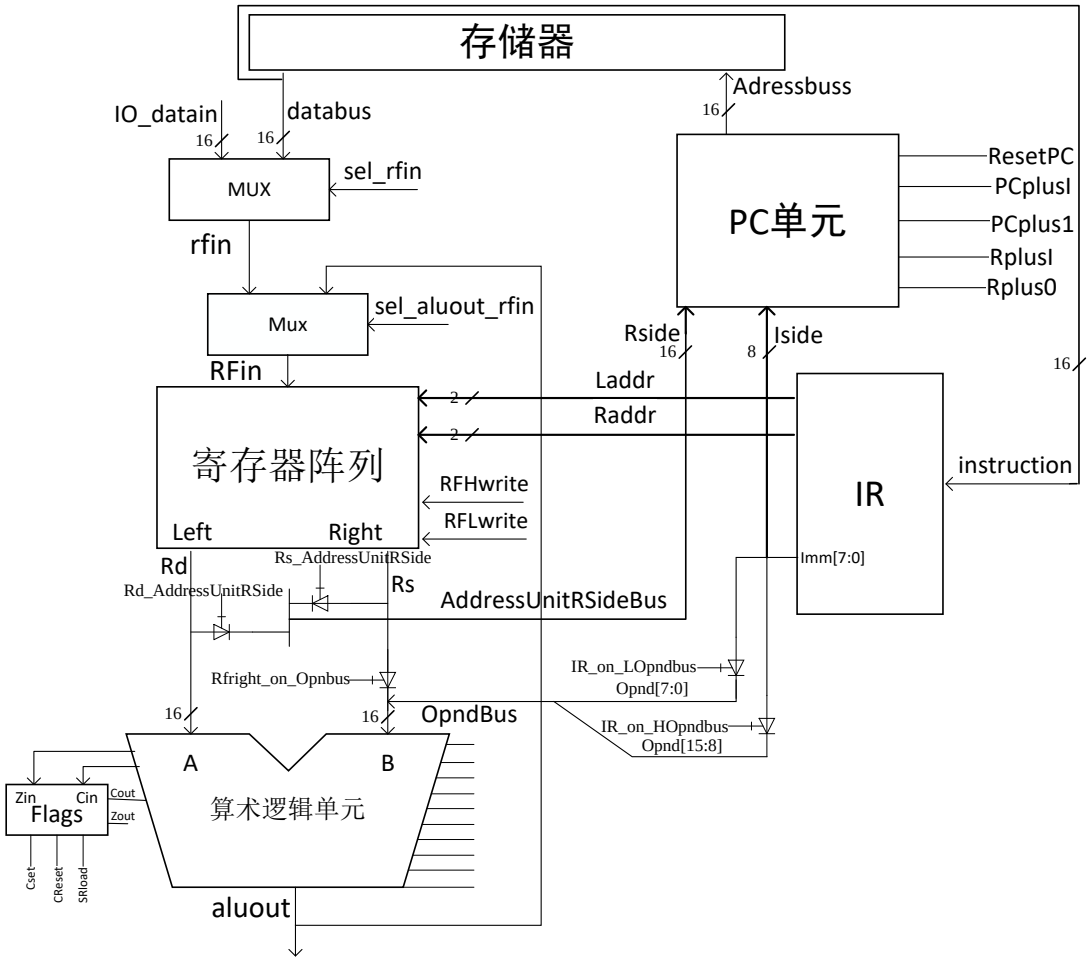
接口名称	输入/输出	位宽(bits)	功能描述	信号来源/去向
ResetPC	input	1	有效时复位寻址单元	控制器
PCplusI	input	1	将 PC 与 Iside 的和送给输出	控制器
PCplus1	input	1	将 PC 与 1 的和送给输出	控制器
RplusI	input	1	将 Iside 送给输出	控制器
Rplus0	input	1	将 Rside 的值送给输出	控制器
PCenable	input	1	PC 使能	控制器
Rside	input	16	输入 Rside 值	通用寄存器阵列
Iside	input	8	输入 Iside 值	指令寄存器
clk	input	1	系统时钟	分频器
Address	output	16	输出地址值	存储器

任务 2:

目标：根据电路图整合 5 个功能模块，汇总为完整的数据通路。在数据通路中分别例化 ALU，寄存器阵列，寻址单元。

任务计划：在第三次课中完成数据通路的整合。

实验原理: 数据通路部分主要由以下基本部件组成：寻址单元（由程序计数器和地址逻辑 组成)、算术逻辑单元、通用寄存器阵列、指令寄存器、状态寄存器。电路图如下。



任务三：

目标：在控制通路中，添加 mvr 指令

在控制通路中，添加 mil 和 mih 指令

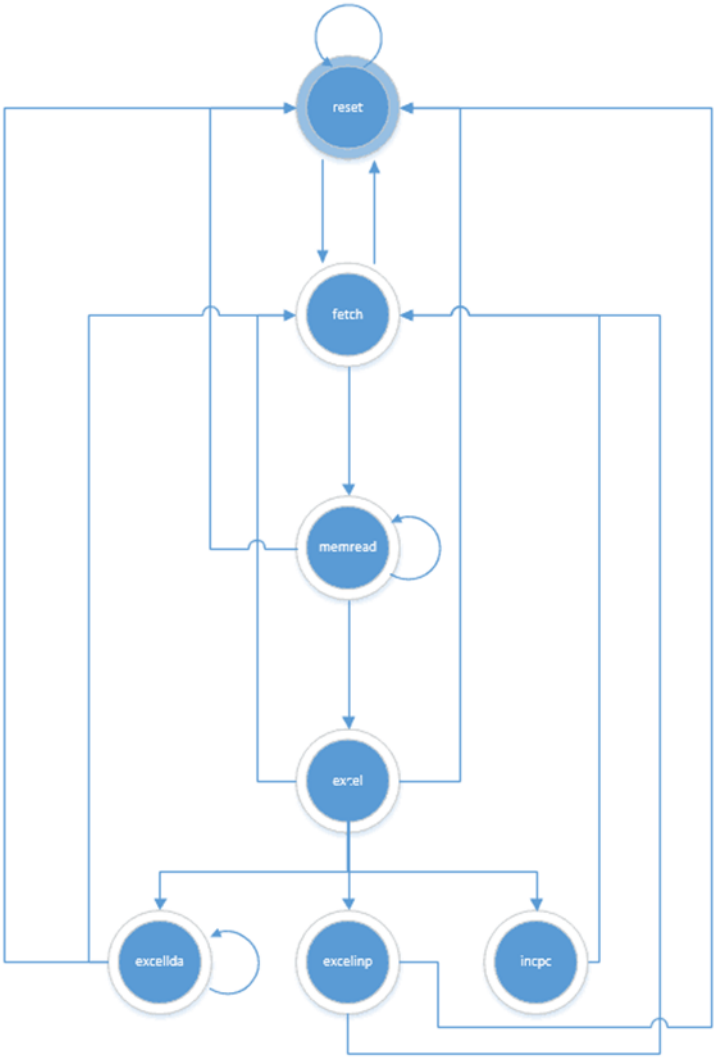
设置 CPU.v 为顶层文件，对 mil、mih 指令进行仿真

任务计划：在第四次课中完成控制通路的设计

实验原理：控制器：控制器是 7 状态的状态机，不同状态下向数据通路发送不同的控制信 号。控制器采用 Huffman 风格的编码，通过把下一状态值赋给 reg 类型的变量 Nstate，来实现状态转化。控制状态：参数声明里定义了 7 个状态。Reset 状态为状态机的初始状态。在 fetch 状态下，状态机读取 16 比特的指令。在 memread 状态下，控制器等存储器准备好 数据后发送 memDataReady 信

号。memread 的下一状态是 execl, 在 execl 状态下执 行指令。指令 lda 和 inp 在状态 execl 下开始执行，这两个状态用于发送读取命令， 但是“载入指定地址数据”和“从 I/O 端口输入”这两个操作需要额外的状态（execllda） 和状态（excelinp)来完成读操作，由于在实验的 CPU 设计中，当从存储器和 IO 时读取数据的时候，是需要延迟 1 个时钟才会出来数据的，所以要处理来自存储器和 IO 的数据，就必须用两个状态，1 个状态（lda 和 inp）用来发送读取命令，第 2 个额 外的状态（execllda 和 excelinp）用来处理读到的数据。因此需要在“载入指定地址 数据”和“从 I/O 端口输入”这两个操作时单独增加两个状态用于处理数据。在执 行状态，大部分指令会让程序计数器的值加 1，但是使用地址总线的某些指令不进 行这个操作，它们在 incpc 状态下让程序计数器加 1。状态转移图如图 3-12 所示： 组合逻辑块: 这个组合逻辑块是一个 always 语句，其中 case 语句根据状态机的当前状态选择状态转化和控制信号的输出。always 语句开始时设置所用控制信号为 无效状态, 避免输出产生锁存器。时序逻辑块: 控制器程序的最后一部分是时序的 always 语句，在时钟有效沿把 Pstate 的值赋给 Nstate。控制状态寄存器和所有数据寄存器是下降沿触发，它们的 值一直保持到下一个时钟下降沿到来时。

助记符及定义	比特 15 ： 0	注释
nop 空操作	0000-0000-0000-0000	无操作
scf 对进位标志位置1	0000-0001-0000-0000	C<=’ 1’
ccf 清除进位标志位	0000-0010-0000-0000	C<=’ 0’
jpr 跳转到相关位置	0000-0011-[I(8bit)]	PC<=PC+I
jnz 零标志位为条件，跳转到指定位置	0000-0100-[I(8bit)]	If(Z=0), PC<={8’ d0,I}
mvr 转移寄存器数据	0001-D-S-0000-0000	Rd<=Rs
lda 载入指定地址数据	0010-D-S-0000-0000	Rd<=(Rs)
sta 储存数据到指定地址	0011-D-S-0000-0000	(Rd)<=Rs
inp 从端口输入	0100-D-S-[I(8bit)]	把端口I的数据写入Rd
oup 输出到端口	0101-D-S-[I(8bit)]	把Rs的数据送给端口I
and 寄存器数据与操作	0110-D-S-0000-0000	Rd<=Rd&Rs
orr 寄存器数据或操作	0111-D-S-0000-0000	Rd<=Rd|Rs
not 寄存器数据非操作	1000-D-S-0000-0000	Rd<=~Rs
shl 左移	1001-D-S-0000-0000	Rd<=Rs左移
shr 右移	1010-D-S-0000-0000	Rd<=Rs右移
add 寄存器数据相加	1011-D-S-0000-0000	Rd<=Rd+Rs+C
sub 寄存器数据相减	1100-D-S-0000-0000	Rd<=Rd-Rs-C
mul 寄存器数据相乘	1101-D-S-0000-0000	Rd<=Rd*Rs(8比特乘法)
cmp 寄存器数据相比较	1110-D-S-0000-0000	比较Rd和Rs，如果Rd〉Rs，则C=1
mil 将立即数放到低8位	1111-D-00-[I(8bit)]	Rd<={8’ BZ,I}
mih 将立即数放到高8位	1111-D-01-[I(8bit)]	Rd<={I,8’ BZ}
jpa 跳转到指定位置	1111-00-10-[I(8bit)]	PC<={8’d0,I}



任务 4

目标：1、根据指令集把流水灯汇编代码翻译为“0”“1”组合的机器码，预先存入 RAM 中。

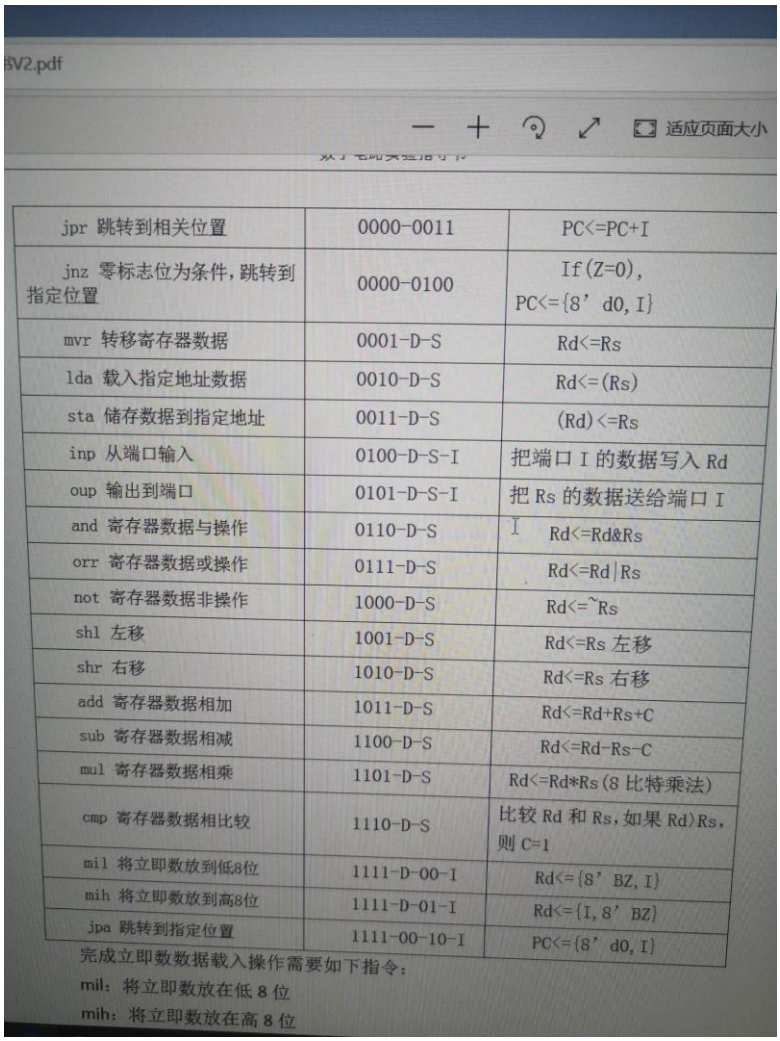
2、软件仿真

3、下板验证流水灯

任务计划：在第五次课中根据指令集翻译机器码，并进行软件仿真和下板验证。

实验原理：指令集的重要作用是反映计算机的基本功能，是软件设计与硬件设计的主要分 界，影响程序执行的时间和效率。设计时要满足正交性，规整性，可

扩充性和对称 性等准则。 本实验 CPU 使用通用寄存器存储变量，所使用的数据变量无法从存储器直接调 用，必须先下载到通用寄存器中才能在运算中使用。将变量分配给通用寄存器，不 但能够减少存储器的通信量，加快程序的执行速度，而且和存储器相比，还可以用 更少的地址位来寻址寄存器，从而能够有效改进程序的目标代码大小。通用寄存器 型指令集结构的一个主要优点就是能够使编译器有效地使用寄存器。这不仅体现在 表达式求值方面，更重要的是体现在利用寄存器存放变量所带来的优越性。 本实验设计的 RISC_CPU 指令长度为 16 位，能够处理 16 位数据，指令中需要 操作符，寄存器地址和立即数等字段。



jpr 跳转到相关位置	0000-0011	PC<=PC+I
jnz 零标志位为条件，跳转到指定位置	0000-0100	If(Z=0), PC<={8' d0, I}
mvr 转移寄存器数据	0001-D-S	Rd<=Rs
lda 载入指定地址数据	0010-D-S	Rd<=(Rs)
sta 储存数据到指定地址	0011-D-S	(Rd)<=Rs
inp 从端口输入	0100-D-S-I	把端口 I 的数据写入 Rd
oup 输出到端口	0101-D-S-I	把 Rs 的数据送给端口 I
and 寄存器数据与操作	0110-D-S	Rd<=Rd&Rs
orr 寄存器数据或操作	0111-D-S	Rd<=Rd|Rs
not 寄存器数据非操作	1000-D-S	Rd<=~Rs
shl 左移	1001-D-S	Rd<=Rs 左移
shr 右移	1010-D-S	Rd<=Rs 右移
add 寄存器数据相加	1011-D-S	Rd<=Rd+Rs+C
sub 寄存器数据相减	1100-D-S	Rd<=Rd-Rs-C
mul 寄存器数据相乘	1101-D-S	Rd<=Rd*Rs(8 比特乘法)
cmp 寄存器数据相比较	1110-D-S	比较 Rd 和 Rs，如果 Rd)Rs，则 C=1
mil 将立即数放到低8位	1111-D-00-I	Rd<={8' BZ, I}
mih 将立即数放到高8位	1111-D-01-I	Rd<={I, 8' BZ}
jpa 跳转到指定位置	1111-00-10-I	PC<={8' d0, I}

完成立即数数据载入操作需要如下指令：
mil：将立即数放在低 8 位
mih：将立即数放在高 8 位

完成立即数数据载入操作需要如下指令： mil：将立即数放在低 8 位 mih：将立即数放在高 8 位 因为一条指令无法载入完整 16 比特立即数数据，设计指令

格式中用于存放立 即数的字段为 8bits，将 16bits 数据传递到通用寄存器需要 2 条指令，“mil R1,A(低 8 位)”将立即数 A 的低 8 位传递给通用寄存器 R1，“mih R1，A(高 8 位)”将立即数 A 的高 8 位传递给通用寄存器 R1。 完成存储器或 I/O 数据载入与存储的操作需要如下指令： lda：载入指定地址数据 sta：储存数据到指定地址 inp：从端口输入 oup：输出到端口 因为存储器中有些地址的数据可能是有工程意义的，对这些地址上的数据的处 理是必不可少的。“lda Rd Rs”将通用寄存器 Rs 的数据作为指定地址，将存储器中 该地址上的数据载入到通用寄存器 Rd 中，“sta Rd Rs”将通用寄存器 Rd 的数据作 为指定地址，将通用寄存器 Rs 的数据储存到存储器该地址上。 完成通用寄存器阵列内数据运算操作需要如下指令： and：寄存器数据与操作 orr：寄存器数据或操作 not：寄存器数据非操作 shl：左移 shr：右移 add：寄存器数据相加 sub：寄存器数据相减 mul：寄存器数据相乘 cmp：寄存器数据相比较 这些是本 CPU 设计能够完成的数据处理操作，有 3 点需要注意：1、所有操作 的数据必须储存于通用寄存器中 2、乘法运算只能进行 8 比特数据相乘，溢出则取 其低 8 位数据相乘 3、cmp 指令的结果会影响标志位，该标志位可作为分支操作的 条件，但执行 cmp 指令之前建议先清除相关标志位。 完成对标志位的处理操作需要如下指令： scf：对进位标志位置 1 ccf：清除进位标志位 设计这些指令为分支操作的执行创造了条件，还需要注意其它指令在执行过程 中同样可能影响标志位的值。 完成指令跳转、分支操作需要如下指令： jpa：跳转到指定位置 jpr：跳转到相关位置 jnz：以零标志位为条件的分支 jpa、jpr 指令为程序的循环执行创造了基础，jnz 指令为程序的分支执行创造了 基础。如果一个程序不只是顺序执行，那么这些指令是必不可少的。 完成无操作、需要如下指令： nop：无操作。

五、实验步骤

任务 1 的步骤如下:

1、首先打开 Quartus II 软件。

2、打开桌面 CPU 文档中的 cpu_top.qpf 文件，可以直接打开 cpu_top 工程文件。

3、点击 Project Navigator 窗口的 Files，选择 ArithmeticUnit.v 模块，单击右键选择‘Set as Top-Level Entity’。

4、打开 ArithmeticUnit.v 文件之后，补全未完成的代码，将代码写到本文件中。单击“保存”，双击 Compile Design 完成编译功能。也可以单击快捷栏 进行编译，若出现错误则改正。

5、点击编译 ，编译通过之后，准备编 写 testbench 激励文件。

6、在激励文件的 initial 部分添加代码（如 果仿真的是时序电路，还需在 always 后添加相应代码）。

7、进行 RTL 仿真，点击 Tools/Run Simulation Tool/RTL simulation。

8、分析各端口信号波形是否符合要求。

9、对寄存器阵列和寻址单元重复上述操作。

任务 2 的步骤如下:

1、首先打开 Quartus II 软件。

2、打开桌面 CPU 文档中的 cpu_top.qpf 文件，可以直接打开 cpu_top 工程文件。

3、点击 Project Navigator 窗口的 Files，选择 Datapath.v 模块，单击右键选择‘Set as Top-Level Entity’。

4、打开 Datapath.v 文件后，补全未完成的代码，将代码写到本文件中。单击“保存”，双击 Compile Design 完成编译功能。也可以单击快捷栏 进行编译，若出现错误则改正。

任务 3 的步骤如下：

1、首先打开 Quartus II 软件。

2、打开桌面 CPU 文档中的 cpu_top.qpf 文件，可以直接打开 cpu_top 工程文件。

3、点击 Project Navigator 窗口的 Files，选择 control.v 模块，单击右键选择‘Set as Top-Level Entity’。

4、打开 control.v 文件后，补全未完成的代码，将代码写到本文件中。单击“保存”，双击 Compile Design 完成编译功能。也可以单击快捷栏 进行编译，若出现错误则改正。

5、设置 CPU.v 为顶层文件，点击编译 ，编译通过之后，准备编 写 testbench 激励文件。

6、在激励文件的 initial 部分添加代码（如 果仿真的是时序电路，还需在 always 后添加相应代码）。

7、进行 RTL 仿真，点击 Tools/Run Simulation Tool/RTL simulation。

8、分析各端口信号波形是否符合要求。

任务 4 的步骤如下：

1、首先打开 Quartus II 软件。

2、打开桌面 CPU 文档中的 cpu_top.qpf 文件，可以直接打开 cpu_top 工程文件。

3、利用 QuartusII 软件提供的 IP 核例化分频器例化分频模块 PLL。

4、利用 QuartusII 软件提供的 IP 核例化分频器例化存储器 ，将汇编代码翻译成机器码，用十六进制代码生成 mif 文件，并用 mif 文件初始化 RAM。

5、当加入外围模块后，再对顶层文件进行编译，编译通过后，按照上述模块仿真的步 骤生成和编写激励文件，进行 RTL 行为级的仿真。

6、单击快捷栏“Start Compilatiom”编译工程，确定无错。由于演示需要程序 指令内的分频数值很低，但不利于 LED 观测，将分频增大。把 RAM 的初始化 mif 文件改为 test 文件。

7、编译成功后配置管脚。

8、正确连接平台硬件各部件后进行下板验证。

六、实验结果

任务 1:

ALU:

1、Verilog 关键代码：

```
//-------------------------------------------------------------------
//--SAYEH (Simple Architecture Yet Enough Hardware) CPU
//-------------------------------------------------------------------
//Arithmetic Logic Unit (ALU)
`timescale 1 ns /1 ns
//宏定义
`define B15to0H 10'b1000000000
`define AandBH   10'b0100000000
`define AorBH    10'b0010000000
`define notBH    10'b0001000000
`define shlBH    10'b0000100000
`define shrBH    10'b0000010000
`define AaddBH   10'b0000001000
`define AsubBH   10'b0000000100
`define AmulBH   10'b0000000010
`define AcmpBH   10'b0000000001
```

```
module ArithmeticUnit (
	A, B,
	B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB,
	aluout, cin,    cout, zout
);
//输入输出
input [15:0] A, B;
input B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB;
input cin;
output [15:0] aluout;
output cout,zout;
reg [15:0] aluout;
reg    cout,zout;

	always @(
		 A or B or B15to0 or AandB or AorB or notB or shlB or shrB or AaddB or AsubB    or
AmulB    or AcmpB or cin
	)
	begin
		cout = 0; aluout = 0;zout=0;
		case({B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB})
			//code

			`B15to0H: begin
				aluout = B;//输出 B
			end
			`AandBH: begin
				aluout = A&B;//输出 A 与 B
			end
			`AorBH: begin
				aluout = A|B;//输出 A 或 B
			end
			`notBH: begin
				aluout = ~B;//将 B 按位取反
			end
			`shlBH: begin
				aluout = {B[14:0],B[15]};//将 B  左移一位
			end
			`shrBH: begin
				aluout = {B[0],B[15:1]};//将 B 右移一位
			end
			`AaddBH: begin
				{cout,aluout }= A+B+cin;//输出 A+B+cin
```

```
                end
                `AsubBH: begin
                    {cout,aluout} = A-B-cin;//输出 A-B-cin
                end
                `AmulBH: begin
                    aluout = A[7:0]*B[7:0];//输出 A*B
                end
                `AcmpBH: begin
                    if(A == B)zout = 1;else zout = 0;
                    if (A> B) cout = 1; else cout = 0;//输出 A 比 B
                end
                default: aluout = 0;
        endcase
         end
endmodule
```

2、testbench 测试文件:

```
   // Copyright (C) 1991-2013 Altera Corporation
// Your use of Altera Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Altera Program License
// Subscription Agreement, Altera MegaCore Function License
// Agreement, or other applicable license agreement, including,
// without limitation, that your use is for the sole purpose of
// programming logic devices manufactured by Altera and sold by
// Altera or its authorized distributors.  Please refer to the
// applicable agreement for further details.

// ********************************************************************************
// This file contains a Verilog test bench template that is freely editable to
// suit user's needs .Comments are provided in each section to help the user
// fill out necessary details.
// ********************************************************************************
// Generated on "09/09/2016 16:28:39"

// Verilog Test Bench template for design : ArithmeticUnit
//
// Simulation tool : ModelSim-Altera (Verilog)
//
```

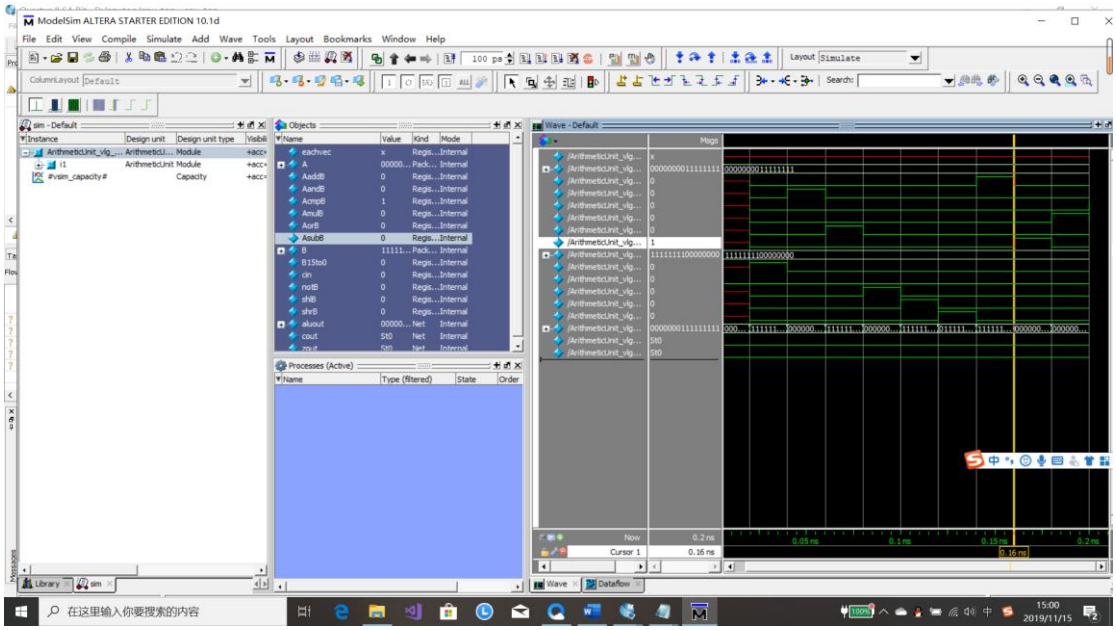
```
`timescale 1 ps/ 1 ps
`define B15to0H 10'b1000000000
`define AandBH    10'b0100000000
`define AorBH     10'b0010000000
`define notBH     10'b0001000000
`define shlBH     10'b0000100000
`define shrBH     10'b0000010000
`define AaddBH    10'b0000001000
`define AsubBH    10'b0000000100
`define AmulBH    10'b0000000010
`define AcmpBH    10'b0000000001
module ArithmeticUnit_vlg_tst();
// constants
// general purpose registers
reg eachvec;
// test vector input registers
reg [15:0] A;
reg AaddB;
reg AandB;
reg AcmpB;
reg AmulB;
reg AorB;
reg AsubB;
reg [15:0] B;
reg B15to0;
reg cin;
reg notB;
reg shlB;
reg shrB;
// wires
wire [15:0]    aluout;
wire cout;
wire zout;

// assign statements (if any)
ArithmeticUnit i1 (
// port map - connection between master ports and signals/registers
      .A(A),
      .AaddB(AaddB),
      .AandB(AandB),
```

```
        .AcmpB(AcmpB),
        .AmulB(AmulB),
        .AorB(AorB),
        .AsubB(AsubB),
        .B(B),
        .B15to0(B15to0),
        .aluout(aluout),
        .cin(cin),
        .cout(cout),
        .notB(notB),
        .shlB(shlB),
        .shrB(shrB),
        .zout(zout)
);
initial
begin
A=16'b0000000011111111;
B=16'b1111111100000000;
cin=0;
        begin
        #20
        {B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB}=`B15to0H;
        #20
        {B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB}=`AandBH;
        #20
        {B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB}=`AorBH;
        #20
        {B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB}=`notBH;
        #20
        {B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB}=`shlBH;
        #20
        {B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB}=`shrBH;
        #20
        {B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB}=`AaddBH;
        #20
        {B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB}=`AsubBH;
        #20
        {B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB}=`AmulBH;
        #20
        {B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB}=`AcmpBH;
        end
```

end

endmodule

3、仿真结果截图



从上到下依次输出 A,AaddB,AandB,AcmpB,AmulB，AorB,AsubB,B,B15to0,cin,notB,shlB,shrB,aluout,cout,zout 当输入为 A=16'b0000000011111111;

B=16'b1111111100000000 时，0.02ns 时，B15to0 上升；0.04ns 时，AandB 上升； 0.06ns 时 AorB 上升；0.08ns 时，AorB 上升；0.1ns 时，shlB 上升；0.12ns 时，shrB，0.14ns 时 AaddB 上升；0.16ns 时，AsubB 上升；0.18ns 时 AmulB 上升。

寄存器阵列：

1、Verilog 关键代码：

```
   // Register File
`timescale 1 ns /1 ns
module RegisterFile (
```

```
input [15:0] in,//数据输入
input clk, RFLwrite, RFHwrite,//时钟输入，低 8 位写有效，高 8 位写有效
input [1:0] Laddr, Raddr,//目的寄存器地址，源寄存器地址
output [15:0] Lout, Rout//输出目的寄存器数据，输出源寄存器数据
);
reg [15:0] MemoryFile [0:3];
reg [15:0] TempReg;
assign Lout = MemoryFile [Laddr];
assign Rout = MemoryFile [Raddr]
    always @(negedge clk) begin
    TempReg=MemoryFile[Laddr];
    //code
    if(RFLwrite==1)
    begin
    TempReg[7:0] = in[7:0];//tempreg 的低 8 位=in 的低八位
    end
    if(RFHwrite==1)
    begin
    TempReg[15:8] = in[15:8];//tempreg 的高 8 位=in 的高八位
    end
    MemoryFile[Laddr] = TempReg;
    end
```

endmodule

2、testbench 测试文件

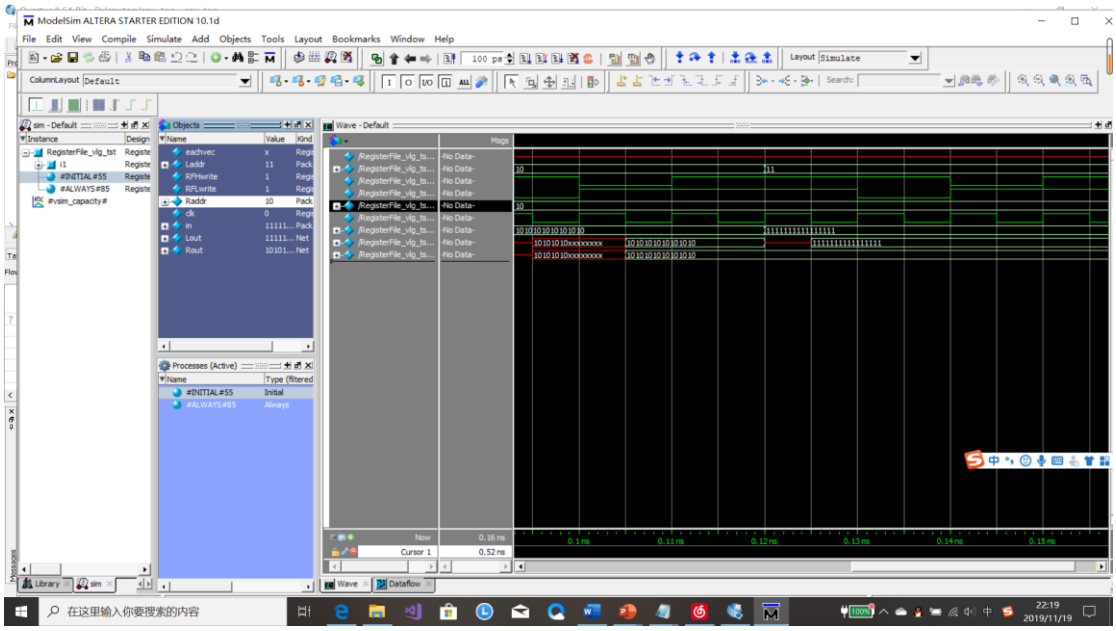
```
        // Copyright (C) 1991-2013 Altera Corporation
// Your use of Altera Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Altera Program License
// Subscription Agreement, Altera MegaCore Function License
// Agreement, or other applicable license agreement, including,
// without limitation, that your use is for the sole purpose of
// programming logic devices manufactured by Altera and sold by
// Altera or its authorized distributors.  Please refer to the
// applicable agreement for further details.
// ********************************************************************************
// This file contains a Verilog test bench template that is freely editable to
// suit user's needs .Comments are provided in each section to help the user
// fill out necessary details.
// ********************************************************************************
// Generated on "11/16/2019 19:23:56"
```

```
// Verilog Test Bench template for design : RegisterFile
//
// Simulation tool : ModelSim-Altera (Verilog)
//
`timescale 1 ps/ 1 ps
module RegisterFile_vlg_tst();
// constants
// general purpose registers
reg eachvec;
// test vector input registers
reg [1:0] Laddr;
reg RFHwrite;
reg RFLwrite;
reg [1:0] Raddr;
reg clk;
reg [15:0] in;
// wires
wire [15:0]    Lout;
wire [15:0]    Rout;
// assign statements (if any)
RegisterFile i1 (
// port map - connection between master ports and signals/registers
```

```
        .Laddr(Laddr),
        .Lout(Lout),
        .RFHwrite(RFHwrite),
        .RFLwrite(RFLwrite),
        .Raddr(Raddr),
        .Rout(Rout),
        .clk(clk),
        .in(in)
);
initial
begin
clk=1;
RFLwrite=1;
RFHwrite=1;
in=16'b0000_0000_1010_1010;
Laddr=2'b00;
Raddr=2'b00;
#10 RFLwrite=0;RFHwrite=1;
#10 RFLwrite=1;RFHwrite=0;
#10 RFLwrite=1;RFHwrite=1;
#10 in=16'b0000_0000_1111_1111;
        Laddr=2'b01;
```

```
RFHwrite=0;RFLwrite=0;Raddr=2'b01;
#10 RFLwrite=0;RFHwrite=1;
#10 RFLwrite=1;RFHwrite=0;
#10 RFLwrite=1;RFHwrite=1;
#10 in=16'b1010_1010_1010_1010;
      Laddr=2'b10;
RFHwrite=0;RFLwrite=0;Raddr=2'b10;
#10 RFLwrite=0;RFHwrite=1;
#10 RFLwrite=1;RFHwrite=0;
#10 RFLwrite=1;RFHwrite=1;
#10 in=16'b1111_1111_1111_1111;
      Laddr=2'b11;
#10 RFLwrite=0;RFHwrite=1;
#10 RFLwrite=1;RFHwrite=0;
#10 RFLwrite=1;RFHwrite=1;
#10 $stop;
end
always
begin
#5 clk=~clk;
end
endmodule
```

3、仿真结果截图



Clk 的周期为 0.01ns，0.07ns 时，RFHwrite 上升为 1；0.08ns 时，RFLwrite 和 RFHwrite 下降为 0；0.09ns 时，RFHwrite 上升为 1；0.1ns 时，RFHwrite 下降为 0，RFLwrite 上升为 1；0.11ns 时，RFHwrite 上升为 1；0.13ns 时，RFLwrite 下降为 0；0.14ns 时，RFHwrite 下降为 0，RFLwrite 上升为 1；0.15ns 时，RFHwrite 上升为 1.

寻址单元

1、Verilog 关键代码

```
   PC：//-------------------------------------------------------------------------
//--SAYEH (Simple Architecture Yet Enough Hardware) CPU
//-------------------------------------------------------------------------
//Program Counter
`timescale 1 ns /1 ns
module ProgramCounter (in, enable, clk, out);
input [15:0] in;
input enable, clk;
output [15:0] out;
reg [15:0] out;
   always @(negedge clk)
            //code
```

```
    begin
      if(enable==1)//只有使能端有效时，输入才有效
          out=in;
    end
endmodule
AL：//-------------------------------------------------------------------
//--SAYEH (Simple Architecture Yet Enough Hardware) CPU
//-------------------------------------------------------------------
// Address Logic

`timescale 1 ns /1 ns
`define ResetPCH 5'b10000//进行宏定义
`define PCplusIH 5'b01000
`define PCplus1H 5'b00100
`define RplusIH 5'b00010
`define Rplus0H 5'b00001
module AddressLogic (
    PCside, Rside, Iside, ALout, ResetPC, PCplusI, PCplus1, RplusI, Rplus0
);
input [15:0] PCside, Rside;
input [7:0] Iside;
input ResetPC, PCplusI, PCplus1, RplusI, Rplus0;
output [15:0] ALout;
reg [15:0] ALout;
    always @(PCside or Rside or Iside or ResetPC or PCplusI or PCplus1 or RplusI or Rplus0)
    begin
        case ({ResetPC, PCplusI, PCplus1, RplusI, Rplus0})
            //code
            `ResetPCH:ALout=16'b0;//复位且使输出为 0
            `PCplusIH:ALout=PCside+Iside;
            `PCplus1H:ALout=PCside+1;
            `RplusIH:ALout={{8'b0},Iside};//将 Iside 扩展至 16 位
            `Rplus0H:ALout=Rside;
            default: ALout = PCside;//AL 无效时，PCside 直接传到输出
        endcase
    end
endmodule
AdressingUnit：//-------------------------------------------------------------------
------
//--SAYEH (Simple Architecture Yet Enough Hardware) CPU
//-------------------------------------------------------------------
// Addressing Unit

`timescale 1 ns /1 ns
```

```
module AddressingUnit (
input [15:0] Rside,
input [7:0] Iside,
output [15:0] Address,
input clk,ResetPC, PCplusI, PCplus1, RplusI, Rplus0, PCenable
);
wire [15:0] PCout;
    ProgramCounter PC (Address, PCenable, clk, PCout);
    AddressLogic AL (PCout, Rside, Iside, Address, ResetPC, PCplusI, PCplus1, RplusI, Rplus0);//进行调用
endmodule
```

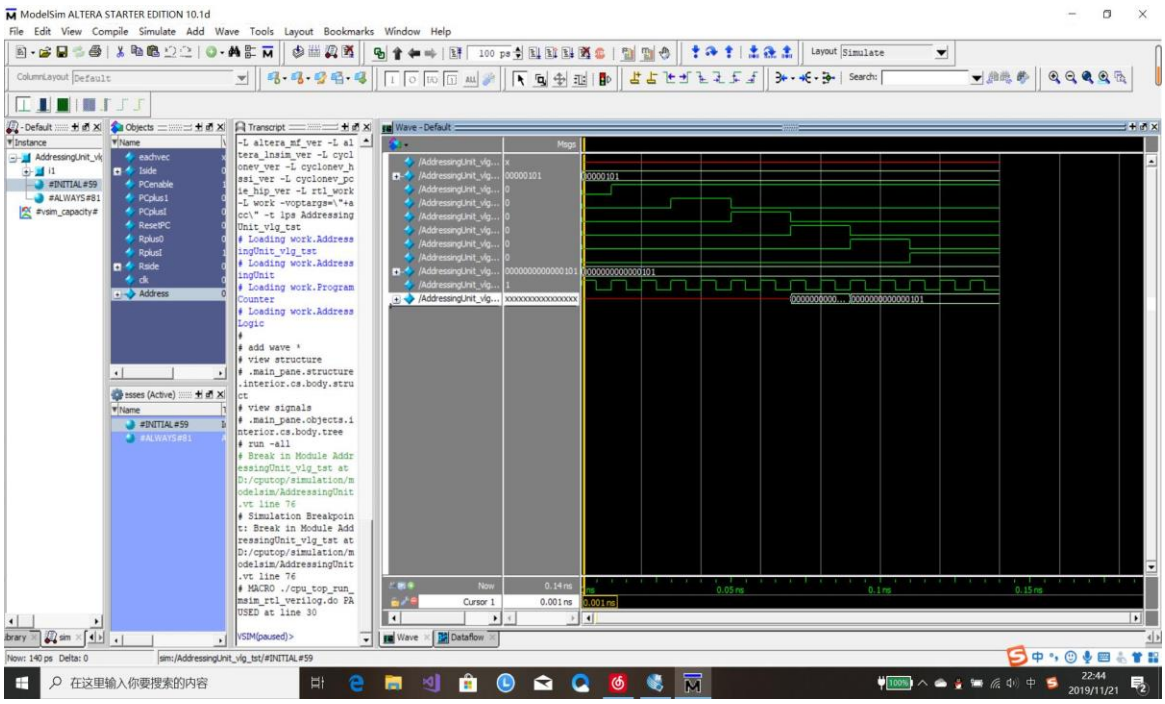
2、testbench 测试文件

```
// Copyright (C) 1991-2013 Altera Corporation
// Your use of Altera Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Altera Program License
// Subscription Agreement, Altera MegaCore Function License
// Agreement, or other applicable license agreement, including,
// without limitation, that your use is for the sole purpose of
// programming logic devices manufactured by Altera and sold by
// Altera or its authorized distributors.  Please refer to the
// applicable agreement for further details.
// ********************************************************************************
// This file contains a Verilog test bench template that is freely editable to
// suit user's needs .Comments are provided in each section to help the user
// fill out necessary details.
// ********************************************************************************
// Generated on "11/21/2019 21:54:08"

// Verilog Test Bench template for design : AddressingUnit
//
// Simulation tool : ModelSim-Altera (Verilog)
//
`timescale 1 ps/ 1 ps
module AddressingUnit_vlg_tst();
// constants
// general purpose registers
```

```
reg eachvec;
// test vector input registers
reg [7:0] Iside;
reg PCenable;
reg PCplus1;
reg PCplusI;
reg ResetPC;
reg Rplus0;
reg RplusI;
reg [15:0] Rside;
reg clk;
// wires
wire [15:0]    Address
// assign statements (if any)
AddressingUnit i1 (
// port map - connection between master ports and signals/registers
      .Address(Address),
      .Iside(Iside),
      .PCenable(PCenable),
      .PCplus1(PCplus1),
      .PCplusI(PCplusI),
      .ResetPC(ResetPC),
      .Rplus0(Rplus0),
      .RplusI(RplusI),
      .Rside(Rside),
      .clk(clk)
);
initial
begin
// code that executes only once
// insert code here --> begin
clk=1;
PCenable=0;
Rside=16'b0000_0000_0000_0101;
Iside=8'b0000_0101;
PCplus1=0;PCplusI=0;ResetPC=0;Rplus0=0;RplusI=0;
#10 PCenable=1;
#20 PCplus1=1;PCplusI=0;ResetPC=0;Rplus0=0;RplusI=0;
#20 PCplus1=0;PCplusI=1;ResetPC=0;Rplus0=0;RplusI=0;
#20 PCplus1=0;PCplusI=0;ResetPC=1;Rplus0=0;RplusI=0;
#20 PCplus1=0;PCplusI=0;ResetPC=0;Rplus0=1;RplusI=0;
```

```
#20 PCplus1=0;PCplusI=0;ResetPC=0;Rplus0=0;RplusI=1;

#30 $stop;

// --> end

end
always
// optional sensitivity list
// @(event1 or event2 or .... eventn)
begin
// code executes for every event on sensitivity list
// insert code here --> begin

#5 clk=~clk;
// --> end
end
endmodule
```

3、仿真结果截图



在时钟进入下降沿时，就会进入 always，从而检查变量是否发生改变，PC 使能端有效，AL 无效时，输出一直等于输入；当 PC 和 AL 同时有效时，就可以产生相应功能的输出。

任务 2

数据通路:

1、Verilog 关键代码：//----------

//--SAYEH (Simple Architecture Yet Enough Hardware) CPU

//----------

// Data Path

`timescale 1 ns /1 ns

module DataPath (

clk, Databus, Addressbus,

ResetPC, PCplusI, PCplus1, RplusI, Rplus0,

Rs_on_AddressUnitRSide, Rd_on_AddressUnitRSide, EnablePC,

B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB,aluout,sel_aluout_rfin,sel_rfin,

RFLwrite, RFHwrite,

IRload, SRload,

Address_on_Databus, ALU_on_Databus, IR_on_LOpndBus, IR_on_HOpndBus, RFright_on_OpndBus,

Cset, Creset,

Instruction, Cout,Zout,IO_datain

);

```
input clk;
input    [15:0] Databus;
input    [15:0] IO_datain;
output [15:0] Addressbus;
output [15:0] aluout;
input
    ResetPC, PCplusl, PCplus1, Rplusl, Rplus0,
    Rs_on_AddressUnitRSide, Rd_on_AddressUnitRSide, EnablePC,
    B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB,
AcmpB,sel_aluout_rfin,sel_rfin,
    RFLwrite, RFHwrite,
    IRload, SRload,
    Address_on_Databus, ALU_on_Databus, IR_on_LOpndBus, IR_on_HOpndBus,
RFright_on_OpndBus,
    Cset, Creset;
output [15:0] Instruction;
output Cout,Zout;

wire [15:0] Address, AddressUnitRSideBus;
wire [15:0] Right, Left, OpndBus, ALUout, IRout;
wire SRCin, SRCout,SRZin,SRZout;
wire
```

```
        ResetPC, PCplusI, PCplus1, RplusI, Rplus0,
        Rs_on_AddressUnitRSide, Rd_on_AddressUnitRSide, EnablePC,
        B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB, AcmpB,
        RFLwrite, RFHwrite,
        IRload, SRload,
        Address_on_Databus, ALU_on_Databus, IR_on_LOpndBus, IR_on_HOpndBus,
RFright_on_OpndBus,
        Cset, Creset;
wire [1:0] Laddr, Raddr;
wire [15:0] RFin,rfin;
assign aluout=ALUout;
AddressingUnit AU//对寻址单元进行例化
(
//code
.Rside    (AddressUnitRSideBus),
.Iside    (IRout[7:0]),
.Address    (Adress),
.clk          (clk),
.ResetPC    (ResetPC),
.PCplusI    (PCplusI),
.PCplus1    (PCplus1),
.RplusI     (RplusI),
```

```
.Rplus0     (Rplus0),
.PCenable   (EnablePC)
);
ArithmeticUnit AL//对算术逻辑单元进行例化
(
//code
.A              (Left),
.B              (OpndBus),
.B15to0   (B15to0),
.AandB    (AandB),
.AorB     (AorB),
.notB     (notB),
.shlB     (shlB),
.shrB     (shrB),
.AaddB    (AaddB),
.AsubB (AsubB),
.AmulB    (AmulB),
.AcmpB    (AcmpB),
.aluout (Aluout),
.cin       (SRCout),
.cout     (SRCin),
.zout     (SRZin),
```

```
);
RegisterFile RF//对寄存器阵列进行例化(
//code
.in              (RFin),
.clk             (clk),
.RFLwrite    (RFLwrite),
.RFHwrite    (RFHwrite),
.Laddr (Laddr),
.Raddr (Raddr),
.Lout     (Left),
.Rout (Right),
);
InstrunctionRegister IR
(
.in           (Databus),
.IRload    (IRload),
.clk         (clk),
.out         (IRout)
);
StatusRegister SR
(
.Cin           (SRCin),
```

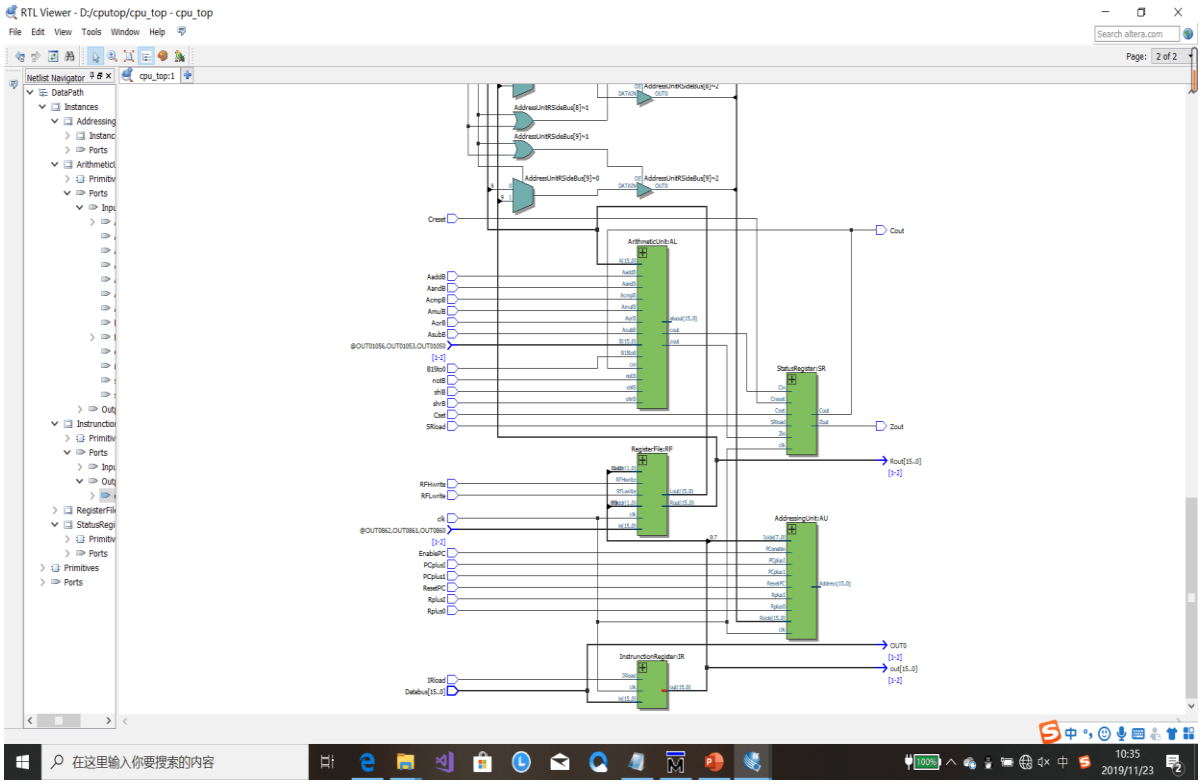
```
.Zin          (SRZin),
.SRload     (SRload),
.clk          (clk),
.Cset         (Cset),
.Creset     (Creset),
.Zout         (SRZout),
.Cout         (SRCout)
);
      assign AddressUnitRSideBus =
           (Rs_on_AddressUnitRSide) ? Right : (Rd_on_AddressUnitRSide) ? Left :
16'bZZZZZZZZZZZZZZZZ;
      assign Addressbus = Address;
      assign OpndBus[07:0] = IR_on_LOpndBus == 1 ? IRout[7:0] : 8'bZZZZZZZZ;
      assign OpndBus[15:8] = IR_on_HOpndBus == 1 ? IRout[7:0] : 8'bZZZZZZZZ;
      assign OpndBus = RFright_on_OpndBus == 1 ? Right :
16'bZZZZZZZZZZZZZZZZ;
      assign Zout = SRZout;
      assign Cout = SRCout;
      assign Instruction = IRout[15:0];
      assign Laddr =    IRout[11:10];
      assign Raddr =    IRout[09:08];
      assign RFin = sel_aluout_rfin ? ALUout : rfin;
```

assign rfin = sel_rfin ? IO_datain : Databus;

endmodule

2、关键代码分析

生成的数据电路如图



AU：Rside 对应 AddressUnitRSideBus,

Iside 对应 IRout[7:0],

Address 对应 Address,

clk 对应 clk,

ResetPC 对应 ResetPC,

PCplusI 对应 PCplusI,

PCplus1 对应 PCplus1,

RplusI 对应 RplusI,

Rplus0 对应 Rplus0,

PCenable 对应 EnablePC

AL：A 对应 Left,

B 对应 OpndBus,

B15to0 对应 B15to0,

AandB 对应 AandB,

AorB 对应 AorB,

notB 对应 notB,

shlB 对应 shlB,

shrB 对应 shrB,

AaddB 对应 AaddB,

AsubB 对应 AsubB,

AmulB 对应 AmulB,

AcmpB 对应 AcmpB,

aluout 对应 ALUout,

cin 对应 SRCout,

cout 对应 SRCin,

zout 对应 SRZin

RF：in 对应 RFin,

clk 对应 clk,

RFLwrite 对应 RFLwrite,

RFHwrite 对应 RFHwrite,

Laddr 对应 Laddr,

Raddr 对应 Raddr,

Lout 对应 Left,

Rout 对应 Right

任务三：控制通路

1、Verilog 关键代码

```
  //------------------------------------------------------------------
-----
//--SAYEH (Simple Architecture Yet Enough Hardware) CPU
//------------------------------------------------------------------
---
//Controller
`timescale 1 ns /1 ns
module control (
     ExternalReset, clk,
      ResetPC, PCplusl, PCplus1, Rplusl, Rplus0,
      Rs_on_AddressUnitRSide, Rd_on_AddressUnitRSide, EnablePC,
      B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB,
AcmpB,sel_aluout_rfin,sel_rfin,
      RFLwrite, RFHwrite,
      IRload, SRload,
      Address_on_Databus, ALU_on_Databus, IR_on_LOpndBus, IR_on_HOpndBus,
```

```
RFright_on_OpndBus,
        ReadMem, WriteMem, ReadIO, WriteIO, Cset, Creset,
       Instruction,
       Cflag,Zflag ,memDataReady
)
input ExternalReset, clk;
output
       ResetPC, PCplusl, PCplus1, Rplusl, Rplus0,
       Rs_on_AddressUnitRSide, Rd_on_AddressUnitRSide, EnablePC,
        B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB,
AcmpB,sel_aluout_rfin,sel_rfin,
        RFLwrite, RFHwrite,
        IRload, SRload,
        Address_on_Databus, ALU_on_Databus, IR_on_LOpndBus, IR_on_HOpndBus,
RFright_on_OpndBus,
        ReadMem, WriteMem, ReadIO, WriteIO, Cset, Creset;
reg
       ResetPC, PCplusl, PCplus1, Rplusl, Rplus0,
       Rs_on_AddressUnitRSide, Rd_on_AddressUnitRSide, EnablePC,
        B15to0, AandB, AorB, notB, shlB, shrB, AaddB, AsubB, AmulB,
AcmpB,sel_aluout_rfin,sel_rfin,
        RFLwrite, RFHwrite,
```

```
        WPreset, WPadd, IRload, SRload,
        Address_on_Databus,  ALU_on_Databus,  IR_on_LOpndBus,IR_on_HOpndBus,
RFright_on_OpndBus,
        ReadMem, WriteMem, ReadIO, WriteIO, Cset, Creset;
input[15:0] Instruction;
input Cflag, Zflag,memDataReady;
parameter [3:0]
      reset = 0,
      fetch = 1,
      memread = 2,
      exec1 = 3,
      exec1lda = 4,
      //exec2lda = 7,
      exec1inp=5,
      incpc = 6;
parameter b0000 = 4'b0000;
parameter b1111 = 4'b1111;
parameter nop = 4'b0000;
parameter scf = 4'b0001;
parameter ccf = 4'b0010;
parameter jpr = 4'b0011;
parameter jz    = 4'b0100
```

```
parameter mvr = 4'b0001;

parameter lda = 4'b0010;

parameter sta = 4'b0011;

parameter inp = 4'b0100;

parameter oup = 4'b0101;

parameter anl = 4'b0110;

parameter orr = 4'b0111;

parameter nol = 4'b1000;

parameter shl = 4'b1001;

parameter shr = 4'b1010;

parameter add = 4'b1011;

parameter sub = 4'b1100;

parameter mul = 4'b1101;

parameter cmp = 4'b1110;

parameter mil = 2'b00;

parameter mih = 2'b01;

parameter jpa = 2'b10;

reg[3:0] Pstate, Nstate;

//------------------------------------------------------------------

-

    always @ (Instruction or Pstate or ExternalReset or Cflag  or Zflag or

memDataReady)
```

```
begin
    ResetPC             = 1'b0;
    PCplusl             = 1'b0;
    PCplus1             = 1'b0;
    Rplusl              = 1'b0;
    Rplus0              = 1'b0;
    EnablePC            = 1'b0;
    B15to0                  = 1'b0;
    AandB               = 1'b0;
    AorB                = 1'b0;
    notB                = 1'b0;
    shrB                = 1'b0;
    shlB                = 1'b0;
    AaddB                   = 1'b0;
    AsubB               = 1'b0;
    AmulB                   = 1'b0;
    AcmpB                   = 1'b0;
    RFLwrite            = 1'b0;
    RFHwrite            = 1'b0;
    //WPreset               = 1'b0;
    //WPadd                 = 1'b0;
    IRload              = 1'b0;
```

```
SRload                  = 1'b0;
sel_aluout_rfin = 1'b0;
sel_rfin                = 1'b0;
Address_on_Databus      = 1'b0;
ALU_on_Databus          = 1'b0;
IR_on_LOpndBus          = 1'b0;
IR_on_HOpndBus          = 1'b0;
RFright_on_OpndBus      = 1'b0;
ReadMem                 = 1'b0;
WriteMem                = 1'b0;
ReadIO                  = 1'b0;
WriteIO                 = 1'b0;
//Shadow                = 1'b0;
Cset                    = 1'b0;
Creset                  = 1'b0;
//Zset                  = 1'b0;
//Zreset                = 1'b0;
Rs_on_AddressUnitRSide = 1'b0;
Rd_on_AddressUnitRSide = 1'b0;
case (Pstate)
reset : // 0000
    if(ExternalReset == 1'b1) begin
```

```
            //WPreset = 1'b1;
            ResetPC = 1'b1;
            EnablePC=1'b1;
            Creset = 1'b1;
            //Zreset = 1'b1;
            Nstate = reset;
        end
        else
            Nstate = fetch;
    fetch : // 0001
        if(ExternalReset == 1'b1)
            Nstate = reset;
        else begin
            ReadMem = 1'b1;
            Nstate = memread;
        end
    memread : // 0010
        if(ExternalReset == 1'b1)
            Nstate = reset;
        else begin
            if (memDataReady == 1'b0) begin
                ReadMem = 1'b1;
```

```
                Nstate = memread;
            end
            else begin
                IRload = 1'b1;
                Nstate = exec1;
            end
        end
exec1 : // 0011
        if(ExternalReset == 1'b1)
            Nstate = reset;
        else begin
            case (Instruction[15:12])
            b0000 :
                case (Instruction[11:8])
                nop :
                    begin
                        PCplus1 = 1'b1;
                        EnablePC=1'b1;
                        Nstate = fetch;
                    end
                scf : begin
                        Cset = 1'b1;
```

```
            PCplus1 = 1'b1;
            EnablePC=1'b1;
            Nstate = fetch;
end
ccf : begin
           Creset = 1'b1;
            PCplus1 = 1'b1;
            EnablePC=1'b1;
            Nstate = fetch;
end
jpr : begin
      PCplusl = 1'b1;
      EnablePC=1'b1;
      Nstate = fetch;
end
jz : begin
     if(Zflag==1'b0)begin
     Rplusl=1'b1;
      EnablePC=1'b1;
      Nstate = fetch;
      end
      else
```

```
            begin
            PCplus1 = 1'b1;
            EnablePC = 1'b1;
            Nstate = fetch;
            end
        end
        default: begin
            PCplus1 = 1'b1;
            EnablePC = 1'b1;
            Nstate = fetch;
        end
        endcase
    mvr : begin
        //添加代码
        RFright_on_OpndBus=1'b1;
        B15to0=1'b1;
        ALU_on_Databus=1'b1;
        sel_aluout_rfin=1'b1;
        RFLwrite=1'b1;
        RFHwrite=1'b1;
        SRload=1'b1;
        PCplus1=1'b1;
```

```
    EnablePC=1'b1;
    Nstate=fetch;

end

lda : begin
      Rplus0 = 1'b1;
      Rs_on_AddressUnitRSide = 1'b1;
      ReadMem = 1'b1;
      Nstate = exec1lda;
end
sta : begin
      Rplus0 = 1'b1;
      Rd_on_AddressUnitRSide = 1'b1;
      RFright_on_OpndBus = 1'b1;
      B15to0 = 1'b1;
      WriteMem = 1'b1;
      Nstate = incpc;
end
inp : begin
      ReadIO = 1'b1;
      Nstate = exec1inp;
```

```
end
oup : begin
      //Rplus0 = 1'b1;
      //Rd_on_AddressUnitRSide = 1'b1;
      RFright_on_OpndBus = 1'b1;
      B15to0 = 1'b1;
      WriteIO = 1'b1;
      PCplus1 = 1'b1;
    EnablePC = 1'b1;
    Nstate = fetch;
end
anl : begin
      RFright_on_OpndBus = 1'b1;
      AandB = 1'b1;
      ALU_on_Databus = 1'b1;
      sel_aluout_rfin=1'b1;
      RFLwrite = 1'b1;
      RFHwrite = 1'b1;
      SRload = 1'b1;
      PCplus1 = 1'b1;
      EnablePC=1'b1;
      Nstate = fetch;
```

```
end
orr : begin
    RFright_on_OpndBus = 1'b1;
    AorB = 1'b1;
    ALU_on_Databus = 1'b1;
    sel_aluout_rfin=1'b1;
    RFLwrite = 1'b1;
    RFHwrite = 1'b1;
    SRload = 1'b1;
    PCplus1 = 1'b1;
    EnablePC=1'b1;
    Nstate = fetch;
end
nol : begin
    RFright_on_OpndBus = 1'b1;
    notB = 1'b1;
    ALU_on_Databus = 1'b1;
    sel_aluout_rfin=1'b1;
    RFLwrite = 1'b1;
    RFHwrite = 1'b1;
    SRload = 1'b1;
    PCplus1 = 1'b1;
```

```
        EnablePC=1'b1;
        Nstate = fetch;
    end
    shl : begin
        RFright_on_OpndBus = 1'b1;
        shlB = 1'b1;
        ALU_on_Databus = 1'b1;
        sel_aluout_rfin=1'b1;
        RFLwrite = 1'b1;
        RFHwrite = 1'b1;
        SRload = 1'b1;
        PCplus1 = 1'b1;
        EnablePC=1'b1;
        Nstate = fetch;
    end
    shr : begin
        RFright_on_OpndBus = 1'b1;
        shrB = 1'b1;
        ALU_on_Databus = 1'b1;
        sel_aluout_rfin=1'b1;
        RFLwrite = 1'b1;
        RFHwrite = 1'b1;
```

```
        SRload = 1'b1;
        PCplus1 = 1'b1;
        EnablePC=1'b1;
        Nstate = fetch;
    end
    add : begin
        RFright_on_OpndBus = 1'b1;
        AaddB = 1'b1;
        ALU_on_Databus = 1'b1;
        sel_aluout_rfin=1'b1;
        RFLwrite = 1'b1;
        RFHwrite = 1'b1;
        SRload = 1'b1;
        PCplus1 = 1'b1;
        EnablePC=1'b1;
        Nstate = fetch;
    end
    sub : begin
        RFright_on_OpndBus = 1'b1;
        AsubB = 1'b1;
        ALU_on_Databus = 1'b1;
        sel_aluout_rfin=1'b1;
```

```
        RFLwrite = 1'b1;
        RFHwrite = 1'b1;
        SRload = 1'b1;
        PCplus1 = 1'b1;
        EnablePC=1'b1;
        Nstate = fetch;
    end
    mul : begin
        RFright_on_OpndBus = 1'b1;
        AmulB = 1'b1;
        ALU_on_Databus = 1'b1;
        sel_aluout_rfin=1'b1;
        RFLwrite = 1'b1;
        RFHwrite = 1'b1;
        SRload = 1'b1;
        PCplus1 = 1'b1;
        EnablePC=1'b1;
        Nstate = fetch;
    end
    cmp : begin
        RFright_on_OpndBus = 1'b1;
        AcmpB = 1'b1;
```

```
        SRload = 1'b1;
        PCplus1 = 1'b1;
        EnablePC=1'b1;
        Nstate = fetch;
    end
    b1111 :
        case (Instruction[9: 8])
        mil : begin
            //添加代码
            IR_on_LOpndBus=1'b1;
            B15to0=1'b1;
            ALU_on_Databus=1'b1;
            sel_aluout_rfin=1'b1;
            RFLwrite = 1'b1;
            SRload = 1'b1;
            PCplus1 = 1'b1;
            EnablePC=1'b1;
            Nstate = fetch;
        end
        mih : begin
            //添加代码
            IR_on_HOpndBus=1'b1;
```

```
            B15to0=1'b1;
            ALU_on_Databus=1'b1;
            sel_aluout_rfin=1'b1;
            RFHwrite = 1'b1;
            SRload = 1'b1;
            PCplus1 = 1'b1;
            EnablePC=1'b1;
            Nstate = fetch;
        end
        jpa : begin
            Rd_on_AddressUnitRSide = 1'b1;
            RplusI = 1'b1;
            EnablePC=1'b1;
            Nstate = fetch;
        end
        default:begin
                RplusI = 1'b1;
            EnablePC=1'b1;
            Nstate = fetch;
        end
        endcase
    default :
```

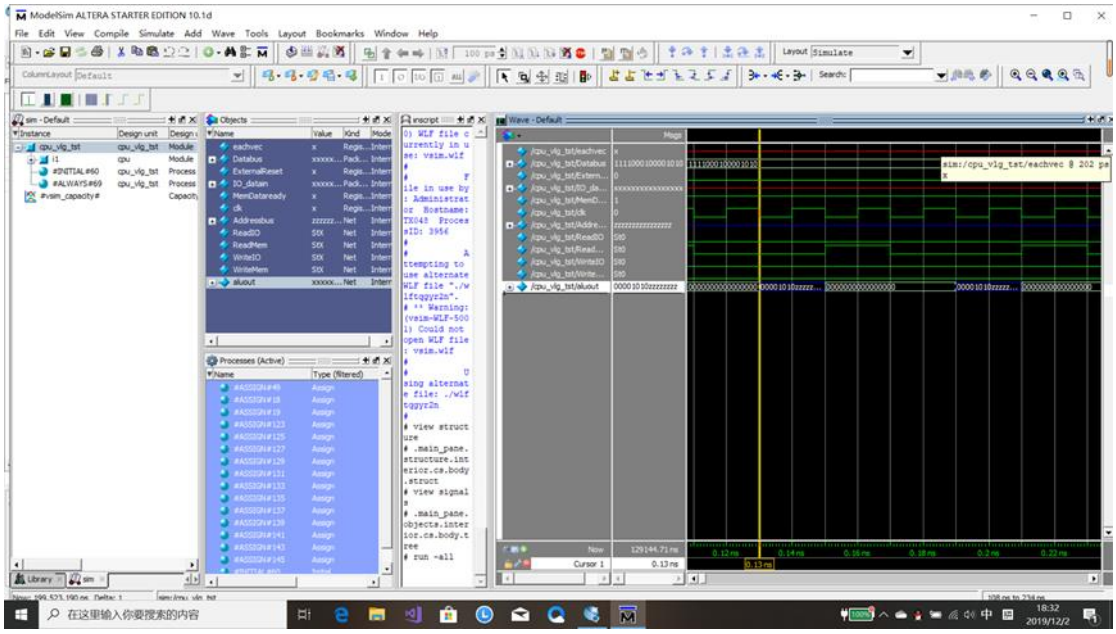
```
            Nstate = fetch;
        endcase
    end
exec1lda :
    if(ExternalReset == 1'b1)
        Nstate = reset;
    else begin
        if (memDataReady == 1'b0) begin
            Rplus0 = 1'b1;
            Rs_on_AddressUnitRSide = 1'b1;
            ReadMem = 1'b1;
            Nstate = exec1lda;
        end
        else begin
            RFLwrite = 1'b1;
            RFHwrite = 1'b1;
          sel_aluout_rfin = 1'b0;
            PCplus1 = 1'b1;
            EnablePC=1'b1;
            Nstate = fetch;
        end
    end
```

```
    exec1inp :
        if(ExternalReset == 1'b1)
            Nstate = reset;
        else begin
            RFLwrite = 1'b1;
            RFHwrite = 1'b1;
            sel_aluout_rfin = 1'b0;
            sel_rfin =1'b1;
            PCplus1 = 1'b1;
            EnablePC=1'b1;
            Nstate = fetch;
            end
    incpc : begin
        PCplus1 = 1'b1;
        EnablePC = 1'b1;
        Nstate = fetch;
    end
    default:   Nstate = reset;
    endcase
end
always @ (negedge clk)
    Pstate = Nstate;
```

Endmodule

2、关键代码分析

仿真波形如图



mvr：首先选择寄存器阵列的 Rd 和 Rs 分别作为 ALU 的输入数据 A 和 B，然后置标志 位 B15to0，此时 ALU 模块收到标志位后就会运算出相应的结果输出到 aluout。根据 指令定义存储到寄存器阵列中的 Rd 中，因此通过 sel_aluout_rfin 信号选通 aluout 输入到寄存器阵列，通过置位 RFLwrite、RFHwrite 存储结果，置位 SRload 重载状态位。执行完毕后，PC+1 操作使控制通路跳转到 fetch 状态准备执行 下一条指令。

mil：首先选择指令寄存器的 Imm 作为 ALU 的输入数据 ，然后置标志 位 B15to0，此时 ALU 模块收到标志位后就会运算出相应的结果输出到 aluout。根据 指令定义存储到寄存器阵列中的 Rd 中，因此通过 sel_aluout_rfin 信号选通 aluout 输入到寄存器阵列，通过置位 RFLwrite 存储结果，置位 SRload 重载状态位。执行完毕后，PC+1 操作使控制通路跳转到 fetch 状态准备执行 下一条

指令。

mih：首先选择指令寄存器的 Imm 作为 ALU 的输入数据 ，然后置标志 位 B15to0，此时 ALU 模块收到标志位后就会运算出相应的结果输出到 aluout。根据 指令定义存储到寄存器阵列中的 Rd 中，因此通过 sel_aluout_rfin 信号选通 aluout 输入到寄存器阵列，通过置位 RFHwrite 存储结果，置位 SRload 重载状态位。执行完毕后，PC+1 操作使控制通路跳转到 fetch 状态准备执行 下一条指令。

任务 4

1、汇编代码对应的机器码

1111100000000001

1111100100000000

1111000000001000

1111000100000000

1111010000000001

1111010100000000

0101010100000010

1001010100000000

1111110000000010

1111110100000000

1100111000000000

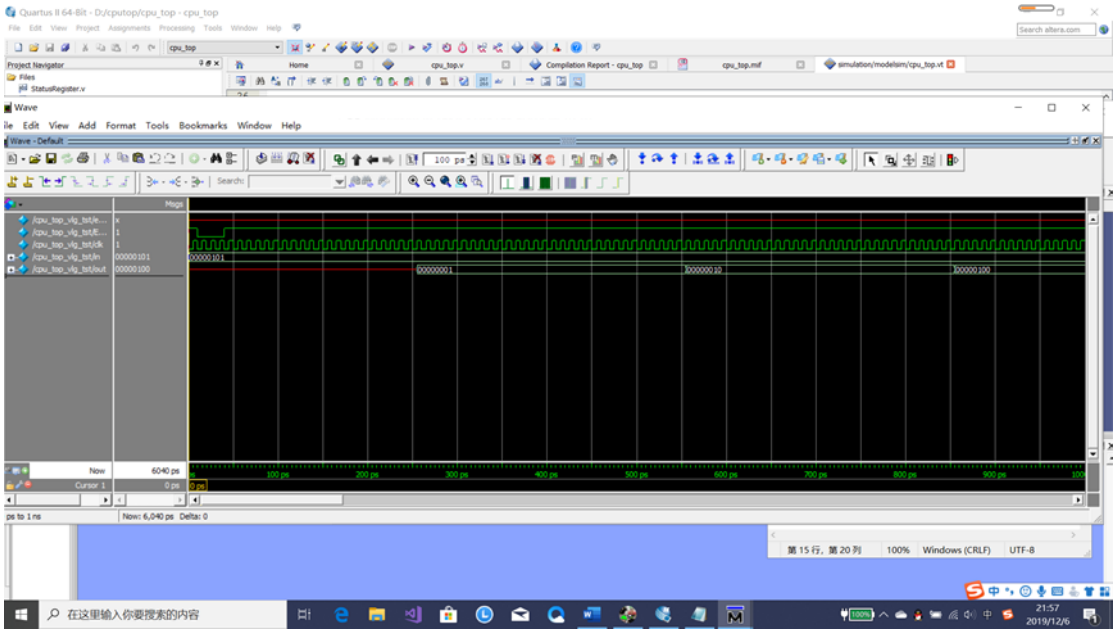
0000010000001010

1100001000000000

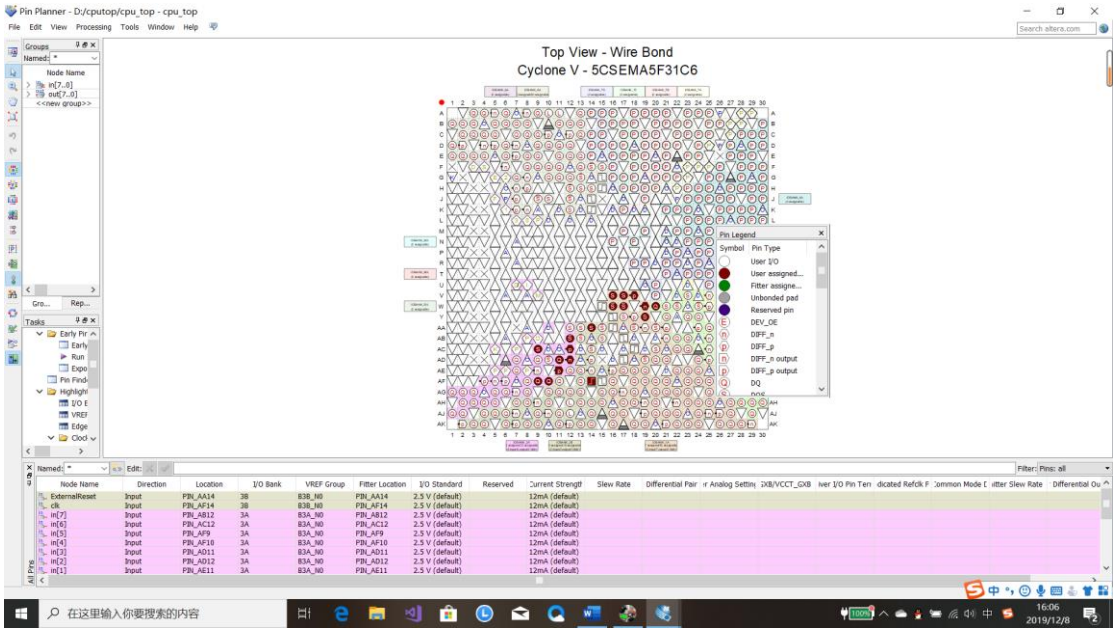
0000010000000110

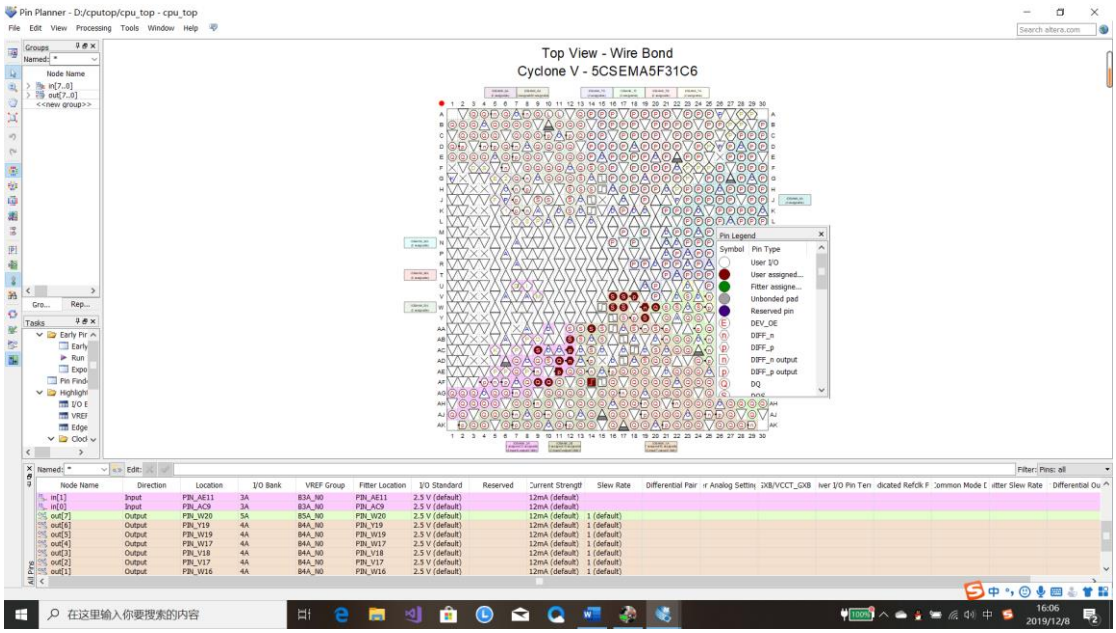
1111001000000010

仿真波形如图



2、下板引脚绑定





七、自主设计部分

无

八、实验中遇到的问题及解决方法

1、进行 ALU 的编写时出现报错显示格式错误

解决方法：将每段代码依次注释掉后重新进行编译，最后定位到错误所在。

2、在对 cpu-top 进行仿真时，出现波形不方便观察的情况

解决方法：将代码中 PLL 部分注释掉，并将 pll 生成的时钟全部改为原时钟，将地址 8 的指令改为 1111110000000010 以减小流水灯的变化间隔，最后成功做出仿真波形。

九、思考题

1.该 cpu 执行一条指令需要几个时钟周期？每个时钟周期分别完成那些功能？

需要 5 个时钟周期，最长的操作每个时候总周期分别完成：取指令、译码寄存器文件、执行 ALU、存储器读写、写寄存器文件这五步操作

2.该 cpu 寄存器阵列共有多少个寄存器？如何用指令给这些寄存器赋值？

共有 4 个寄存器

分别标记为 00/01/10/11，使用机器码中的相应命令对其进行赋值

3.如果没有标志位寄存器，会对 cpu 哪些功能造成什么影响？

如果没有标志位寄存器，CPU 可能无法从这里获得需要执行的指令，也无法对 CPU 工作进行控制

十、实验分工说明

周子涵 ： 100%

十一、实验感想和建议

本次数字电路实验提高了我的动手能力，增强了我对 verilog 代码的理解，提高了我对机器语言的认识。建议可以在完成实验的基础上具体地给出该 cpu 还可以在哪些地方有所改进，以进一步提高教学成果。