计算机系统结构实验指导书-LAB5

1. OVERVIEW

1.1 实验名称

简单的类 MIPS 单周期处理器的实现 – 整体调试

1.2 实验目的

1. 理解简单的类 MIPS 单周期处理器的工作原理(即几类基本指令执行时所需的数据通路和与之对应的控制线路及其各功能部件间的互联定义、逻辑选择关系)
2. 完成简单的类 MIPS 单周期处理器
   1) 9 条 MIPS 指令(lw, sw, beq, add, sub, and, or, slt, j) CPU 的实现与调试
   2) 拓展至 16 条指令(增加 addi, andi, ori, sll, srl, jal, jr) CPU 的实设计与实现

      Ps：若事先考虑妥当准备充分允许直接完成 16 条指令执行的设计
3. 仿真测试
4. 上板验证（不做）

1.3 实验报告与验收办法

应深入细致完成本实验报告。线上自查分析单周期 CPU 指令执行的仿真结果并截图

1.4 注意事项

本实验建立在前几个实验的基础上，旨在理解和掌握简单的类 MIPS 单周期处理器的设计与整体调试

1.5 实验预计时间

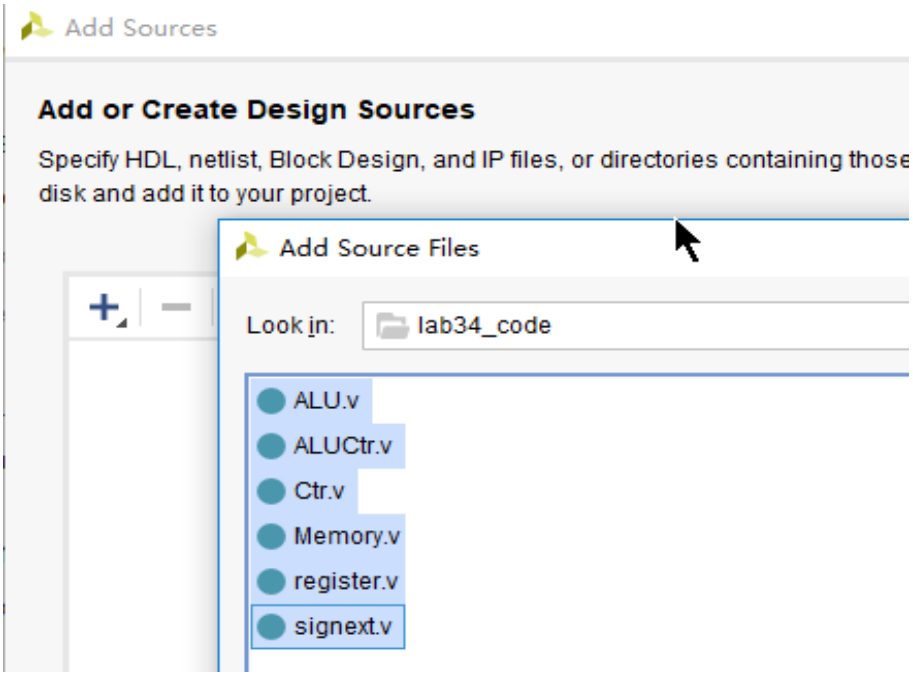
240~480 分钟

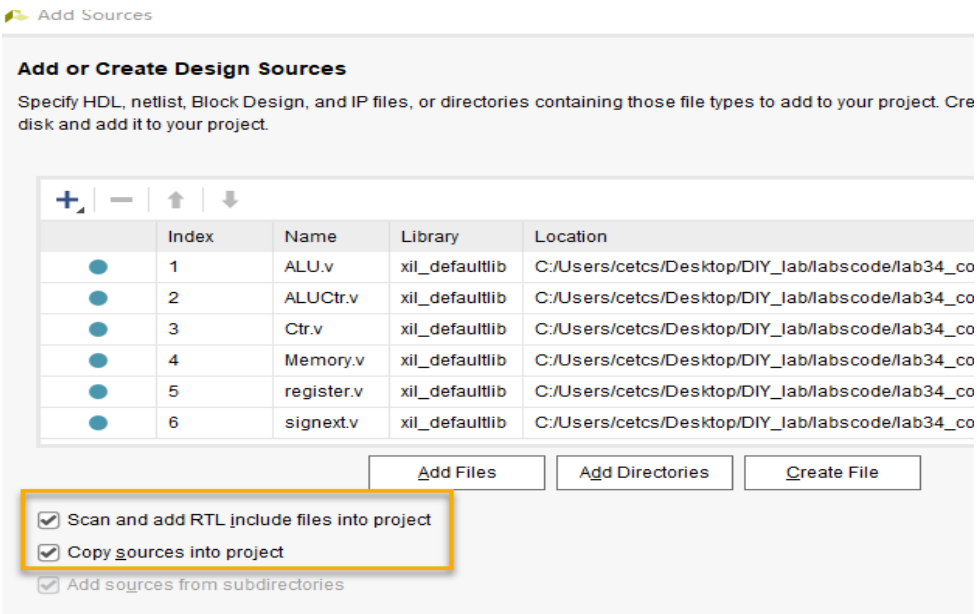
2. 新建工程，导入文件

2.1 实验描述

2.1.1 新建工程

1. 启动 Vivado
2. 新建工程 lab5
3. 将此前两次实验中的已有模块添加到 lab5 工程目录下：





3. 创建顶层模块（TOP）

3.1 类 MIPS 单周期处理器实验原理描述

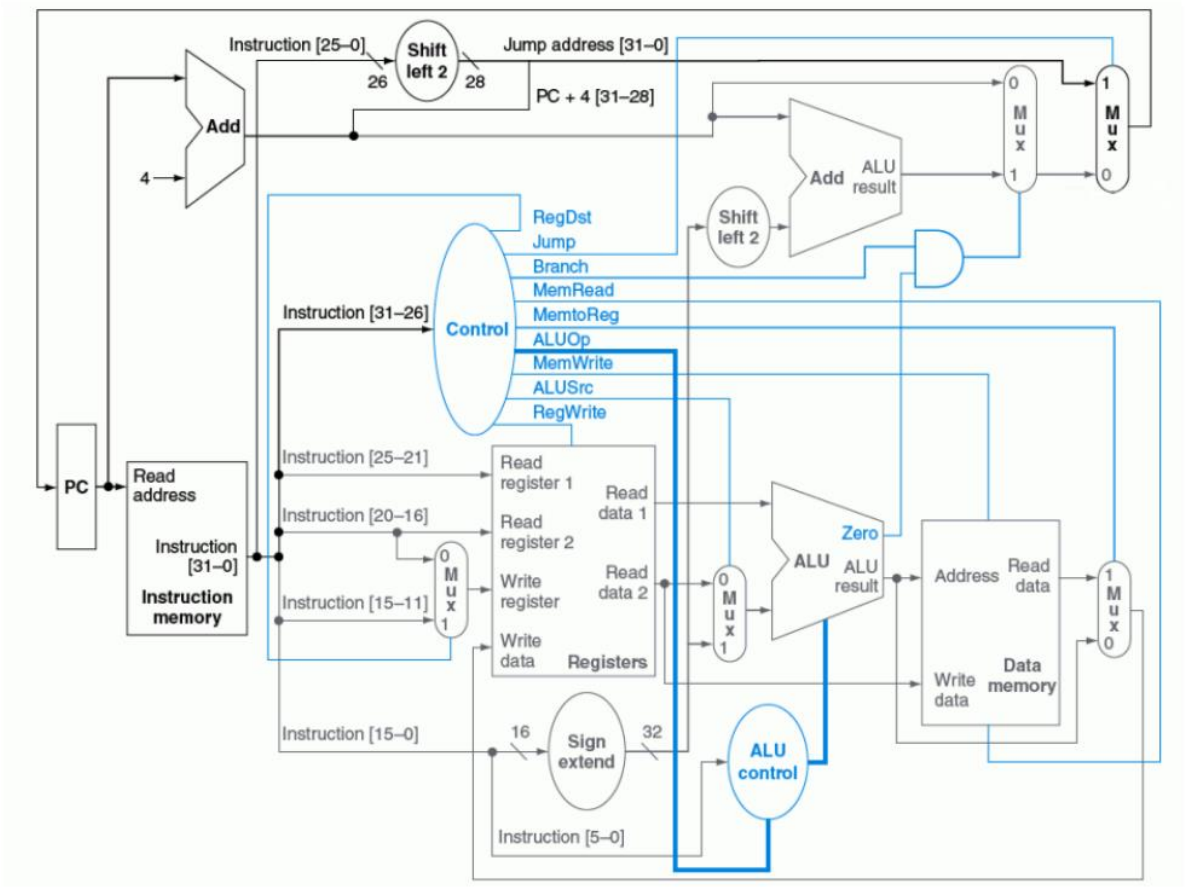


图 1. 能处理跳转等几类 MIPS 指令的简单的单周期处理器电路设计图

PS:

1) 图中由于各种变量或信号名比较众多复杂，需要创建一套命名规则，方便代码的编写和阅读。
2) 除了实验 3、4 已编写好的代码模块外，大家可以凭借个人喜好选择将图 1 中的其它功能部件或单元设计成模块文件。
3) 顶层模块是图 1 所有数据通路和控制通路的总体功能模块，顶层模块里含有各功能子模块间的互联定义。

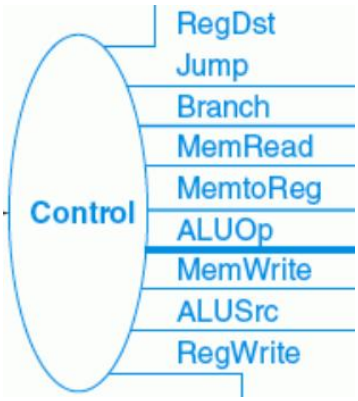
3.1.1 创建顶层模块文件，可命名 Top

3.1.2 定义信号线

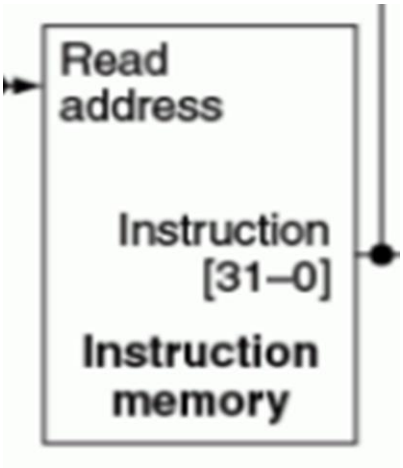
为 Top 模块内的每一根连接的信号线命名，并在 Top 模块中声明定义它们。

例如，主控制模块输出端口上的连线：

```
47      wire REG_DST,
48           JUMP,
49           BRANCH,
50           MEM_READ,
51           MEM_TO_REG,
52           MEM_WRITE;
53      wire[1:0] ALU_OP;
54      wire ALU_SRC,
55           REG_WRITE;
```



3.1.3 指令存储器(Instruction memory)



此图是图 1 中指令存储器部件，可以考虑设计成模块文件来描述，可取名 InstMemory.v

3.1.4 程序计数器 PC

程序计数器是这个简单 CPU 能够跑起来的关键。定义一个 32 位 reg 类型 PC，在时钟上升沿(下降沿已在 Lab4 被用作寄存器的写)做 PC<=PC+4。

PS：1）简单的讲，在组合逻辑中用阻塞赋值“=”，时序逻辑中用非阻塞赋值“<=”。两者综合出来的电路不一样，具体区别查阅有关参考。时序逻辑和组合逻辑不要放在同一个 always 块中。

2）PC 也可以考虑用模块来实现

3.1.5 RESET

PC 置 0x00000000，各寄存器清零，这是 reset 要做的工作。同步或异步，边沿或电平，同学们可以自由实现。

寄存器清零，要适当修改之前的 registers 模块，并给模块添加 reset 信号。

PS：添加 reset 要注意，需写在原来“写”的 always 块中。假如新加一个 always 块，当个两个“写”always 同时满足时，就会混乱不知赋什么值了。

3.1.6 模块实例化，连接各模块

实例化前两次实验中编写的模块和新编写的模块，实例化的过程中连接模块的端口。实例化有以下两种方法：

1) 严格按照模块定义的端口顺序来连接，不用表明原模块定义时规定的端口名：

    模块　模块名(连接端口 1 信号名，连接端口信号名 2…)

2) 在连接时用“.”符号，表明原模块是定义时规定的端口名：

    模块　模块名(.端口 1 名(信号 1),　.端口 2　名(信号 2)…)

实验中推荐用第 2 种实例化方法。

以主控制模块为例，以下为 Ctr 模块的实例化、并连接其端口。其中 mainCtr 就是该模块的实例化名，INST 是定义好的指令存储器输出的连接信号，其它信号线我们应当在之前的众模块编写小节中定义好。

```
100     Ctr mainCtr(
101             .opcode(INST[31:26]),
102             .regDst(REG_DST),
103             .jump(JUMP),
104             .branch(BRANCH),
105             .memRead(MEM_READ),
106             .memToReg(MEM_TO_REG),
107             .aLUOp(ALU_OP),
108             .memWrite(MEM_WRITE),
109             .aLUSrc(ALU_SRC),
110             .regWrite(REG_WRITE));
```

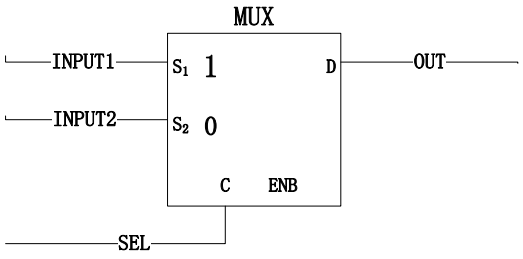
实例化 Ctr 模块

3.1.7 连接其它信号线

1. 多路选择器 MUX
   MUX 实现简单，一个三目运算符

   Assign　OUT = SEL ? INPUT1 : INPUT2;

   OUT, SEL, INPUT1, INPUT2 都是预先定义的信号



2. 左移两位，用移位运算符：左移(“<<”)。另，右移(“>>”)
3. 加法器，直接用无符号加法运算。
   注：verilog 中寄存器类型被解释成无符号数，整数类型(integer)被解释成二进制补码形式的有符号数。因此要综合成无符号算术算符需要使用寄存器类型，而要得到有符号算术算符就需要使用整数。网线类型被解释成无符号数。

4. 与门，使用位运算符&(位与)。注意&和&&的区别。
5. MUX 也可用模块来实现

注意：若要设计支持给定的 16 条 mips 指令的单周期 CPU，则图 1 的 CPU 电路设计原理图或许要适当的作些修改，上述已完成的且相关的模块也需做些修改

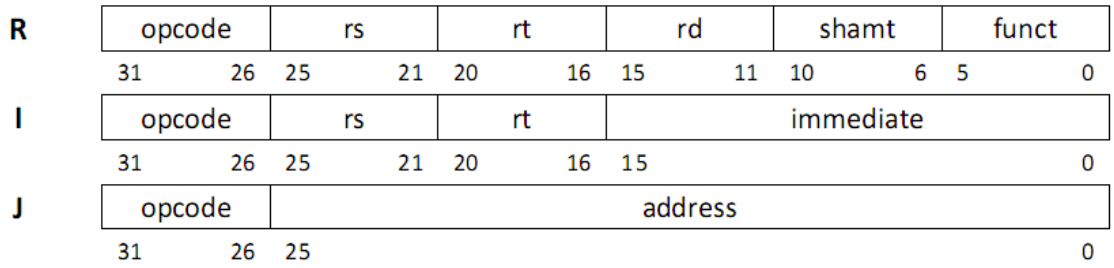
4. 仿真测试

1. 编写相关二进制测试程序

   9 条或 16 条指令的处理器设计完成后请编写自己的测试汇编。下面提供一个简易汇编器供参考。

   一些相关的基本知识：

   指令格式：



Mips 基本指令格式

汇编格式：注意汇编中寄存器的顺序跟指令格式中的不一样

add $1,$2,$3 ： $1=$2 + $3

sub $1,$2,$3 ： $1=$2 - $3

and $1,$2,$3 ： $1=$2 & $3

or $1,$2,$3 ： $1=$2 | $3

slt $1,$2,$3 ： if($2<$3) $1=1 else $1=0

lw $1,10($2) ： $1=memory[$2+10]

sw $1,10($2) ： memory[$2+10]=$1

beq $1,$2,10 ： if($1==$2) goto PC+4+40

[10 是 PC+4 后的指令间隔数，故为 PC+4+40]

j 10000 ： goto 10000

2. 系统任务$readmemb 和$readmemh，放在 initial 初始化块中。

   Verilog 中这两个系统任务用来从外部文件中读取数据到三大存储器中而作为初始化数值，

   格式如下：

   1) 相对路径：

   $readmemh(“Data”, memoryName);

   $readmemh("Instruction", InstMemFile);

相对路径时，其中外部文件 Data 或 Instruction 需事先存放在类似于“C:\Archlabs\lab05\lab05.sim\sim_1\behav**xsim**”的子文件夹里

2) 绝对路径：

$readmemh("D:/Archlab/lab05/lab05.srcs/Src/Data", MemFile);

或：

$readmemh("D:/Archlab/lab05/lab05.srcs/Src/Data", processor.data_memory.MemFile);

3. 这里，给出内存和指令内存的 mem_data 和 mem_inst 两个样例文件，分别用来装入 data memory 和 instruction memory。

mem_data：

```
00000000
00000001
00000002
00000003
00000004
00000005
00000006
00000007
00000008
```

… …

……

mem_inst：

```
00001000000000000000000000000100        // j a
00000000000000000000000000000000        // nop
00000000000000000000000000000000        // nop
00000000000000000000000000000000        // nop
10001100000000010000000100010100        // lw $1, 276($0)
10001100000000010000000100001000        // lw $2, 264($0)
00000000001000100001100000100000        // add $3, $1, $2
00000000001000100010000000100010        // sub $4, $1, $2
```

… …

……

4. 其它常用系统任务(若需要用的话)

$monitor 提供监控和输出参数列表中的表达式或变量的值功能。

如，$monitor(“rxd=%b”, rxb);

$time 返回当前仿真时刻值。

如，$monitor($time);

5. 编写激励代码，进行仿真测试：

   1) 初始化 data memory、instruction memory 和 register 三大存储模块，这里仅以初始化 instruction memory 为例。如下图，Verilog 中调用了系统任务$readmemh 将 Instruction 文件中的指令数据读到 InstMemFile 数组中。

```
26  reg [31:0] InstMemFile [0:255];
27  initial begin
28      $readmemh("Instruction",InstMemFile);
29  end
```

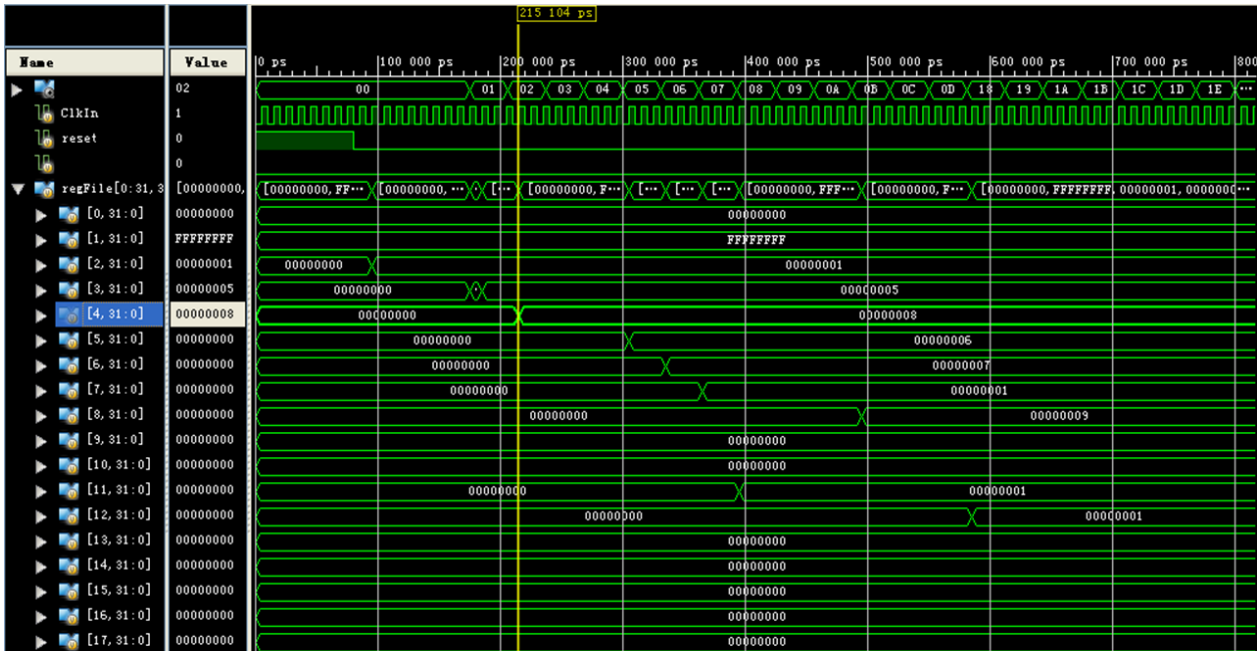
   2) 添加时钟激励和其它输入信号并给定初始值等。

```
42      initial begin
43          // Initialize Inputs
44          Clk = 0;
45          reset = 0;
46          // Wait 100 ns for global reset to finish
47          #100;
48          reset =1;
49          #200;
50          reset =0;
51
52       end
```

   3) 添加 register 模块中的 regfile 寄存器数组以及其它需要的变量或信号到仿真波形窗口，观察各个相关数值的变化情况。

   4) 在 Tcl Console 窗口中分别输入 restart 和 run 2000ns 两条命令，以便重新进行仿真。观察仿真波形与你设计编排的 MIPS 指令预期运行的结果进行比对……

6. 下面给出一个 9 条指令执行的仿真样例：



5. 上板调试 （不需要做）

1. 利用 switch、led 甚至是七段数码管等来观察指令运行结果是否预期
2. 可参考 lab2 相关外设上板验证的方法