合肥工业大学计算机与信息学院

计算机组成原理实验教程

即

基于 Verilog HDL 的 CPU 设计实验教程

安鑫

2019.2.14

目录

实验一、Verilog 与 ModelSim 基础

一、实验目的：

熟悉并掌握 Verilog HDL 与 ModelSim 的使用

二、实验环境：

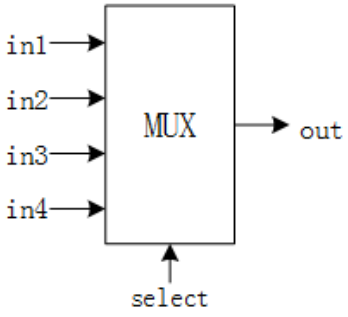
ModelSim

三、实验内容：

学习使用 Verilog 完成 4 选 1 多路选择器的设计和实现,并使用 ModelSim 工具对设计进行仿真和分析验证。

四、实验原理

多路选择器（MUX）是一种在多路数据传送过程中，能够根据需要将其中任意一路选出来的电路，其原理图和真值表如下图所示。



选择输入	数据输入				输出
select	in1	in2	in3	in4	out
0 0	in1	×	×	×	in1
0 1	×	in2	×	×	in2
1 0	×	×	in3	×	in3
1 1	×	×	×	in4	in4

图 1　4 选 1 多路选择器及其真值表

五、实验内容（步骤）

5.1、Verilog 关键代码实现

表 1. MUX 模块功能描述

输入	4 位输入信号 in1、in2、in3、in4 和 2 位选择信号 select
输出	4 位输出信号 out
功能	根据选择信号 select 的值把相应输入信号赋值给 out 输出

MUX 模块的 verilog 代码如下：

```
module mux41(
  input wire [3:0] in1, in2, in3, in4,
  input wire [1:0] select,
  output reg [3:0] out
);

  always@* begin
    case (select)
      2'b00: out = in1;
      2'b01: out = in2;
      2'b10: out = in3;
      2'b11: out = in4;
      default: out = 4'bx;
    endcase
  end

endmodule
```

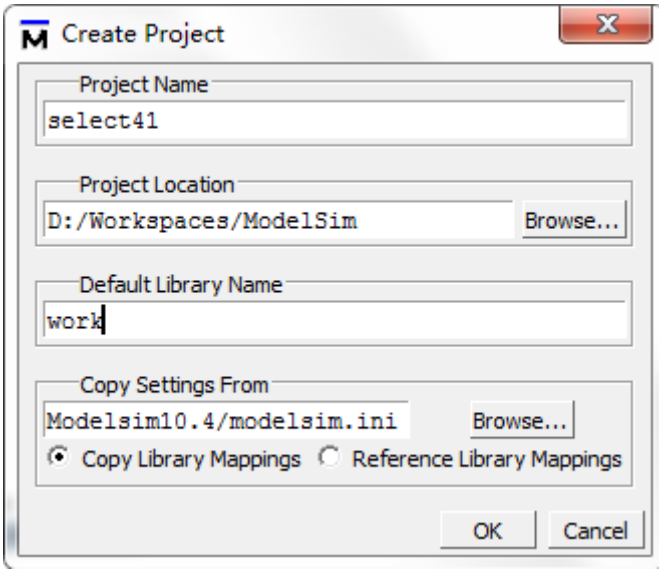
5.2、测试文件(TestBench)关键代码描述

```
module mux41_tb;

  reg [3:0] in1, in2, in3, in4;
  reg [1:0] select;
  wire [3:0] out;

  initial begin
    in1 = 4'b0001;
    in2 = 4'b0011;
    in3 = 4'b0111;
    in4 = 4'b1111;
    select = 2'b00;

    #10 select = 2'b01;

    #10 select = 2'b10;

    #10 select = 2'b11;

    #10 $stop;
  end

  mux41 uut(
   .in1(in1), .in2(in2), .in3(in3), .in4(in4),
   .select(select),
   .out(out)
  );

endmodule
```

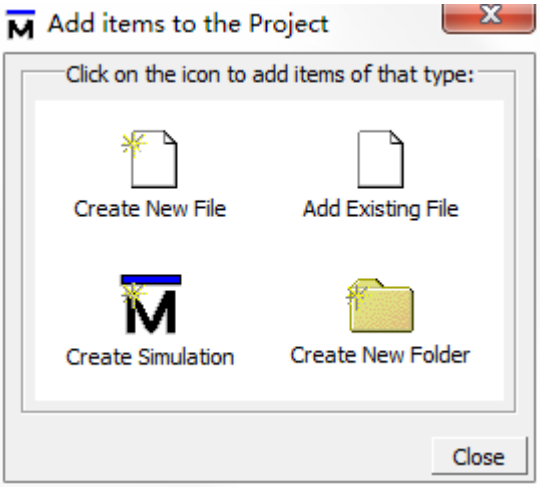
5.3、ModelSim 仿真及分析

5.3.1 建立 ModelSim 工程

打开 ModelSim，选择 File->New->Project，出现 Create Project 对话框，填写工程名（Project Name），选择保存目录（Project Location），注意保存目录中不要有中文，如下图所示：

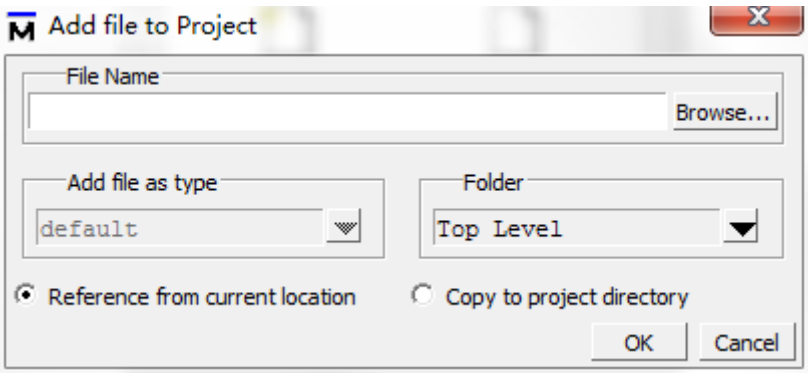


单击 OK 按钮后，会出现下图界面：



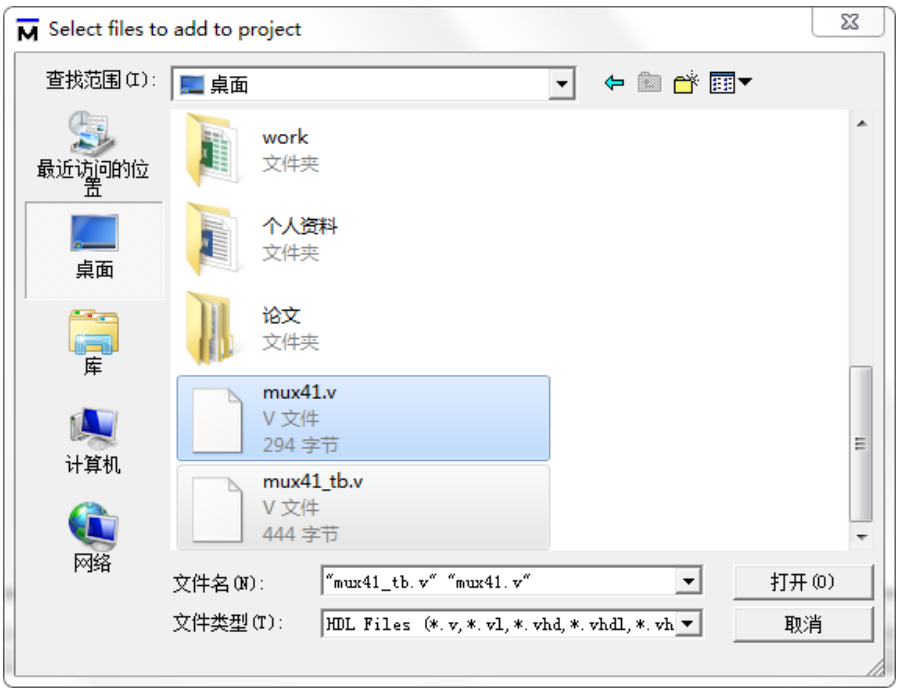
现在可以选 Create New File（新建文件）或者 Add Existing File（添加已存在文件）。这里

我们选择 Add Existing File，也就是添加 5.1 和 5.2 中的 Verilog 代码，会出现下图界面：

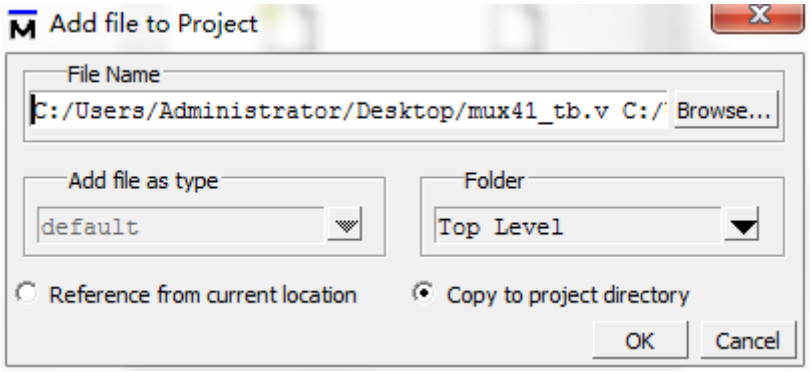


点击 Browse 按钮，添加 5.1 中的 mux41.v 和 5.2 中的测试文件 mux41_tb.v，会出现下图

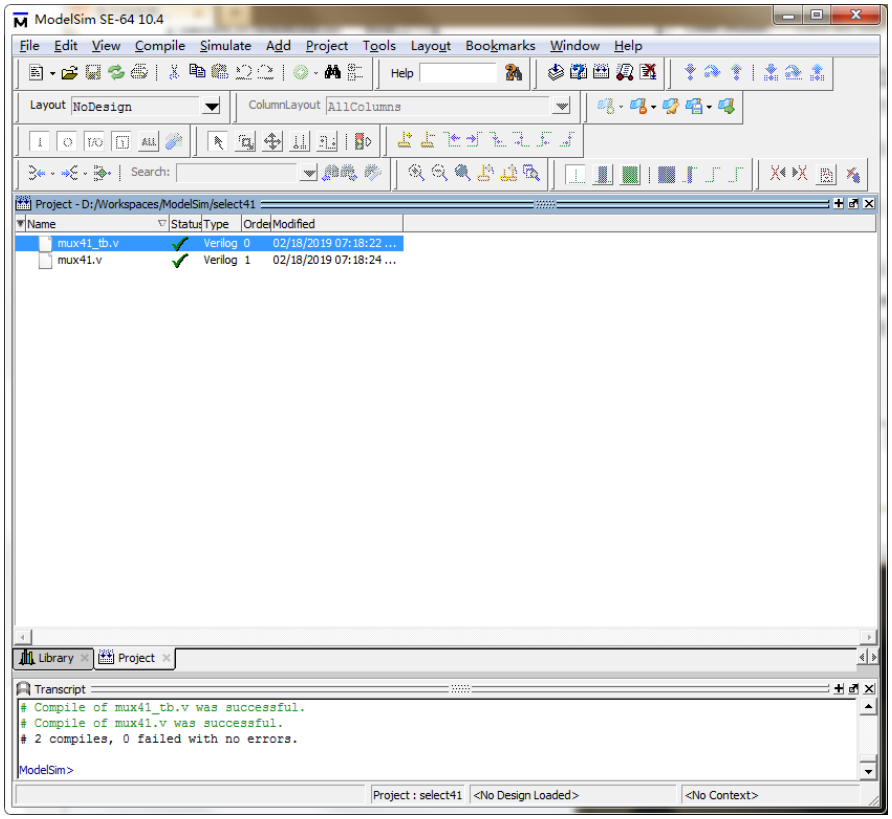
界面：



选择要添加的文件后，单击“打开”按钮，即添加完成，会出现下图界面，在其中选择 copy to project directory，这样就会将 mux41.v 和 mux41_tb.v 文件复制到新的工程目录下，单击 OK 按钮。

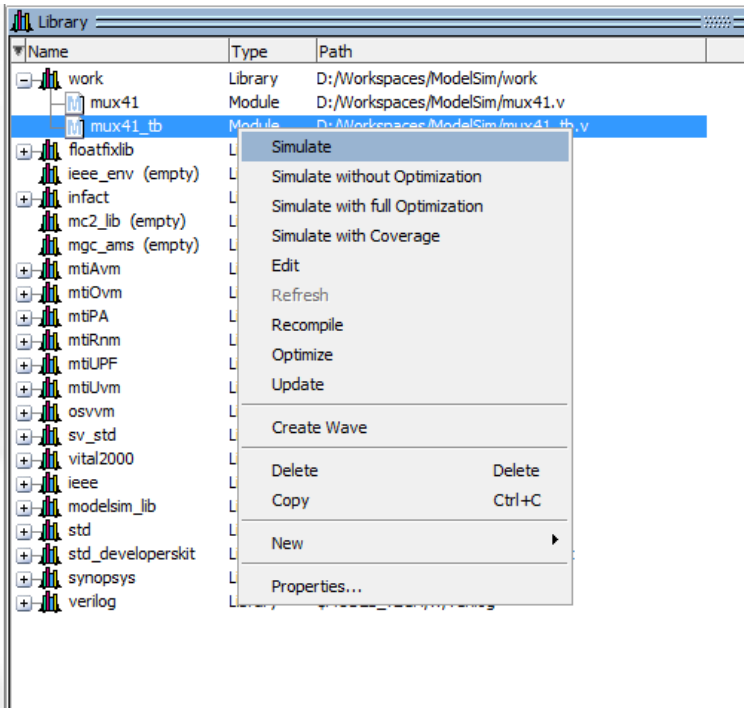


文件添加完成后，ModelSim 主界面会显示所有文件的状态。选中任意一个文件，右键单击，选择 Compile->Compile All，即开始编辑所有文件，会出现下图界面。没有出错，文件状态应该都是绿色的对号，否则点击屏幕下方的 Transcript，查看出错信息，直至无误。

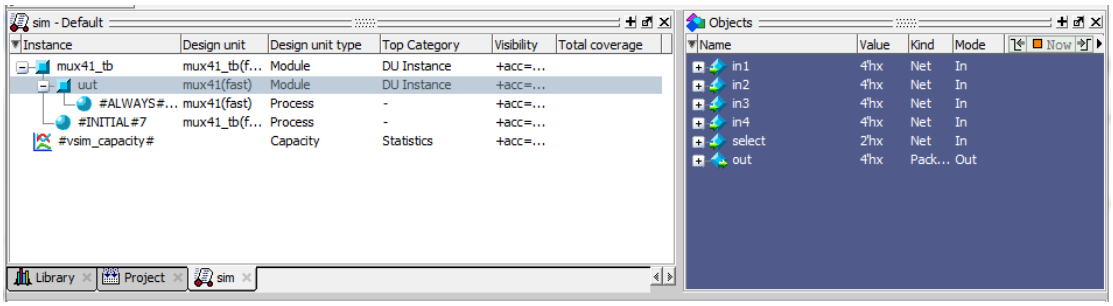


5.3.2 开始仿真

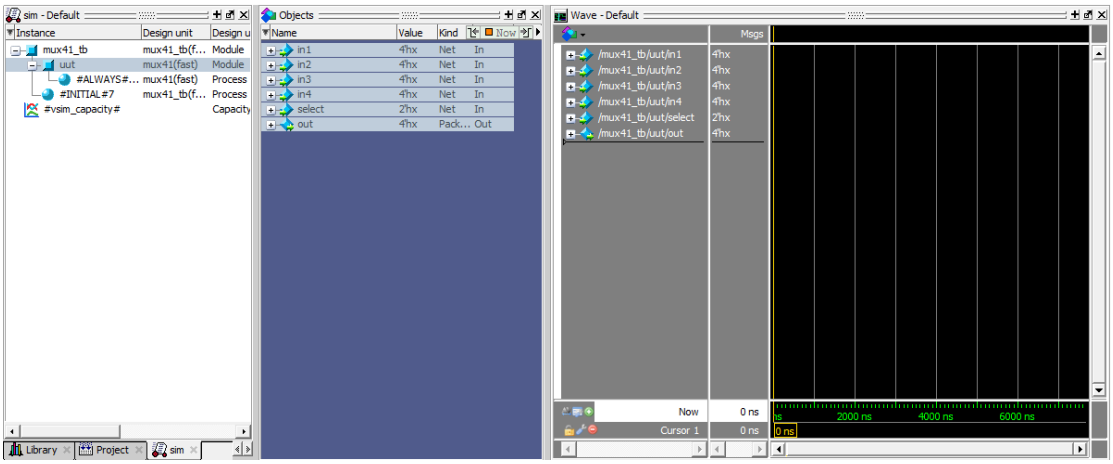
切换到 Library，然后展开 work 目录，在 mux41_tb.v 文件上单击右键，在弹出菜单中选择 Simulate （without Optimization），如下图界面：



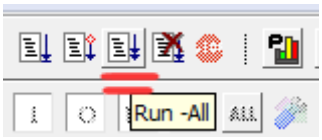
此时会出现一个名为 sim 的界面，展开其中的 mux41_tb 节点，选择 uut，会在 Objects 窗口显示所有信号，如下图所示：(若没有出现 Objects 窗口，可以通过菜单 View->Objects 调出该窗口)



选择 Objects 窗口的所有信号 (Ctrl+A)，然后单击右键，在弹出菜单中选择 Add to->Wave->Selected Signals，如下图所示：



单击工具栏中的 Run-All 按钮，便开始仿真，如下图所示：



仿真效果图，如下图所示：

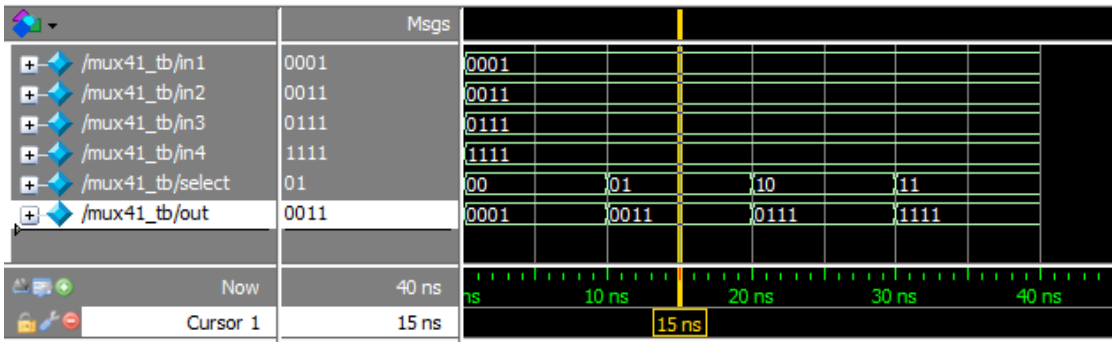
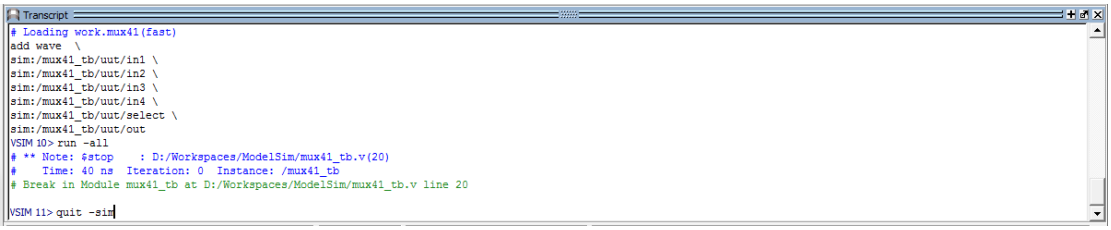


图 X. 多路选择器仿真结果

结束后，请在 Transcript 中输入 quit –sim 命令退出，如下图所示：



六、进一步实验

请使用 Verilog 完成 4 位全加器、8 位比较器、74138 译码器等模块设计,然后编写测试文件使用 ModelSim 进行仿真验证。

实验二、CPU 部件实现之 ALU 和寄存器堆

一、实验目的：

理解和掌握 CPU 中的算术逻辑运算部件（ALU）和寄存器堆（Register File）的工作原理，并使用 Verilog 和 ModelSim 进行设计和仿真。

二、实验内容：

1. 使用 Verilog 完成 ALU 的设计,并编写测试仿真文件验证其正确性。要求：
   - ALU 支持 16 位的加、减、与、或以及移位运算。
2. 使用 Verilog 完成通用寄存器堆的设计，并编写测试仿真文件验证其正确性。要求
   - 寄存器堆包含 8 个 16 位的寄存器；
   - 寄存器堆有两个读端口和一个写端口。

三、实验原理

请根据实验内容要求分别介绍 ALU 和寄存器堆的原理图和工作原理

四、实验步骤

参照实验一，依次补充本部分内容。

1. Verilog 关键代码描述
2. 测试文件描述
3. ModelSim 仿真及分析

五、总结

实验三、CPU 部件实现之 PC 和半导体存储器 RAM

一、实验目的：

理解和掌握 CPU 中程序计数器 PC 和半导体存储器 RAM 的工作原理，并使用 Verilog 和 ModelSim 进行设计和仿真。

二、实验内容：

1. 使用 Verilog 完成程序计数器 PC 的设计，要求：
   - PC 为 8 位计数器
2. 使用 Verilog 完成数据存储器的设计，并编写测试仿真文件验证其正确性。要求
   - 存储字长 16 位，存储容量 1K 字节；
   - 一根读写控制信号线控制读写，低电平有效。

三、实验原理

请根据实验内容要求分别介绍 PC 和 RAM 的原理图和工作原理

四、实验步骤

1. Verilog 关键代码描述
2. 测试文件描述
3. ModelSim 仿真（含仿真分析与总结）

五、总结

实验四、单周期 CPU 设计与实现——单指令 CPU

一、实验目的：

通过设计并实现支持一条指令的 CPU，理解和掌握 CPU 设计的基本原理和过程。

二、实验内容：

设计和实现一个支持**加法指令**的单周期 CPU。要求该加法指令（表示为 **add r1，r2，r3**）格式约定如下：

- 采用寄存器寻址，r1，r2，r3 为寄存器编号，r1 和 r2 存放两个源操作数，r3 为目标寄存器，其功能为[r1] + [r2] -> r3；
- 指令字长 16 位，操作码和地址码字段分配如下所示：

15 9	8 6	5 3	2 0
OpCode	r1	r2	r3

三、实验原理

单周期 CPU 是指所有指令均在一个时钟周期内完成的 CPU。CPU 由数据通路及其控制部件两部分构成，因而要完成一个支持若干条指令 CPU 的设计，需要依次完成以下两件事：

1）根据指令功能和格式设计 CPU 的数据通路；

2）根据指令功能和数据通路设计控制部件。

3.1 根据功能和格式完成 CPU 的数据通路设计

本实验需要设计的 CPU 只需要支持一条加法指令，而该指令的功能是在一个时钟周期内从寄存器组中 r1 和 r2 中取出两个操作数，然后送到 ALU 进行加法运算，最后把计算结果保存到 r1 寄存器中。下图给出了改加法指令的数据通路图。

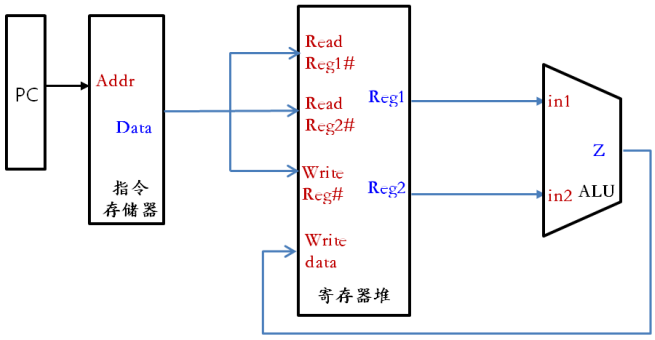


图 3.1. 加法指令 add r1,r2,r3 数据通路

此外，还需要确定各个部件的位数，为了简单起见，我们假设目标 CPU 的机器字长、存储字长和指令字长相等均为 16 位，存储单元个数假设为 256，按字寻址，并取 PC 位数为 8。

3.2 根据指令功能、数据通路完成控制单元的设计

控制单元的功能是为当前要执行的指令产生微操作命令从而完成该指令的执行。为了能够完成加法指令的执行，结合**图 1**，控制单元需要在取出指令后根据指令操作码（本例中是加法指令），控制 ALU（参考实验二）做加法（通过给 alu_op 信号线相应赋值），并把结果写回寄存器组（参考实验三）中（通过给 wr_en 赋值为 true）。图 2 给出了整合控制单元后目标 CPU 的原理图，系统时钟信号也已标注。

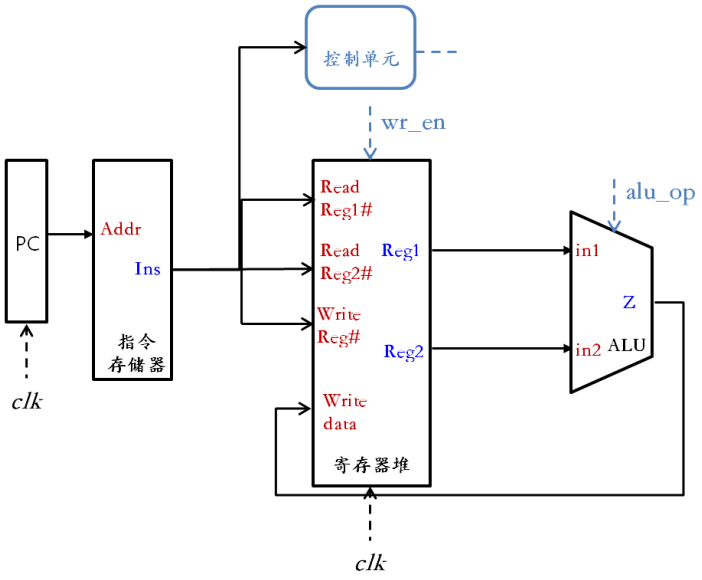


图 3.2. 单指令 CPU 原理图

四、实验步骤

在第三部分通过对该 CPU 实现细节的分析、设计，并得到该 CPU 的原理图后，就可以依次实现各个模块，并进行仿真验证了。

4.1 CPU 各模块 Verilog 实现

在前面实验中，已经分别设计和实现了 PC、指令存储器、寄存器组和 ALU，这里只给出各个模块的功能描述及其接口定义 ,具体实现可以直接使用或者调整前面试验的实现代码。

1）PC 模块

表 4.1 PC 模块功能描述

输入	时钟信号 clk、重置信号 rst
输出	指令地址 pc（8 位）
功能	每个时钟上升沿 PC 的值自动加 1，并输出

Verilog 关键代码：

2）指令存储器模块

表 4.2 指令存储器模块功能描述

输入	8 位指令地址 Addr
输出	16 位指令 Ins
功能	存放待执行的指令（初始化），并根据地址输出指令

Verilog 关键代码：

3）寄存器堆

表 4.3 寄存器堆模块功能描述

输入	时钟信号 clk、读写控制线 wr_en、读寄存器编号 read_reg1 和 read_reg2、写寄存器编号 write_reg、写入数据 write_data
输出	对应两个读寄存器编号的寄存器值 reg1 和 reg2
功能	根据读寄存器编号给出对应寄存器的值；在写允许情况下，把写入端的数据在 clk 下降沿写到写寄存器编号对应的寄存器

Verilog 关键代码：

4）ALU

表 4.4 ALU 模块功能描述

输入	操作数 in1 和 in2、操作选择信号 alu_op
输出	ALU 运算结果 Z
功能	根据操作选择信号计算 in1 和 in2 的运算结果 Z

Verilog 关键代码：

5）控制单元

表 4.5 控制单元模块功能描述

输入	指令（操作码）
输出	寄存器堆的读写控制线 wr_en、ALU 的操作选择信号 alu_op
功能	根据当前指令功能对 wr_en 和 alu_op 赋值

Verilog 关键代码：

4.2 CPU 顶层文件封装实现

通过根据图 2 将以上定义的模块进行连接、封装就得到了目标 CPU ,该 CPU 的输入为系统时钟信号 clk 和重置信号 reset。

Verilog 关键代码：

4.3 CPU 模拟仿真

为了仿真验证所实现的 CPU，需要定义测试文件并在测试文件中对指令存储器和寄存器堆中的相应寄存器的值进行初始化，并通过仿真波形图查看是否指令得到了正确执行。

1）TestBench 关键代码

2、ModelSim 仿真及分析

五、总结

通过实验，请思考你认为完成一个 CPU 的设计与实现主要由哪几个步骤完成？主要注意事项有哪些？

六、进一步实验

1）在本节实现的单指令 CPU 基础上，添加存数指令 st r1,addr，实现一个可以支持加

法和存数指令的 CPU，并使用 ModelSim 进行仿真验证。

指令 st r1, addr: **[r1] -> mem[addr]**的格式如下：

15 9	8 6	5 0
st	r1	addr

实验分析提示

首先根据新增加的访存指令功能设计其数据通路，如图 6.1。为使 CPU 能够支持前面的加法指令和该访存指令，只需要将两个数据通路进行合并（即图 3.1 和图 6.1），并最终得到该 CPU 的原理图，如图 6.2 所示。

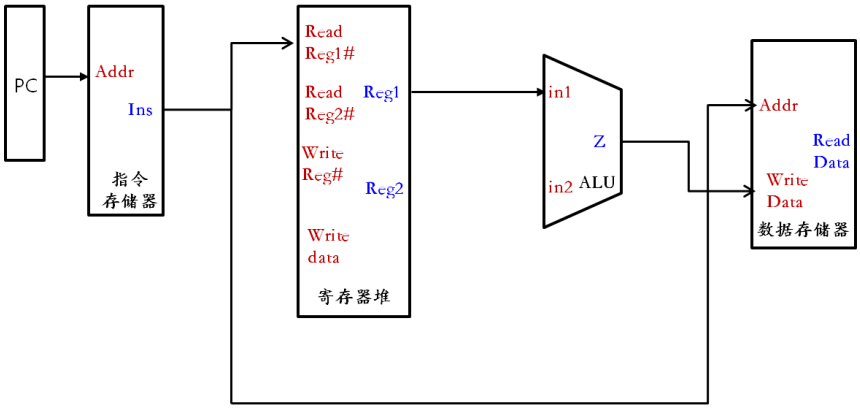


图 6.1 访存指令数据通路

如果要设计支持更多条指令的 CPU，只需根据每条指令设计其数据通路，并采用合并的方式构建支持所有指令的 CPU 数据通路，然后在进行控制单元的设计即可。在合并过程中需要注意的是：当某个部件的输入端口有多个输入来源的时候需在此端口前添加一个多路选择器从而允许控制部件根据执行的需要选择所需的数据来源。

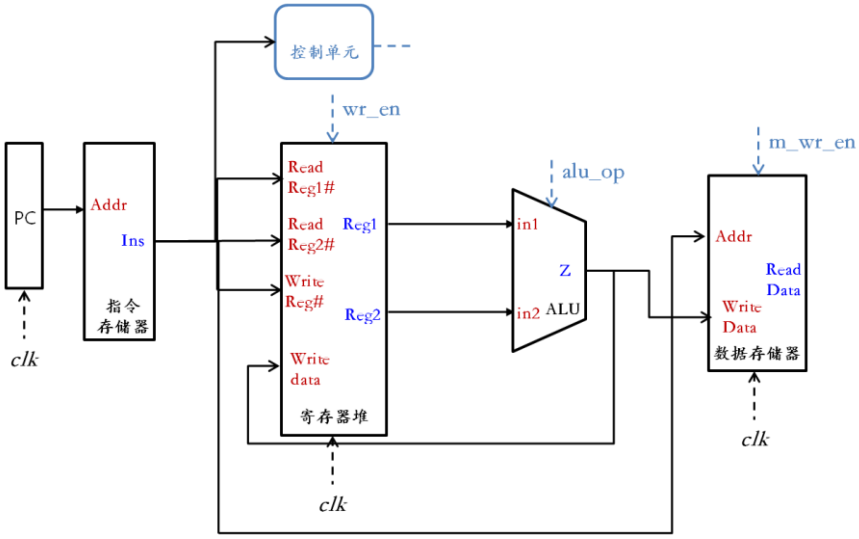


图 6.2 支持加法和访存指令的 CPU 原理图

2）设计和实现一个支持加法 add、立即数加 addi、存数 st、取数 ld 四条指令的 CPU（机器字长=指令字长=存储字长），要求指令格式如下：

- 加法 add: **[r1] + [r2] -> r1**

15 12	11 9	8 6	5 0
add	r1	r2	

- 立即数加 addi r1,Imm: **[r1]+SignExt[Imm] -> r1**

15 12	11 9	8 0
addi	r1	imm

- 存数 st r1, addr: **[r1] -> mem[addr]**

15 12	11 9	8 0

st	r1	addr

- 取数 ld r1, addr: **mem[addr] -> r1**

15 12	11 9	8 0
ld	r1	addr

实验五、单周期 CPU 设计与实现——十条指令 CPU

一、实验目的：

通过设计并实现支持 10 条指令的 CPU，进一步理解和掌握 CPU 设计的基本原理和过程。

二、实验内容：

设计和实现一个支持如下十条指令的单周期 CPU。

- **非访存指令**

◆ 清除累加器指令 CLA

◆ 累加器取反指令 COM

◆ 算术右移一位指令 SHR：将累加器 ACC 中的数右移一位，结果放回 ACC

◆ 循环左移一位指令 CSL：对累加器中的数据进行操作

◆ 停机指令 STP

- **访存指令**

◆ 加法指令 ADD　X：[X] + [ACC] –〉ACC，X 为存储器地址，直接寻址

◆ 存数指令 STA　X，采用直接寻址方式

◆ 取数指令 LDA　X，采用直接寻址

- **转移类指令**

◆ 无条件转移指令 JMP　imm：signExt(imm) -> PC

◆ 有条件转移（负则转）指令 BAN　X:　ACC 最高位为 1 则（PC）+ X -> PC,否则 PC 不变

三、实验原理

参照上次实验，依次给出指令格式定义、数据通路和控制单元的设计，并给出目标 CPU

的原理图。

四、实验步骤

参照上次实验，对实验过程进行描述和分析。

五、总结

实验六、多周期 CPU 设计与实现

一、实验目的：

通过设计并实现支持若干条指令的多周期 CPU ,理解和掌握多周期 CPU 设计的基本原理和过程。

二、实验内容

设计一个支持以下三条指令的包含取指、译码 (取操作数) 和执行三个工作周期的多周期 CPU。

- 加法 add r1,r2: **[r1]+ [r2] -> r1**

15 12	11 9	8 6	0
addi	r1	r2	XX

- 存数 st r1, addr: **[r1] -> mem[addr]**

15 12	11 9	8 0
st	r1	addr

- 无条件跳转 jmp imm: **SignExt[imm] -> PC**

15 12	11 0
addi	imm

三、实验原理

多周期 CPU 是将单条指令的执行划分为若干个工作周期，每个工作周期用一个时钟周期完成，每条指令的具体功能决定了完成该指令需要的工作周期数目。指令执行过程中在工作周期的转换由有限状态机进行实现 ,控制单元根据指令和其所处工作周期发出为操作命令

信号来控制指令的执行。

类似于单周期 CPU 的设计，多周期 CPU 的设计也需要完成 1）数据通路设计和 2）控制单元设计。

3.1 数据通路设计

首先对三条指令的功能依次进行分析：

- 加法 add
  - 取指：PC->Imem->IR；PC+1 -> PC；
  - 译码：RegFile[IR.r1]->A；RegFile[IR.r2]->B；
  - 执行：A、B->ALU-> RegFile[IR.r1]
- 存数 st
  - 取指：PC->Imem->IR；PC+1 -> PC；
  - 译码：无；
  - 执行：RegFile[IR.r1]->mem[IR.addr]
- 无条件跳转 jmp
  - 取指：PC->Imem->IR；PC+1 -> PC；
  - 译码：SignExt[IR.imm]->PC

根据三条指令执行分析，可以依次获得各条指令的数据通路，合并后可以得到如图 6.1 的支持该三条指令的多周期 CPU 数据通路图。特别的，相对于单周期 CPU，多周期 CPU 中添加了由红色标注的 IR 和操作数暂存部件 A、B。

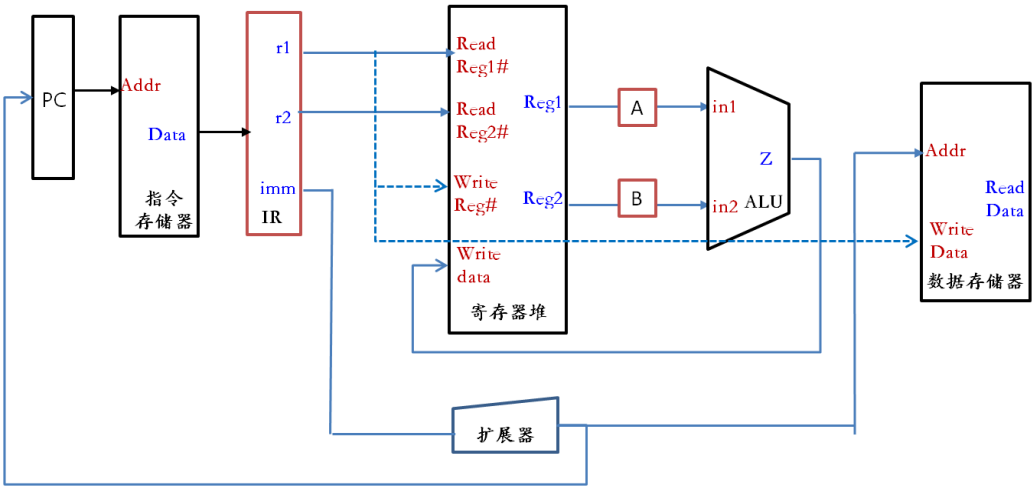


图 6.1 支持三条指令的多周期 CPU 数据通路图

3.2 控制单元设计

对于多周期 CPU，由于同一条指令在不同工作周期完成的操作不同，控制单元除了需要根据所执行的指令外，还需要根据其所出工作周期发出控制信号，从而确保指令的正确执行。 根据指令功能分析，可以明确控制单元的输入和输出控制信号如下表。

表 6.1. 控制单元的输入输出及说明

输入	Instruction	当前预执行指令
	Period	指令所处的工作周期
输出	PC_Plus_en	使 PC 加 1
	PC_Up_Imm_en	使 PC 根据立即数更新其值
	IR_wr_en	允许 IR 写入
	RegFile_wr_en	允许写入到 Reg. File 中的相应寄存器
	AluOpSel	ALU 执行操作选择信号（或算术逻辑操作选择信号）
	A_B_wr_en	暂存器 A、B 允许写入信号

通过对指令功能的分析，我们可以看到不同指令需要的工作周期数是不一样的，通常采用**有限状态机（FSM）**来对指令所处的工作周期进行实现。图 6.2 给出了实现本实验中三条指令在各个工作周期的转换的 FSM，其中 FI、ID、Exe 分别表示取指、译码和执行周期。

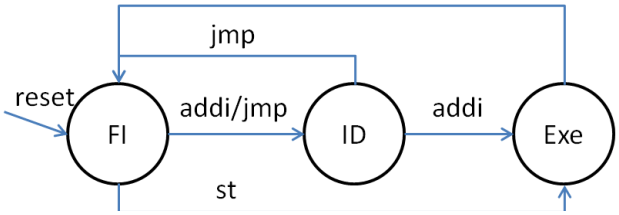


图 6.2. 实现指令执行工作周期（Period）状态转换的 FSM。

状态机由状态寄存器（记录当前所处状态的寄存器）、下一状态逻辑（根据当前状态和输入决定下一状态的组合逻辑电路）和输出逻辑（由当前状态*和输入*决定当前输出的组合逻辑电路）构成，能够根据控制信号按照预先设定的状态进行状态转移，常用于构建转移状态有限的系统。状态机通常分为两类：若输出只和状态有关而与输入无关，则称为 Moore 状态机；若输出不仅和状态有关而且和输入有关系，则称为 Mealy 状态机。图 6.3 给出了 Mealy 状态机的原理图，当把输入到产生输出的组合逻辑之间的连线去掉就是 Moore 状态机的原理图了。

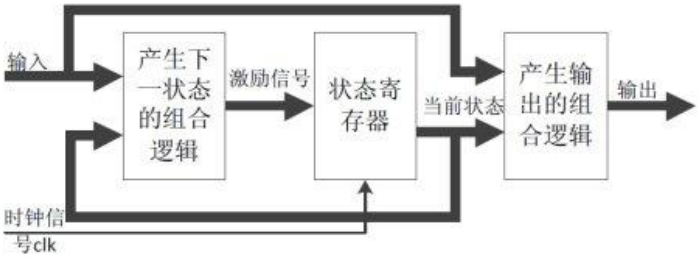


图 6.3 Mealy 状态机原理图

使用 Verilog 来描述状态机的方法与描述常规时许电路类似，其设计可以采用三段式设计，也就是根据原理图把状态机的设计分为以下三段：

- 当前状态：采用同步时序方式描述状态转移（如每一个时钟周期触发）；

- 下一状态逻辑：采用组合逻辑方式判断状态转移条件并产生下一状态；
- 输出逻辑：采用组合逻辑方式描述输出。

四、实验步骤

参照上次实验，对实验过程进行描述和分析。

五、总结

六、进一步实验

根据实验五设计的单周期 CPU，将其设计为包含取指、译码和执行三个工作周期的多周期 CPU。