中国农业大学
China Agricultural University

2021夏季
计算机组成与体系结构I课程设计

——Minisys-1 CPU设计

第一部分：课设要求

课程讨论群和课程资源

中国农业大学

计算机组成与体系结构课程设计

首页 | 基本信息 | 课程通知 | 课程资源 | 课程设计报告 | 单元学习 | 课程活动

《计算机组成与体系结构课程设计》 一、主要目标和主要内容： 本课程是计算机科学与技术专业的专业必修实践课（课程设计）。融会贯通计算机组成原理课程内容，加深对计算机系统各模块的工作原理及相互联系的认识，主要内容是进行CPU 的设计。旨在通过这些实验，在理论与实践相结合的基础上，加深学生的整机概念，进一步弄清计算机的内部结构和时空关系，学会指令功能的电路实现方法和技巧。学习运用EDA 技术进行设计和调试的基本步骤和方法，熟悉集成开发软件中设计调试工具链，体会软硬件协同开发技术的特点。认真执行这一教学过程，使学生在完成项目过程中，做到理论与实践相结合、硬件与软件相结合，培养学生综合运用所学的知识去解决处理实际问题的能力。 二、授课教师和授课对象： 计算机科学与技术 三、课程类型和学时学分： 必修 四、教学方式（授课形式和考核方式）： 实践。见大纲 五、教材与参考书目： 1、袁春风 主编 杨若瑜、王帅 、唐杰 编著。计算机组成与系统结构(第2版)，北京：清华大学出版社，2015 2、白中英 、 戴志涛。计算机组成原理（第五版·立体化教材）（含光盘），北京：科学出版社,2013

2021夏季学期-计算机组成课设群

该二维码7天内(7月14日前)有效，重新进入将更新

课设分组及安排：

【腾讯文档】2021夏计组课设分组表格

https://docs.qq.com/sheet/DWmhxV3R3YXdraHpM

进度

- 7月12-14日 完成分组、任务分工、制定设计方案，7月14日下午检查设计方案
- 7月15日-17日 完成各电路模块源码及仿真文件编写、并验证通过
- 7月18日-19日 使用顶层仿真文件组合各电路模块组成CPU，编写测试汇编代码，译为机器码，并在CPU上测试代码，下板验证
- 7月19日-20日 完善设计，准备验收及汇报PPT
- 7月21日 撰写报告、下午做PPT汇报
- 考试部分单独安排时间

课程设计团队

3位教师：黄岚、史银雪、赵景博

3名助教：刘静静、王楠、彭展佳

1位企业导师：依元素公司工程师

课程设计主要目标

目标1

加深对计算机数据通路的理解
加深对计算机指令系统、微指令的认识
通过时序控制，完整设计CPU

目标2

锻炼计算机系统分析和设计能力
锻炼分析、定位和排除故障的能力

核心任务

设计**基础组件**

以数字电路为基础

设计实现
基于MIPS32指令集
的MiniSys
功能型CPU

设计数据通路、功能单元

设计CPU

以组成原理为基础

以组件为原料

设计**接口组件（选做）**

以数字电路为基础

设计流程

原理分析 → **指令集分析** → **CPU架构分析** → **单元、顶层设计**

- 计算机的组成
- 计算机的基本组成部件有哪些?
- 计算机各组成部件间如何关联?
- CPU的结构和功能?
- CPU的工作原理?
- CPU的数据通路结构?

- MIPS32指令中最常用的31条指令
- 32个32位寄存器功能结构
- 32位定长格式指令
- 5种寻址方式

- 如何从1条指令推导出数据通路?
- 如何把多个数据通路组合成完整数据通路?
- 如何设计控制指令执行的控制系统?
- 如何让CPU与外界联系起来(I/O部件）？

- 顶层整合
- 组合成一个完整的CPU

设计流程

原理分析 → 指令集分析 → CPU架构分析 → 单元、顶层设计

资源：

- 计算机组成与体系结构I
- 其他网络资源

资源：

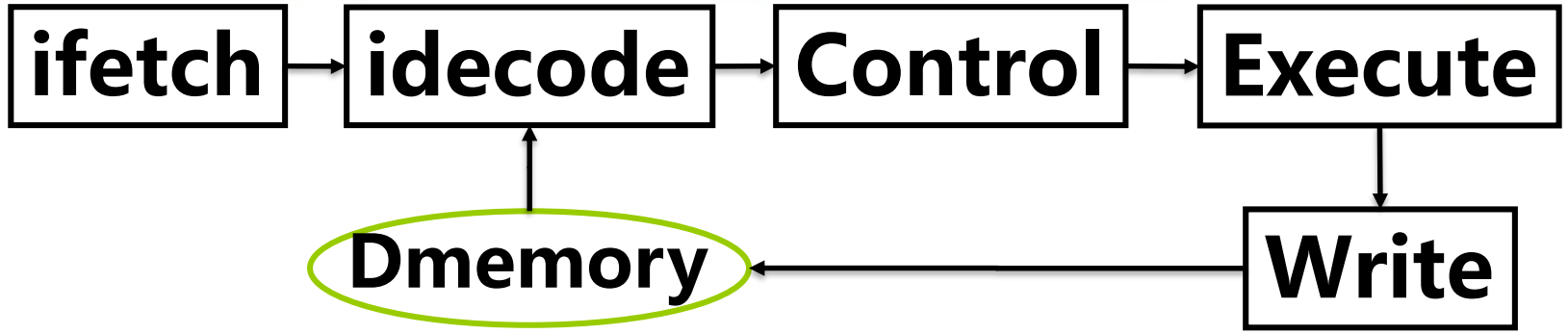
- 计算机组成课程设计手册
- 计算机组成与系统结构(第2版)
- 网络平台PPT
- 网上资源，CSDN等

资源

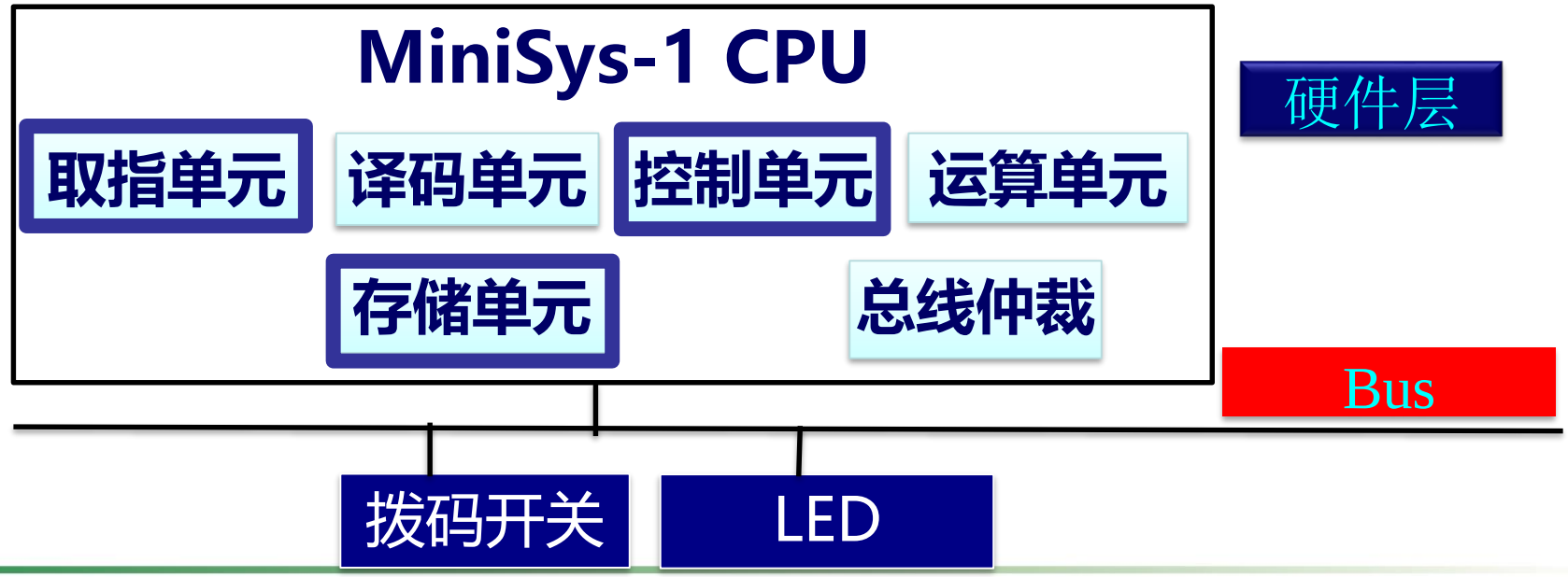
- minisys.v
- 网络平台参考ppt
- 其他网络资源

设计流程

CPU架构分析

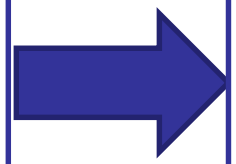


单元,顶层设计

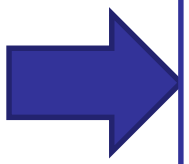


设计流程

数据通路设计
控制器设计
取指部件设计
译码部件设计
运算器设计
存储器设计
简单I/O



控制器模块(仿真)
取指模块(仿真)
译码模块(仿真)
执行模块(仿真)
存储器模块(仿真)
(RAM,ROM)
IO模块(仿真)



封装模块为系统：

Verilog的元件例化方法

Block Design

传统的电路系统设计方法

从状态图简化，写出最简逻辑表达式

↓

采用通用逻辑元器件(通常采用74系列和CMOS4000系列的产品)进行设计

↓

在系统硬件设计的后期进行调试和仿真

↓

当设计调试完毕后，形成电原理图(包括元器件型号和信号之间的互连关系等)

只有在部分或全部硬件电路连接完毕，才可以进行电路调试，一旦考虑不周到，系统设计存在较大缺陷，则要重新设计，使设计周期延长。

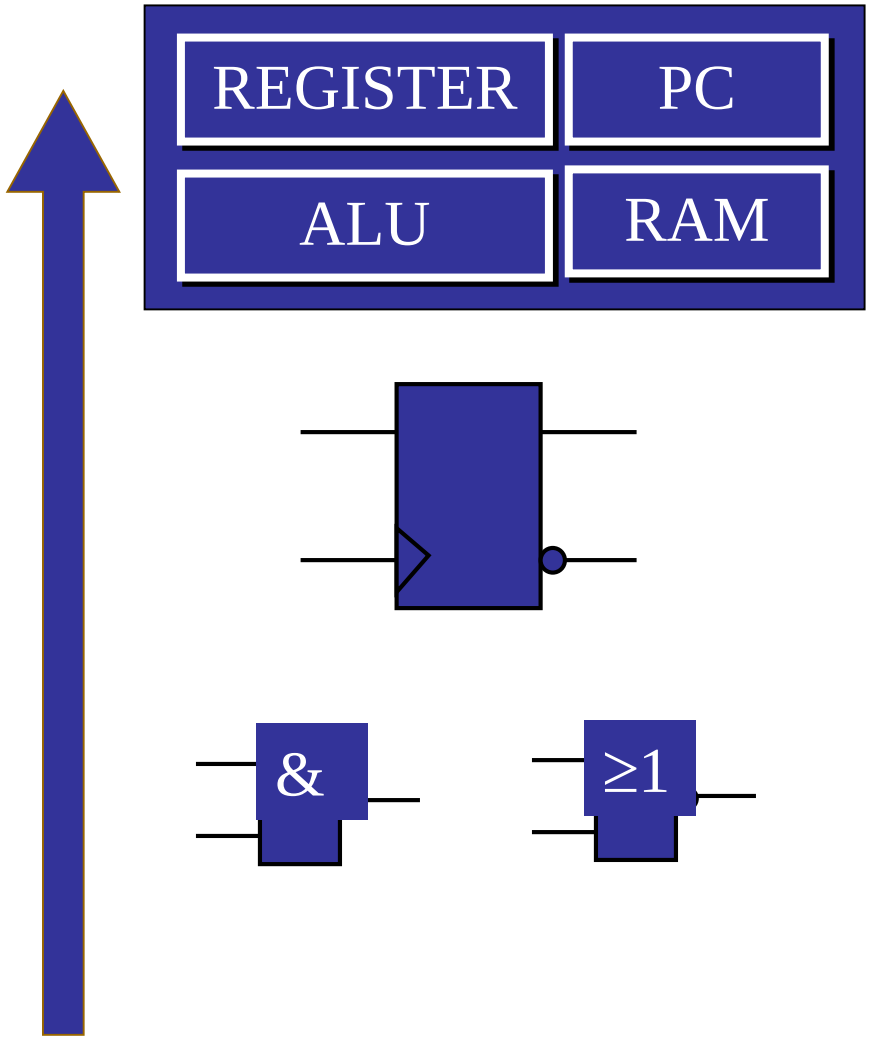
Bottom Up的设计方法

4. 完成整个系统测试与性能分析

3. 由各个功能模块连成一个完整系统

2. 由逻辑单元组成各个独立的功能模块

1. 由基本门构成各个组合与时序逻辑



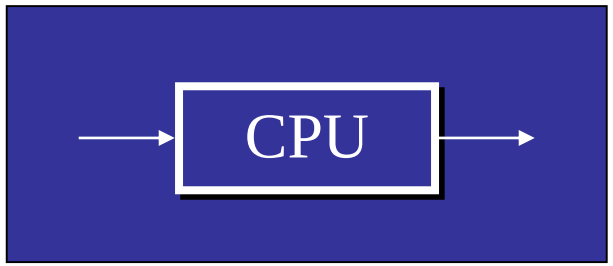
Top Down的设计方法

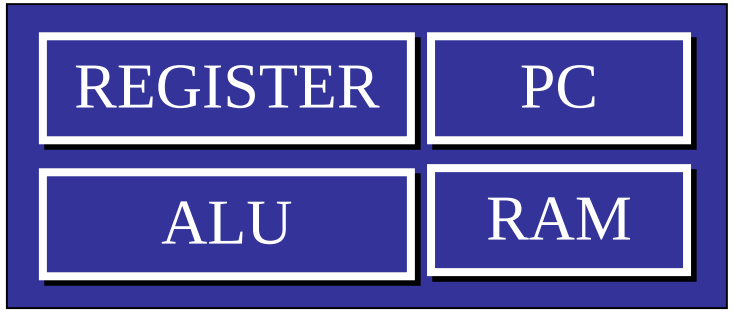
1. 系统层：顶层模块，行为级描述，功能模拟和性能评估

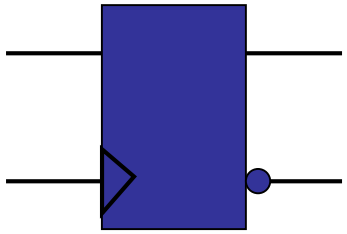
2. 各个功能模块划分，设计和验证

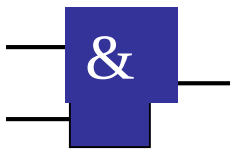
3. 各个功能模块系统级联合验证

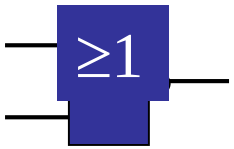
4. 工艺库映射







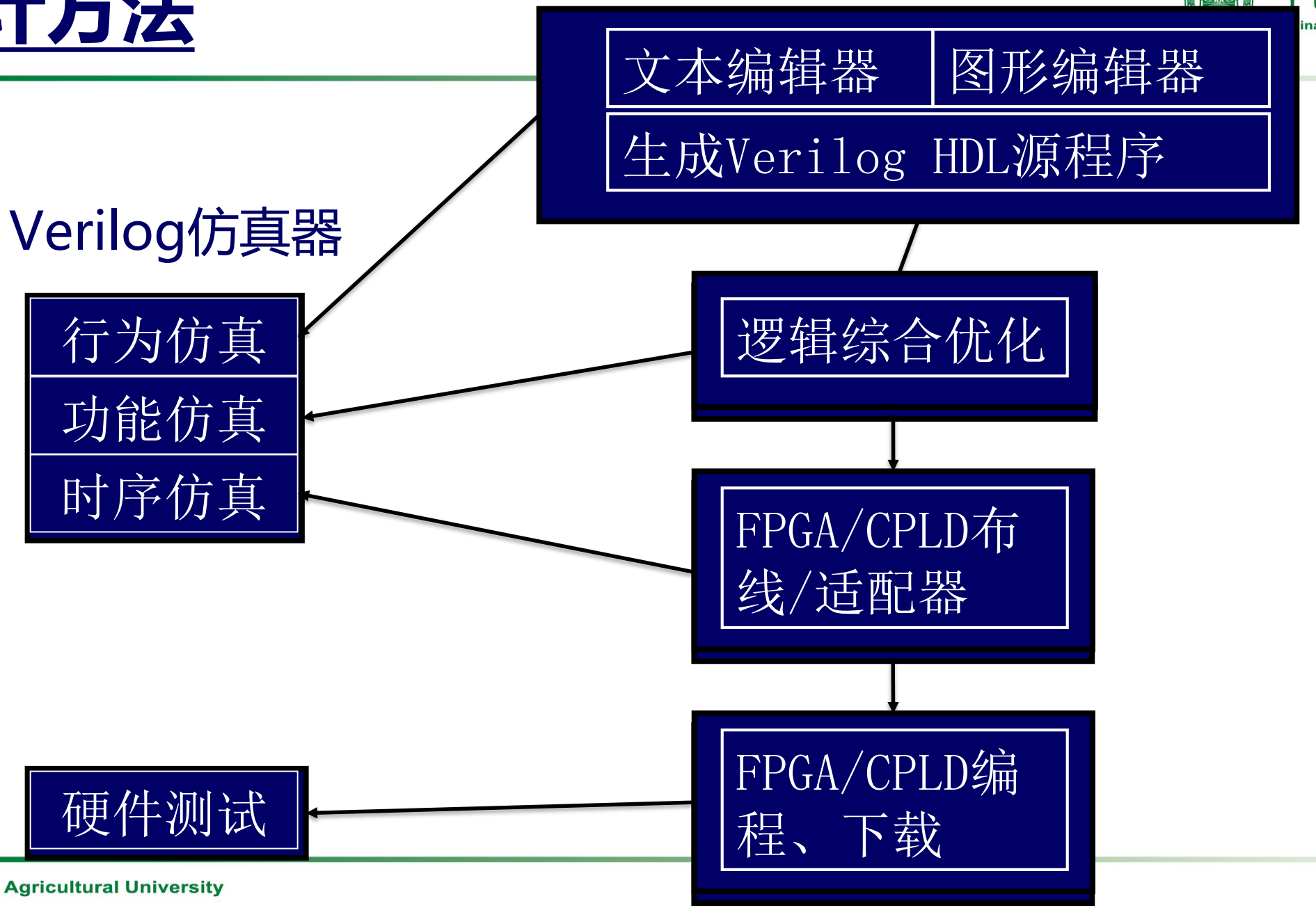




现代芯片设计方法

- **硬件描述语言**
  - **HDL（Hardware Description Language）**
    - Verilog HDL
  - **专门描述硬件工作原理的语言**
- **与程序设计语言（C）的主要区别**
  - 语言内置的并行性/并发性
  - 不仅描述逻辑，而且**描述时序**
    - 软件：1 + 1的计算结果等于2
    - 硬件：1 + 1的计算结果等于2 & **什么时候完成这个计算**

设计方法


文本编辑器
图形编辑器
生成Verilog HDL源程序
Verilog仿真器
行为仿真
功能仿真
时序仿真
逻辑综合优化
FPGA/CPLD布线/适配器
FPGA/CPLD编程、下载
硬件测试

设计工具

工具软件Vivado 2015.4

Minisys实验板 xc7a100tfgg484-1

设计验收

源代码：

- 完成31条MIPS指令的Minisys-1单周期CPU设计
- 包含取指、译码、控制、运算、存储等模块
- 指令存储器和数据存储器的哈佛结构存储
- 具有简单的LED和拨码开关功能
- 整机功能仿真结果

课程设计报告

- 请按照课程设计报告格式
- **每位同学均需写课程设计报告**

答辩PPT

- 按照excel分组答辩，要说明分组设计的特色点。

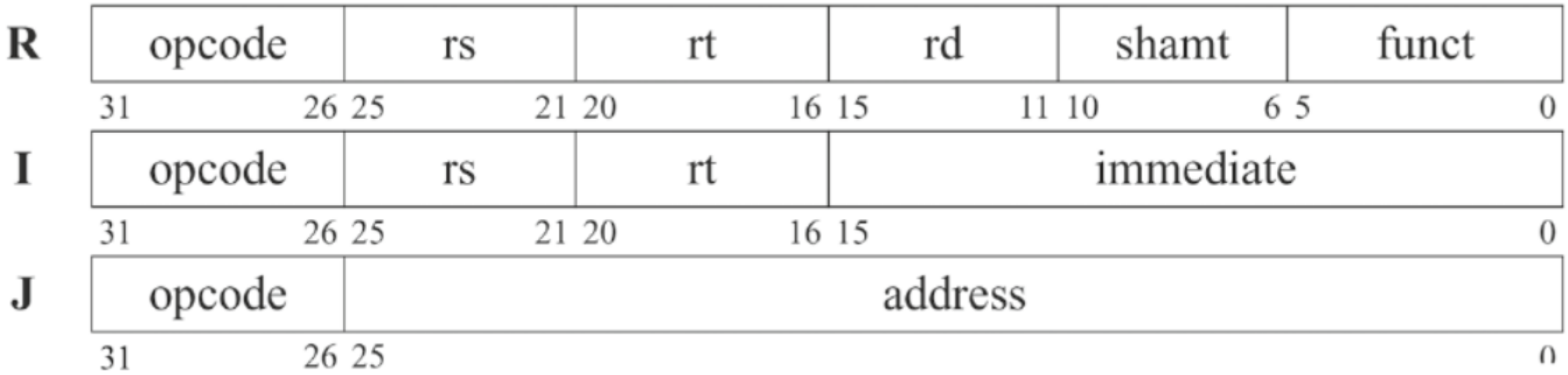
第二部分：MIPS指令集、处理器回顾

MIPS寄存器组

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

MIPS指令格式

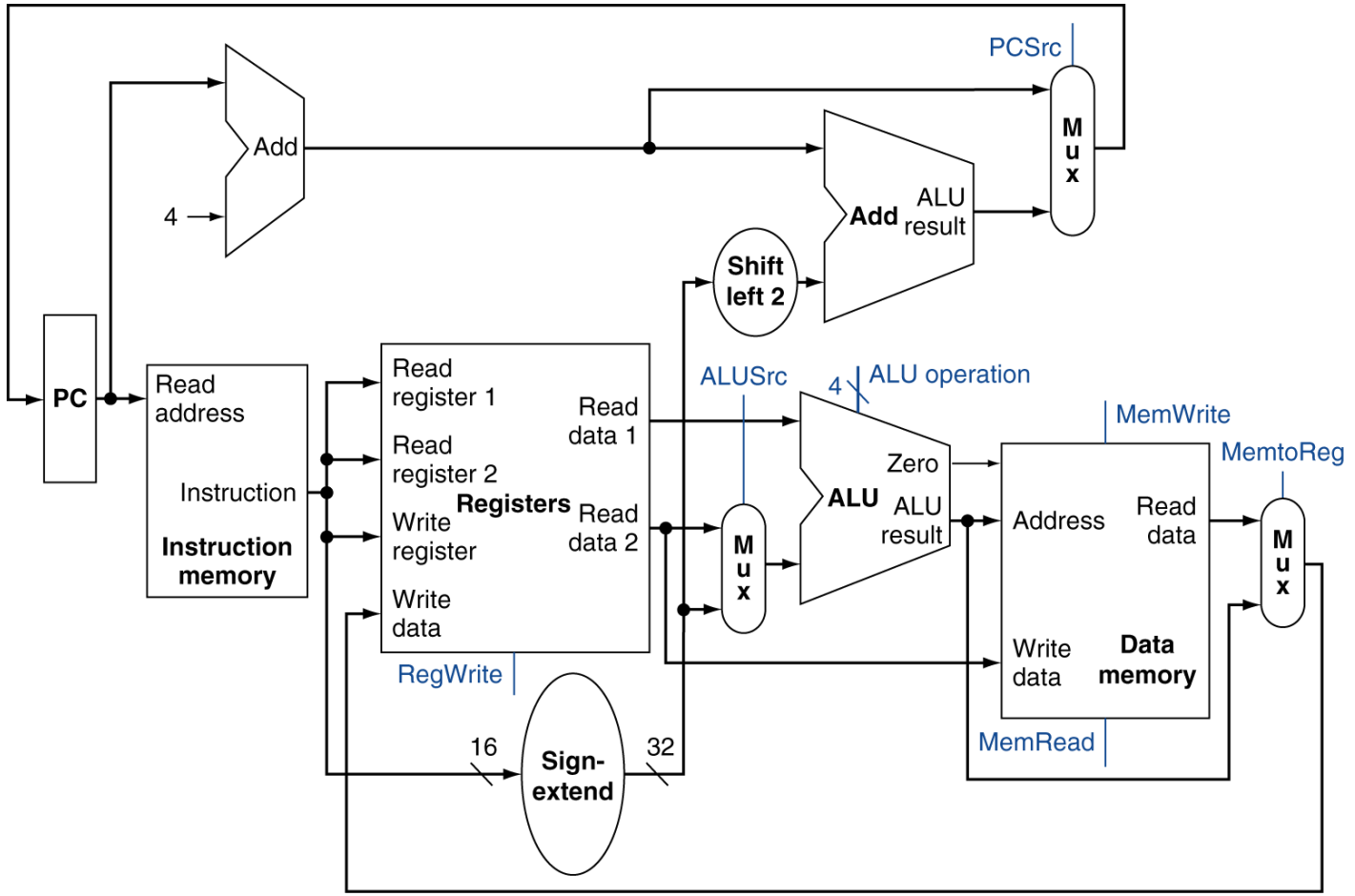

R
opcode
rs
rt
rd
shamt
funct
31
26
25
21
20
16
15
11
10
6
5
0
I
opcode
rs
rt
immediate
31
26
25
21
20
16
15
0
J
opcode
address
31
26
25
0

基本MIPS指令

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)
Add	add	R	R[rd] = R[rs] + R[rt]
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]
And	and	R	R[rd] = R[rs] & R[rt]
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr
Jump	j	J	PC=JumpAddr
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr
Jump Register	jr	R	PC=R[rs]
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])
Or	or	R	R[rd] = R[rs] | R[rt]
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 :
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt
Shift Right Logical	srl	R	R[rd] = R[rt] >>> shamt
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0) (2) 29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt] (2) $2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt] (1) $0 / 22_{hex}$
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt] $0 / 23_{hex}$

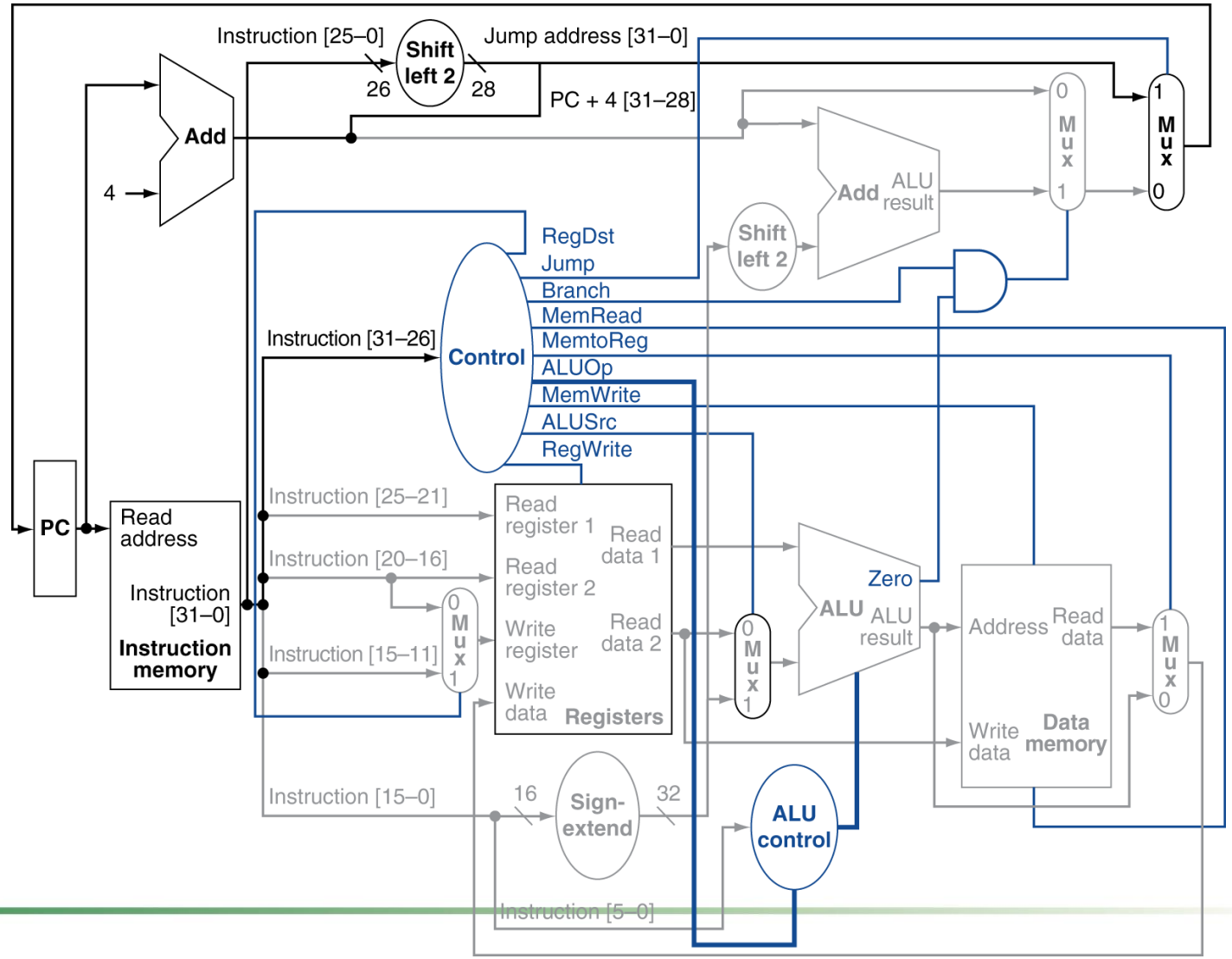
32位MIPS单周期CPU回顾

- 完整数据通路

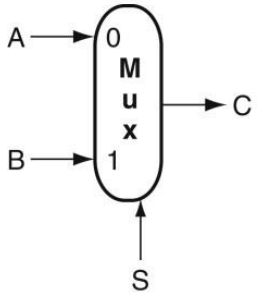


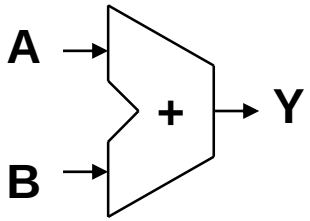
32位MIPS单周期CPU回顾

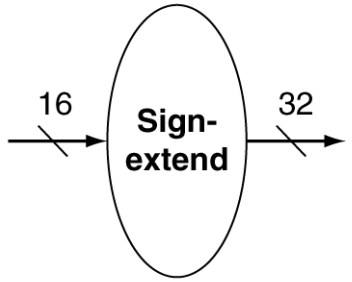
- 带控制器数据通路
- 只实现了部分指令
- 需要扩展

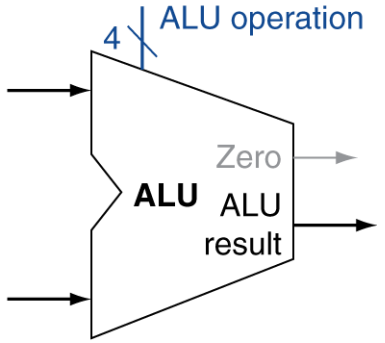


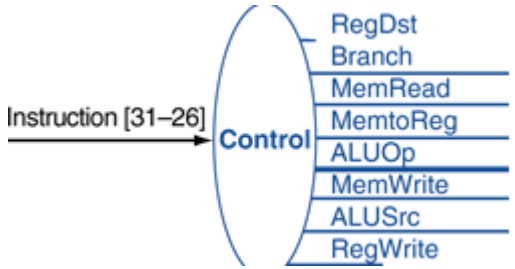
组合逻辑元件

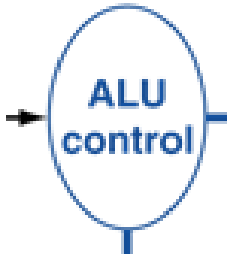

A
B
0
M
u
x
1
C
S


A
B
+
Y

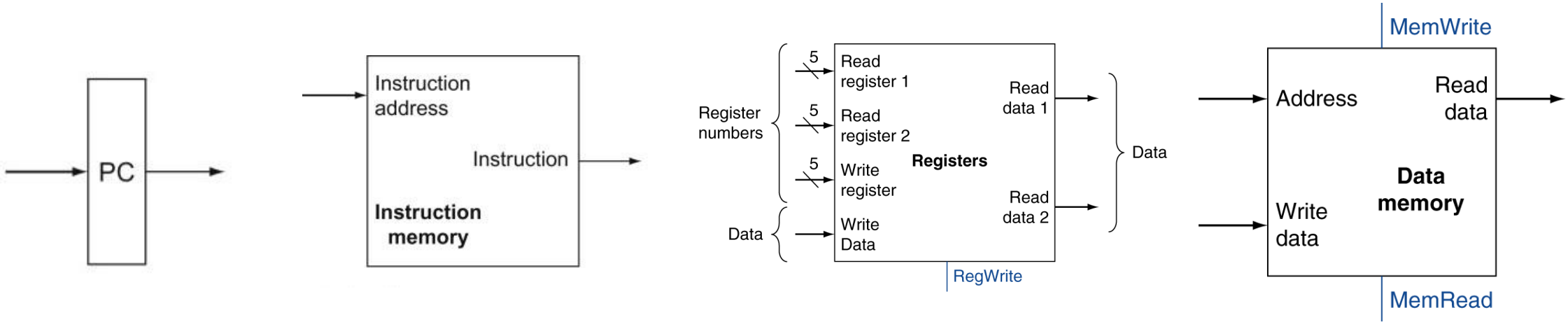

16
Sign-
extend
32


4
ALU operation
ALU
Zero
ALU
result


Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite


ALU
control

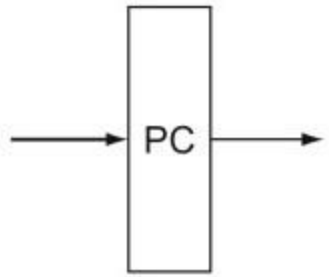
时序逻辑元件


PC
Instruction address
Instruction
Instruction memory
Register numbers
5
5
5
Read register 1
Read register 2
Write register
Registers
Write Data
Read data 1
Read data 2
Data
Data
RegWrite
MemWrite
Address
Read data
Data memory
Write data
MemRead

设计步骤及验证步骤

- 分别实现各个组合逻辑元件及时序逻辑元件
  - 注意时序元件输入为时钟信号(CLK)及重置信号(Reset)
  - 注意有两个控制器要实现：Control及ALUControl
- 对每个电路元件使用仿真文件(Test Bench)进行测试(输入覆盖所有可能的组合)
- 利用一个顶层文件(Top Module) 对电路器件进行组合
- 编写测试用MIPS汇编代码
- 利用Minisys汇编辅助翻译MIPS汇编代码至机器码，将机器码置入指令内存、数据置入数据内存(readmemh()或readmemb()函数)或使用BRAM IP核设计数据内存、指令内存并置入数据
- 利用一个顶层仿真文件(Testbench)给CPU提供输入
- 使用Vivado的仿真器运行CPU并观察寄存器中数值的变化是否符合设计要求
- 下板验证，观察LED灯输出

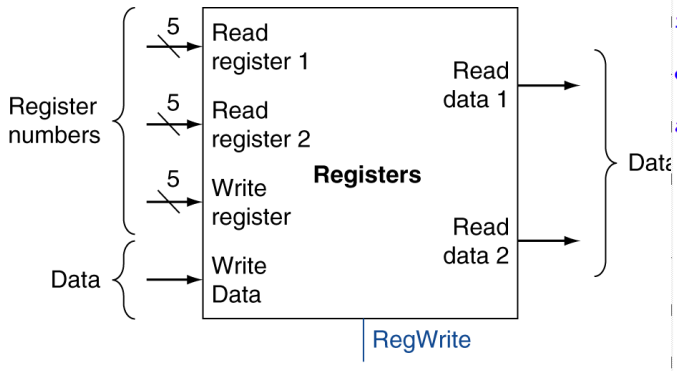
实现举例



Program Counter 程序计数器（未考虑跳转）

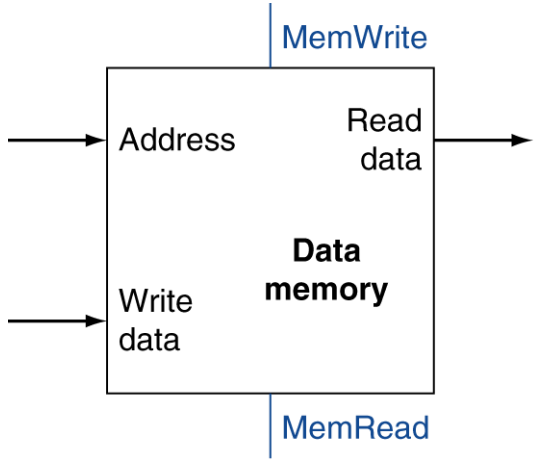
```
module program_counter(rst, clk, cout);
input rst, clk;
output reg [31:0] cout;
reg [31:0] counter;

    initial begin
        counter <= 0;
        cout <= 0;
    end

    always @ (rst or posedge clk) begin
        if (~rst)
            begin
            counter <= 0;
            cout <= 0;
            end
        else
            begin
            cout <= counter;
            counter <= counter + 1;
            end
    end

endmodule
```

实现举例



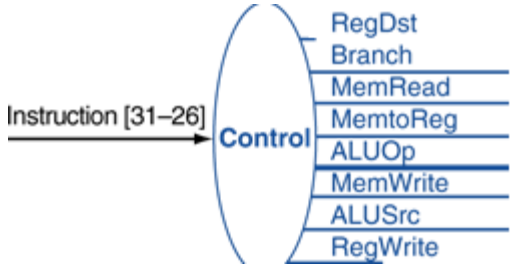

```
module register_file(rst, clk, RegWrite, read_register_1, read_register_2, write_register_1, read_data_1, read_data_2, write_data);
input rst, clk, RegWrite;
input [31:0] read_register_1, read_register_2, write_register_1, write_data;
output reg [31:0] read_data_1, read_data_2;
reg [31:0] registers [31:0];

initial begin
    registers[0] <= 0;
end

always @ (rst or posedge clk) begin
    if (~rst)
        begin
        read_data_1 <= 0;
        read_data_2 <= 0;
        end
    else
        begin
        if (RegWrite == 1)
            begin
                registers[write_register_1] <= write_data;
            end
        end
        read_data_1 <= registers[read_register_1];
        read_data_2 <= registers[read_register_2];
end

endmodule
```

实现举例




```
module data_memory(rst, clk, mem_write, mem_read, address, read_data, write_data);
input rst,clk;
input mem_write,mem_read;
input [31:0] address;
input [31:0] write_data;
output reg [31:0] read_data;
reg [31:0] memory [99:0];

initial begin
    memory[0] = 1;
    memory[1] = 2;
    memory[2] = 3;
    memory[3] = 4;
end

always @ (rst, posedge clk) begin
    if (~rst)
        begin
        read_data <= 0;
        end
    else
        if (mem_read == 1)
        begin
        read_data <= memory[address];
        end

        if (mem_write == 1)
        begin
        memory[address] <= write_data;
        end
end

endmodule
```

实现举例




```
module control(Op, RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, ALUOp1, ALUOp0);

input [5:0] Op;
output reg RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, ALUOp1, ALUOp0;

always @ (Op) begin
    if (Op == 6'b000000)
        begin
        RegDst <= 1;
        ALUSrc <= 0;
        MemtoReg <= 0;
        RegWrite <= 1;
        MemRead <= 0;
        MemWrite <= 0;
        Branch <= 0;
        ALUOp1 <= 1;
        ALUOp0 <= 0;
        end
    else if (Op == 6'b100011)
        begin
        RegDst <= 0;
        ALUSrc <= 1;
        MemtoReg <= 1;
        RegWrite <= 1;
        MemRead <= 1;
        MemWrite <= 0;
```

实现I/O扩展

- 编写按键、拨码开关模块
- 编写LED灯模块
- 扩展CPU的数据、地址和控制通路
- 能够使用按键、拨码开关控制CPU的启动、复位
- 能够使用LED灯显示CPU运行时的运算结果

选做（无考核要求）

- VGA控制器，使用VGA显示器显示运算结果

- 浮点数运算指令支持

中國農業大學
China Agricultural University

Fighting!