形状, 箭头

描述已自动生成

**32位MIPS多周期CPU设计方案**

（本科生）

**2022年春季学期计算机组成原理课程**

**负责人：韩旭达 联系方式：**

**刘文博 林骋楷 李清扬**

**‬班级：2020级数据科学四班**

**2022年5月20日星期五**

1. **项目简介**
2. **项目环境**
   1. **设计语言：**

Verilog硬件设计语言

* 1. **仿真环境：**

Vivado 2018.3

1. **任务目标**
   1. **课程设计要求：**

设计一个兼顾32位MIPS或RISC-V指令集的多周期CPU（核）

* 1. **设计目标：**

1. 实现无流水多周期CPU；
2. 兼容32位MIPS指令集（部分）；
3. 实现指令覆盖R型、I型计算类、I型取数类、I型存数类、I型条件判断类、J型；
4. 前仿成功；
5. **参考资料**

[数字逻辑与组成原理实践教程.清华大学出版社[引用日期2022-05-20]](http://www.tup.com.cn/bookscenter/book_07600401.html)

1. **项目文件夹结构**

* MultiCycleCPU
  + Code
  + Project
  + Materials
  + Result
  + 设计方案.docx
  + 设计方案.pdf
  + 设计方案.pptx
  + 讲解视频.mov
  + Readme.txt

1. **实现细节**
2. **指令集选取**
   1. **指令集：**

MIPS32

* 1. **指令类型：**

R型、I型、J型

* 1. **指令格式：**

表格

描述已自动生成

* 1. **指令操作：**

1. R型：该类型指令从寄存器堆中读取两个源操作数，计算结果写回寄存器堆。具体操作由、结合指定，和是源寄存器的编号，是目的寄存器的编号。
2. I类型，该类型指令使用一个16位的立即数作为一个源操作数。具体操作由指定，指令的低16位是立即数，运算时要将其扩展至32位，然后作为其中一个源操作数参与运算。
3. J类型，该类型指令使用一个26位的立即数作为跳转的目标地址。具体操作由指定，一般是跳转指令，低26位是字地址，用于产生跳转的目标地址。
   1. **完整MIPS指令集如下：**

表格

描述已自动生成

表格

描述已自动生成

* 1. **实现指令：**

本设计我们选取MIPS32指令集中的七条指令进行实现，分别覆盖R型、I型、J型：

1. ADD：R型，求和指令；
2. SUB：R型，求差指令；
3. OR：R型，按位或指令；
4. LW：I型，取数指令；
5. SW：I型，存数指令；
6. BEQ：I型，条件判断跳转指令；
7. J：J型，跳转指令；
8. **设计原理**

多周期CPU的中心思想是把一条指令的执行过程分成若干个小周期完成，根据每条指令的复杂程度，使用不同数量的小周期去执行，许多个小周期加在一起相当于单周期CPU中的一个周期。

在我们实现的7条指令中，最复杂的指令之一是，它需要5个周期，其整个执行的过程如下：

1. 根据取指令；
2. 完成加4；
3. 读出和两个寄存器的数据并锁存，比较是否相等；
4. 相等，ALU计算转移地址并锁存，否则指令结束；
5. 将转移地址写入，指令结束；

而最简单的指令是，具有以下三个周期：

1. 根据取指令；
2. 完成加4；
3. 指令中的左移两位与的高4位拼接成32位地址写入；

ALU计算类型的指令需要以下4个周期：

1. 根据取指令；
2. 完成加4；
3. 读出和两个寄存器的内容进行运算；
4. 把运算结果写入寄存器堆中的（或）寄存器；

访问内存类型的指令，需要以下4个周期：

1. 根据取指令；
2. 完成加4；
3. 寄存器的内容与指令中的偏移量相加，计算得到的存储器地址；
4. 使用计算好的地址访问存储器，从中读写出一个32位的数据；

由上述给出的特定指令执行过程可以看出，多周期CPU在处理指令时，一般需要经过（取指）、（译码）、（执行）、（访存）、（写回）五个阶段。执行一条指令最长需要五个时钟周期。下图为多周期CPU状态转移图，状态转移图不是唯一的，只要能实现各条指令所经过的周期即可：

一些文字和图案

低可信度描述已自动生成

在多周期CPU核中确定状态转移图及输出逻辑的部件是控制单元，我们用有限状态机实现多周期CPU的控制部件，可以用时序电路来实现多周期CPU的控制单元。多周期控制器逻辑结构图如下：

图示

描述已自动生成

1. **模块设计**
   1. **控制器设计：**

在控制器中设计一个状态寄存器（D触发器）用于保存当前状态，属于时序逻辑电路，当RST复位信号来临时初始化状态为‘000’（阶段）。再设计两个组合逻辑电路，一个（状态机）用于产生下一阶段的状态，另一个（硬布线控制器）用于产生每个阶段的控制信号。下一状态取决于指令操作码和当前状态，而每个阶段的控制信号取决于指令操作码、当前状态和反映运算结果的zero状态标志。

* **控制器结构**：

图表, 瀑布图

描述已自动生成

* **控制器描述：**

`timescale 1ns / 1ps

module ControlUnit(opcode, clk, reset, zero, PCWre, InsMemRW, IRWre, WrRegData, RegWre, ALUSrcA, ALUSrcB, DataMemRW, ALUM2Reg, ExtSel, RegOut, PCSrc, ALUOp);

    input [5:0]opcode;

    input zero, clk, reset;

    output PCWre, InsMemRW, IRWre, WrRegData, RegWre,ALUSrcA, ALUSrcB, DataMemRW, ALUM2Reg;

    output [1:0]ExtSel, RegOut, PCSrc;

    output [2:0]ALUOp;

    wire [2:0]i\_state, o\_state;

    DFlipFlop DFlipFlop(i\_state, reset, clk, o\_state);

    NextState NextState(o\_state, opcode, i\_state);

    OutputFunc OutputFunc(o\_state, opcode, zero, PCWre, InsMemRW, IRWre, WrRegData, RegWre, ALUSrcA, ALUSrcB, DataMemRW, ALUM2Reg, ExtSel, RegOut, PCSrc, ALUOp);

endmodule

* **状态寄存器（D触发器）描述：**

`timescale 1ns / 1ps

module DFlipFlop(i\_state, reset, clk, o\_state);

    input [2:0]i\_state;

    input reset, clk;

    output reg[2:0]o\_state;

    always @(posedge clk) begin

        if (reset) o\_state = 3'b000;

        else o\_state = i\_state;

    end

endmodule

* **状态机描述：**

`timescale 1ns / 1ps

module NextState(i\_state, opcode, next\_state);

    input [2:0]i\_state;

    input [5:0]opcode;

    output reg[2:0]next\_state;

    parameter [2:0] IF = 3'b000,

                    ID = 3'b001,

                    aEXE = 3'b110,

                    bEXE = 3'b101,

                    cEXE = 3'b010,

                    MEM = 3'b011,

                    aWB = 3'b111,

                    cWB = 3'b100;

    always @(i\_state or opcode) begin

        case (i\_state)

            IF: next\_state = ID;

            ID: begin

                case (opcode[5:3])

                    3'b110: begin

                        if (opcode == 6'b110100) next\_state = bEXE;

                        else next\_state = cEXE;

                    end

                    3'b111: next\_state = IF;

                    default: next\_state = aEXE;

                endcase

            end

            aEXE: next\_state = aWB;

            bEXE: next\_state = IF;

            cEXE: next\_state = MEM;

            MEM: begin

                if (opcode == 6'b110001) next\_state = cWB;

                else next\_state = IF;

            end

            aWB: next\_state = IF;

            cWB: next\_state = IF;

            default: next\_state = IF;

        endcase

    end

endmodule

* **硬布线控制器描述：**

`timescale 1ns / 1ps

module OutputFunc(state, opcode, zero, PCWre, InsMemRW, IRWre, WrRegData, RegWre, ALUSrcA, ALUSrcB, DataMemRW, DBDataSrc, ExtSel, RegDst, PCSrc, ALUOp);

    input [2:0]state;

    input [5:0]opcode;

    input zero;

    output reg PCWre, InsMemRW, IRWre, WrRegData, RegWre, ALUSrcA, ALUSrcB, DataMemRW, DBDataSrc;

    output reg[1:0]ExtSel, RegDst, PCSrc;

    output reg[2:0]ALUOp;

    parameter [2:0] IF = 3'b000,

                    ID = 3'b001,

                    aEXE = 3'b110,

                    bEXE = 3'b101,

                    cEXE = 3'b010,

                    MEM = 3'b011,

                    aWB = 3'b111,

                    cWB = 3'b100;

    parameter [5:0] add = 6'b000000,

                    sub = 6'b000001,

                    sw = 6'b110000,

                    lw = 6'b110001,

                    beq = 6'b110100,

                    j = 6'b111000,

                    Or = 6'b010000,

                    halt = 6'b111111;

    always @(state) begin

        if (state == IF && opcode != halt) PCWre = 1;

        else PCWre = 0;

        InsMemRW = 1;

        if (state == IF) IRWre = 1;

        else IRWre = 0;

        if (state == aWB || state == cWB) WrRegData = 1;

        else WrRegData = 0;

        if (state == aWB || state == cWB) RegWre = 1;

        else RegWre = 0;

    ALUSrcA = 0;

        if (opcode == sw || opcode == lw) ALUSrcB = 1;

        else ALUSrcB = 0;

        if (state == MEM && opcode == sw) DataMemRW = 1;

        else DataMemRW = 0;

        if (state == cWB) DBDataSrc = 1;

        else DBDataSrc = 0;

        ExtSel = 2'b10;

    if (opcode == lw) RegDst = 2'b01;

        else RegDst = 2'b10;

        case(opcode)

            j: PCSrc = 2'b11;

            beq: begin

                if (zero) PCSrc = 2'b01;

                else PCSrc = 2'b00;

            end

            default: PCSrc = 2'b00;

        endcase

        case(opcode)

            sub: ALUOp = 3'b001;

            Or: ALUOp = 3'b101;

            beq: ALUOp = 3'b001;

            default: ALUOp = 3'b000;

        endcase

        if (state == IF) begin

            RegWre = 0;

            DataMemRW = 0;

        end

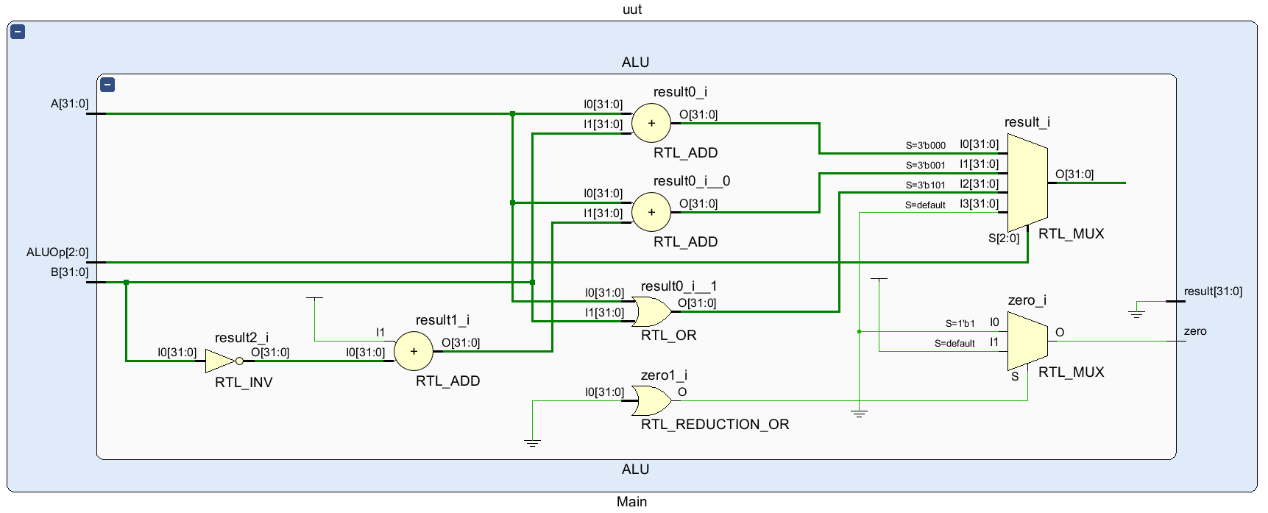
    end

endmodule

* 1. **ALU设计：**

ALU模块接收两个源操作数和控制码作为输入，输出计算结果和零标志位。初始化输出结果为0。在多周期CPU的（执行）阶段，每当时钟上升沿到来时，根据控制模块给出的控制信号进行相应计算。

* **ALU结构：**

****

* **ALU描述：**

`timescale 1ns / 1ps

module ALU(A, B, ALUOp, zero, result);

    input [31:0] A, B;

    input [2:0] ALUOp;

    reg [31:0] reg\_A,reg\_B;

    output zero;

    output reg [31:0] result;

    initial begin

        result = 0;

    end

    assign zero = (result? 0 : 1);

    always @(A or B or ALUOp) begin

        case(ALUOp)

            3'b000: begin reg\_A=A; reg\_B=B; result = reg\_A + reg\_B; end

            3'b001: begin reg\_A=A; reg\_B=(~B+1);result = reg\_A + reg\_B; end

            3'b101: result = A | B;

            default: result = 0;

        endcase

    end

endmodule

* 1. **选择器设计：**

多周期CPU核中涉及多种多路选择器：2选1、3选1、4选1，但其结构类似。多路选择器根据控制器给出的控制信号选择相对应的输入进行输出。

* **多路选择器结构：**

**图表

描述已自动生成**

* **多路选择器描述：**

`timescale 1ns / 1ps

module DataSelector\_4to1(A, B, C, D, Control, Result);

    input [31:0] A, B, C, D;

    input [1:0]Control;

    output reg[31:0] Result;

    always @(Control or A or B or C or D) begin

        case(Control)

            2'b00: Result = A;

            2'b01: Result = B;

            2'b10: Result = C;

            2'b11: Result = D;

            default: Result = 0;

        endcase

    end

endmodule

* 1. **存储器设计:**

多周期CPU核中涉及两种存储器：指令存储器、数据存储器，两者结构类似。其核心功能负责连续读取/写入4个8位二进制数。

* **指令存储器结构：**

**图示

描述已自动生成**

* **指令存储器描述：**

`timescale 1ns / 1ps

module InstructionMEM (addr, InsMemRW, instruction);

    input InsMemRW;

    input [31:0] addr;

    output reg [31:0] instruction;

    reg [7:0] mem [0:127];

    initial begin

        $readmemb("E:\\Courses\\Computer\_Organization\\MultiCycleCPU\\Code\\instructions.txt", mem);

        instruction = 0;

    end

    always @(addr or InsMemRW) begin

        if (InsMemRW) begin

            instruction[31:24] = mem[addr];

            instruction[23:16] = mem[addr+1];

            instruction[15:8] = mem[addr+2];

            instruction[7:0] = mem[addr+3];

        end

    end

endmodule

* **数据存储器结构：**

**图表

中度可信度描述已自动生成**

* **数据存储器描述：**

`timescale 1ns / 1ps

module DataMEM (i\_data, addr, DataMemRW, o\_data);

    input [31:0] i\_data;

    input [31:0] addr;

    input DataMemRW;

    output reg [31:0] o\_data;

    reg [7:0] memory [0:63];

    initial begin

        o\_data = 0;

    end

    always @(addr or i\_data or DataMemRW) begin

        if (DataMemRW) begin

            memory[addr] = i\_data[31:24];

            memory[addr+1] = i\_data[23:16];

            memory[addr+2] = i\_data[15:8];

            memory[addr+3] = i\_data[7:0];

        end

        else begin

          o\_data[31:24] = memory[addr];

          o\_data[23:16] = memory[addr+1];

          o\_data[15:8] = memory[addr+2];

          o\_data[7:0] = memory[addr+3];

        end

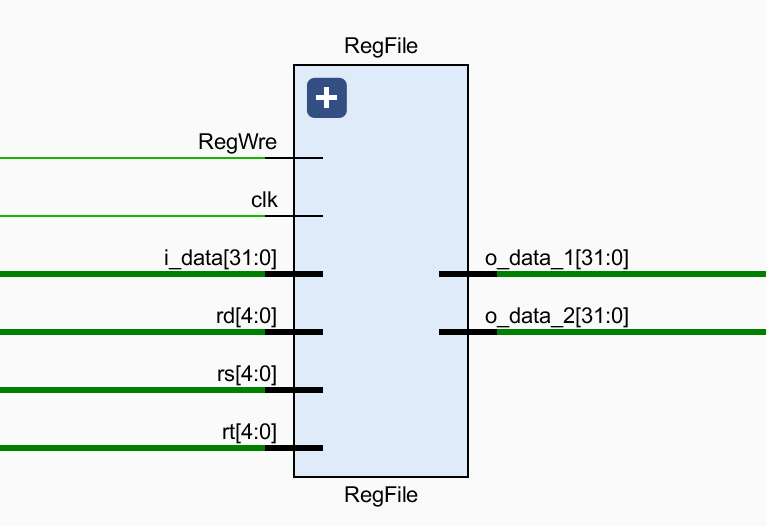
    end

endmodule

* 1. **寄存器设计：**

多周期CPU核具有一个由32个32位寄存器构成的寄存器组。初始化0号寄存器的值为0。

* **寄存器结构：**

****

* **寄存器描述：**

`timescale 1ns / 1ps

module RegFile (rs, rt, rd, i\_data, RegWre, clk, o\_data\_1, o\_data\_2);

    input [4:0] rs, rt, rd;

    input [31:0] i\_data;

    input RegWre, clk;

    output [31:0] o\_data\_1, o\_data\_2;

    reg [31:0] register [0:31];

    integer i;

    initial begin

        i = 1;

        register[0]=0;

        while (i<32) begin

            register[i]=1;

            i = i+1;

        end

    end

    assign o\_data\_1 = register[rs];

    assign o\_data\_2 = register[rt];

    always @(i\_data or rd) begin

        if ((rd != 0) && (RegWre == 1)) begin

            register[rd] = i\_data;

        end

    end

endmodule

* 1. **位扩展器设计：**

多周期CPU核具有一个位扩展器，可以根据控制器给出的控制信号进行零扩展、符号位扩展等操作。本CPU核所实现指令中，只有指令涉及位扩展器有关操作，且为立即数符号位扩展。

* **位扩展器结构：**

**图表

描述已自动生成**

* **位扩展器描述：**

`timescale 1ns / 1ps

module SignExtend(i\_num, ExtSel, o\_num);

    input [15:0] i\_num;

    input [1:0] ExtSel;

    output reg[31:0] o\_num;

    initial begin

        o\_num = 0;

    end

    always @(i\_num or ExtSel) begin

        case(ExtSel)

            2'b00: o\_num <= {{27'o000000000}, i\_num[10:6]};

            2'b01: o\_num <= {{16'h0000}, i\_num[15:0]};

            2'b10: o\_num <= {{16{i\_num[15]}}, i\_num[15:0]};

            default: o\_num <= {{16{i\_num[15]}}, i\_num[15:0]};

        endcase

    end

endmodule

* 1. **指令寄存器设计：**

多周期CPU核具有一个指令寄存器，用于临时存储指令供下一阶段使用。

* **指令寄存器结构：**

**图示

描述已自动生成**

* **指令寄存器描述：**

`timescale 1ns / 1ps

module IR(i\_data, clk, IRWre, o\_data);

    input clk, IRWre;

    input [31:0] i\_data;

    output reg[31:0] o\_data;

    always @(negedge clk) begin

        if (IRWre) begin

            o\_data = i\_data;

        end

    end

endmodule

* 1. **操作数寄存器设计：**

多周期CPU核具有两个操作数寄存器，用于临时存储源操作数供下一阶段使用。

* **操作数寄存器结构（ADR）：**

**图示

描述已自动生成**

* **操作数寄存器结构（BDR）：**

**图示

描述已自动生成**

* **操作数寄存器描述（ADR）：**

`timescale 1ns / 1ps

module ADR(i\_data, clk, o\_data);

    input clk;

    input [31:0] i\_data;

    output reg[31:0] o\_data;

    always @(posedge clk) begin

        o\_data = i\_data;

    end

endmodule

* **操作数寄存器描述（BDR）：**

`timescale 1ns / 1ps

module BDR(i\_data, clk, o\_data);

    input clk;

    input [31:0] i\_data;

    output reg[31:0] o\_data;

    always @(posedge clk) begin

        o\_data = i\_data;

    end

endmodule

* 1. **结果寄存器设计：**

多周期CPU核具有一个结果寄存器，用于临时存储ALU运算结果供下一阶段使用。

* **结果寄存器结构：**

**图示

描述已自动生成**

* **结果寄存器描述：**

`timescale 1ns / 1ps

module ALUoutDR(i\_data, clk, o\_data);

    input clk;

    input [31:0] i\_data;

    output reg[31:0] o\_data;

    always @(posedge clk) begin

        o\_data = i\_data;

    end

endmodule

* 1. **数据寄存器设计：**

多周期CPU核具有一个数据寄存器，用于临时存储ALU运算结果或数据存储器访存结果供下一阶段使用。

* **数据寄存器结构：**

**图示

描述已自动生成**

* **数据寄存器描述：**

`timescale 1ns / 1ps

module DBDR(i\_data, clk, o\_data);

    input clk;

    input [31:0] i\_data;

    output reg[31:0] o\_data;

    always @(posedge clk) begin

        o\_data = i\_data;

    end

endmodule

1. **数据通路**

多周期CPU核数据通路如下：

图示, 示意图

描述已自动生成

Vivado Schematic：

图示, 示意图

描述已自动生成

1. **仿真测试**
2. **测试指令：**

为了对CPU运行状态进行测试，我们编写测试指令如下，覆盖7条MIPS指令：

**j 2 //J型：跳转指令**

**111000 00000000000000000000000010**

**E000 0002**

**\\空指令：J指令跳转测试**

**00000000000000000000000000000000**

**sw $1 0($0) \\I型：存数指令**

**110000 00000 00001 0000000000000000**

**C001 0000**

**sw $2 4($0) \\I型：存数指令**

**110000 00000 00010 0000000000000100**

**C002 0004**

**add $1 $1 $2 \\R型：求和指令**

**000000 00001 00010 00001 00000 000000**

**0022 0800**

**add $1 $1 $2 \\R型：求和指令**

**000000 00001 00010 00001 00000 000000**

**0022 0800**

**sub $2 $1 $2 \\R型：求差指令**

**000001 00001 00010 00010 00000 000000**

**0422 1000**

**or $2 $1 $2 \\R型：按位或指令**

**010000 00001 00010 00010 00000 000000**

**4022 1000**

**lw $1 0($0) \\I型：取数指令**

**110001 00000 00001 0000000000000000**

**C401 0000**

**lw $2 4($0) \\I型：取数指令**

**110001 00000 00010 0000000000000100**

**C402 0004**

**beq $1 $2 1\\I型：条件判断指令**

**110100 00001 00010 0000000000000001**

**D022 0001**

**\\空指令：beq指令跳转测试**

**00000000000000000000000000000000**

**halt \\停机指令**

**111111 00000000000000000000000000**

**FC00 0000**

1. **仿真结果：**

从下面的仿真结果可以看出，指令正常读取执行，CPU核正常运行：

图形用户界面

描述已自动生成

1. **性能指标分析**

* 支持7条指令，多周期CPI=3.9；
* 时钟周期：100ns；
* 等效频率：10MHz；
* 多周期CPU中各部件的利用率依然偏低。

1. **总结**

通过本次课程设计任务，我们学习了多周期CPU的基本设计原理，对于CPU的构成有了更加深入的了解。

在设计数据通路的过程中，我们学习了各类型MIPS指令的执行过程，让我们对于精简指令集有了更加充分的认识，对精简指令集的优势有了更加直观的体会。

在编写测试指令的过程中，我们学习了有关MIPS指令集的基本汇编语言和各类型MIPS指令的算数逻辑功能、寄存器功能，掌握了有关MIPS指令集的基本知识。