

**数学计算机科学 学院实验报告**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验课程： | 计算机组成原理实验 | | | |
| 实验编号： | 综合设计 | | | |
| 实验名称： | MIPS微处理器设计与验证(MIPS32-C7指令集) | | | |
| 实验人员 | 151111206033 孙浩 | | | |
| 实验日期： |  | | | |
| 实验室： | 学苑南楼1幢305 | | | |
|  |  | | | |
| 实验评价： |  | | | |
| 实验成绩： |  | 评价日期： |  |
|  | 指导教师： |  | | |

# MIPS微处理器设计与验证(MIPS32-C7指令集)

# 一、实验目的

## 1. 理解MIPS微处理器工作原理。

## 2.掌握MIPS微处理器设计工程化设计方法。

# 二、实验工具

## 1.Xilinx Vivado 2014.2软件。

## 2.Windows系统PC机。

# 三、实验要求

## 1.设计一个MIPS微处理器，支持MIPS32-C7指令集。

MIPS32-C7微处理器指令字长为32位，通用寄存器为32位，存储单元地址为32位，支持MIPS32-C7指令集＝{addu，subu，ori，lw，sw，beq，jal}。所有运算类指令均可以不支持溢出。

### (1)指令：addu

指令名称：无符号加

指令格式：addu rd,rs,rt

指令功能：GPR[rd] ←GPR[rs] + GPR[rt]

指令编码：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31..26 | 25..21 | 20..16 | 15..11 | 10..6 | 5..0 |
| op | rs | rt | rd | shamt | func |
| 000000 | rs | rt | rd | 00000 | 100001 |

### (2)指令：subu

指令名称：无符号减

指令格式：subu rd,rs,rt

指令功能：GPR[rd] ←GPR[rs] -GPR[ rt]

指令编码：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31..26 | 25..21 | 20..16 | 15..11 | 10..6 | 5..0 |
| op | rs | rt | rd | shamt | func |
| 000000 | rs | rt | rd | 00000 | 100011 |

### (3)指令：ori

指令名称：或立即数

指令格式：ori rt,rs,imm16

指令功能：GPR[rt ]←GPR[ rs] | (zero-extend)imm16

指令编码：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31..26 | 25..21 | 20..16 | 15..11 | 10..6 | 5..0 |
| op | rs | rt | rd | shamt | func |
| 001101 | rs | rt | imm16 | | |

### (4)指令：lw

指令名称：加载字

格式：lw rt, imm16(rs)

指令功能：GPR[rt ]← memory[GPR[rs] + (sign-extend)imm16]

指令编码：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31..26 | 25..21 | 20..16 | 15..11 | 10..6 | 5..0 |
| op | rs | rt | rd | shamt | func |
| 100011 | rs | rt | imm16 | | |

### (5)指令：sw

指令名称：存储字

指令格式：sw rt, imm16(rs)

指令功能：memory[GPR[rs] + (sign-extend)imm16] ←GPR[rt]

指令编码：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31..26 | 25..21 | 20..16 | 15..11 | 10..6 | 5..0 |
| op | rs | rt | rd | shamt | func |
| 101011 | rs | rt | imm16 | | |

### (6)指令：beq

指令名称：等于转移

指令格式：beq rs,rt, imm16

指令功能： if (GPR[rs] ==GPR[ rt]) PC ← PC+4 + (sign-extend)imm16<<2

指令编码：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31..26 | 25..21 | 20..16 | 15..11 | 10..6 | 5..0 |
| op | rs | rt | rd | shamt | func |
| 000100 | rs | rt | imm16 | | |

### (7)指令：jal

指令名称：跳转并链接

指令格式：jal addr26

指令功能：$31<-PC+4；PC ←(PC+4)[31..28],addr26,0,0

指令编码：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31..26 | 25..21 | 20..16 | 15..11 | 10..6 | 5..0 |
| op | rs | rt | rd | shamt | func |
| 000011 | addr26 | | | | |

## 2.在设计好的MIPS微处理器上执行一段程序，给出程序执行结果。

MIPS32-C7处理器为多周期设计，如图1所示。

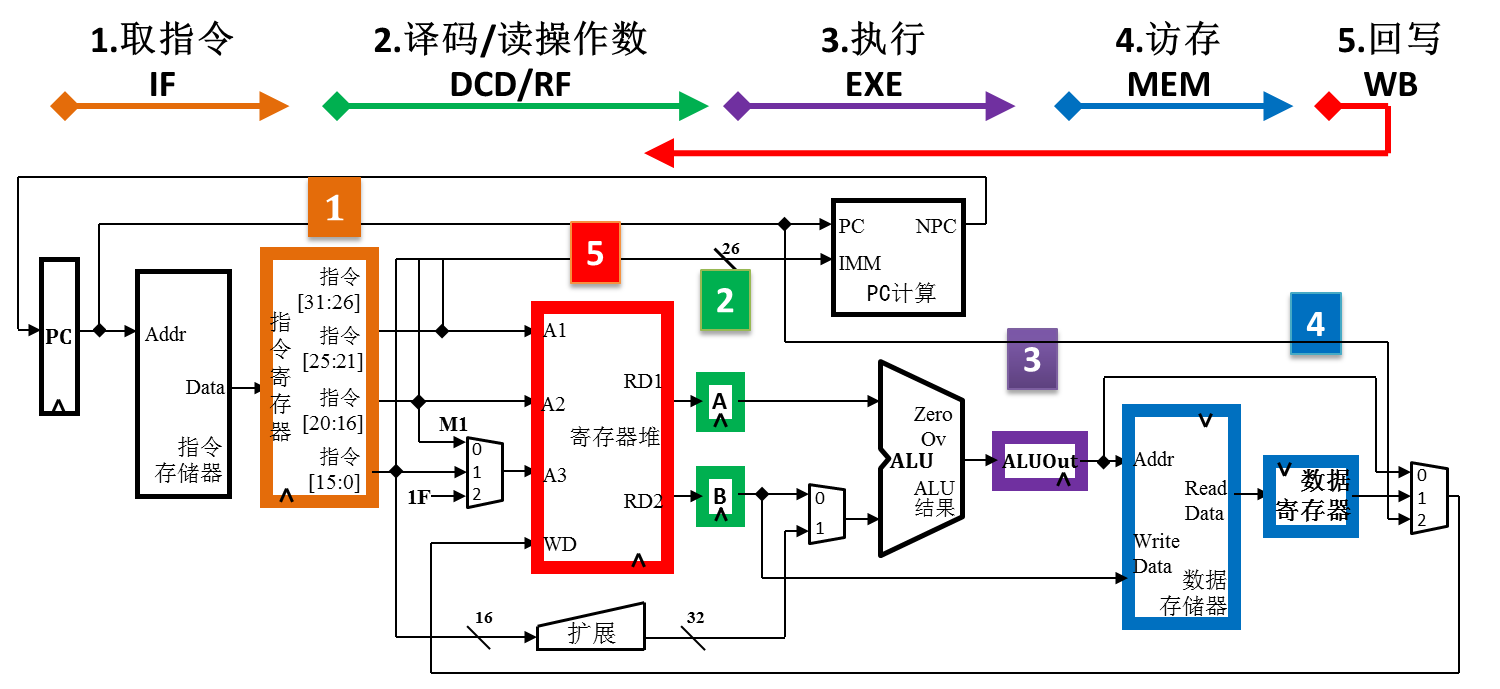


图1 五周期数据通路

表1 五周期设计

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **分段通路** | **段内**  **平行功能** | **起始**  **(读寄存器)** | **中间逻辑** | **结束**  **(写寄存器)** | **依赖** |
| 读取指令 |  | PC | IM | IR |  |
| 读操作数 | 读2个寄存器 | IR | ***RF*** | A/B | 读取指令 |
| 读1个寄存器，立即数扩展 |  |  | A(/***B***)，***EXT*** |
| 执行 | R-R运算 | A，B | ALU | ALUOut | 读操作数 |
| R-I运算 | A，EXT |
| 访存 | 读存储 | ALUOut | DM | DR | 执行  执行 |
| 写存储 | ALUOut，B |  | DM |
| 回写 | 存储回写 | DR |  | RF | 读存储@访存 |
| 计算回写 | ALUOut |  | RF | 执行 |

多周期处理器由datapath(数据通路)和controller(控制器)组成。

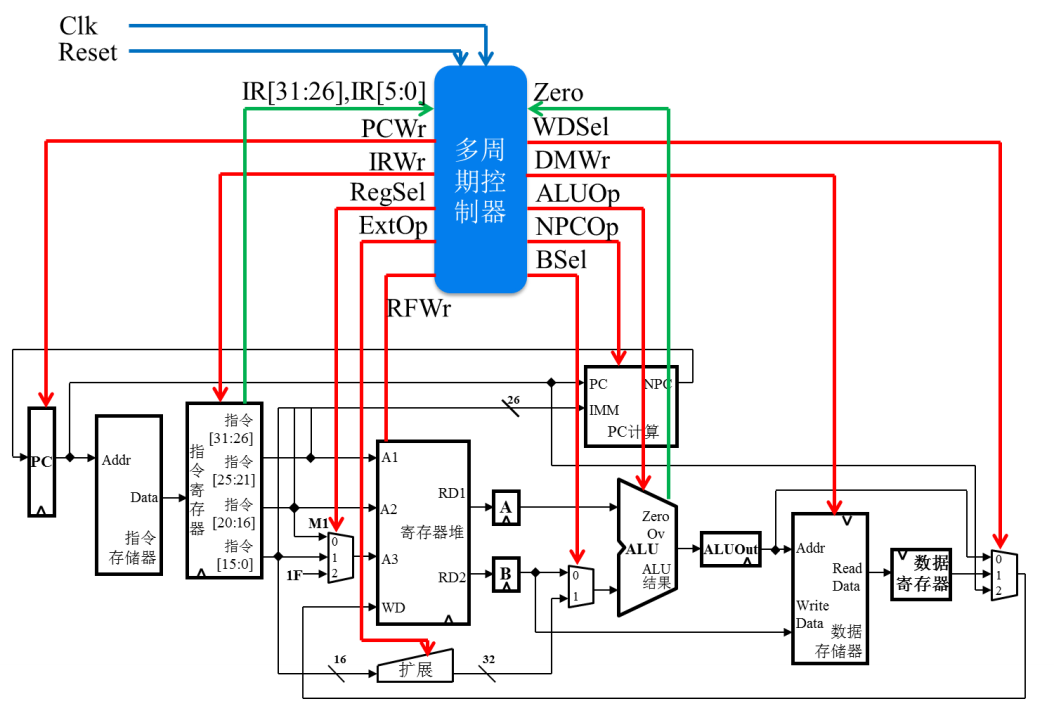


图2 多周期处理器

数据通路应至少包括如下module：PC(程序计数器)、NPC(NextPC计算单元)、RF (通用寄存器组，也称为寄存器文件、寄存器堆)、ALU(算术逻辑单元)、EXT(扩展单元)、IM(指令存储器)、DM(数据存储器)、DR(数据寄存器)等。控制器由状态机控制。

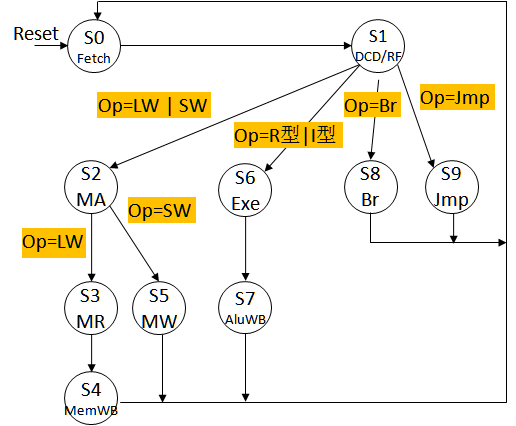


图3 控制器的状态机

图4为参考的数据通路架构图。该图支持MIPS32-C7指令集。如果你对数据通路做了比较大的调整，请注意务必不要与要求矛盾。

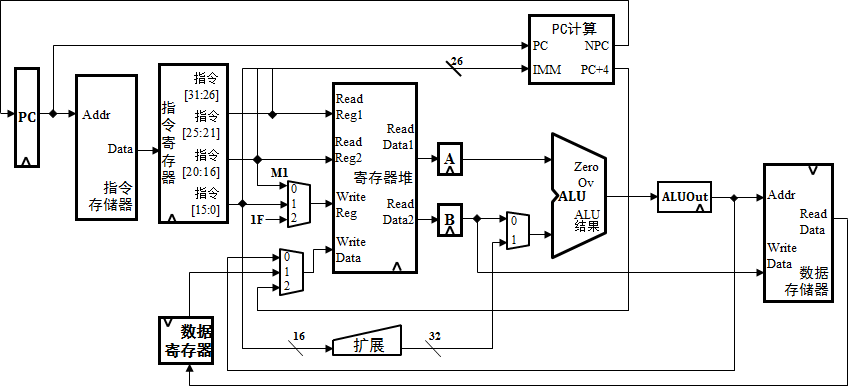


图4 多周期数据通路

MIPS32-C7多周期数据通路应包括PC、NPC、IM、DM这4个独立模块。其中，IM容量为4KB(32bit×1024字)；DM容量为4KB(32bit×1024字)。

层次及模块实例化命名必须满足下列要求：

* 本project的顶层设计文件命名：mips.v。
* PC必须被实例化命名：U\_PC。下面代码为示例。

|  |
| --- |
| pc U\_PC(…) ; // 实例化PC(程序计数器) |

* 指令存储器必须被实例化命名：U\_IM。
* 数据存储器必须被实例化命名：U\_DM。
* 寄存器文件必须被实例化命名：U\_RF。

建议datapath中的每个module都由一个独立的Verilog HDL文件组成。建议所有mux（包括不同位数、不同端口数等）都建模在一个mux.v中。同一个.v文件中可以有多个module。为使得代码更加清晰可读，建议多使用宏定义，并将宏定义组织在1个或多个头文件中。

PC复位后初值为0x0000\_3000，目的是与MARS的Memory Configuration相配合。场测试用的测试程序将通过MARS产生，其配置模式如图5所示。

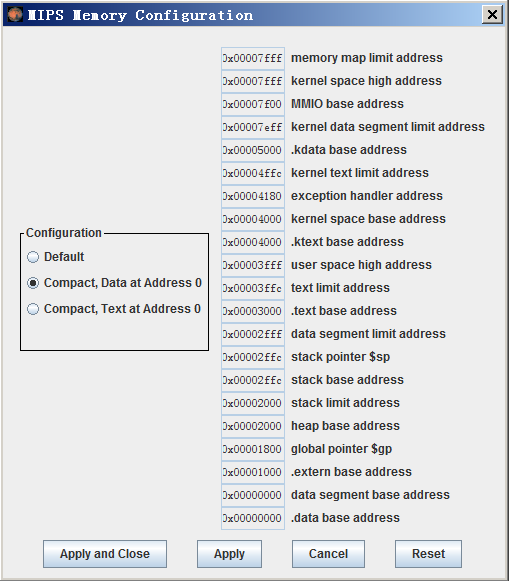


图5 MARS存储配置模式(MARS memory configuration)

图3中的状态机设计仅供参考。可以根据对指令的理解去构造不同的状态机。

但仍然建议遵循下述原则：

(1)按类构造状态分支，而不是按指令。

(2)状态分支不宜过多。

# 四、实验内容

## 1.逻辑部件设计

仿照下面给出的PC模块定义，给出表中所有功能部件的模块定义：

1)必须在Verilog HDL设计中建模这些模块。

2)不允许修改模块名称、端口各信号的名称/类型/位宽。

表2 模块及模块接口定义

|  |  |  |
| --- | --- | --- |
| 序号 | 文件 | 模块接口定义 |
| 1 | PC.v | module PC( clk, rst, PCWr, NPC, PC );  input clk;  input rst;  input PCWr;  input [31:2] NPC;  output [31:2] PC; |
| 2 | NPC.v | module NPC( PC, NPCOp, IMM, NPC );  input [31:2] PC;  input [1:0] NPCOp;  input [25:0] IMM;  output [31:2] NPC; |
| 3 | dm.v | module dm\_4k( addr, din, DMWr, clk, dout ) ;  input [11:2] addr ; // address bus  input [31:0] din ; // 32-bit input data  input DMWr ; // memory write enable  input clk ; // clock  output [31:0] dout ; // 32-bit memory output |
| 4 | im.v | module im\_4k( addr, dout ) ;  input [11:2] addr ; // address bus  output [31:0] dout ; // 32-bit memory output |
| 5 | EXT.v | module EXT( Imm16, EXTOp, Imm32 );  input [15:0] Imm16;  input [1:0] EXTOp;  output [31:0] Imm32; |
| 6 | alu.v | module alu (A, B, ALUOp, C, Zero);  input [31:0] A, B;  input [1:0] ALUOp;  output [31:0] C;  output Zero; |
| 7 | IR.v | module IR (clk, rst, IRWr, im\_dout, instr);  input clk;  input rst;  input IRWr;  input [31:0] im\_dout;  output [31:0] instr; |
| 8 | mux.v |  |
| 9 | flopr.v | module flopr #(parameter WIDTH = 8) (clk, rst, d, q);  input clk;  input rst;  input [WIDTH-1:0] d;  output [WIDTH-1:0] q; |
| 10 | RF.v | module RF( A1, A2, A3, WD, clk, RFWr, RD1, RD2 );  input [4:0] A1, A2, A3;  input [31:0] WD;  input clk;  input RFWr;  output [31:0] RD1, RD2; |
| 11 | global\_def.v  ctrl\_encode\_def.v  instruction\_def.v |  |
| 12 | mips.v | module mips(clk, rst) ;  input clk ; // clock  input rst ;// reset |
| 13 | ctrl.v | module ctrl(clk, rst, Zero, Op, Funct,  RFWr, DMWr, PCWr, IRWr,  EXTOp, ALUOp, NPCOp, GPRSel,  WDSel, BSel);  input clk, rst, Zero;  input [5:0] Op;  input [5:0] Funct;  output RFWr;  output DMWr;  output PCWr;  output IRWr;  output [1:0] EXTOp;  output [1:0] ALUOp;  output [1:0] NPCOp;  output [1:0] GPRSel;  output [1:0] WDSel;  output BSel; |

### 1)PC模块

#### (1)基本描述

PC是指令计数器，主要功能是完成输出当前指令地址并保存下一条指令地址。复位后，PC指向0x0000\_3000，此处为第一条指令的地址。



#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [31:2]NPC | I | 下条指令的地址 |
| PCWr | I | PC写使能  1：允许NPC写入PC内部寄存器  0：禁止NPC写入PC内部寄存器 |
| clk | I | 时钟信号 |
| rst | I | 复位信号。  1：复位  0：无效 |
| [31:2]PC | O | 30位指令存储器地址(最低2位省略) |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | if rst=1，PC🠄0x0000\_3000。 |
| 2 | 保存NPC并输出 | Clk上升沿。  if rst=0,PC🠄 NPC。 |

#### (4)模块描述

module PC( clk, rst, PCWr, NPC, PC );

input clk;

input rst;

input PCWr;

input [31:2] NPC;

output [31:2] PC;

reg [31:2] PC;

reg [1:0] tmp;

always @(posedge clk or posedge rst) begin

if ( rst )

{PC, tmp} <= 32'h0000\_3000;

else if ( PCWr )

PC <= NPC;

end // end always

endmodule

### 2)NPC模块

#### (1)基本描述

NPC是下条指令计数器，主要功能是计算下一条指令地址，NPCOp[1:0]决定如何计算NPC。



#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC[31:2] | I | 30位本条指令的地址(最低2位省略) |
| Imm[25:0] | I | 立即数(偏移量) |
| NPCOp[1:0] | I | 计算方式 |
| NPC[31:2] | O | 30位下一条指令地址(最低2位省略) |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算下一条指令地址 | |  |  |  | | --- | --- | --- | | NPCOp | 功能 | 操作 | | 00 | 顺序地址 | NPC 🠄 PC + 1 | | 01 | 计算B指令转移地址 | NPC 🠄 PC +{ sign\_ext(imm16) } | | 10 | 计算J类指令转移地址 | NPC 🠄 {PC[31:28], imm26} | | 11 | ... | ... | |

#### (4)模块描述

`include "ctrl\_encode\_def.v"

module NPC( PC, NPCOp, IMM, NPC );

input [31:2] PC;

input [1:0] NPCOp;

input [25:0] IMM;

output [31:2] NPC;

reg [31:2] NPC;

always @(\*) begin

case (NPCOp)

`NPC\_PLUS4: NPC = PC + 1;

`NPC\_BRANCH: NPC = PC + {{14{IMM[15]}}, IMM[15:0]};

`NPC\_JUMP: NPC = {PC[31:28], IMM[25:0]};

default: ;

endcase

end // end always

endmodule

### 3)DM模块

#### (1)基本描述

DM是数据存储器，主要功能是根据读写控制信号DMWr，读写对应addr地址的32位数据。



#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| din [31:0] | I | 需要写回的数据 |
| DMWr | I | 读写操作的写使能端  0：禁止写  1：允许写 |
| addr[11:2] | I | 访问地址 |
| dout[31:0] | O | 读出的数据 |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读存储 | 输出地址所对应的数据，dout 🠄dmem[addr] |
| 2 | 写存储 | 当写使能有效时，将待写数据写入对应地址  clk上升沿时  if DMWr then  dmem[addr] 🠄 din |

#### (4)模块描述

module dm\_4k( addr, din, DMWr, clk, dout );

input [11:2] addr;

input [31:0] din;

input DMWr;

input clk;

output [31:0] dout;

reg [31:0] dmem[1023:0];

always @(posedge clk) begin

if (DMWr)

dmem[addr] <= din;

end // end always

assign dout = dmem[addr];

endmodule

### 4)IM模块

#### (1)基本描述

IM是指令存储器，主要功能是根据读控制信号DMWr，读写对应addr地址的32位数据。



#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr[11:2] | I | 访问地址 |
| dout[31:0] | O | 读出的指令 |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读指令存储器 | 输出地址所对应的指令，dout 🠄imem[addr] |

#### (4)模块描述

module im\_4k( addr, dout );

input [11:2] addr;

output [31:0] dout;

reg [31:0] imem[1023:0];

assign dout = imem[addr];

endmodule

### 5)EXT模块

#### (1)基本描述

EXT主要功能是将16位的数据扩展为32位数据。



#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Imm16 [15:0] | I | 需要进行扩展的数据 |
| EXTOp[1:0] | I | 扩展方式的控制信号  00：0扩展  01：符号扩展  10：将立即数扩展到高位 |
| Imm32 [31:0] | O | 扩展结果 |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 扩展 | |  |  |  | | --- | --- | --- | | EXTOp | 功能 | 操作 | | 00 | 无符号扩展 | IMM32 🠄 {16’b0 ， Imm16} | | 01 | 符号扩展 | IMM32 🠄 {{16{Imm16[15]}}, Imm16} | | 10 | 高位扩展 | IMM32 🠄 {Imm16 ， 16’b0} | | 11 |  | 。。。 | |

#### (4)模块描述

`include "ctrl\_encode\_def.v"

module EXT( Imm16, EXTOp, Imm32 );

input [15:0] Imm16;

input [1:0] EXTOp;

output [31:0] Imm32;

reg [31:0] Imm32;

always @(\*) begin

case (EXTOp)

`EXT\_ZERO: Imm32 = {16'd0, Imm16};

`EXT\_SIGNED: Imm32 = {{16{Imm16[15]}}, Imm16};

`EXT\_HIGHPOS: Imm32 = {Imm16, 16'd0};

default: ;

endcase

end // end always

endmodule

### 6)ALU模块

#### (1)基本描述

ALU主要功能是完成对输入数据的算数逻辑计算，包括加法、减法、按位或运算以及判断两个操作数是否相等。



#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A [31:0] | I | 操作数A |
| B [31:0] | I | 操作数B |
| ALUOp[1:0] | I | 需要进行的运算  00：加法  01：减法  10：或运算 |
| Zero | O | 两操作数是否相等 |
| C [31:0] | O | 运算结果 |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 运算 | |  |  |  | | --- | --- | --- | | ALUOp | 功能 | 操作 | | ---- | A等于B? | Zero 🠄 (A==B) ? 1 : 0 | | 0000 | 加 | C 🠄 A + B | | 0001 | 减 | C 🠄 A – B | | 0010 | 与 | C 🠄 A & B | | 0011 | 或 | C 🠄 A | B | | 0100 | 异或 | C 🠄 A ^ B | | ... | ... | ... | |

#### (4)模块描述

`include "ctrl\_encode\_def.v"

module alu (A, B, ALUOp, C, Zero);

input [31:0] A, B;

input [1:0] ALUOp;

output [31:0] C;

output Zero;

reg [31:0] C;

always @( A or B or ALUOp ) begin

case ( ALUOp )

`ALUOp\_ADDU: C = A + B;

`ALUOp\_SUBU: C = A - B;

`ALUOp\_OR: C = A | B;

default: ;

endcase

end // end always;

assign Zero = (A == B) ? 1 : 0;

endmodule

### 7)IR模块

#### (1)基本描述

IR主要功能是完成对来自IM的指令的缓冲。



#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [31:0] im\_dout; | I | 指令输入 |
| IRWr | I | 写使能信号 |
| rst | I | 复位信号 |
| clk | I | 时钟信号 |
| [31:0] instr; | O | 指令输出 |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | if rst=1, instr🠄0 |
| 2 | 缓冲 | if rst=0, instr🠄im\_dout |

#### (4)模块描述

module IR (clk, rst, IRWr, im\_dout, instr);

input clk;

input rst;

input IRWr;

input [31:0] im\_dout;

output [31:0] instr;

reg [31:0] instr;

always @(posedge clk or posedge rst) begin

if ( rst )

instr <= 0;

else if (IRWr)

instr <= im\_dout;

end // end always

endmodule

### 8)mux模块

#### (1)基本描述

mux主要功能是多路选择器。mux.v文件包含二选一、四选一、八选一、十六选一4中多路选择器。实例化多路选择器时，可使用#(XXX)，实例化位宽为XXX的多路选择器。



#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| d0、d1、d2... | I | 供选择数据（d0、d1） |
| s | I | 片选信号 |
| y | O | 片选后的数据 |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择 | |  |  |  | | --- | --- | --- | | s | 功能 | 操作 | | 00 | 选择d0 | y 🠄 d0 | | 01 | 选择d1 | y 🠄 d1 | | 10 | 选择d2 | y 🠄 d2 | | ... | ... | ... | |

#### (4)模块描述

// mux2

module mux2 #(parameter WIDTH = 8)

(d0, d1,

s, y);

input [WIDTH-1:0] d0, d1;

input s;

output [WIDTH-1:0] y;

assign y = ( s == 1'b1 ) ? d1:d0;

endmodule

// mux4

module mux4 #(parameter WIDTH = 8)

(d0, d1, d2, d3,

s, y);

input [WIDTH-1:0] d0, d1, d2, d3;

input [1:0] s;

output [WIDTH-1:0] y;

reg [WIDTH-1:0] y\_r;

always @( \* ) begin

case ( s )

2'b00: y\_r = d0;

2'b01: y\_r = d1;

2'b10: y\_r = d2;

2'b11: y\_r = d3;

default: ;

endcase

end // end always

assign y = y\_r;

endmodule

// mux8

module mux8 #(parameter WIDTH = 8)

(d0, d1, d2, d3,

d4, d5, d6, d7,

s, y);

input [WIDTH-1:0] d0, d1, d2, d3;

input [WIDTH-1:0] d4, d5, d6, d7;

input [2:0] s;

output [WIDTH-1:0] y;

reg [WIDTH-1:0] y\_r;

always @( \* ) begin

case ( s )

3'd0: y\_r = d0;

3'd1: y\_r = d1;

3'd2: y\_r = d2;

3'd3: y\_r = d3;

3'd4: y\_r = d4;

3'd5: y\_r = d5;

3'd6: y\_r = d6;

3'd7: y\_r = d7;

default: ;

endcase

end // end always

assign y = y\_r;

endmodule

// mux16

module mux16 #(parameter WIDTH = 8)

(d0, d1, d2, d3,

d4, d5, d6, d7,

d8, d9, d10, d11,

d12, d13, d14, d15,

s, y);

input [WIDTH-1:0] d0, d1, d2, d3;

input [WIDTH-1:0] d4, d5, d6, d7;

input [WIDTH-1:0] d8, d9, d10, d11;

input [WIDTH-1:0] d12, d13, d14, d15;

input [3:0] s;

output [WIDTH-1:0] y;

reg [WIDTH-1:0] y\_r;

always @( \* ) begin

case ( s )

4'd0: y\_r = d0;

4'd1: y\_r = d1;

4'd2: y\_r = d2;

4'd3: y\_r = d3;

4'd4: y\_r = d4;

4'd5: y\_r = d5;

4'd6: y\_r = d6;

4'd7: y\_r = d7;

4'd8: y\_r = d8;

4'd9: y\_r = d9;

4'd10: y\_r = d10;

4'd11: y\_r = d11;

4'd12: y\_r = d12;

4'd13: y\_r = d13;

4'd14: y\_r = d14;

4'd15: y\_r = d15;

default: ;

endcase

end // end always

assign y = y\_r;

endmodule

### 9)flopr模块

#### (1)基本描述

DR、A、B、ALUOut由flopr模块实例化，主要功能是数据缓冲，由带复位的D触发器构成。



#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [WIDTH-1:0] d | I | 输入数据 |
| rst | I | 复位信号 |
| clk | I | 时钟信号 |
| [WIDTH-1:0] q | O | 输出数据 |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 数据复位 | if rst=0, q🠄d |
| 2 | 数据缓冲 | if rst=1, q🠄0 |

#### (4)模块描述

module flopr #(parameter WIDTH = 8)

(clk, rst, d, q);

input clk;

input rst;

input [WIDTH-1:0] d;

output [WIDTH-1:0] q;

reg [WIDTH-1:0] q\_r;

always @(posedge clk or posedge rst) begin

if ( rst )

q\_r <= 0;

else

q\_r <= d;

end // end always

assign q = q\_r;

endmodule

### 10)RF模块

#### (1)基本描述

RF主要功能是保存寄存器文件，并支持对通用寄存器的访问。



#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A1 [4:0] | I | 需要读的寄存器1的地址 |
| A2 [4:0] | I | 需要读的寄存器2的地址 |
| A3 [4:0] | I | 需要写的寄存器的地址 |
| WD [31:0] | I | 需要写的寄存器的数据 |
| RFWr | I | 寄存器写使能端  0：寄存器不写  1：寄存器写 |
| clk | I | 时钟信号 |
| RD1 [31:0] | O | 需要读的寄存器1的数据 |
| RD2 [31:0] | O | 需要读的寄存器2的数据 |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读取通用寄存器 | 根据输入的RS、RT域的值，输出相应通用寄存器所存储的数据。  RD1🠄RF[A1]；RD2🠄RF[A2] |
| 2 | 写回通用寄存器 | 当RF写使能有效时，将待写的数据写入给定地址的通用寄存器中。  Clk上升沿时  if (RFWr) then  RF[A3]🠄WD |

#### (4)模块描述

`include "global\_def.v"

module RF( A1, A2, A3, WD, clk, RFWr, RD1, RD2 );

input [4:0] A1, A2, A3;

input [31:0] WD;

input clk;

input RFWr;

output [31:0] RD1, RD2;

reg [31:0] rf[31:0];

integer i;

initial begin

for (i=0; i<32; i=i+1)

rf[i] = 0;

end

always @(posedge clk) begin

if (RFWr)

rf[A3] <= WD;

`ifdef DEBUG

$display("R[00-07]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", 0, rf[1], rf[2], rf[3], rf[4], rf[5], rf[6], rf[7]);

$display("R[08-15]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[8], rf[9], rf[10], rf[11], rf[12], rf[13], rf[14], rf[15]);

$display("R[16-23]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[16], rf[17], rf[18], rf[19], rf[20], rf[21], rf[22], rf[23]);

$display("R[24-31]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[24], rf[25], rf[26], rf[27], rf[28], rf[29], rf[30], rf[31]);

`endif

end // end always

assign RD1 = (A1 == 0) ? 32'd0 : rf[A1];

assign RD2 = (A2 == 0) ? 32'd0 : rf[A2];

endmodule

### 11)宏定义文件

#### (1)global\_def.v

定义DEBUG宏，控制是否调试。

`define DEBUG 1

#### (2)instruction\_def.v

定义7条指令的OPCODE/FUNCT域的。

// OP

`define INSTR\_RTYPE\_OP 6'b000000

`define INSTR\_LW\_OP 6'b100011

`define INSTR\_SW\_OP 6'b101011

`define INSTR\_ORI\_OP 6'b001101

`define INSTR\_BEQ\_OP 6'b000100

`define INSTR\_JAL\_OP 6'b000011

// Funct

`define INSTR\_ADDU\_FUNCT 6'b100001

`define INSTR\_SUBU\_FUNCT 6'b100011

#### (3)ctrl\_encode\_def.v

相关控制信号的宏。其中使用Verilog描述控制器的控制信号时，需与该宏保持一致。也可根据情况增加控制信号的宏。

// NPC control signal

`define NPC\_PLUS4 2'b00

`define NPC\_BRANCH 2'b01

`define NPC\_JUMP 2'b10

// B control signal

`define EXT 1'b1

`define B 1'b0

// EXT control signal

`define EXT\_ZERO 2'b00

`define EXT\_SIGNED 2'b01

`define EXT\_HIGHPOS 2'b10

// ALU control signal

`define ALUOp\_ADDU 2'b00

`define ALUOp\_SUBU 2'b01

`define ALUOp\_OR 2'b10

// RF control signal

`define RFSel\_RD 2'b00

`define RFSel\_RT 2'b01

`define RFSel\_31 2'b10

`define WDSel\_FromALU 2'b00

`define WDSel\_FromMEM 2'b01

`define WDSel\_FromPC 2'b10

## 2.数据通路设计

数据通路设计即为mips模块设计，完成mips模块内部组件的实例化和信号连接。

### 1)基本描述

mips模块是一个CPU，只含有复位信号rst和时钟信号clk，内部由PC、NPC、DM、IM、EXT、ALU、IR、Ctrl等模块以及一些多路选择器和缓冲器组成。

### 2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| rst | I | 复位信号 |

### 3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 构建CPU数据通路 | 连接内部组成模块，构建数据通路。 |

### 4)数据通路设计

#### (1) addu rd,rs,rt

##### a)RTL建模分析

|  |  |  |
| --- | --- | --- |
| 周期 | 阶段 | 操作 |
| 1 | Fetch(读取指令) | * + Op1：IM中读出的数据(即指令)写入IR   + Op2：计算PC=PC+4(PC指向下条指令)     - Op2部署在同一个cycle的理由：硬件设计基本原则之一是“尽早执行”     - 即便后续操作需要再次改变PC也无所谓   + 注意1：由于寄存器时序特点，因此Op2不影响Op1   + 注意2：所有指令都必须包括该step |
| 2 | DCD/RF(读取操作数) | * + Op1和Op2分别把RS和RT写入A、B   + 所有指令都必须包括该step     - 该周期隐含包括另一重要步骤：指令译码(控制器内完成) |
| 3 | Exe(执行) | * + Op：ALU完成无符号加法计算并写入ALUOut |
| 4 | ALUWB(结果回写) | * + Op：ALUOut写入RF的RD寄存器 |

##### b)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch(读取指令) | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | IR  NPC  PC | IRWr🠄1;  NPCOp🠄+4;  PCWr🠄1 |
| 2 | DCD/RF(读取操作数) | 2个操作数存入A/B | A🠄RF[rs]; B🠄RF[rt] |  |  |
| 3 | Exe(执行) | 执行加法，结果存入ALUOut | ALUOut🠄ALU(A,B) | ALU | ALUOp🠄`ALUOp\_ADDU |
| 4 | ALUWB(结果回写) | 计算结果回写至rd寄存器 | RF[rd]🠄ALUOut | RF | RFWr🠄1 |

##### c)数据通路

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **目标模块** | **周期** | **NPC** | | | **PC** | | | **IM** | **IR** | | | **RF** | | | | | **A** | **B** | **EXT** | | **ALU** | | | **ALUOut** | **DM** | | | **DR** |
| **PC** | **IMM** | **NPCOp** | **NPC** | **PCWr** | **rst** | **addr** | **im\_dout** | **IRWr** | **rst** | **A1** | **A2** | **A3** | **WD** | **RFWr** |  |  | **Imm16** | **EXTOp** | **A** | **B** | **ALUOp** |  | **addr** | **din** | **DMWr** | **d** |
| **ADDU rd,rs,rt** | **1** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  | **rs** | **rt** |  |  |  | **RF.RD1** | **RF.RD2** |  |  |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **A.q** | **B.q** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |
| **4** |  |  |  |  |  |  |  |  |  |  |  |  | **rd** | **ALUOut.q** | **Ctrl.RFWr** |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** | **rt** | **rd** | **aLUOut.q** | **ctrl.RFWr** | **RF.RD1** | **RF.RD2** |  |  | **A.q** | **B.q** |  |  |  |  |  |  |

#### (2) subu rd,rs,rt

##### a)RTL建模分析

|  |  |  |
| --- | --- | --- |
| 周期 | 阶段 | 操作 |
| 1 | Fetch(读取指令) | * + Op1：IM中读出的数据(即指令)写入IR   + Op2：计算PC=PC+4(PC指向下条指令)     - Op2部署在同一个cycle的理由：硬件设计基本原则之一是“尽早执行”     - 即便后续操作需要再次改变PC也无所谓   + 注意1：由于寄存器时序特点，因此Op2不影响Op1   + 注意2：所有指令都必须包括该step |
| 2 | DCD/RF(读取操作数) | * + Op1和Op2分别把RS和RT写入A、B   + 所有指令都必须包括该step     - 该周期隐含包括另一重要步骤：指令译码(控制器内完成) |
| 3 | Exe(执行) | * + Op：ALU完成无符号减法计算并写入ALUOut |
| 4 | ALUWB(结果回写) | * + Op：ALUOut写入RF的RD寄存器 |

##### b)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch(读取指令) | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | IR  NPC  PC | IRWr🠄1;  NPCOp🠄+4;  PCWr🠄1 |
| 2 | DCD/RF(读取操作数) | 2个操作数存入A/B | A🠄RF[rs]; B🠄RF[rt] |  |  |
| 3 | Exe(执行) | 执行加法，结果存入ALUOut | ALUOut🠄ALU(A,B) | ALU | ALUOp🠄`ALUOp\_SUBU |
| 4 | ALUWB(结果回写) | 计算结果回写至rd寄存器 | RF[rd]🠄ALUOut | RF | RFWr🠄1 |

##### c)数据通路

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **目标模块** | **周期** | **NPC** | | | **PC** | | | **IM** | **IR** | | | **RF** | | | | | **A** | **B** | **EXT** | | **ALU** | | | **ALUOut** | **DM** | | | **DR** |
| **PC** | **IMM** | **NPCOp** | **NPC** | **PCWr** | **rst** | **addr** | **im\_dout** | **IRWr** | **rst** | **A1** | **A2** | **A3** | **WD** | **RFWr** |  |  | **Imm16** | **EXTOp** | **A** | **B** | **ALUOp** |  | **addr** | **din** | **DMWr** | **d** |
| **subu rd,rs,rt** | **1** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  | **rs** | **rt** |  |  |  | **RF.RD1** | **RF.RD2** |  |  |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **A.q** | **B.q** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |
| **4** |  |  |  |  |  |  |  |  |  |  |  |  | **rd** | **ALUOut.q** | **Ctrl.RFWr** |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** | **rt** | **rd** | **aLUOut.q** | **ctrl.RFWr** | **RF.RD1** | **RF.RD2** |  |  | **A.q** | **B.q** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |

#### (3) ori rt,rs,imm16

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch(读取指令) | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | IR  NPC  PC | IRWr🠄1;  NPCOp🠄+4;  PCWr🠄1 |
| 2 | DCD/RF(读取操作数) | 操作数存入A  无符号扩展 | A🠄RF[rs]; EXT(Imm16) | EXT | EXTOp🠄`UE |
| 3 | Exe(执行) | 执行加法，结果存入ALUOut | ALUOut🠄ALU(A,EXT) | ALU | ALUOp🠄` ALUOp\_OR |
| 4 | ALUWB(结果回写) | 计算结果回写至rt寄存器 | RF[rt]🠄ALUOut | RF | RFWr🠄1 |

##### b)数据通路

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **目标模块** | **周期** | **NPC** | | | **PC** | | | **IM** | **IR** | | | **RF** | | | | | **A** | **B** | **EXT** | | **ALU** | | | **ALUOut** | **DM** | | | **DR** |
| **PC** | **IMM** | **NPCOp** | **NPC** | **PCWr** | **rst** | **addr** | **im\_dout** | **IRWr** | **rst** | **A1** | **A2** | **A3** | **WD** | **RFWr** |  |  | **Imm16** | **EXTOp** | **A** | **B** | **ALUOp** |  | **addr** | **din** | **DMWr** | **d** |
| **ori rt,rs,imm16** | **1** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  | **rs** |  |  |  |  | **RF.RD1** |  | **Imm16** | **Ctrl.EXTOp** |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **A.q** | **EXT.** **Imm32** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |
| **4** |  |  |  |  |  |  |  |  |  |  |  | **rt** |  | **ALUOut.q** | **Ctrl.RFWr** |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** | **rt** | **rd** | **aLUOut.q** | **ctrl.RFWr** | **RF.RD1** |  | **Imm16** | **Ctrl.EXTOp** | **A.q** | **EXT.** **Imm32** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |

#### (4) lw rt, imm16(rs)

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch  取指令 | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | IR  NPC  PC | IRWr🠄1;  NPCOp🠄+4;  PCWr🠄1 |
| 2 | DCD/RF  读操作数 | 基地址存入A；  偏移符号扩展 | A🠄RF[rs];  ***EXT(Imm16)*** | EXT | EXTOp🠄`SE |
| 3 | MA  计算地址 | 执行加法，结果存入ALUOut | ALUOut🠄ALU(A,EXT) | ALU | ALUOp🠄`ALUOp\_ADD |
| 4 | MR  读存储器 | 读取DM，数据存储DR | DR🠄DM[ALUOut] |  |  |
| 5 | MemWB  回写 | DR写入rt寄存器 | RF[rt]🠄DR | RF | RFWr🠄1 |

##### b)数据通路

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **目标模块** | **周期** | **NPC** | | | **PC** | | | **IM** | **IR** | | | **RF** | | | | | **A** | **B** | **EXT** | | **ALU** | | | **ALUOut** | **DM** | | | **DR** |
| **PC** | **IMM** | **NPCOp** | **NPC** | **PCWr** | **rst** | **addr** | **im\_dout** | **IRWr** | **rst** | **A1** | **A2** | **A3** | **WD** | **RFWr** |  |  | **Imm16** | **EXTOp** | **A** | **B** | **ALUOp** |  | **addr** | **din** | **DMWr** | **d** |
| **lw rt, imm16(rs)** | **1** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  | **rs** |  |  |  |  | **RF.RD1** |  | **Imm16** | **Ctrl.EXTOp** |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **A.q** | **EXT.** **Imm32** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |
| **4** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **ALUOut.q** |  |  | **DM.dout** |
| **5** |  |  |  |  |  |  |  |  |  |  |  |  | **rt** | **DR.q** | **Ctrl.RFWr** |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** |  | **rt** | **DR.q** | **ctrl.RFWr** | **RF.RD1** |  | **Imm16** | **Ctrl.EXTOp** | **A.q** | **EXT.** **Imm32** | **Ctrl.ALUOp** | **ALUOut.d** | **ALUOut.q** |  |  | **DM.dout** |

#### (5) sw rt, imm16(rs)

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch  取指令 | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | IR  NPC  PC | IRWr🠄1  NPCOp🠄+4  PCWr🠄1 |
| 2 | DCD/RF  读操作数 | 基地址存入A；  偏移符号扩展 | A🠄RF[rs]  EXT(Imm16) | EXT | EXTOp🠄`SE |
| 3 | MA  计算地址 | 执行加法，结果存入ALUOut | ALUOut🠄ALU(A,EXT) | ALU | ALUOp🠄`ALUOp\_ADD |
| 4 | MW  写存储器 | rt寄存器写入DM | DM[ALUOut]🠄RF[rt] | DM | DMWr🠄1 |

##### b)数据通路

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **目标模块** | **周期** | **NPC** | | | **PC** | | | **IM** | **IR** | | | **RF** | | | | | **A** | **B** | **EXT** | | **ALU** | | | **ALUOut** | **DM** | | | **DR** |
| **PC** | **IMM** | **NPCOp** | **NPC** | **PCWr** | **rst** | **addr** | **im\_dout** | **IRWr** | **rst** | **A1** | **A2** | **A3** | **WD** | **RFWr** |  |  | **Imm16** | **EXTOp** | **A** | **B** | **ALUOp** |  | **addr** | **din** | **DMWr** | **d** |
| **sw rt, imm16(rs)** | **1** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  | **rs** |  |  |  |  | **RF.RD1** |  | **Imm16** | **Ctrl.EXTOp** |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **A.q** | **EXT.** **Imm32** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |
| **4** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **ALUOut.q** | **RF.RD2** | **Ctrl.DMWr** |  |
|  | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** |  |  |  |  | **RF.RD1** |  | **Imm16** | **Ctrl.EXTOp** | **A.q** | **EXT.** **Imm32** | **Ctrl.ALUOp** | **ALUOut.d** | **ALUOut.q** | **RF.RD2** | **Ctrl.DMWr** |  |

#### (6) beq rs,rt, imm16

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch  取指令 | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | NPC  PC  IR | IRWr🠄1  NPCOp🠄+4  PCWr🠄1 |
| 2 | DCD/RF  读操作数 | RS操作数存入A；  RS操作数存入B | A🠄RF[rs];  B🠄RF[rt]; |  |  |
| 3 | Br  执行 | 执行减法，判断Zero | ALUOut🠄ALU(A,B)  PC🠄NPC(PC,imm16) | ALU  NPC  PC | ALUOp🠄`ALUOp\_SUB  NPC🠄`BNPC  PCWr🠄Zero |

##### b)数据通路

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **目标模块** | **周期** | **NPC** | | | **PC** | | | **IM** | **IR** | | | **RF** | | | | | **A** | **B** | **EXT** | | **ALU** | | | **ALUOut** | **DM** | | | **DR** |
| **PC** | **IMM** | **NPCOp** | **NPC** | **PCWr** | **rst** | **addr** | **im\_dout** | **IRWr** | **rst** | **A1** | **A2** | **A3** | **WD** | **RFWr** |  |  | **Imm16** | **EXTOp** | **A** | **B** | **ALUOp** |  | **addr** | **din** | **DMWr** | **d** |
| **beq rs,rt, imm16** | **1** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  | **rs** | **rt** |  |  |  | **RF.RD1** | **RF.RD2** | **Imm16** | **Ctrl.EXTOp** |  |  |  |  |  |  |  |  |
| **3** |  | **Imm16** | **Ctrl.NPCOp** |  | **Ctrl.PCWr** |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **A.q** | **B.q** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |
|  | **PC.PC** | **Imm16** | **Ctrl.NPCOp** | **NPC.NPC** | **Ctrl.PCWr** |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** | **rt** |  |  |  | **RF.RD1** | **RF.RD2** | **Imm16** | **Ctrl.EXTOp** | **A.q** | **B.q** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |

#### (7) jal imm26

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch  取指令 | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | NPC  PC  IR | IRWr🠄1  NPCOp🠄+4  PCWr🠄1 |
| 2 | DCD/RF  读操作数 |  |  |  |  |
| 3 | JMP  执行 | 计算并保存转移PC；  保存PC | RF[31]🠄PC  PC🠄NPC(PC,imm26) | RF  NPC  PC | RFWr🠄1  NPCOp🠄`JNPC  PCWr🠄1 |

##### b)数据通路

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **目标模块** | **周期** | **NPC** | | | **PC** | | | **IM** | **IR** | | | **RF** | | | | | **A** | **B** | **EXT** | | **ALU** | | | **ALUOut** | **DM** | | | **DR** |
| **PC** | **IMM** | **NPCOp** | **NPC** | **PCWr** | **rst** | **addr** | **im\_dout** | **IRWr** | **rst** | **A1** | **A2** | **A3** | **WD** | **RFWr** |  |  | **Imm16** | **EXTOp** | **A** | **B** | **ALUOp** |  | **addr** | **din** | **DMWr** | **d** |
| jal addr26 | **1** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **3** | **PC.PC** | **Imm26** | **Ctrl.NPCOp** |  | **Ctrl.PCWr** |  |  |  |  |  |  |  | **‘d31** | **PC.PC** | **Ctrl.RFWr** |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **PC.PC** | **Imm26** | **Ctrl.NPCOp** | **NPC.NPC** | **Ctrl.PCWr** |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  |  |  | **‘d31** | **PC.PC** | **Ctrl.RFWr** |  |  |  |  |  |  |  |  |  |  |  |  |

### 5)数据通路合成

综合各指令的数据通路，得表：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **目标模块** | **NPC** | | | **PC** | | | **IM** | **IR** | | | **RF** | | | | | **A** | **B** | **EXT** | | **ALU** | | | **ALUOut** | **DM** | | | **DR** |
| **PC** | **IMM** | **NPCOp** | **NPC** | **PCWr** | **rst** | **addr** | **im\_dout** | **IRWr** | **rst** | **A1** | **A2** | **A3** | **WD** | **RFWr** |  |  | **Imm16** | **EXTOp** | **A** | **B** | **ALUOp** |  | **addr** | **din** | **DMWr** | **d** |
| **addu rd,rs,rt** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** | **rt** | **rd** | **aLUOut.q** | **ctrl.RFWr** | **RF.RD1** | **RF.RD2** |  |  | **A.q** | **B.q** |  |  |  |  |  |  |
| **subu rd,rs,rt** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** | **rt** | **rd** | **aLUOut.q** | **ctrl.RFWr** | **RF.RD1** | **RF.RD2** |  |  | **A.q** | **B.q** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |
| **ori rt,rs,imm16** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** | **rt** | **rd** | **aLUOut.q** | **ctrl.RFWr** | **RF.RD1** |  | **Imm16** | **Ctrl.EXTOp** | **A.q** | **EXT.** **Imm32** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |
| **lw rt, imm16(rs)** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** |  | **rt** | **DR.q** | **ctrl.RFWr** | **RF.RD1** |  | **Imm16** | **Ctrl.EXTOp** | **A.q** | **EXT.** **Imm32** | **Ctrl.ALUOp** | **ALUOut.d** | **ALUOut.q** |  |  | **DM.dout** |
| **sw rt, imm16(rs)** | **PC.PC** |  | **Ctrl.NPCOp** | **NPC.NPC** |  |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** |  |  |  |  | **RF.RD1** |  | **Imm16** | **Ctrl.EXTOp** | **A.q** | **EXT.** **Imm32** | **Ctrl.ALUOp** | **ALUOut.d** | **ALUOut.q** | **RF.RD2** | **Ctrl.DMWr** |  |
| **beq rs,rt, imm16** | **PC.PC** | **Imm16** | **Ctrl.NPCOp** | **NPC.NPC** | **Ctrl.PCWr** |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** | **rt** |  |  |  | **RF.RD1** | **RF.RD2** | **Imm16** | **Ctrl.EXTOp** | **A.q** | **B.q** | **Ctrl.ALUOp** | **ALUOut.d** |  |  |  |  |
| **jal addr26** | **PC.PC** | **Imm26** | **Ctrl.NPCOp** | **NPC.NPC** | **Ctrl.PCWr** |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  |  |  | **‘d31** | **PC.PC** | **Ctrl.RFWr** |  |  |  |  |  |  |  |  |  |  |  |  |
| 合成 | **PC.PC** | **Imm26** | **Ctrl.NPCOp** | **NPC.NPC** | **Ctrl.PCWr** |  | **PC.PC** | **IM.dout** | **Ctrl.IRWr** |  | **rs** | **rt** | **rd** | **DR.q** | **Ctrl.RFWr** | **RF.RD1** | **RF.RD2** | **Imm16** | **Ctrl.EXTOp** | **A.q** | **B.q** | **Ctrl.ALUOp** | **ALUOut.d** | **ALUOut.q** | **RF.RD2** | **Ctrl.DMWr** | **DM.dout** |
| **rt** | **aLUOut.q** |
| **EXT. Imm32** |
| **‘d31** | **PC.PC** |
| 多路选择器 |  |  |  |  |  |  |  |  |  |  |  |  | **MUX4\_RF\_A3** | **MUX4\_RF\_WD** |  |  |  |  |  |  | **MUX2\_ALU\_B** |  |  |  |  |  |  |

### 6)模块描述

module mips( clk, rst );

input clk;

input rst;

wire RFWr;

wire DMWr;

wire PCWr;

wire IRWr;

wire [1:0] EXTOp;

wire [1:0] ALUOp;

wire [1:0] NPCOp;

wire [1:0] RF\_A3\_sel;

wire [1:0] RF\_WD\_sel;

wire B\_sel;

wire Zero;

wire [29:0] PC, NPC;

wire [31:0] im\_dout, dm\_dout;

wire [31:0] DR\_dout;

wire [31:0] instr;

wire [4:0] rs;

wire [4:0] rt;

wire [4:0] rd;

wire [5:0] Op;

wire [5:0] Funct;

wire [15:0] Imm16;

wire [31:0] Imm32;

wire [25:0] Imm26;

wire [4:0] A3;

wire [31:0] WD;

wire [31:0] RD1, RD1\_r, RD2, RD2\_r;

wire [31:0] B, C, C\_r;

assign Op = instr[31:26];

assign Funct = instr[5:0];

assign rs = instr[25:21];

assign rt = instr[20:16];

assign rd = instr[15:11];

assign Imm16 = instr[15:0];

assign Imm26 = instr[25:0];

ctrl U\_CTRL (

.clk(clk), .rst(rst), .Zero(Zero), .Op(Op), .Funct(Funct),

.RFWr(RFWr), .DMWr(DMWr), .PCWr(PCWr), .IRWr(IRWr),

.EXTOp(EXTOp), .ALUOp(ALUOp), .NPCOp(NPCOp), .RF\_A3\_sel(RF\_A3\_sel),

.RF\_WD\_sel(RF\_WD\_sel), .B\_sel(B\_sel)

);

PC U\_PC (

.clk(clk), .rst(rst), .PCWr(PCWr), .NPC(NPC), .PC(PC)

);

NPC U\_NPC (

.PC(PC), .NPCOp(NPCOp), .IMM(Imm26), .NPC(NPC)

);

im\_4k U\_IM (

.addr(PC[9:0]) , .dout(im\_dout)

);

IR U\_IR (

.clk(clk), .rst(rst), .IRWr(IRWr), .im\_dout(im\_dout), .instr(instr)

);

RF U\_RF (

.A1(rs), .A2(rt), .A3(A3), .WD(WD), .clk(clk),

.RFWr(RFWr), .RD1(RD1), .RD2(RD2)

);

mux4 #(5) U\_MUX4\_RF\_A3 (

.d0(rd), .d1(rt), .d2(5'd31), .d3(/\*undefine\*/5'd0),

.s(RF\_A3\_sel), .y(A3)

);

mux4 #(32) U\_MUX4\_RF\_WD (

.d0(C\_r), .d1(DR\_dout), .d2({PC,2'b00}), .d3(/\*undefine\*/32'd0),

.s(RF\_WD\_sel), .y(WD)

);

flopr #(32) U\_A (

.clk(clk), .rst(rst), .d(RD1), .q(RD1\_r)

);

flopr #(32) U\_B (

.clk(clk), .rst(rst), .d(RD2), .q(RD2\_r)

);

EXT U\_EXT (

.Imm16(Imm16), .EXTOp(EXTOp), .Imm32(Imm32)

);

mux2 #(32) U\_MUX2\_ALU\_B (

.d0(RD2\_r), .d1(Imm32), .s(B\_sel) , .y(B)

);

alu U\_ALU (

.A(RD1\_r), .B(B), .ALUOp(ALUOp), .C(C), .Zero(Zero)

);

flopr #(32) U\_ALUOut (

.clk(clk), .rst(rst), .d(C), .q(C\_r)

);

dm\_4k U\_DM (

.addr(C\_r[11:2]), .din(RD2\_r), .DMWr(DMWr), .clk(clk), .dout(dm\_dout)

);

flopr #(32) U\_DR (

.clk(clk), .rst(rst), .d(dm\_dout), .q(DR\_dout)

);

endmodule

## 3.控制器设计

控制器设计即为ctrl模块设计，按多周期方式实现每条指令各阶段对微处理器各组件的执行控制。

### 1)基本描述

ctrl主要功能是通过使用状态机，根据指令产生数据通路各模块在对应状态中的控制信号。

### 2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| rst | I | 复位信号 |
| OP[5:0] | I | 指令格式中的OPCODE域 |
| Funct[5:0] | I | 指令格式中的FUNCT域 |
| Zero | I | ALU输出信号  0：ALU两操作数不等  1：ALU两操作数相等 |
| B\_sel | O | ALU第二操作数的片选信号  0：RT域对应的数据  1：16位立即数通过EXT模块扩张后的数据 |
| RF\_WD\_sel[1:0] | O | 寄存器写入数据的片选信号  0：ALU计算结果  1：DM读出数据  2：PC（针对JAL指令） |
| RFWr | O | RF的写使能信号  0：禁止写  1：允许写 |
| DMWr | O | 数据存储器写使能信号  0：禁止写  1：允许写 |
| NPCOp[1:0] | O | NPC的片选信号  00：选择PC+4  01：选择分支地址  10：选择跳转地址 |
| EXTOp[1:0] | O | 数据扩展模式选择信号  00：零扩展  01：符号扩展  10：16位立即数扩展到高位 |
| ALUOp[1:0] | O | 传送给ALU的运算控制信号。  00：加法  01：减法  10：或运算 |
| PCWr | O | PC写使能，控制PC写入下一条地址  0：禁止写  1：允许写 |
| IRWr | O | IR写使能，控制IR写入下一条指令  0：禁止写  1：允许写 |
| RF\_A3\_sel[1:0] | O | RF的写回寄存器的地址片选信号  00：指令格式中的Rd域  01：指令格式中的Rs域  10：$31寄存器（针对JAL） |

### 3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 控制RF回写寄存器地址的选择 | 选择正确的RF回写地址 |
| 2 | 控制ALU第二操作数的选择 | 选择正确的ALU第二操作数 |
| 3 | 控制RF回写数据的选择 | 选择正确的RF回写数据 |
| 4 | 控制RF写使能 | 控制RF的写使能 |
| 5 | 控制数据存储器的写使能 | 控制数据存储器的写使能端 |
| 6 | 控制下一条指令地址的选择 | 00：PC+4  01：分支地址  10：跳转地址 |
| 7 | 控制EXT单元的扩展方式 | 控制正确的立即数扩展方式 |
| 8 | 控制ALU执行的运算操作 | 控制ALU执行的运算操作。 |

### 4)状态表设计

#### (1) addu rd,rs,rt

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch(读取指令) | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | IR  NPC  PC | IRWr🠄1;  NPCOp🠄+4;  PCWr🠄1 |
| 2 | DCD/RF(读取操作数) | 2个操作数存入A/B | A🠄RF[rs]; B🠄RF[rt] |  |  |
| 3 | Exe(执行) | 执行加法，结果存入ALUOut | ALUOut🠄ALU(A,B) | ALU | `ALUOp\_ADDU |
| 4 | ALUWB(结果回写) | 计算结果回写至rd寄存器 | RF[rd]🠄ALUOut | RF | RFWr🠄1 |

##### b)状态机



##### c)控制信号设计

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 控制信号 | | NPCOp | PCWr | IRWr | RF\_WD\_sel  (MUX4\_RF\_WD) | RF\_A3\_sel  (MUX4\_RF\_A3) | RFWr | EXTOp | B\_sel  (MUX) | ALUOp | DMWr |
| addu rd,rs,rt | S0:Fetch | S0:+4 | S0 | S0 |  |  |  |  |  |  |  |
| S1:DCD/RF |  |  |  |  |  |  |  |  |  |  |
| S6:Exe |  |  |  |  |  |  |  | S6:`B | S6:`ALUOp\_ADDU |  |
| S7:ALUWB |  |  |  | S7:`ALUOut | S7:`RD | S7 |  |  |  |  |
|  |  | S0:+4 | S0 | S0 | S7:`ALUOut | S7:`RD | S7 |  | S6:`B | S6:`ALUOp\_ADDU |  |

#### (2) subu rd,rs,rt

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch(读取指令) | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | IR  NPC  PC | IRWr🠄1;  NPCOp🠄+4;  PCWr🠄1 |
| 2 | DCD/RF(读取操作数) | 2个操作数存入A/B | A🠄RF[rs]; B🠄RF[rt] |  |  |
| 3 | Exe(执行) | 执行加法，结果存入ALUOut | ALUOut🠄ALU(A,B) | ALU | `ALUOp\_SUBU |
| 4 | ALUWB(结果回写) | 计算结果回写至rd寄存器 | RF[rd]🠄ALUOut | RF | RFWr🠄1 |

##### b)状态机



##### c)控制信号设计

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 控制信号 | | NPCOp | PCWr | IRWr | RF\_WD\_sel  (MUX4\_RF\_WD) | RF\_A3\_sel  (MUX4\_RF\_A3) | RFWr | EXTOp | B\_sel  (MUX) | ALUOp | DMWr |
| subu rd,rs,rt | S0:Fetch | S0:+4 | S0 | S0 |  |  |  |  |  |  |  |
| S1:DCD/RF |  |  |  |  |  |  |  |  |  |  |
| S6:Exe |  |  |  |  |  |  |  | S6:`B | S6:`ALUOp\_SUBU |  |
| S7:ALUWB |  |  |  | S7:`ALUOut | S7:`RD | S7 |  |  |  |  |
|  |  | S0:+4 | S0 | S0 | S7:`ALUOut | S7:`RD | S7 |  | S6:`B | S6:`ALUOp\_SUBU |  |

#### (3) ori rt,rs,imm16

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch(读取指令) | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | IR  NPC  PC | IRWr🠄1;  NPCOp🠄+4;  PCWr🠄1 |
| 2 | DCD/RF(读取操作数) | 操作数存入A  无符号扩展 | A🠄RF[rs]; EXT(Imm16) | EXT | EXTOp🠄`UE |
| 3 | Exe(执行) | 执行加法，结果存入ALUOut | ALUOut🠄ALU(A,EXT) | ALU | ALUOp🠄 `ALUOp\_OR |
| 4 | ALUWB(结果回写) | 计算结果回写至rt寄存器 | RF[rt]🠄ALUOut | RF | RFWr🠄1 |

##### b)状态机



##### c)控制信号设计

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 控制信号 | | NPCOp | PCWr | IRWr | RF\_WD\_sel  (MUX4\_RF\_WD) | RF\_A3\_sel  (MUX4\_RF\_A3) | RFWr | EXTOp | B\_sel  (MUX) | ALUOp | DMWr |
| ori rt,rs,imm16 | S0:Fetch | S0:+4 | S0 | S0 |  |  |  |  |  |  |  |
| S1:DCD/RF |  |  |  |  |  |  | S1:`UE |  |  |  |
| S6:Exe |  |  |  |  |  |  |  | S6:`EXT | S6:`ALUOp\_OR |  |
| S7:ALUWB |  |  |  | S7:`ALUOut | S7:`RT | S7 |  |  |  |  |
|  |  | S0:+4 | S0 | S0 | S7:`ALUOut | S7:`RT | S7 | S1:`UE | S6:`EXT | S6:`ALUOp\_OR |  |

#### (4) lw rt,imm16(rs)

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch  取指令 | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | IR  NPC  PC | IRWr🠄1;  NPCOp🠄+4;  PCWr🠄1 |
| 2 | DCD/RF  读操作数 | 基地址存入A；  偏移符号扩展 | A🠄RF[rs];  ***EXT(Imm16)*** | EXT | EXTOp🠄`SE |
| 3 | MA  计算地址 | 执行加法，结果存入ALUOut | ALUOut🠄ALU(A,EXT) | ALU | ALUOp🠄`ALUOp\_ADD |
| 4 | MR  读存储器 | 读取DM，数据存储DR | DR🠄DM[ALUOut] |  |  |
| 5 | MemWB  回写 | DR写入rt寄存器 | RF[rt]🠄DR | RF | RFWr🠄1 |

##### b)状态机



##### c)控制信号设计

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 控制信号 | | NPCOp | PCWr | IRWr | RF\_WD\_sel  (MUX4\_RF\_WD) | RF\_A3\_sel  (MUX4\_RF\_A3) | RFWr | EXTOp | B\_sel  (MUX) | ALUOp | DMWr |
| lw rt,imm16(rs) | S0:Fetch | S0:+4 | S0 | S0 |  |  |  |  |  |  |  |
| S1:DCD/RF |  |  |  |  |  |  | S1:`SE |  |  |  |
| S2:MA |  |  |  |  |  |  |  | S2: `EXT | S2:`ALUOp\_ADD |  |
| S3:MR |  |  |  |  |  |  |  |  |  |  |
| S4:MemWB |  |  |  | S4:`DR | S4:`RT | S4 |  |  |  |  |
|  |  | S0:+4 | S0 | S0 | S4:`DR | S4:`RT | S4 | S1:`SE | S2: `EXT | S2:`ALUOp\_ADD |  |

#### (5) sw rt, imm16(rs)

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch  取指令 | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | IR  NPC  PC | IRWr🠄1  NPCOp🠄+4  PCWr🠄1 |
| 2 | DCD/RF  读操作数 | 基地址存入A；  偏移符号扩展 | A🠄RF[rs]  EXT(Imm16) | EXT | EXTOp🠄`SE |
| 3 | MA  计算地址 | 执行加法，结果存入ALUOut | ALUOut🠄ALU(A,EXT) | ALU | ALUOp🠄`ALUOp\_ADD |
| 4 | MW  写存储器 | rt寄存器写入DM | DM[ALUOut]🠄RF[rt] | DM | DMWr🠄1 |

##### b)状态机



##### c)控制信号设计

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 控制信号 | | NPCOp | PCWr | IRWr | RF\_WD\_sel  (MUX4\_RF\_WD) | RF\_A3\_sel  (MUX4\_RF\_A3) | RFWr | EXTOp | B\_sel  (MUX) | ALUOp | DMWr |
| sw rt,imm16(rs) | S0:Fetch | S0:+4 | S0 | S0 |  |  |  |  |  |  |  |
| S1:DCD/RF |  |  |  |  |  |  | S1:`SE |  |  |  |
| S2:MA |  |  |  |  |  |  |  | S2: `EXT | S2:`ALUOp\_ADD |  |
| S5:MW |  |  |  |  |  |  |  |  |  | S5 |
|  |  | S0:+4 | S0 | S0 |  |  |  | S1:`SE | S2: `EXT | S2:`ALUOp\_ADD | S5 |

#### (6) beq rs,rt, imm16

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch  取指令 | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | NPC  PC  IR | IRWr🠄1  NPCOp🠄+4  PCWr🠄1 |
| 2 | DCD/RF  读操作数 | RS操作数存入A；  RS操作数存入B | A🠄RF[rs];  B🠄RF[rt]; |  |  |
| 3 | Br  执行 | 执行减法，判断Zero | ALUOut🠄ALU(A,B)  PC🠄NPC(PC,imm16) | ALU  NPC  PC | ALUOp🠄`ALUOp\_SUB  NPC🠄`BNPC  PCWr🠄Zero |

##### b)状态机



##### c)控制信号设计

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 控制信号 | | NPCOp | PCWr | IRWr | RF\_WD\_sel  (MUX4\_RF\_WD) | RF\_A3\_sel  (MUX4\_RF\_A3) | RFWr | EXTOp | B\_sel  (MUX) | ALUOp | DMWr |
| beq rs,rt,imm16 | S0:Fetch | S0:+4 | S0 | S0 |  |  |  |  |  |  |  |
| S1:DCD/RF |  |  |  |  |  |  |  |  |  |  |
| S8:Branch | S8:`BNPC | S8 |  |  |  |  |  | S8: `B | S8:`ALUOp\_SUB |  |
|  |  | S0:+4  S8:`BNPC | S0  S8:Zero | S0 |  |  |  | S1:`HE | S8: `B | S8:`ALUOp\_SUB |  |

#### (7) jal imm26

##### a)RTL描述表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 周期 | 阶段 | 语义 | RTL | 需控制的功能部件 | 功能部件控制信号 |
| 1 | Fetch  取指令 | 读取指令；  计算下条指令地址 | IR🠄IM[PC]; PC🠄NPC(PC) | NPC  PC  IR | IRWr🠄1  NPCOp🠄+4  PCWr🠄1 |
| 2 | DCD/RF  读操作数 |  |  |  |  |
| 3 | JMP  执行 | 计算并保存转移PC；  保存PC | RF[31]🠄PC  PC🠄NPC(PC,imm26) | RF  NPC  PC | RFWr🠄1  NPCOp🠄`JNPC  PCWr🠄1 |

##### b)状态机



##### c)控制信号

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 控制信号 | | NPCOp | PCWr | IRWr | RF\_WD\_sel  (MUX4\_RF\_WD) | RF\_A3\_sel  (MUX4\_RF\_A3) | RFWr | EXTOp | B\_sel  (MUX) | ALUOp | DMWr |
| jal imm26 | S0:Fetch | S0:+4 | S0 | S0 |  |  |  |  |  |  |  |
| S1:DCD/RF |  |  |  |  |  |  |  |  |  |  |
| S9:Jmp | S9:`BNPC | S9 |  |  |  |  |  |  |  |  |
|  |  | S0:+4  S9:`JNPC | S0  S9 | S0 | S9:`PC | S9:`0x1F | S9 |  |  |  |  |

### 5)状态机和控制信号合成

#### (1)状态机合成

综合各指令的分状态机状态分支处需要增加判断条件。



#### (2)控制信号合成

将各指令的控制信号合成一张完整表格。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 控制信号 | NPCOp | PCWr | IRWr | RF\_WD\_sel  (MUX4\_RF\_WD) | RF\_A3\_sel  (MUX4\_RF\_A3) | RFWr | EXTOp | B\_sel  (MUX) | ALUOp | DMWr |
| addu rd,rs,rt | S0:+4 | S0 | S0 | S7:`ALUOut | S7:`RD | S7 |  | S6:`B | S6:`ALUOp\_ADDU |  |
| subu rd,rs,rt | S0:+4 | S0 | S0 | S7:`ALUOut | S7:`RD | S7 |  | S6:`B | S6:`ALUOp\_SUBU |  |
| ori rt,rs,imm16 | S0:+4 | S0 | S0 | S7:`ALUOut | S7:`RT | S7 | S1:`UE | S6:`EXT | S6:`ALUOp\_OR |  |
| lw rt,imm16(rs) | S0:+4 | S0 | S0 | S4:`DR | S4:`RT | S4 | S1:`SE | S2: `EXT | S2:`ALUOp\_ADD |  |
| sw rt,imm16(rs) | S0:+4 | S0 | S0 |  |  |  | S1:`SE | S2: `EXT | S2:`ALUOp\_ADD | S5 |
| beq rs,rt,imm16 | S0:+4  S8:`BNPC | S0  S8:Zero | S0 |  |  |  |  | S8: `B | S8:`ALUOp\_SUB |  |
| jal imm26 | S0:+4  S9:`JNPC | S0  S9 | S0 | S9:`PC | S9:`0x1F | S9 |  |  |  |  |

(3)状态控制信号表

综合控制信号合成表，获取每个周期的控制信号。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 状态 | NPCOp | PCWr | IRWr | RF\_WD\_sel  (MUX4\_RF\_WD) | RF\_A3\_sel  (MUX4\_RF\_A3) | RFWr | EXTOp | B\_sel  (MUX) | ALUOp | DMWr |
| S0:Fetch | +4 | 1 | 1 |  |  | 0 |  |  |  | 0 |
| S1:DCD/RF |  | 0 | 0 |  |  | 0 | ori:`UE  lw|sw:`SE |  |  | 0 |
| S2:MA |  | 0 | 0 |  |  | 0 |  | `EXT | `ALUOp\_ADD | 0 |
| S3:MR |  | 0 | 0 |  |  | 0 |  |  |  | 0 |
| S4:MemWB |  | 0 | 0 | `DR | `RT | 1 |  |  |  | 0 |
| S5:MW |  | 0 | 0 |  |  | 0 |  |  |  | 1 |
| S6:Exe |  | 0 | 0 |  |  | 0 |  | addu|subu:`B  ori:`EXT | addu:`ALUOp\_ADDU  subbu:`ALUOp\_SUBU  ori:`ALUOp\_OR | 0 |
| S7:ALUWB |  | 0 | 0 | `ALUOut | addu|subu:`RD  ori:`RT | 1 |  |  |  | 0 |
| S8:Branch | `BNPC | Zero | 0 |  |  | 0 |  | `B | `ALUOp\_SUB | 0 |
| S9:Jmp | `JNPC | 1 | 0 | `PC | `0x1F | 1 |  |  |  | 0 |

### 6)模块描述

`include "ctrl\_encode\_def.v"

`include "instruction\_def.v"

module ctrl(clk, rst, Zero, Op, Funct,

RFWr, DMWr, PCWr, IRWr,

EXTOp, ALUOp, NPCOp, RF\_A3\_sel,

RF\_WD\_sel, B\_sel);

input clk, rst, Zero;

input [5:0] Op;

input [5:0] Funct;

output RFWr;

output DMWr;

output PCWr;

output IRWr;

output [1:0] EXTOp;

output [1:0] ALUOp;

output [1:0] NPCOp;

output [1:0] RF\_A3\_sel;

output [1:0] RF\_WD\_sel;

output B\_sel;

parameter Fetch = 4'b0000,

DCD = 4'b0001,

Exe = 4'b0010,

MA = 4'b0011,

Branch = 4'b0100,

Jmp = 4'b0101,

MR = 4'b0110,

MW = 4'b0111,

WB = 4'b1000,

MemWB = 4'b1001;

wire RType; // Type of R-Type Instruction

wire IType; // Type of Imm Instruction

wire BrType; // Type of Branch Instruction

wire JType; // Type of Jump Instruction

wire LdType; // Type of Load Instruction

wire StType; // Type of Store Instruction

wire MemType; // Type pf Memory Instruction(Load/Store)

assign RType = (Op == `INSTR\_RTYPE\_OP);

assign IType = (Op == `INSTR\_ORI\_OP );

assign BrType = (Op == `INSTR\_BEQ\_OP );

assign JType = (Op == `INSTR\_JAL\_OP );

assign LdType = (Op == `INSTR\_LW\_OP );

assign StType = (Op == `INSTR\_SW\_OP );

assign MemType = LdType || StType;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\* FSM \*\*\*\*\*\*/

reg [3:0] nextstate;

reg [3:0] state;

always @(posedge clk or posedge rst) begin

if ( rst )

state <= Fetch;

else

state <= nextstate;

end // end always

always @(\*) begin//state transition

case (state)

Fetch: nextstate = DCD;

DCD: begin

if ( RType || IType )

nextstate = Exe;

else if ( MemType )

nextstate = MA;

else if ( BrType )

nextstate = Branch;

else if ( JType )

nextstate = Jmp;

else //if Op wrong, then fetch next one.

nextstate = Fetch;

end

Exe: nextstate = WB;

MA: begin

if ( LdType )

nextstate = MR; //LW

else if ( StType )

nextstate = MW; //SW

end

Branch: nextstate = Fetch;

Jmp: nextstate = Fetch;

MR: nextstate = MemWB;

MW: nextstate = Fetch;

WB: nextstate = Fetch;

MemWB: nextstate = Fetch;

default: ;

endcase

end // end always

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\* Control Signal \*\*\*\*\*\*/

reg RFWr;

reg DMWr;

reg PCWr;

reg IRWr;

reg [1:0] EXTOp;

reg [1:0] ALUOp;

reg [1:0] NPCOp;

reg [1:0] RF\_A3\_sel;

reg [1:0] RF\_WD\_sel;

reg B\_sel;

always @( \* ) begin//output

case ( state )

Fetch: begin

PCWr = 1'b1;

NPCOp = `NPC\_PLUS4;

IRWr = 1'b1;

RFWr = 1'b0;

DMWr = 1'b0;

end // end Fetch

DCD: begin

PCWr = 1'b0;

IRWr = 1'b0;

RFWr = 1'b0;

DMWr = 1'b0;

if (IType)

EXTOp = `EXT\_ZERO;

else if (MemType)

EXTOp = `EXT\_SIGNED;

else

EXTOp = 0;

end // end DCD

Exe: begin

PCWr = 1'b0;

IRWr = 1'b0;

RFWr = 1'b0;

DMWr = 1'b0;

DMWr = 0;

if (IType)

B\_sel = `EXT;

else

B\_sel = `B;

if (Op == `INSTR\_ORI\_OP)

ALUOp = `ALUOp\_OR;

else if (Op == `INSTR\_RTYPE\_OP) begin

case (Funct)

`INSTR\_ADDU\_FUNCT: ALUOp = `ALUOp\_ADDU;

`INSTR\_SUBU\_FUNCT: ALUOp = `ALUOp\_SUBU;

default: ;

endcase

end

end // end Exe

MA: begin

PCWr = 1'b0;

IRWr = 1'b0;

RFWr = 1'b0;

DMWr = 1'b0;

B\_sel = `EXT;

ALUOp = `ALUOp\_ADDU;

end // end MA

Branch: begin

PCWr=Zero;

NPCOp = `NPC\_BRANCH;

IRWr = 1'b0;

RFWr = 1'b0;

DMWr = 1'b0;

B\_sel = `B;

ALUOp = `ALUOp\_SUBU;

end // end Branch

Jmp: begin

PCWr = 1'b1;

NPCOp = `NPC\_JUMP;

IRWr = 1'b0;

RFWr = 1'b1;

DMWr = 1'b0;

RF\_A3\_sel = `RFSel\_31;

RF\_WD\_sel = `WDSel\_FromPC;

end // end Jmp

MR: begin

PCWr = 1'b0;

IRWr = 1'b0;

RFWr = 1'b0;

DMWr = 1'b0;

end // end MR

MW: begin

PCWr = 1'b0;

NPCOp = 0;

IRWr = 1'b0;

RFWr = 1'b0;

DMWr = 1'b1;

EXTOp = 0;

RF\_A3\_sel = 0;

RF\_WD\_sel = 0;

B\_sel = 0;

ALUOp = 0;

end // end MW

WB: begin

PCWr = 1'b0;

IRWr = 1'b0;

RFWr = 1'b1;

DMWr = 1'b0;

if (IType)

RF\_A3\_sel = `RFSel\_RT;

else

RF\_A3\_sel = `RFSel\_RD;

RF\_WD\_sel = `WDSel\_FromALU;

B\_sel = 0;

ALUOp = 0;

end // end WB

MemWB: begin

PCWr = 1'b0;

IRWr = 1'b0;

RFWr = 1'b1;

DMWr = 1'b0;

RF\_A3\_sel = `RFSel\_RT;

RF\_WD\_sel = `WDSel\_FromMEM;

end // end MemWB

default: begin

PCWr = 1'b0;

NPCOp = 0;

IRWr = 1'b0;

RFWr = 1'b0;

DMWr = 1'b0;

EXTOp = 0;

RF\_A3\_sel = 0;

RF\_WD\_sel = 0;

B\_sel = 0;

ALUOp = 0;

end // end default

endcase

end // end always

endmodule

## 4.测试

完成mips\_tb模块设计，实现对Test\_7\_Instr.asm程序的测试。

### 1)Test\_7\_Instr.asm是一个MIPS32汇编程序文件

# Test File for 7 Instruction, include:

# ADDU/SUBU/LW/SW/ORI/BEQ/JAL

################################################################

### Make sure following Settings :

# Settings -> Memory Configuration -> Compact, Data at address 0

.text

ori $29, $0, 12

ori $2, $0, 0x1234

ori $3, $0, 0x3456

addu $4, $2, $3

subu $6, $3, $4

sw $2, 0($0)

sw $3, 4($0)

sw $4, 4($29)

lw $5, 0($0)

beq $2, $5, \_lb2

\_lb1:

lw $3, 4($29)

\_lb2:

lw $5, 4($0)

beq $3, $5, \_lb1

jal F\_Test\_JAL # $31 change

# Never return

F\_Test\_JAL:

subu $6, $6, $2

sw $6, -4($29)

\_loop:

beq $3, $4, \_loop

# Never return back

### 2)mips\_tb模块

#### (1)基本描述

激励文件。初始化时钟周期等信号。

#### (2)模块描述

module mips\_tb();

reg clk, rst;

mips U\_MIPS(

.clk(clk), .rst(rst)

);

initial begin

$readmemh( "code.txt" , U\_MIPS.U\_IM.imem ) ;

$monitor("PC = 0x%8X, IR = 0x%8X", U\_MIPS.U\_PC.PC, U\_MIPS.instr );

clk = 1 ;

rst = 0 ;

#5 ;

rst = 1 ;

#20 ;

rst = 0 ;

end

always

#(50) clk = ~clk;

endmodule

**【试用Mars将Test\_7\_Instr.asm转换成生成16进制机器指令文件，用Verilog HDL建模方法对以上各模块进行建模，并通过mips\_tb进行仿真】**

# 五、实验思考

## 1.状态机设计通常没有唯一答案。

下图为两个均可行的状态机。状态机设计思路的主要差异在于在译码状态后，根据指令的性质设置了不同的状态分支。每位设计者的设计构思可能都不尽相同。请详细描述设计构思，特别是描述为什么要这样设计状态分支。

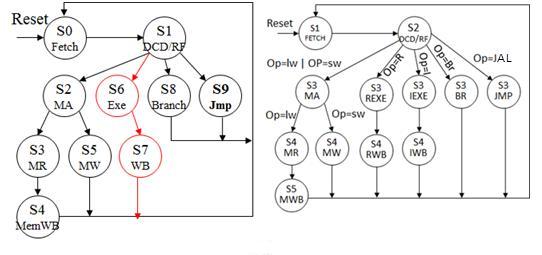


图 多周期控制器状态机参考设计

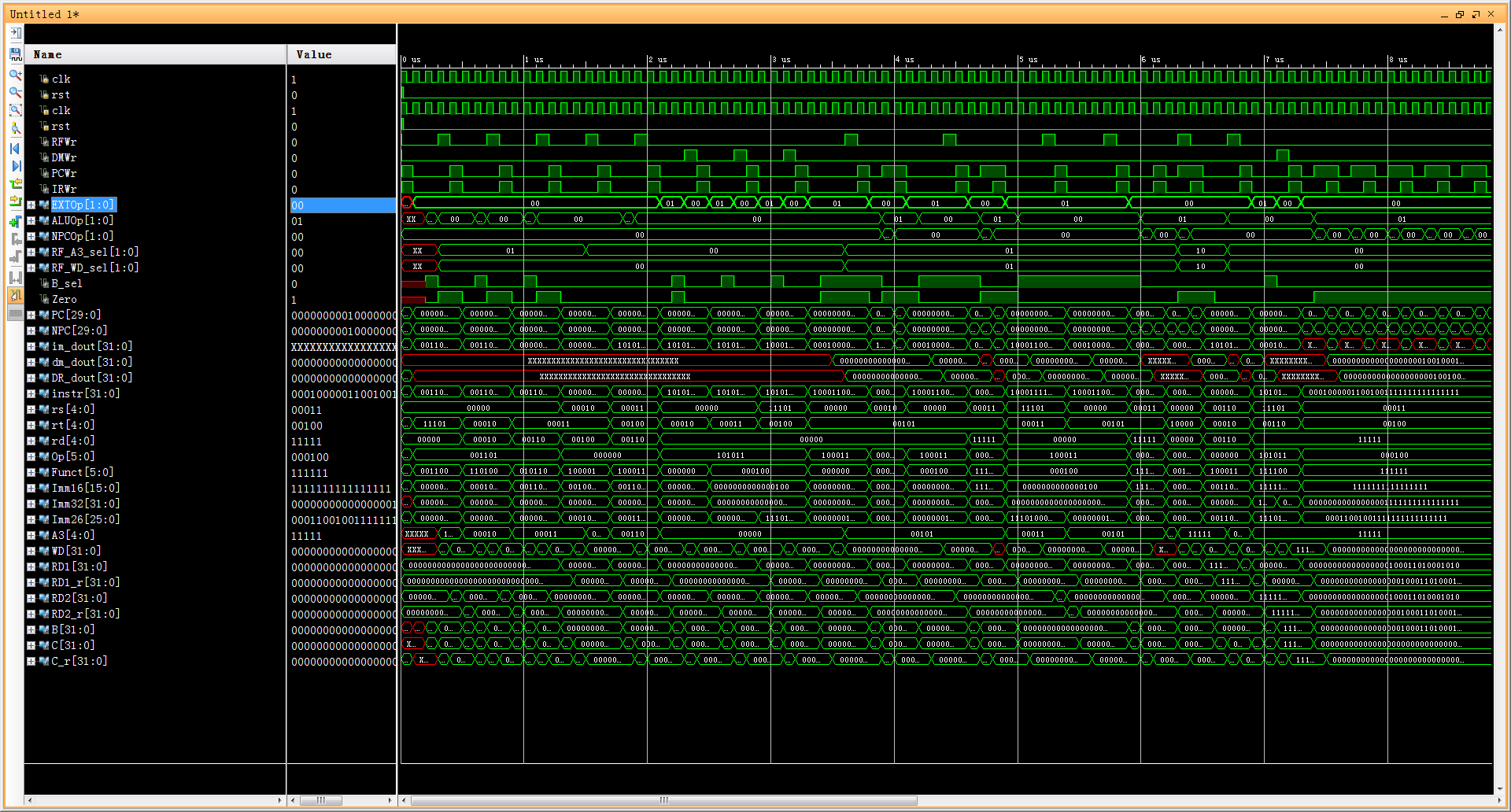
**【答】对于左图，在执行了取指令及指令译码并从寄存器取数后，若是分支指令或跳转指令，则直接执行继续取下一条指令。若是SW指令，则需要先得到写入数据的主存地址，再将数据写入存储器中。（S0->S1->S2->S5）。若是LW指令，则需要先得到数据的主存地址,再从该内存单元读出数据写入特定寄存器。（S0->S1->S2->S3->S4）。若是计算型指令，则在ALU单元中进行算术运算再将结果写入某个寄存器或内存单元（S0->S1->S6->S7）。简言之，左图状态机按照指令具体执行目的分类。**

**对于右图，在执行了取指令及指令译码并从寄存器取数后，若是分支指令或跳转指令，则直接执行继续取下一条指令。若是R型指令，先对RS和RT内容进行运算最终把ALU的计算结果送到目的寄存器RD（S1->S2->S3->S4）。若是I型指令，涉及对16位立即数进行符号扩展或零扩展，然后和RS的内容进行运算，最终把ALU的计算结果送目的寄存器RT（S1->S2->S3->S4）。显然，R型和I型指令功能段划分相同。若是SW指令，则需要先得到写入数据的主存地址，再将数据写入存储器中。（S1->S2->S3->S4）。若是LW指令，则需要先得到数据的主存地址,再从该内存单元读出数据写入特定寄存器。(S1->S2->S3->S4->S5)。简言之，右图按操作码分类。**

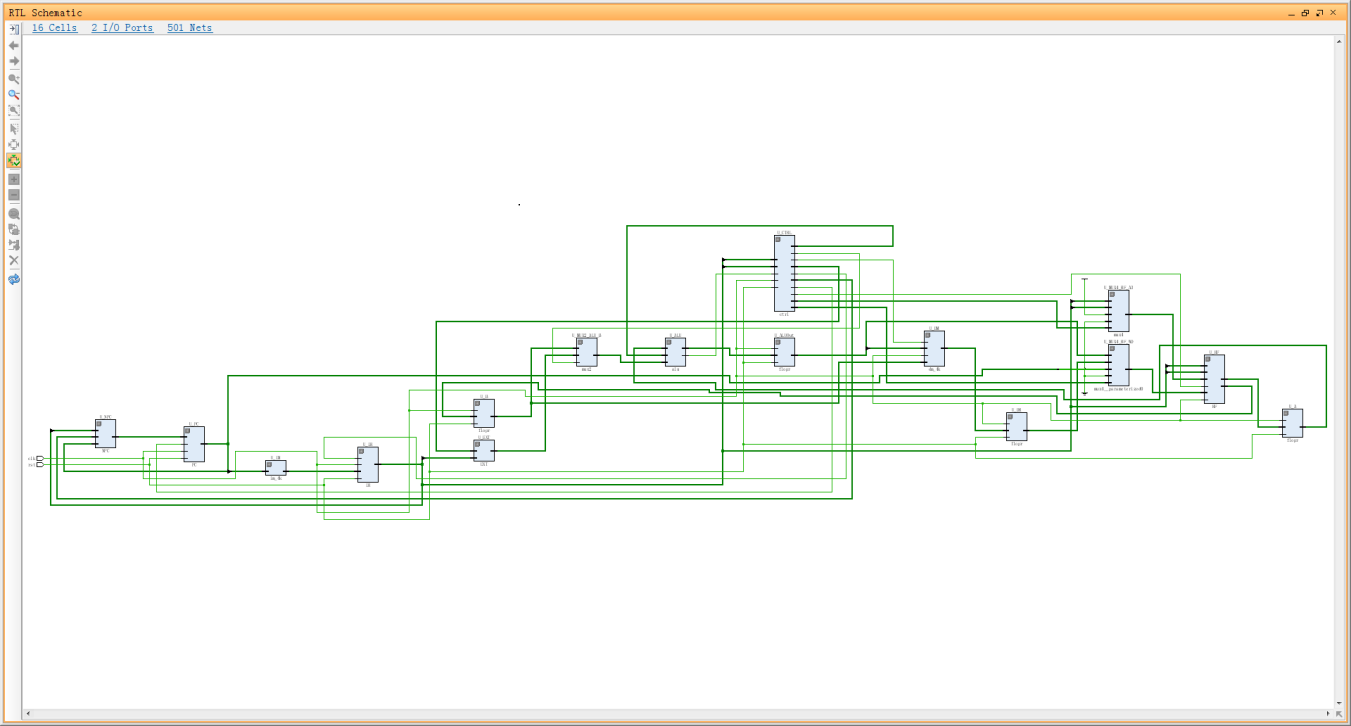
## 2.MIPS微处理器设计的关键工作是那两部分设计任务？

**【答】**

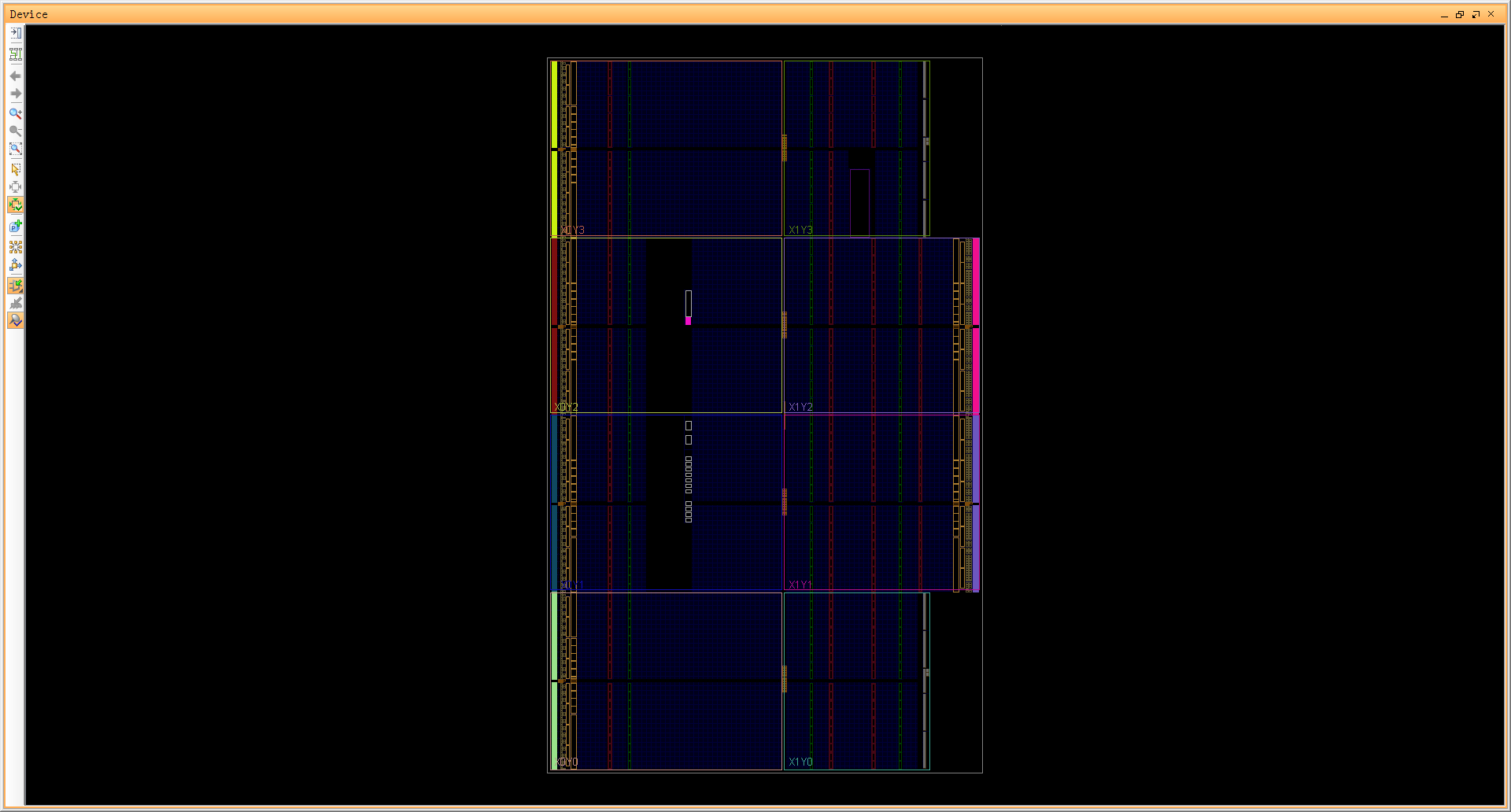
1. **数据通路设计即为mips模块设计，完成mips模块内部组件的实例化和信号连接。**
2. **控制器设计即为ctrl模块设计，按多周期方式实现每条指令各阶段对微处理器各组件的执行控制**。

仿真波形图：

RTL schematic:



Systhesis:



**小组分工：**

程舸帆：带领组员研读综合实验文档，对组员的工作进行分工，学习了R-型、I-型指令的完整执行过程，并分析不同状态机的设机构思。

徐思聪、谷梦园：学习综合实验文档，进行代码的研读与整合，得到MIPS处理器最后的仿真波形与综合图等。

赵翔、郑州：学习综合实验文档与store、Load-型指令，分支与跳转指令的完整执行过程。

**组员心得：**

程舸帆：通过这次期末综合实验，我不仅学习了mips处理器具体的实现过程，确实受益匪浅，更懂得了如何进行团队合作，根据每个组员不同个人情况及特长进行分工，力求将综合实验做到最好。具体到实验内容，在做这个实验前我也只是学习了一些课本上对于指令执行过程的解释，通过这个实验，我才深刻了解到多周期处理器的工作方式，每条指令的执行被分成若干阶段，每个阶段内完成一次读写或ALU操作，最后通过对内部组件实例化并连接信号，实现了处理器设计。

徐思聪：这次综合实验 让我更深刻的了解了cpu的工作方式，以及对mips微处理器的认识。在看完书上相关的图示后，让我第一次对微处理器的原理图，部件以及各类指令所对应的工作路线有了系统的了解 ，并通过这些了解来用virelog进行相关仿真，这对我们来说作用是非常大的。

谷梦园：经过这次的综合实验的训练,让我对原本模糊不清的指令系统的工作过程有了进一步得了解。我了解到了不管CPU有多复杂,都是由数据通路与控制部件两大部分组成,其中数据通路又包括了众多不同类型指令的构成与设计,而控制单元则起到指令译码与输出控制信号的作用,两者紧密有序得联合才能使整个指令系统有条不紊得工作。通过这次实验,我感受到了计算机内部指令体统严谨的构成与有序的工作,执行任何一条指令都是按部就班,这样才会减少出错的概率,我们平日里做事情也应该做到高效有序得完成任务。

赵翔：也没有做很多事，只是看见了自己很多不足。

郑州：这次的实验还是挺难的，平时的实验都没有认真完成，所以还是靠队友，平时的积累很重要。幸好还有这样的队友，不过不是每次都这么幸运，平时还是要学一些。