# Verilog流水线处理器

20373864 谭立德

## 一、CPU设计方案综述

### （一）总体设计概述

本CPU为verilog实现的流水线MIPS - CPU，支持的指令集包含MIPS-C4={LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、 SUB、 SUBU、 MULT、 MULTU、 DIV、 DIVU、 SLL、 SRL、 SRA、 SLLV、 SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、 XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、 BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO、MTC0、MFC0、ERET}。为了实现这些功能，CPU主要包含了IM、GRF、DM、ALU、PC、CU 等主要模块，这些模块按照自顶向下的顶层设计逐级展开。

### （二）关键模块定义

#### 1. GRF （通用寄存器组，也称为寄存器文件、寄存器堆）

GRF端口定义：

表0 GRF端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号，将32个寄存器中的值全部清零  1：复位  0：无效 |
| We | I | 写使能信号  1：可向GRF中写入数据  0：不可向GEF中写入数据 |
| A1 | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出至RD1 |
| A2 | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出至RD2 |
| A3 | I | 5位地址输入信号，指定32个寄存器中的一个作为写入的目标寄存器 |
| WD3 | I | 32位数据输入信号 |
| RD1 | O | 输出指定的寄存器中的32位数据 |
| RD2 | O | 输出指定的寄存器中的32位数据 |

GRF模块功能定义：

表1 GRF功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 复位 | Reset信号有效时，所有寄存器储存的数值清零 |
| 2 | 读数据 | 读出A1，A2地址对应寄存器中所储存的数据到RD1，RD2 |
| 3 | 写数据 | 当WE有效且时钟上升沿来临时，将WD写入A3所对应的寄存器中 |

1. DM （数据存储器）：

DM端口定义：

表2 DM端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号，将32个寄存器中的值全部清零  1：复位  0：无效 |
| We | I | 写使能信号  1：可向DM中写入数据  0：不可向DM中写入数据 |
| A | I | 5位地址输入信号，指定中储存器上的地址，将其中存储的数据读出至RD1 |
| WD | I | 32位数据输入信号 |
| RD | O | 输出储存器指定地址上的32位数据 |

DM模块功能定义：

表3 DM功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 复位 | Reset信号有效时，储存器储存的所有数值清零 |
| 2 | 读数据 | 读出A地址对应储存器中所储存的数据到RD |
| 3 | 写数据 | 当WE有效且时钟上升沿来临时，将WD写入A3所对应的寄存器中 |

#### ALU （算术逻辑运算单元）：

ALU端口定义：

表4 ALU端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| SrcA | I | 32位运算数输入信号 |
| SrcB | I | 32位运算数输入信号 |
| ALU Control | I | 3位逻辑运算选择信号，选择进行哪种逻辑运算 |
| Zero | O | 输出比较两运算数比较的1位输出 |
| ALU Result | O | 输出对两运算数进行指定逻辑运算后的32位结果 |

ALU模块功能定义：

表5 ALU功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 计算 | 根据控制信号进行对应的逻辑计算并输出 |
| 2 | 比较 | 判断两个输入是否相等 |

#### IM （指令存储器）：

IM端口定义：

表6 IM端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC | I | 5位输入地址信号 |
| Instr | 0 | 输出地址所储存32位指令 |

IM模块功能定义：

表7 IM功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 读指令 | 根据输入输出对应32位指令 |

#### Control Unit （指令译码器）：

Control Unit端口定义：

表8 Control Unit端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Opcode[5:0] | I | 指令操作码 |
| Funct[5:0] | I | 指令功能码 |
| Jump | O | 跳转信号 |
| ToHigh16 | O | 高位置位信号 |
| ExtOp | O | 位扩展方式 |
| MemtoReg | O | 读内存信号 |
| MemWrite | O | 内存写使能信号 |
| Branch | O | 分支信号 |
| ALUCtrl[2:0] | O | ALU控制信号 |
| ALUSrc | O | ALU操作数2的来源0：寄存器1：立即数 |
| RegDst | O | 寄存器写地址选择0：Instr[20:16] 1：Instr[15:11] |
| RegWrite | O | 寄存器写使能信号 |
| DMop[1:0] | O | 存储、读取方式控制信号 |

### （三）重要机制实现方法

#### 1. J类型指令

根据输入判断和ALU模块协同工作算出跳转地址后跳转。

#### 2. R类型指令

根据输入判断和ALU模块协同工作算出结果后存储回寄存器堆中以实现指令R类型指令。

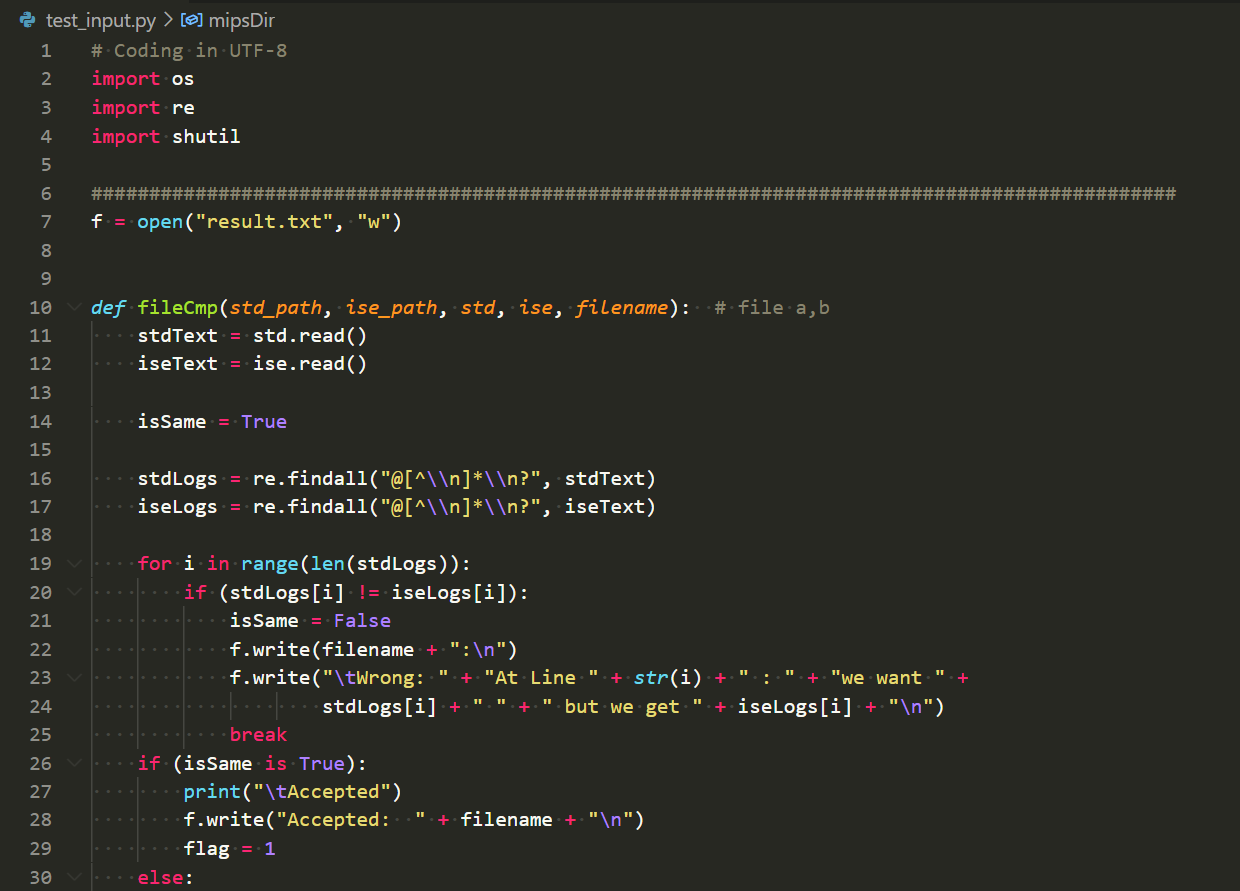
#### 3. I类型指令

根据输入判断和ALU模块和DM模块协同工作支持I类型指令。

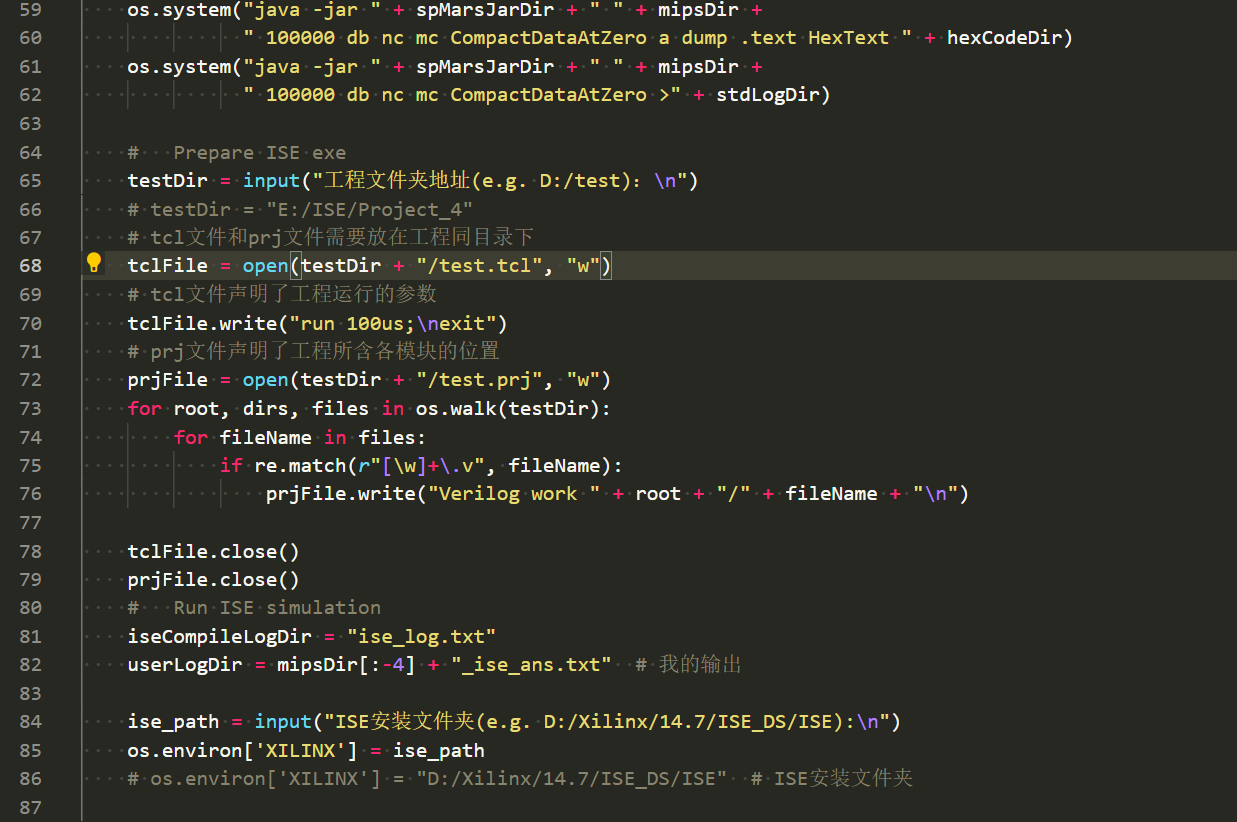
4.暂停&转发

见附页

## 测试方案









1. **CP0设计**

CP0 要干的事就是接收到中断异常时看看是否允许其发生，允许的话记录一下状态交给handler处理。

我们要实现 CP0 中的四个寄存器：SR，Cause，EPC，PrID。

SR 表示系统的状态，比如能不能发生异常

Cause 记录异常的信息，比如是否处于延迟槽以及异常的原因

EPC 记录发生异常的位置，便于处理完中断异常的时候返回

PrID 是一个可以随便定义的寄存器，表示你的 CPU 型号

SR只要实现一部分：SR[15:10]表示允许发生的中断；SR[1]表示是否处于中断异常中（是的话就不能发生中断异常）；SR[0]表示是否允许中断。Cause也只要实现一部分：Cause[31]表示延迟槽标记；Cause[15:10]表示发生了哪个中断；Cause[6:2]表示异常原因。 为了方便定义一些宏。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | `define IM SR[15:10]`define EXL SR[1]  `define IE SR[0]  `define BD Cause[31]  `define hwint\_pend Cause[15:10]  `define ExcCode Cause[6:2] |

异常和中断的条件：

|  |  |
| --- | --- |
| 1  2  3 | wire IntReq = (|(HWInt & `IM)) & !`EXL & `IE; // 允许当前中断 且 不在中断异常中 且 允许中断发生wire ExcReq = (|ExcCodeIn) & !`EXL; // 存在异常 且 不在中断中assign Req = IntReq | ExcReq; |

发生异常的处理方法：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | if (Req) begin // int|exc  `ExcCode <= IntReq ? 5'b0 : ExcCodeIn;  `EXL <= 1'b1;  EPCreg <= tempEPC;  `BD <= bdIn;end |

BD：如果异常发生在延迟槽，那么按照要求我们返回的时候要返回跳转指令。所以如果 BD 信号为真时应该输出上一条指令的 PC。

|  |  |
| --- | --- |
| 1  2  3  4 | wire [31:2] tempEPC = (Req) ? (bdIn ? PC[31:2]-1 : PC[31:2])  : EPCreg;  assign EPCout = {tempEPC, 2'b0}; |

每个时钟上升沿都要更新HWInt：

|  |  |
| --- | --- |
| 1 | `hwint\_pend <= HWInt; |

退出异常的条件是识别到了 eret，我们直接把 EXLClr 接上 M\_eret 就好

1. **Bridge与IO设计**

桥和DM

我们的 CPU 把 DM 中一块特殊的区域用来作为与外设交互的接口，中间通过桥来连接。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | wire selTC1 = (`RAddr >= `StartAddrTC1) && (`RAddr <= `EndAddrTC1),  selTC2 = (`RAddr >= `StartAddrTC2) && (`RAddr <= `EndAddrTC2);wire TCwe1 = selTC1 && PrWE,  TCwe2 = selTC2 && PrWE;wire [31:0] TCout1, TCout2;wire IRQ1, IRQ2;assign PrRD = selTC1 ? TCout1 :  selTC2 ? TCout2 :0;wire [5:0] HWInt = {3'b0, interrupt, IRQ2, IRQ1}; |

DM

注意 DM 写入的条件（WE 接口）为 M\_WE & (!req)。

内部 always@(posedge clk) 中的也要改（考虑到外设）。

|  |  |
| --- | --- |
| 1  2  3 | if (WE && (addr >= `StartAddrDM) && (addr <= `EndAddrDM)) begin  // ...end |

注意下一级寄存器（W\_reg）传入 DM 数据时要判断是否是外设的数据。

|  |  |
| --- | --- |
| 1  2  3  4  5 | W\_REG W\_reg(  // ...  .DM\_in((M\_ALUout >= 32'h0000\_7f00) ? PrRD : M\_DMout),  // ...); |

**(3)异常中断测试**

1.ADEL

(1).ktext 0x4180

mfc0 $k0, $14

addu $k0, $k0, 4

mtc0 $k0, $14

eret

.text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

lui $8, 0x7000

lui $9, 0xf000

lw $9,3($0)

sub $10, $8,$9

or $10, $8, $9

(2).ktext 0x4180

mfc0 $k0, $14

addu $k0, $k0, 4

mtc0 $k0, $14

eret

.text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

lui $8, 0x7000

lui $9, 0xf000

lh $9,3($0)

sub $10, $8,$9

or $10, $8, $9

(3).text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7fffffff

lui $9, 0x1

add $10, $8,$9

lw $a0,0x1000($8)

or $10, $8, $9

2.ADES

(1).text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7f00

lui $9, 0xf000

sw $9,3($0)

sub $10, $8,$9

or $10, $8, $9

(2).text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7f00

lui $9, 0xf000

sh $9,1($0)

sub $10, $8,$9

or $10, $8, $9

3.RI

在其他测试的机器码中插入ffffffff

4.Ov

.ktext 0x4180

mfc0 $k0, $14

sub $8,$8,$8

mtc0 $k0, $14

eret

.text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

lui $8, 0x7fff

lui $9, 0x7fff

add $10, $8,$9

or $10, $8, $9

.text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

lui $8, 0x7fff

lui $9, 0x7fff

addi $10, $8,0x7fff0000

or $10, $8, $9

.text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

lui $8, 0x7fff

ori $8, $8, 0xffff

addi $10, $8, 1

ori $a0,$0,100

.text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

lui $8, 0x7000

lui $9, 0xf000

sub $10, $8,$9

or $10, $8, $9

.text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7f00

lui $9, 0xf000

lw $9,0($8)

sub $10, $8,$9

or $10, $8, $9

.text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7f00

lui $9, 0xf000

sh $9,4($8)

sub $10, $8,$9

or $10, $8, $9

5.延迟槽

（1）.ktext 0x4180

mfc0 $1,$13

sub $9,$9,$9

eret

.text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7fffffff

ori $9, 0x1000

j eee

add $10, $8,$9

lw $a0,0x1000($8)

eee:

or $10, $8, $9

(2) .ktext 0x4180

mfc0 $1,$13

sub $9,$9,$9

eret

.text

ori $8, 0x7fffffff

ori $9, 0x1000

ori $t1 0x00007f00

ori $a0,0x0009

ori $a3,0xfc01

beq $9,$8,eee

add $10,$8,$9

ori $a1,2

sw $a1,4($t1)

eee:

sw $a0,0($t1)

（3）.ktext 0x4180

mfc0 $1,$13

sub $9,$9,$9

eret

.text

ori $8, 0x7fff0000

ori $9, 0x7fff0000

ori $a0,0x0009

ori $a3,0xfc01

beq $9,$8,eee

add $10,$8,$9

ori $a1,2

sw $a1,4($t1)

eee:

sw $a0,0($t1)

（4）.ktext 0x4180

mfc0 $1,$13

sub $8,$8,$8

sub $9,$9,$9

eret

.text

ori $8, 0x7fff0000

ori $9, 0x7fff0000

ori $a0,0x0009

ori $a3,0xfc01

beq $9,$8,eee

add $10,$8,$9

ori $a1,2

sw $a1,4($t1)

eee:

sw $a0,0($t1)

（5）.ktext 0x4180

mfc0 $1,$13

sub $9,$9,$9

eret

.text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7fffffff

ori $9, 0x1000

j eee

add $10, $8,$0

lw $a0,0x1000($8)

eee:

or $10, $8, $9

j end

add $10,$8,$9

end:

ori $a0,$0,0

(6) .ktext 0x4180

mfc0 $1,$13

sub $9,$9,$9

eret

.text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7fffffff

ori $9, 0x1000

j eee

add $10, $8,$0

lw $a0,0x1000($8)

eee:

or $10, $8, $9

j end

sh $1,1($0)

end:

ori $a0,$0,0

(7) .text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7fffffff

ori $9, 0x1000

j eee

add $10, $8,$0

lw $a0,0x1000($8)

eee:

or $10, $8, $9

j end

sw $1,1($0)

end:

ori $a0,$0,0

(8) .text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7fffffff

ori $9, 0x1000

j eee

add $10, $8,$0

lw $a0,0x1000($8)

eee:

or $10, $8, $9

j end

lh $1,1($0)

end:

ori $a0,$0,0

(9) .text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7fffffff

ori $9, 0x1000

j eee

add $10, $8,$0

lw $a0,0x1000($8)

eee:

or $10, $8, $9

j end

lw $1,1($0)

end:

ori $a0,$0,0

(10).text

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $8, 0x7fffffff

ori $9, 0x1000

j eee

add $10, $8,$0

lw $a0,0x1000($8)

eee:

or $10, $8, $9

j end

lhu $1,1($0)

end:

ori $a0,$0,0

6.中断

（1）.ktext 0x4180

mfc0 $1,$13

sub $9,$9,$9

mtc0 $0,$12

eret

.text

ori $8, 0x7fffffff

ori $9, 0x1000

ori $t1 0x00007f00

ori $a0,0x0009

ori $a3,0xfc01

ori $a1,2

sw $a1,4($t1)

sw $a0,0($t1)

mtc0 $a3,$12

or $10, $8, $9

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $28, $0, 0x0010

ori $29, $0, 0x0111

nop

nop

nop

（2）.ktext 0x4180

mfc0 $1,$13

sub $9,$9,$9

mtc0 $0,$12

eret

.text

ori $8, 0x7fffffff

ori $9, 0x1000

ori $t1 0x00007f00

ori $a0,0x0009

ori $a3,0xfc01

ori $a1,2

sw $a1,4($t1)

sw $a0,0($t1)

add $11,$8,$9

mtc0 $a3,$12

or $10, $8, $9

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $28, $0, 0x0010

ori $29, $0, 0x0111

nop

nop

nop

（3）.ktext 0x4180

mfc0 $1,$13

sub $9,$9,$9

sub $29,$29,$29

mtc0 $0,$12

ori $v1,$0,0x00007f00

sw $a3,0($v1)

eret

.text

ori $8, 0x7fffffff

ori $9, 0x1000

add $11,$8,$9

ori $t1 0x00007f00

ori $a0,0x0009

ori $a3,0xfc01

ori $a1,1

sw $a1,4($t1)

sw $a0,0($t1)

mtc0 $a3,$12

nop

ori $a0,1111

j eee

ori $29, $0, 0x0111

eee:

or $10, $8, $9

ori $28, $0, 0x0000

ori $29, $0, 0x0000

ori $29,0x1000

lui $v1,1

j end

add $10,$8,$29

end:

lui $a0,1

lui $a2,2

7.乘除相关

（1）.ktext 0x4180

mfc0 $1,$13

sub $a0,$a0,$a0

eret

.text

ori $8, 0x7fffffff

ori $9, 0x1000

div $8,$9

lui $a0,0x7f00

add $a0,$a0,$a0

mthi $a0

mflo $s7

ori $a1,1

ori $v0,1

addu $a1,$v0,$v0

nop

ori $s0,11

（2）.ktext 0x4180

mfc0 $1,$13

mtc0 $0,$12

mflo $s2

mfhi $s1

ori $v1,$0,0x00007f00

sw $a3,0($v1)

eret

.text

ori $t1,0x00007f00

ori $a0,0x0009

ori $a3,0xfc01

ori $a1,3

sw $a1,4($t1)

sw $a0,0($t1)

mtc0 $a3,$12

mult $t1,$a3

mthi $a3

mfhi $a2

ori $29,0x1000

lui $v1,1

（3）mult 中断

.ktext 0x4180

mfc0 $1,$13

mtc0 $0,$12

mflo $s2

mfhi $s1

ori $v1,$0,0x00007f00

sw $a3,0($v1)

eret

.text

ori $t1,0x00007f00

ori $a0,0x0009

ori $a3,0xfc01

ori $a1,3

sw $a1,4($t1)

sw $a0,0($t1)

mtc0 $a3,$12

mult $t1,$a3

mthi $a3

mfhi $a2

ori $29,0x1000

lui $v1,1

(4) .ktext 0x4180

mfc0 $1,$13

mtc0 $0,$12

mflo $s2

mfhi $s1

ori $v1,$0,0x00007f00

sw $a3,0($v1)

eret

.text

ori $t1,0x00007f00

ori $a0,0x0009

ori $a3,0xfc01

ori $a1,4

sw $a1,4($t1)

sw $a0,0($t1)

mtc0 $a3,$12

mult $t1,$a3

mthi $a3

mfhi $a2

ori $29,0x1000

lui $v1,1

(5) .ktext 0x4180

mfc0 $1,$13

mtc0 $0,$12

mflo $s2

mfhi $s1

ori $v1,$0,0x00007f00

sw $a3,0($v1)

eret

.text

ori $t1,0x00007f00

mult $t1,$t1

ori $a0,0x0009

ori $a3,0xfc01

ori $a1,3

sw $a1,4($t1)

sw $a0,0($t1)

mtc0 $a3,$12

mult $t1,$a3

#mthi $a3

#mfhi $a2

ori $29,0x1000

lui $v1,1

（6）.ktext 0x4180

mfc0 $1,$13

mtc0 $0,$12

mflo $s2

mfhi $s1

ori $v1,$0,0x00007f00

sw $a3,0($v1)

eret

.text

ori $t1,0x00007f00

mult $t1,$t1

ori $a0,0x0009

ori $a3,0xfc01

ori $a1,8

sw $a1,4($t1)

sw $a0,0($t1)

mtc0 $a3,$12

ori $29,0x1000

mthi $a3

ori $a0,0x000b

sw $a0,0($t1)

mthi $a3

mfhi $a2

ori $29,0x1000

mult $t1,$t1

div $a2,$a2

lui $a0,1

110@00003000: $ 9 <= 00007f00

150@00003008: $ 4 <= 00000009

170@0000300c: $ 7 <= 0000fc01

190@00003010: $ 5 <= 00000008

270@00003020: $29 <= 00001000

310@00003028: $ 4 <= 0000000b

370@00003034: $ 6 <= 0000fc01

390@00003038: $29 <= 00001000

550@00004180: $ 1 <= 00000000

590@00004188: $18 <= 3f010000

610@0000418c: $17 <= 00000000

630@00004190: $ 3 <= 00007f00

870@00003044: $ 4 <= 00010000

## 三、思考题

### 1、我们计组课程一本参考书目标题中有“硬件/软件接口”接口字样，那么到底什么是“硬件/软件接口”？(Tips：什么是接口？和我们到现在为止所学的有什么联系？)

解：

硬件

目标：CPU与外设(被控对象)在硬件上连接构成一个有机整体

方法：I/0接口电路(接口、接口控制器)

软件

目标：控制设备工作方式，完成信息传送

方法：接口控制程序(或驱动程序)

硬软件接口就是计算机硬件与软件的交互手段，人类与电脑等信息机器或人类与程序之间的接口称为用户界面。电脑等信息机器硬件组件间的接口叫硬件接口。电脑等信息机器软件组件间的接口叫软件接口。硬件接口指的是两个硬件设备之间的连接方式。硬件接口既包括物理上的接口，还包括逻辑上的数据传送协议。软件不同部分之间的交互接口。通常就是所谓的API――应用程序编程接口，其表现的形式是源代码。

### 2、BE 部件对所有的外设都是必要的吗？

解：

不是，timer只能用lw,sw,用不到BE了。

### 3、请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并分别针对每一种模式绘制状态转移图。

解：

异：

模式0计时结束后，一直保持中断，直到en或IM被修改

模式1计时结束后，中断一个周期，再重新计数

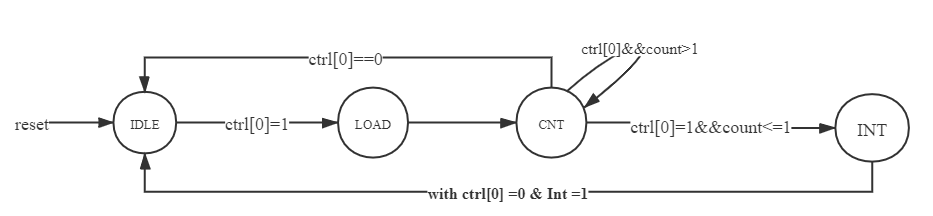
可以理解为中断保持的逻辑不同

同：

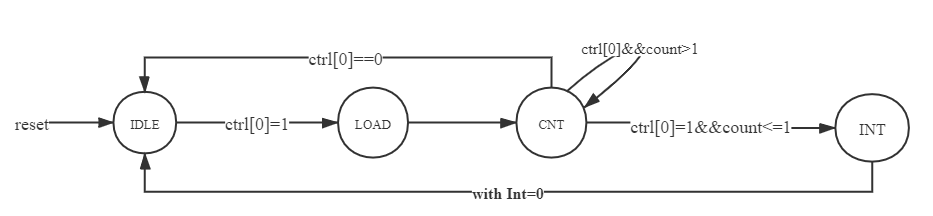
都在CNT到0的时候产生了中断

2.绘制状态转移图

模式0：



模式1：



### 4、请开发一个主程序以及定时器的 exception handler。整个系统完成如下功能：

### （1）定时器在主程序中被初始化为模式 0；

### （2）定时器倒计数至 0 产生中断；

### （3）handler 设置使能 Enable 为 1 从而再次启动定时器的计数器。(2) 及 (3) 被无限重复。

### （4）主程序在初始化时将定时器初始化为模式 0，设定初值寄存器的初值为某个值，如 100 或 1000。(注意，主程序可能需要涉及对 CP0.SR 的编程，推荐阅读过后文后再进行。)

解：

.ktext 0x4180

mfc0 $1,$13

mflo $s2

mfhi $s1

ori $v1,$0,0x00007f00

sw $a0,0($v1)

eret

.text

ori $t1,0x00007f00

mult $t1,$t1

ori $a0,0x0009

ori $a3,0xfc01

ori $a1,2

sw $a1,4($t1)

sw $a0,0($t1)

mtc0 $a3,$12

mult $t1,$a3

ori $29,0x1000

lui $v1,1

lui $v0,2

lui $s1,3

lui $s0,4

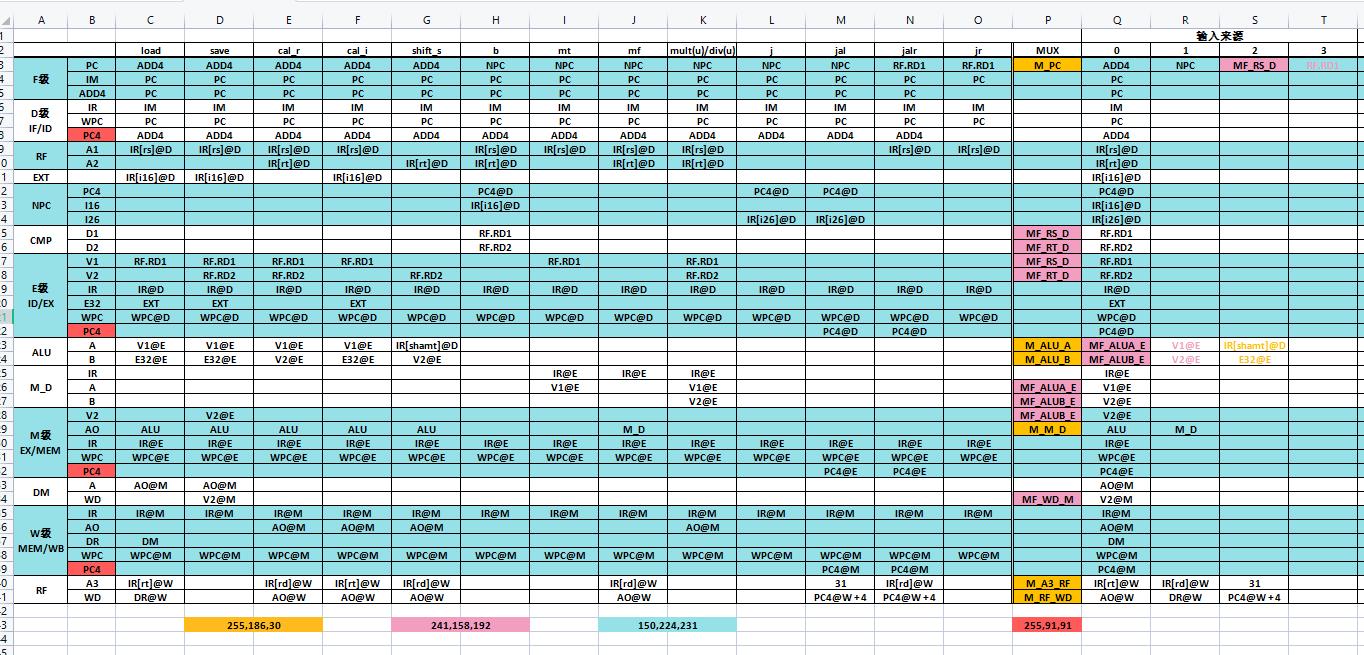
### 5、请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的？

解：

设备实际上包括两部分接口控制器（也称为接口芯片）和设备主体，设备主体不直接与主机连接，而是通过接口控制器与主机连接。鼠标和键盘的输入信号相当于中断，当键盘、鼠标有信息时，产生一个中断然后中断例程会从端口读入数据到寄存器。CPU接收到中断请求之后进入中断处理程序获得鼠标键盘的信息。

附1：

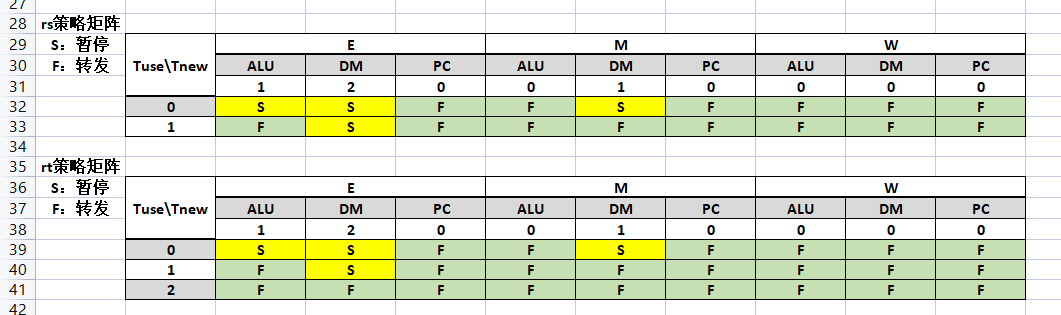
数据通路

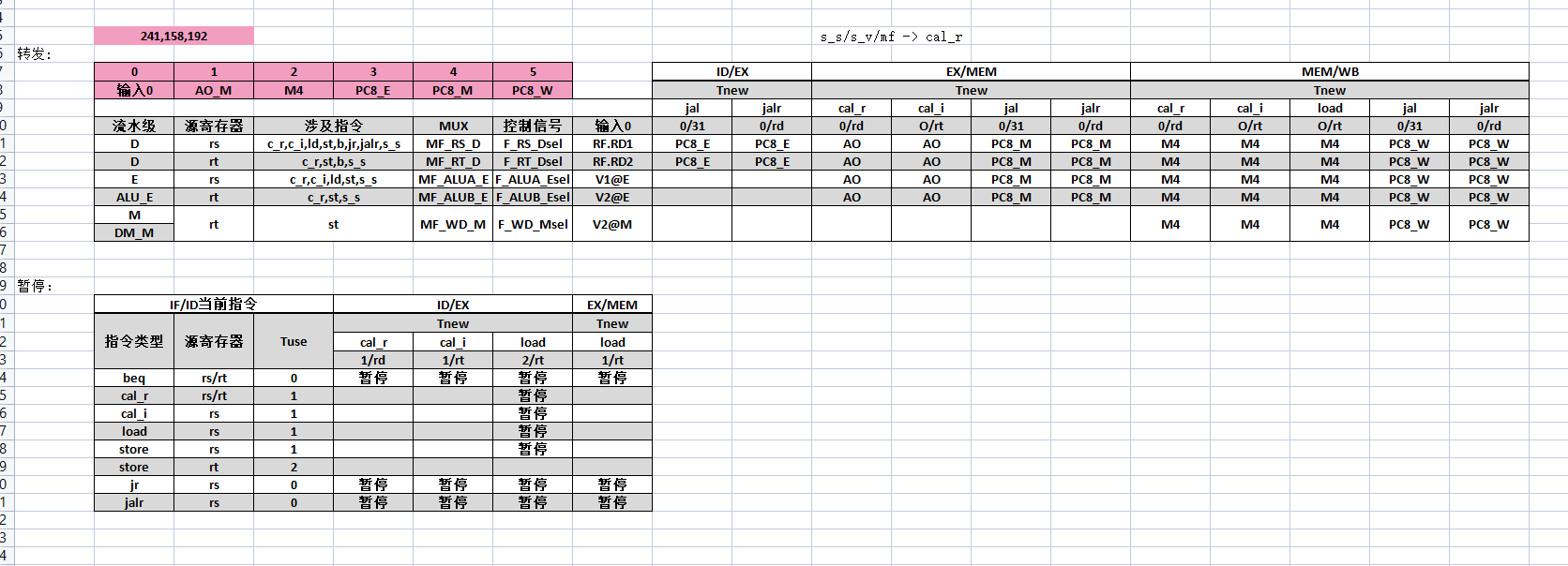


附2：

冲突策略矩阵

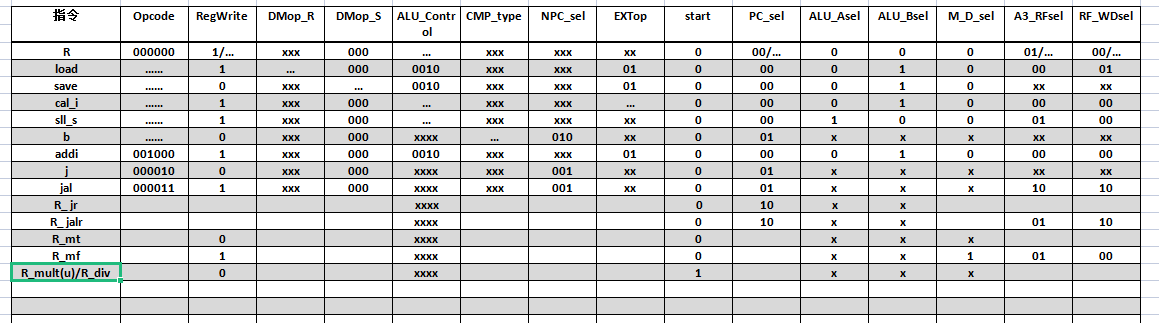


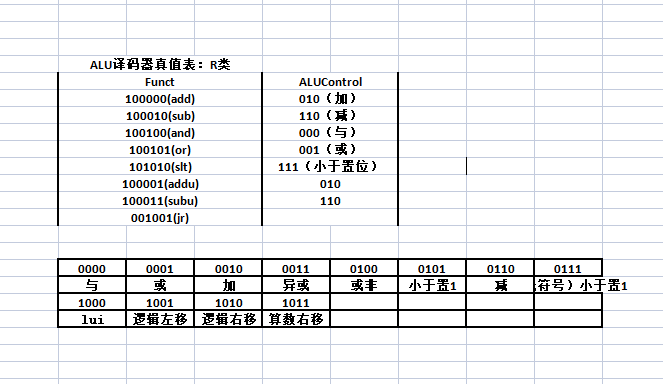




附3：

控制器





附4：

多路选择器控制信号

