`思考题

1. 什么是硬件软件接口

指令集体系结构就是低层次软件和硬件之间的抽象接口，他可让程序员在不直接操纵硬件的前提下，就编写出可以让硬件实现想要达到的目的的正确的程序，包括指令，寄存器，访存和IO。

1. 在现代计算机中，DM和其他外设一样应该位于CPU外部（~~不然我上个月咋安的内存条）~~，通过总线控制器连接到CPU供其访问。
2. 不是。因为不是所有外设都要求支持按字节访问
3. .text

ori $s0,9#低四位分别是1001

ori $s1,1000

sw $s0,0x7f10($0)

sw $s0,0x7014($0)

loop:#开始死循环

beq $s0,$s0,loop

nop

.ktext 0x0004180

ori $s0,9

sw $s0,0x7f10($0)

eret

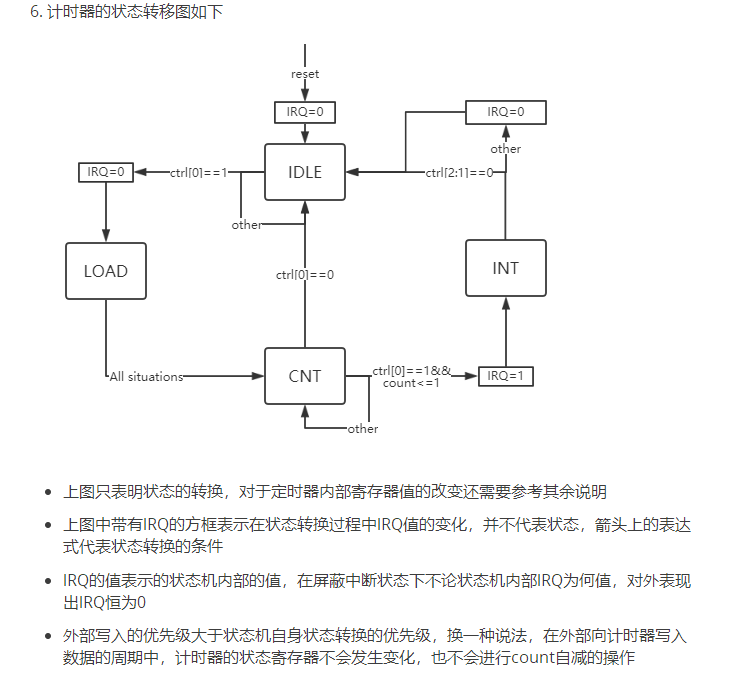
1. 键盘在按键被按下或者抬起时都会发生中断，CPU读取对应键盘端口的寄存器来得知发生了什么事件，通常如果一个按键被按下CPU读取到的扫描码是X那么该按键抬起时CPU读到的就是X+0X80 鼠标同理

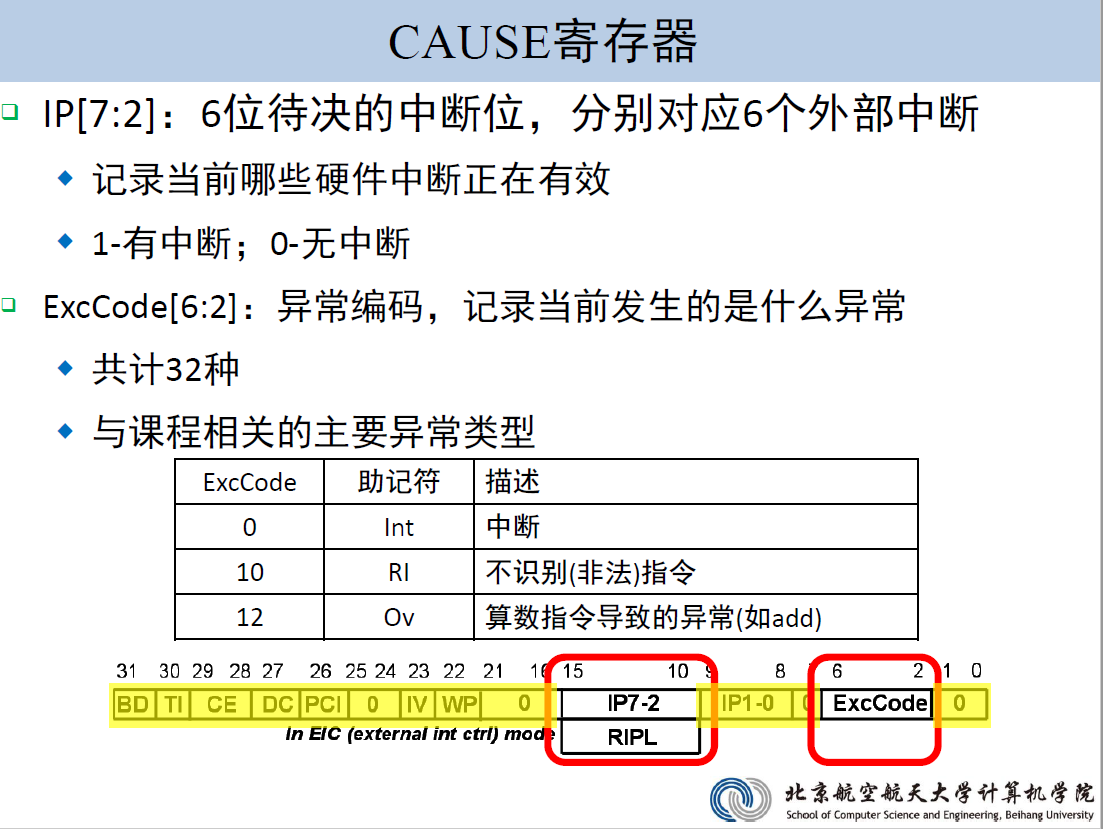
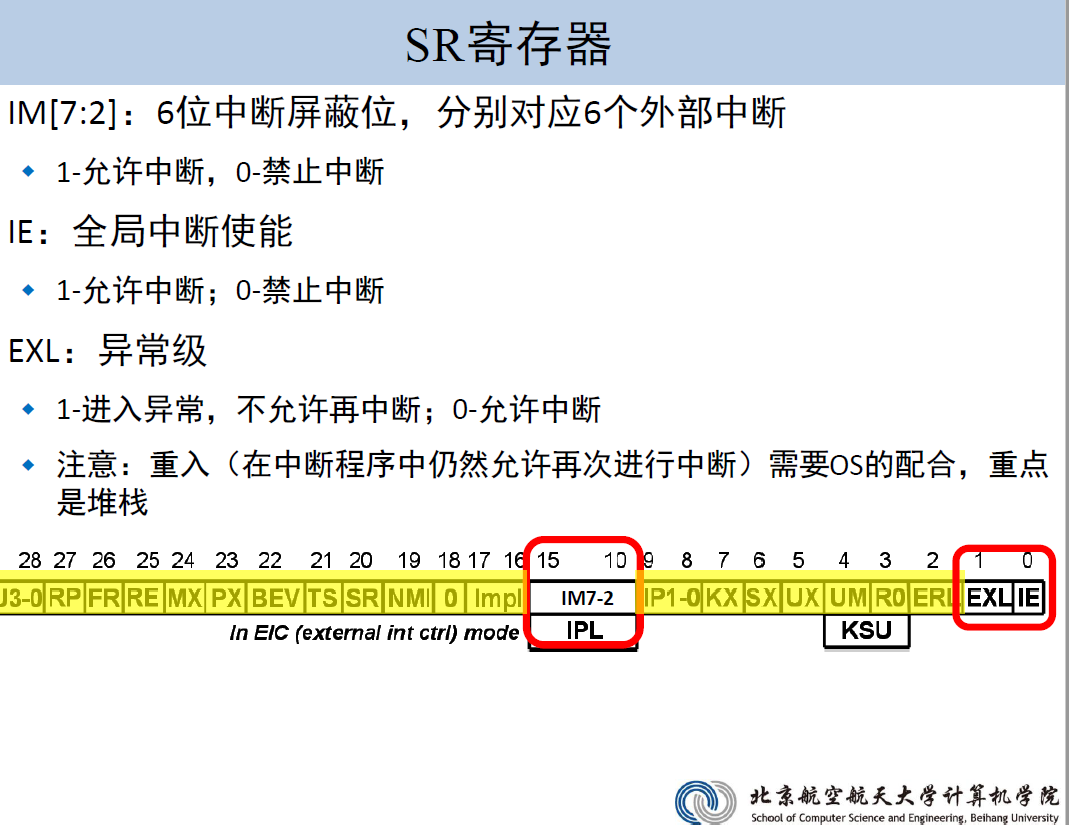
P7 CPU文档

~~（P6/7 工程化写bug报告）~~

数据通路设计：







正常各流水级部件

1. IFU

|  |  |  |
| --- | --- | --- |
| 接口名称 | 类型 | 描述 |
| clk | I | 时钟信号 |
| clr | I | 复位信号 |
| branch | I | 跳转信号，为1时有效 |
| npcout[31:0] | I | 输入在跳转指令时需要跳转的地址 |
| stall | I | 暂停信号 |
| instr | O | 输出取出的指令 |
| pc8 | O | 输出pc+8的值 |
| intreq | I | 中断信号 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当clr信号有效时pc被置为0x00003000 |
| 2. | 取指令 | 取出pc对应地址里的指令 |
| 3 | 生成下一指令地址 | 若branch有效，下一指令地址为npc  否则为pc+4 |
| 4 | stall响应 | 若stall有效，锁定pc寄存器使其值不再改变 |
| 5. | intreq响应 | 若intreq有效跳转至npc（此时是0x00004180） |

1. grf

|  |  |  |
| --- | --- | --- |
| 接口名 | 类型 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 异步复位信号，1为有效 |
| we | I | 写使能信号，1为有效 |
| a1[4:0] | I | RD1端口输出的寄存器的编号 |
| a2[4:0] | I | RD2端口输出的寄存器的编号 |
| a3[4:0] | I | 要写入的寄存器的编号 |
| wd | I | 要写入选定寄存器的数据 |
| rd1 | O | 输出A1端口选中寄存器的值(进行转发) |
| rd2 | O | 输出A2端口选中寄存器的值（进行转发） |

功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | reset信号有效时，所有寄存器的值置为0 |
| 2 | 读寄存器 | RD1,RD2输出A1,A2选中寄存器的值 |
| 3 | 写寄存器 | we信号有效时A3选中的寄存器的值被写为WD的值 |

1. npc

|  |  |  |
| --- | --- | --- |
| 接口名 | 类型 | 描述 |
| npcop[1:0] | I | 功能选择  00 依据Imm字段计算下一指令地址  01 依据指令后26位与pc+4高四位计算地址  02 输出rd1中的地址  03 0x00004180  04 eret执行时需返回的地址 |
| pc4 | I | 输入pc+4 |
| extout | I | 输入imm进行16位扩展后的信号 |
| rd1 | I | 输入gpr 中a1选定寄存器的值 |
| intsr | I | 输入当前正在执行的指令 |
| npcout | O | 输出计算出的下一指令 |
| intreq | I | 中断信号，若有效npcot为0x4180 |
| epc | I | eret跳转地址 |

功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 实现立即数跳转 | 依据Imm字段计算下一指令地址 |
| 2 | 实现j类型跳转 | 依据指令后26位与pc+4高四位计算地址 |
| 3 | 实现寄存器跳转 | 输出rd1中的地址 |
| 4 | 实现handler跳转 | 输出0x4180 |
| 5 | 实现eret跳转 | 输出epc的值（epc值已经过显式转发） |

1. ext

|  |  |  |
| --- | --- | --- |
| 接口名 | 类型 | 描述 |
| imm[15:0] | I | 输入16位待扩展立即数 |
| extout[31:0] | O | 输出扩展后32位数据 |
| extop[1:0] | I | 功能选择  00无符号扩展，01有符号扩展，02Lui扩展 |

功能说明

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 无符号扩展 | 高位补16个0 |
| 2 | 有符号扩展 | 高位补16个符号位 |
| 3 | Lui扩展 | 低位补16个0 |

1. cmp

|  |  |  |
| --- | --- | --- |
| 接口名 | 类型 | 描述 |
| [31:0]a | I | 输入待比较数a |
| [31:0] | I | 输入待比较数b |
| [31:0]zero32 | O | 输出zeroextent(a==b) |
| gtz | O | 输出a>0 |
| gez | O | 输出a>=0 |
| ltz | O | 输出a<0 |
| lez | O | 输出a<=0 |

功能说明

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 相等判断 | 若a==b时有效 |

1. alu

|  |  |  |
| --- | --- | --- |
| 接口名 | 类型 | 描述 |
| a[31:0] | I | 输入alu的第一个操作数 |
| b[31:0] | I | 输入alu的第二个操作数 |
| aluop[1:0] | I | alu功能选择  00输出b，01为加运算，10为减运算，11为或运算， |
| aluout[31:0] | O | 输出运算结果 |
| overflow | O | 输出有无溢出 |

功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加运算 | 输出a1+a2 |
| 2 | 减运算 | 输出a1-a2 |
| 3 | 或运算 | 输出a1|a2 |
| 4 | 输出b | 为提高lui等指令的性能并降低复杂度而预留 |

1. dm

|  |  |  |
| --- | --- | --- |
| 接口名 | 类型 | 描述 |
| addr[31:0] | I | DM输入的地址 |
| wd[31:0] | I | DM要写入的数据 |
| clk | I | 时钟信号 |
| clr | I | 异步复位信号 |
| we | I | 写使能信号，1为有效，将A端口选中的字写为INPUT中数据 |
| dmout[31:0] | O | 输出addr端口选中的字的值 |
| be | I | 输入字节使能 |

功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | clr信号有效时，所有主存的值置为0 |
| 2 | 读主存 | 输出addr端口选中的字的值 |
| 3 | 写主存 | we有效时将addr端口选中的字写为wd中数据 |

异常处理单元

1. Exc\_i

|  |  |  |
| --- | --- | --- |
| 接口名 | 方向 | 描述 |
| instr\_d | I | d级指令 |
| pc8\_i | I | I级指令的pc8值 |
| bd\_i | O | branch delayed信号，若为延迟槽指令则为1 |
| exccode\_i | O | 传递异常代码 |

功能说明

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 判断取指异常 | 若pc越界或不对齐报adel异常 |
| 2 | 延迟槽指令判断 | 若为延迟槽指令则置1 |

1. Exc\_d

|  |  |  |
| --- | --- | --- |
| 接口名 | 方向 | 描述 |
| instr\_d | I | 输入d级指令 |
| exccode\_i | I | 输入i级传递的exccode |
| bd\_i | I | 输入是否延迟槽指令 |
| exccode\_d | O | 输出exccode |
| bd\_d | O | 输出bd |

功能说明

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 检查不识别指令 | 若指令未被识别则报ri异常 |
| 2 | 传递前级异常 | 若本级无异常则传递前级exccode |

1. exc\_e

|  |  |  |
| --- | --- | --- |
| 接口名 | 方向 | 描述 |
| instr\_e | I | 输入e级指令 |
| overflow | I | 输入overflow判断结果 |
| bd\_d | I | 输入前级bd |
| exccode\_d | I | 输入前级exccode |
| bd\_e | O | 输出bd |
| exccode\_e | O | 输出exccode |

功能说明

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 检查算数溢出 | 若add sub算数溢出报ov异常 |
| 2 | 传递前级异常 | 若本级无异常则传递前级exccode |

1. exc\_m

|  |  |  |
| --- | --- | --- |
| 接口名 | 方向 | 描述 |
| instr\_m | I | 输入m级指令 |
| ao\_m | I | 输入m级地址总线 |
| exccode\_e | I | 输入前级exccode |
| bd\_e | I | 输入前级bd |
| exccode\_m | O | 输出exccode |
| bd\_m | O | 输出bd |

功能说明

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 判断读/写异常 | 若读写地址不对齐/地址越界/写不该写的地址  报对应的异常 |
| 2 | 传递前级异常 | 若本级无异常则传递前级exccode |

协处理器单元0

|  |  |  |
| --- | --- | --- |
| 接口名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| clr | I | 复位信号 |
| a1 | I | 读寄存器编号 |
| a2 | I | 写寄存器编号 |
| wdbus | I | 写入数据 |
| pc8\_m | I | 写入m级异常/中断受害指令地址 |
| bdin | I | 写入m级异常/中断受害指令是否为延迟槽指令 |
| hwint | I | 输入6个外部设备中断信号 |
| exccodein | I | 输入m级异常/中断编号 |
| we | I | CP0寄存器写使能 |
| exlclr | I | 关中断使能 |
| cp0\_rd | O | cp0读出数据 |
| cp0\_epc | O | epc寄存器的值 |
| intreq | O | 中断信号 |

功能说明

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 复位 | 若clr信号有效实施复位操作 |
| 2 | 外部中断响应 | 若外部中断到来且全局允许中断且不在中断时  保存epc值为pc\_m(pc\_m-4若为延迟槽)  保存cause寄存器的值  exl置位为1  cause exccode段置位00000 |
| 3 | 异常响应 | 若m级指令异常且全局允许中断且不在中断时  保存epc值为pc\_m(pc\_m-4若为延迟槽)  保存cause寄存器的值  exl置位为1 |
| 4 | 关中断信号 | 若有效exl置位0 |
| 5 | 写CP0寄存器 | 若写使能有效，写入对应编号的寄存器 |
| 6 | 读CP0寄存器 | 输出对应编号的寄存器值 |

Bridge设计

|  |  |  |
| --- | --- | --- |
| 接口名 | 方向 | 描述 |
| we | I | 写使能信号 |
| addrbus | I | 地址总线 |
| wdbus | I | 数据总线 |
| bridge\_rd | O | 读出数据 |
| hwint | O | 输出中断位 |
| t\_addr | O | 向定时器输出地址总线 |
| t\_we | O | 定时器写使能 |
| t\_rd0 | I | 定时器0读出数据 |
| t\_rd1 | I | 定时器1读出数据 |
| t\_int0 | I | 定时器0中断信号 |
| t\_int1 | I | 定时器1中断信号 |

新增乘除处理单元

|  |  |  |
| --- | --- | --- |
| 接口名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| clr | I | 复位信号 |
| a | I | 输入操作数1 |
| b | I | 输入操作数2 |
| a1 | I | 写入寄存器选择 |
| start | I | 启动运算选择 |
| mode | I | 运算模式选择 |
| stop | I | 无条件中止运算使能信号 |
| ret | I | 无条件实施回滚使能信号 |
| we | I | 寄存器写使能（不含回滚） |
| busy | O | 忙信号 |
| hi | O | 输出hi寄存器 |
| lo | O | 输出lo寄存器 |

功能说明

|  |  |  |
| --- | --- | --- |
| 编号 | 功能 | 描述 |
| 1 | 乘除运算 | 根据模式信号选择不同的模式进行运算 |
| 2 | 写寄存器 | 写使能有效时写入hi或lo寄存器 |
| 3 | 中止运算 | 为响应中断，stop信号有效时立即无条件放弃运算 |
| 4 | 回滚寄存器 | 为响应中断,ret信号有效时撤销mthi/mtlo指令操作 |

新增支持半字/字节模块

M级dmdecode

|  |  |  |
| --- | --- | --- |
| 接口名 | 方向 | 描述 |
| wordmode | I | 输入BE的译码模式 |
| addr | I | 输入当前要读取的地址 |
| be | O | 输出BE译码结果 |

功能说明

|  |  |  |
| --- | --- | --- |
| 编号 | 功能 | 描述 |
| 1 | BE译码 | 根据输入输出BE的值 |

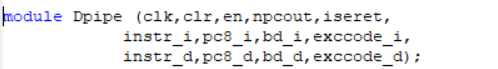
W级dmcut

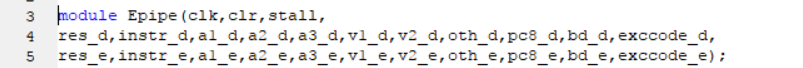
|  |  |  |
| --- | --- | --- |
| 接口名 | 方向 | 描述 |
| dmout | I | 输入完整DM输出结果 |
| ao\_w | I | 输入当前地址 |
| dr\_w | O | 输出需要的结果 |
| wordmode | I | 输入字节/半字/字模式 |

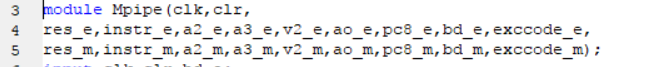
功能说明

|  |  |  |
| --- | --- | --- |
| 编号 | 功能 | 描述 |
| 1 | 无符号半字 | 输出无符号扩展的指定半字 |
| 2 | 有符号半字 | 输出有符号扩展的指定半字 |
| 3 | 无符号字节 | 输出无符号扩展的指定字节 |
| 4 | 有符号字节 | 输出有符号扩展的指定字节 |
| 5 | 整字输出 | 输出完整的字 |

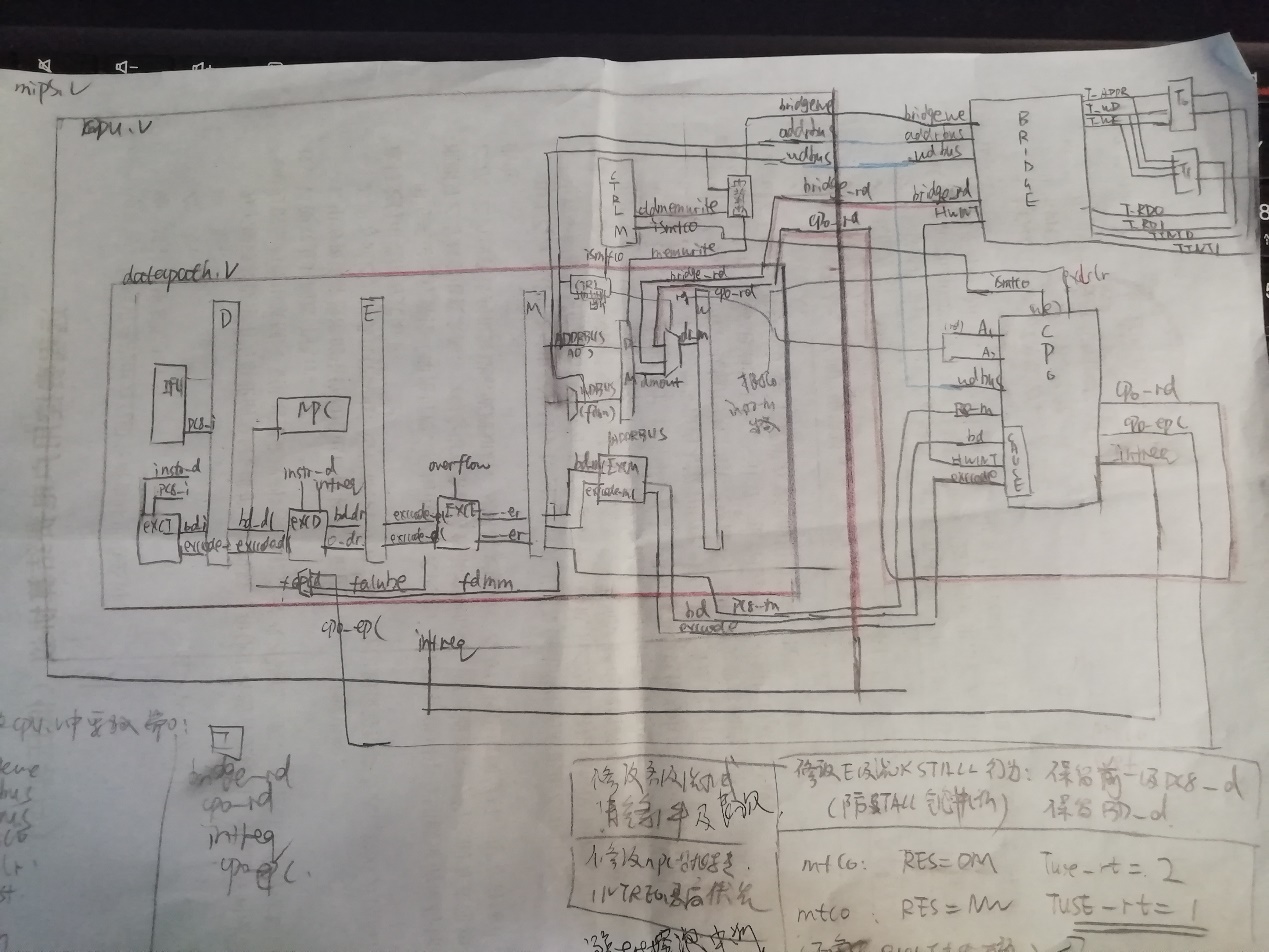
各级流水线寄存器







DATAPATH及CP0，BRIDGE,TIMER缩略图

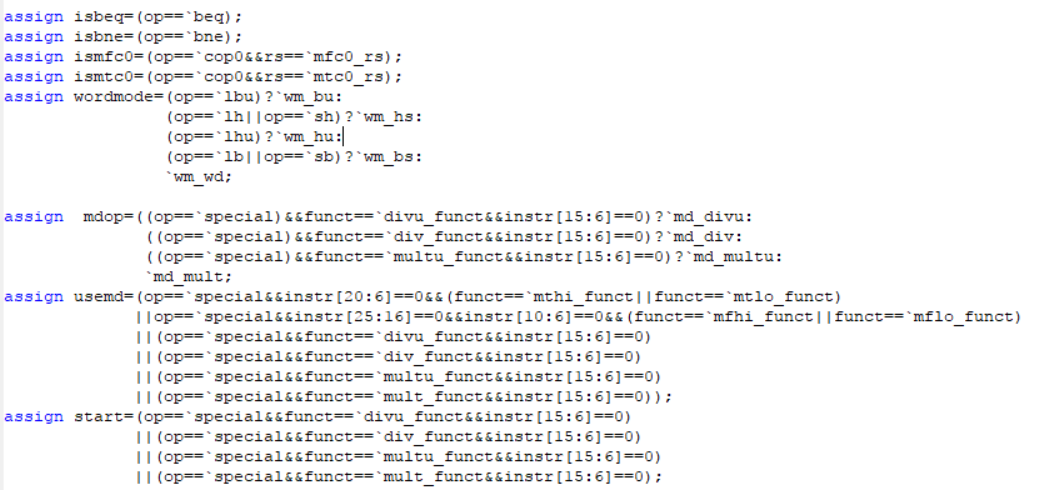


控制器设计：

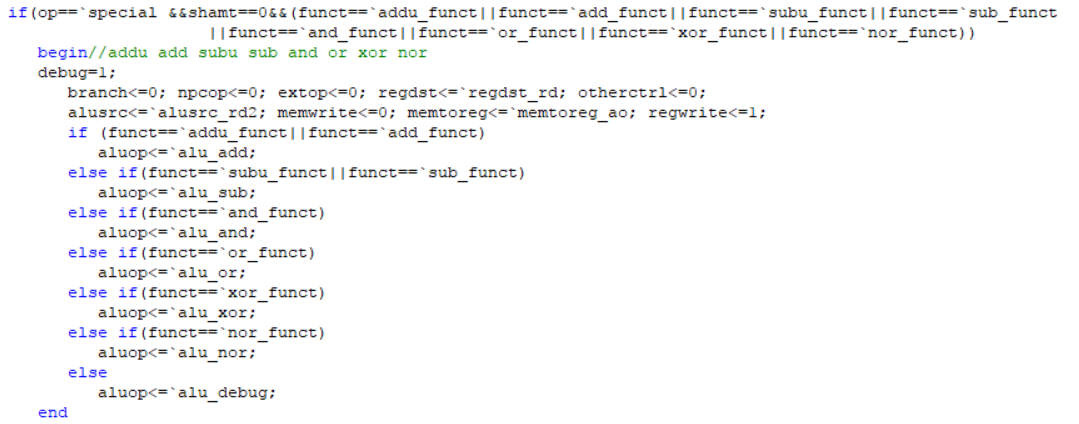
控制器部分基本沿用P5设计

1. 常规控制器设计

线网类型变量：



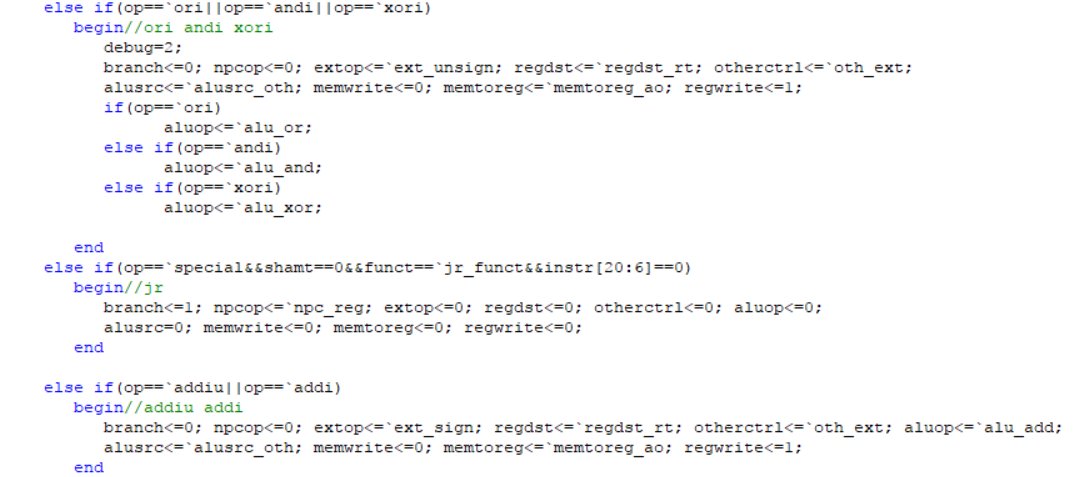
Addu add subu sub and or xor nor



ori andi xori

jr

addiu addi

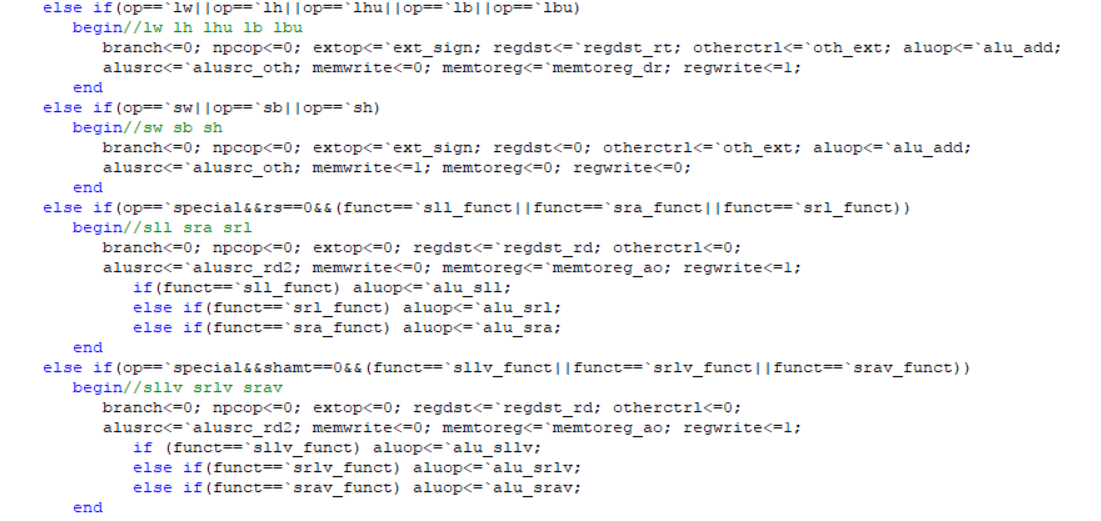


lw lh lhu lb lbu

sw sb sh

sll sra srl

sllv srlv srav

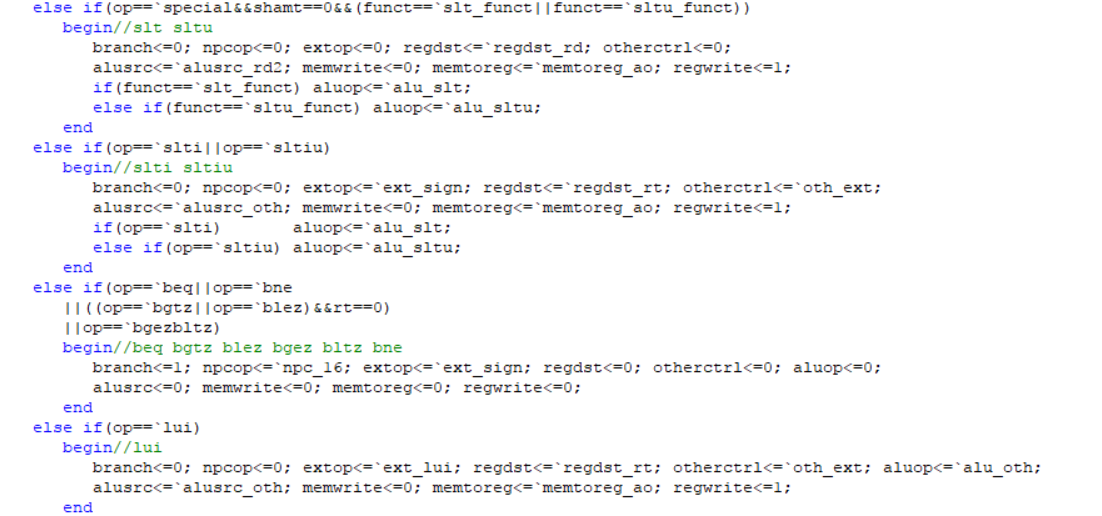


slt sltu

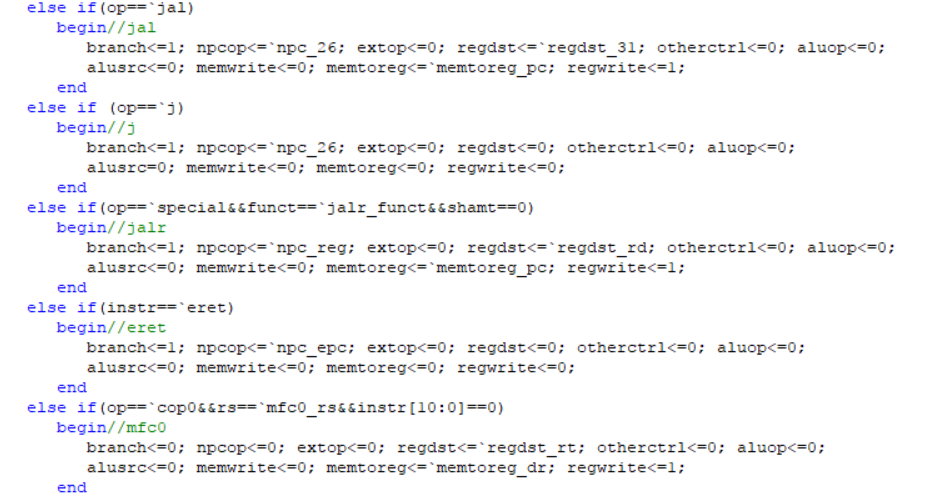
slti sltiu

beq bgtz bgez blez bltz bne

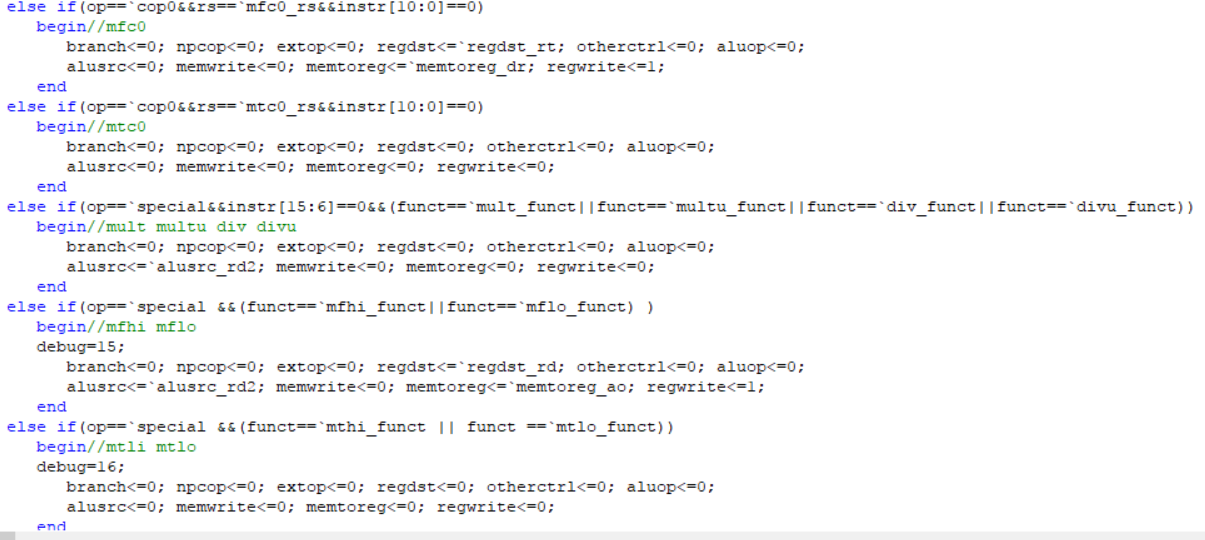
lui



Jal j jalr eret mfc0

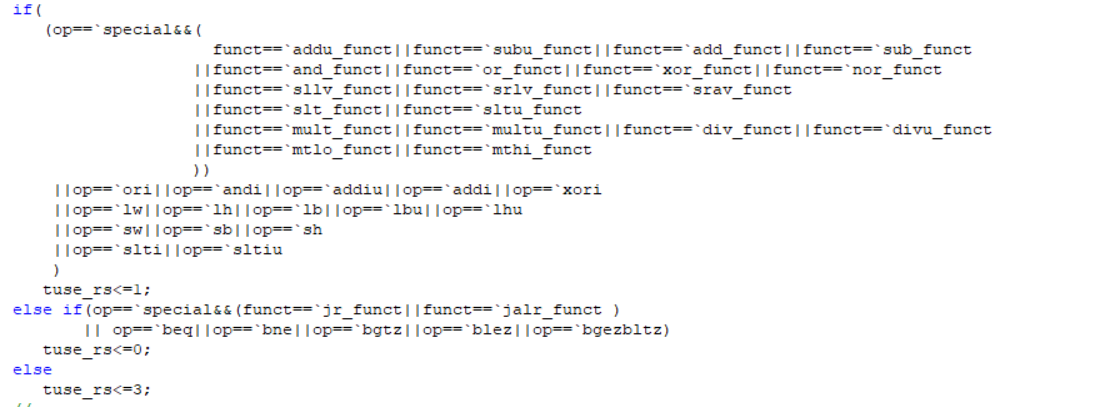


Mtc0 mult multu div divu mfhi mflo mthi mtlo

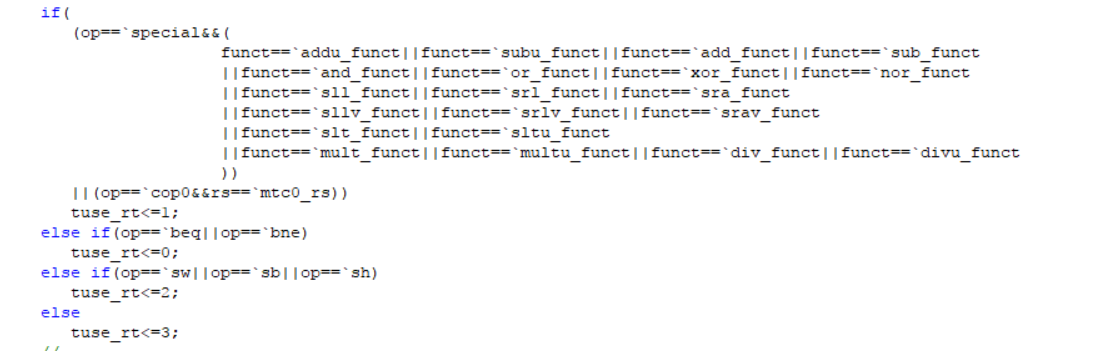


AT控制器

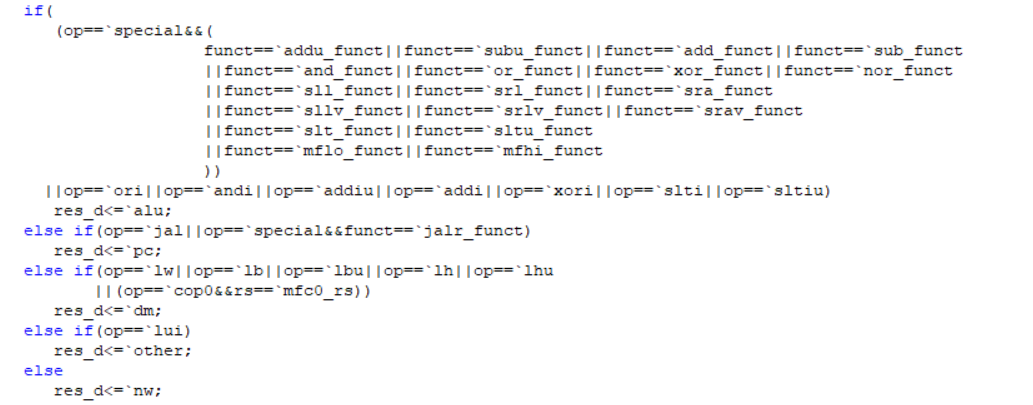
Tuse\_rs



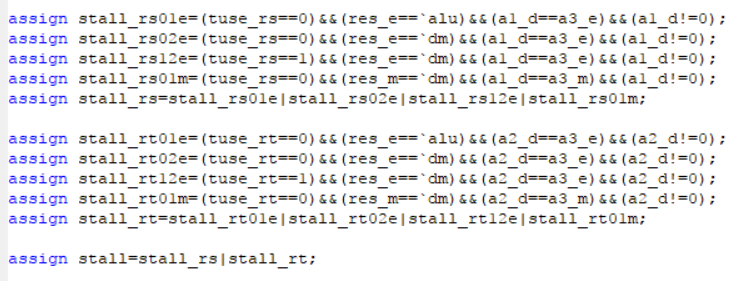
Tuse\_rt



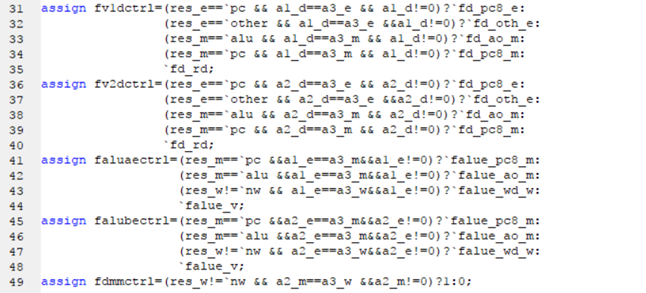
Res



STALL控制器：无任何修改



转发控制器：无任何修改



（epc的转发在datapath中独立完成，不受forward控制器的控制）

（乘除模块的暂停，中止，回滚在datapath中完成）

测试程序

测试程序1：测试了取地址异常和epc的独立转发，mtc0与mfc0部分的转发

.text

ori $s0,0x3005

jr $s0

ori $t0,1

ori $t1,1

ori $t2,1

ori $t3,1

ori $t4,1

.ktext 0x4180

ori $a1,3

mfc0 $a0,$14

add $a0,$a0,$a1

mtc0 $a0,$14

eret

测试程序2测试了算数溢出异常

.text

lui $s0,0x7fff

add $s1,$s0,$s0

ori $t0,1

.ktext 0x4180

ori $a1,4

mfc0 $a0,$14

add $a0,$a0,$a1

mtc0 $a0,$14

eret

测试程序3测试了取数异常和eret后硬件添加的延迟槽

.text

ori $s0,1

lw $s1,0($s0)

ori $t0,1

.ktext 0x4180

ori $a1,4

mfc0 $a0,$14

add $a0,$a0,$a1

mtc0 $a0,$14

eret

ori $t7,1

测试程序4测试了定时器和外部中断的响应

.text

ori $s0,9#低四位分别是1001

ori $s1,1000

sw $s0,0x7f10($0)

sw $s0,0x7014($0)

loop:#开始死循环

beq $s0,$s0,loop

nop

.ktext 0x0004180

ori $s0,9

sw $s0,0x7f10($0)

eret

.text

ori $s0,9

sw $s0,0($0)

ori $s1,1

sw $s0,0x7f00($0)

sw $s1,0x7f04($0)

ori $t0,1

ori $t1,2

lw $t3,0($0)#stall=============

ori $t3,4

ori $t4,5

ori $t5,6

lui $s0,0xffff

ori $s0,0xffff

lui $s1,0x8000

beq $s1,$0,tar

add $s0,$s0,$s1#===============

ori $t6,7

tar:

ori $t7,8

.ktext 0x00004180

mfc0 $1, $1296

mfc0 $1, $13

mfc0 $1, $14

lui $s0,0x0000

lui $s1,0

sw $0,0x7f00($0)

sw $0,0x7f04($0)

ori $a0,$0,4

mtc0 $1,$14

nop

eret