CPU设计文档

目录：

[一、相关规定与设计要求 1](#_Toc533617688)

[1.设计规格说明 1](#_Toc533617689)

[2.设计约束 1](#_Toc533617690)

[3.中断异常行为规范 2](#_Toc533617691)

[二、模块及数据通路增改 3](#_Toc533617692)

[1.异常检测模块 3](#_Toc533617693)

[2.CP0模块 3](#_Toc533617694)

[3.bridge模块 4](#_Toc533617695)

[4.timer计时器 5](#_Toc533617696)

[5.datapath修改 5](#_Toc533617697)

[6.mips模块修改 6](#_Toc533617698)

[三、控制器设计 7](#_Toc533617699)

[1.主控制器 7](#_Toc533617700)

[2.AT编码器 9](#_Toc533617701)

[3.冲突检测器 10](#_Toc533617702)

[4.转发控制器 10](#_Toc533617703)

[四、测试程序 12](#_Toc533617704)

[1.新增指令测试 12](#_Toc533617705)

[2.bridge与timer测试 12](#_Toc533617706)

[3. 中断异常进入测试 12](#_Toc533617707)

[六、思考题 14](#_Toc533617708)

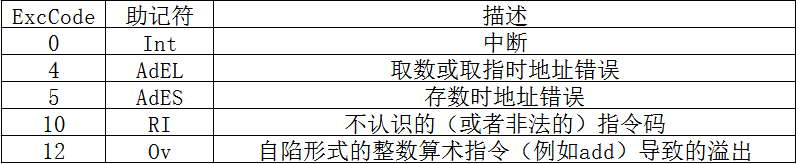
一、相关规定与设计要求

1.设计规格说明

（1）MIPS微系统包括MIPS处理器（CPU）、系统桥（bridge）和2个定时器（timer），支持中断和异常，需要实现系统桥（bridge）和2个设备（device），device为定时器（timer）。

（2）CPU支持MIPS-lite2-int指令集：{ADDU、SUBU、**ADD、SUB、**ORI、LW、SW、BEQ、LUI、J、JAL、JR、NOP、**ERET、MFC0、MTC0**}，为流水线设计，**eret指令需要由硬件实现后跟nop，即当eret指令到达E级时清除D级指令**。

（3）本project需要支持的异常：



（4）为了支持异常和中断，流水线必须支持0号协处理器（CP0），必须实现的CP0寄存器包括:**SR、CAUSE、EPC、PrID**。

（5）MIPS的异常入口只支持**0x00004180**，所有的异常与中断都从这里进入，即**2个timer的异常入口是完全相同的**。

2.设计约束

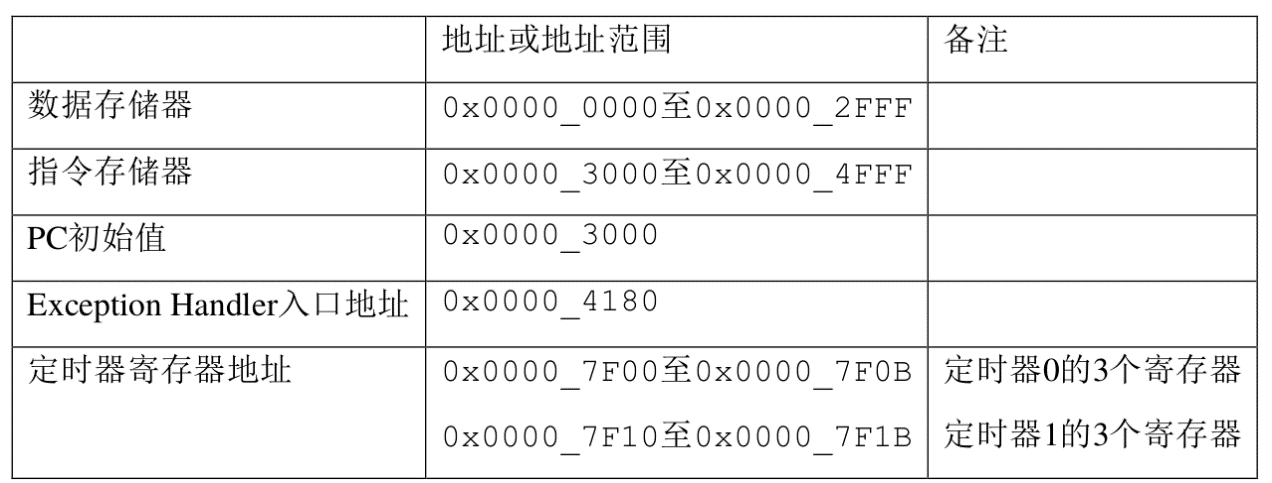
（1）IM和DM：容量为16KB（32bit/word×4Kword）。

（2）整个设计的顶层模块应该至少包括：CPU、bridge和2个timer。

CPU：在P5基础上添加相应扩展。

bridge：作为独立module，不包括在CPU中。

（3）地址空间：



（4）Exception handler的代码属于指令存储器。

（5）timer0 输出的中断请求信号请接入 MIPS 处理器的 HWInt[2]，即最低中断位。timer1 输出的中断请求信号请接入 MIPS 处理器的 HWInt[3] 。

3.中断异常行为规范

（1）对于异常的处理，遵循精确异常的处理规则，受害指令的前序指令的结果保留，受害指令的后序指令的效果不影响异常返回后后序指令执行时的效果。

（2）对于中断的发生，不指定哪条为受害指令，只需保证执行中断处理程序前后每条指令的执行效果不变即可。

（3）当异常或中断发生时，应立即将 EXL 置为 1，表示屏蔽所有中断，并将 ExcCode写入到Cause寄存器中。

（4）中断的优先级高于异常的优先级，不支持中断异常嵌套的情况。在中断异常处理程序结束后（即eret指令后），即可支持中断异常。

（5） 当 jal 的延迟槽指令发生异常时，jal 指令写入结果保存。当异常处理结束后，jal应再次写入。

二、模块及数据通路增改

1.异常检测模块

异常检测模块用于检测除W外每一级的异常，产生ExcCode并存入本级流水寄存器。其中F级同时产生标记延迟槽指令的BD信号，用于CP0处理延迟槽指令异常。

表2.1.1 各级异常类型及描述

|  |  |  |
| --- | --- | --- |
| **级次** | **异常类型** | **异常描述** |
| F | 取指异常AdEL | PC 值没有以4对齐  范围超出0x3000~0x4ff。 |
| 非法指令码RI | IFU 取出一条非给定指令，即op及fun是否为给定 |
| E | 运算溢出Ov | add、sub引起的溢出 |
| 取数异常AdEL | lw计算地址时加法溢出。 |
| 存数异常AdES | sw计算地址时加法溢出。 |
| M | 取数异常AdEL | lw 取数指令时，地址没有按照要求对齐  范围超出0x0000~0x2ffc、0x7F00~0X7F0B、 0x7F10~0X7F1B |
| 存数异常AdES | sw 取数指令时，地址没有按照要求对齐  范围超出0x0000~0x2ffc、0x7F00~0X7F0B、 0x7F10~0X7F1B  试图往定时器的 COUNT 寄存器写入值 |

M级由bridge传入timer的中断信号HWInt，与ExcCode一起进入CP0被处理。

2.CP0模块

CP0主要用于处理中断异常信息，需求信息都来自M级流水寄存器，因此加入M级。支持 SR、Cause、EPC、PRId 四个寄存器，模块规格参考课件上的设计，并加入了BD信号。

图2.2.1 12号SR寄存器与13号Cause寄存器

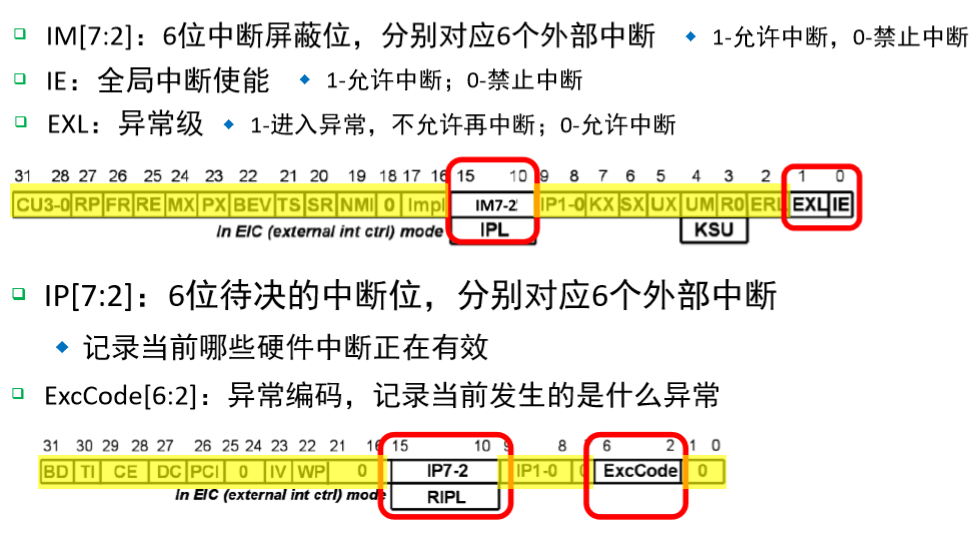


表2.2.1 CP0模块端口定义

|  |  |  |
| --- | --- | --- |
| **端口名** | **方向** | **端口含义** |
| [4:0] A1 | I | 读CP0寄存器的编号 |
| [4:0] A2 | I | 写CP0寄存器的编号 |
| [31:0] DIn | I | MTC0指令写入CP0寄存器的值 |
| [31:2] PC8 | I | 当前M级指令的PC8值 |
| [6:2] ExcCode | I | M级存储当前指令存在的异常类型 |
| [5:0] HWInt | I | bridge传入的中断信号 |
| BD | I | M级存储的当前指令的延迟槽指令标记 |
| We | I | MTC0的写使能信号 |
| clk | I | 时钟信号 |
| rst | I | 复位信号 |
| EXLClr | I | EXL清空信号，用于eret指令 |
| IntReq | O | CP0产生的中断异常处理信号 |
| [31:0] EPC | O | 向NPC传递eret所需的EPC地址 |
| [31:0] DOut | O | MFC0指令输出CP0寄存器的值 |

表2.2. CP0模块功能描述

|  |  |  |
| --- | --- | --- |
| **序号** | **功能** | **功能描述** |
| 1 | 判断中断异常 | IntReq = (|(HWInt & IM) & IE & !EXL)|  ((ExcCode != 5'b0) && !EXL) |
| 2 | 处理中断 | SR存入IM，EXL与IE，EXL置位为1  EPC 根据BD存入 PC-4或PC  Cause存入HWInt，ExcCode为0（中断优先异常） |
| 3 | 处理异常 | SR存入IM，EXL与IE，EXL置位为1  EPC 根据BD存入 PC-4或PC  Cause存入HWInt与ExcCode |
| 4 | 输出寄存器值 | 根据A1的值输出对应寄存器值 |
| 5 | 存入寄存器值 | 写使能有效，根据A2的值向对应寄存器中存入值 |

3.bridge模块

用于连接CPU与两个timer，传递数据。

表2.3.1 bridge模块端口定义

|  |  |  |
| --- | --- | --- |
| **端口名** | **方向** | **端口含义** |
| [31:0] PrAddr | I | CPU读设备的地址 |
| [31:0] PrRD | O | 设备传给CPU的数据 |
| [31:0] PrWD | I | CPU写给设备的数据 |
| [3:2] DEV\_Addr | O | 设备可用的地址 |
| [31:0] DEV\_WD | O | CPU写给设备的数据 |
| [31:0] DEV1\_RD | I | 设备1向CPU输出的数据 |
| [31:0] DEV2\_RD | I | 设备2向CPU输出的数据 |
| Int0 | I | 设备0的中断信号 |
| Int1 | I | 设备1的中断信号 |
| [7:2] HWInt | O | 传给CPU与CP0的待决中断位 |

4.timer计时器

根据设定的时间来定时产生中断信号，用来模拟中断源。用图例状态机描述计时器行为，CPU写入数据的优先级高于自改变的优先级。

表2.4.1 timer模块端口定义

|  |  |  |
| --- | --- | --- |
| **端口名** | **方向** | **端口含义** |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| [3:2] Addr | I | 设备内部偏移地址 |
| We | I | 写使能信号 |
| [31:0] DataIn, | I | CPU写入数据 |
| [31:0] DataOut, | O | 计数器向CPU输出的数据 |
| IRQ | O | 计时器发出的中断信号 |

5.datapath修改

（1）IFU修改

根据CP0产生的IntReq，决定IFU的跳转是handle还是正常指令地址。

更改IM的大小与PC取指的地址宽度，增加handler在指令存储器中的地址位置，增加初始化。

如果发生取指异常或者RI，则将指令改为nop。

（2）NPC修改

当跳转指令为eret时，跳转地址为EPC寄存器存储的值，且优先级高于beq、jr、j和jal。

（3）DM修改

更改DM的大小与地址的宽度。

（4）流水寄存器修改

D级增加BD与ExcCode，增加eret后加nop时清空D级的机制（PC置位为EPC），增加IntReq导致的清空机制。

E级增加BD与ExcCode，更改暂停时清空E级的机制（PC置位为D级PC值），增加IntReq导致的清空机制。

M级增加BD与ExcCode，增加IntReq导致的清空机制。

W级增加IntReq导致的清空机制。

（5）CP0与bridge数据处理机制

做进行类似DM的处理，用指令信息（MTC0与MFC0）与bridge控制信号区分向DM读/写信息与向CP0或设备读/写信息。

（6）加入bridge带来的信息

CPU与bridge的连接与DM相似，向bridge传入ALU计算出的地址与寄存器RD2的值，得到设备传来的输出信息和中断信号。

添加HITDM信号，区分是向设备读写还是向DM读写。

6.mips模块修改

将bridge与连个timer与CPU连接，每个timer的写使能信号为判断bridge得到的地址基地址是否为自身加上原本的DM写使能信号。

三、控制器设计

1.主控制器

在主控制器中加入新加指令的信号。

表3.1.1 非转发暂停控制信号

|  |  |
| --- | --- |
| 控制信号 | 功能名称 |
| branch | branch = 0时，PC端输入为IFU.PC4；  branch = 1时，PC端输入为NPC。 |
| Br | Br = 00时，该指令为分支指令beq，NPC = PC + 4 + sign\_ext(Imm||00)。  Br = 01时，该指令为跳转指令jal或j，NPC = {PC[31:28] ，Jal，0，0}。  Br = 10时，该指令为跳转指令jr，NPC = rs。  Br = 11 时，该指令为跳转指令eret，NPC=EPC，优先级最高。 |
| RegIn | RegIn = 00时，数据写入rt寄存器；  RegIn = 01时，数据写入rd寄存器。  RegIn = 10时，数据写入31号寄存器。 |
| RegWr | RegWr = 0时，GRF不能写入数据；  RegWr = 1时，GRF可以写入数据。 |
| EXTOp | EXTOp = 00时，进行高位0扩展，B = {16{0}，A}；  EXTOp = 01时，进行低位0扩展，B = {A，16{0}}；  EXTOp = 10时，进行符号扩展，B = {16{A[15]}，A}；。 |
| ALUSrc | ALUSrc = 0时，ALU的第二个运算数来自转发复选器MF\_RD2\_E；  ALUSrc = 1时，ALU的第二个运算数来自位扩展之后的立即数。 |
| ALUOp | ALUOp = 00时，进行加运算，C = A + B；  ALUOp = 01时，进行减运算，C = A - B；  ALUOp = 10时，进行或运算，C = A | B； |
| MemWr | MemWr = 0时，DM不能写入数据；  MemWr = 1时，DM可以写入数据。 |
| ToReg | ToReg = 00时，写入寄存器的数据来自ALU；  ToReg = 01时，写入寄存器的数据来自DM。  ToReg = 01时，写入寄存器的数据来自NPC。 |

表3.1.2非转发暂停控制信号产生的真值表

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | opcode | funct | branch | Br | RegIn | RegWr | EXTOp | ALUSrc | ALUOp | MemWr | ToReg |
| addu | 000000 | 100001 | 0 | X | 01 | 1 | X | 0 | 00 | 0 | 00 |
| add | 000000 | 100000 | 0 | X | 01 | 1 | X | 0 | 00 | 0 | 00 |
| subu | 000000 | 100011 | 0 | X | 01 | 1 | X | 0 | 01 | 0 | 00 |
| sub | 000000 | 100010 | 0 | X | 01 | 1 | X | 0 | 01 | 0 | 00 |
| ori | 001101 |  | 0 | X | 00 | 1 | 00 | 1 | 10 | 0 | 00 |
| lw | 100011 |  | 0 | X | 00 | 1 | 10 | 1 | 00 | 0 | 01 |
| sw | 101011 |  | 0 | X | X | 0 | 10 | 1 | 00 | 1 | X |
| beq | 000100 |  | 1 | 00 | X | 0 | X | 0 | X | 0 | X |
| lui | 001111 |  | 0 | X | 00 | 1 | 01 | 1 | 00 | 0 | 00 |
| nop | 000000 | 000000 | 0 | X | 00 | 0 | 00 | 0 | 00 | 0 | 00 |
| j | 000010 |  | 1 | 01 | X | 0 | X | X | X | 0 | X |
| jal | 000011 |  | 1 | 01 | 10 | 1 | X | X | X | 0 | 10 |
| jr | 000000 | 001000 | 1 | 10 | X | 0 | X | X | X | 0 | X |
| mfc0 | 010000 00000 | | 0 | X | 00 | 1 | X | X | X | 0 | 01 |
| ismfc0 决定存入GRF的数据来源是DM还是CP0 | | | | | | | | |
| mtc0 | 010000 00100 | | 0 | X | X | 0 | X | X | X | 0 | X |
| ismtc0 决定数据能否存入CP0， | | | | | | | | |
| eret | 32位全 | | 1 | 11 | X | 0 | X | X | X | 0 | X |
| iseret CP0的EXL清零，D级加nop | | | | | | | | |

附表 支持指令集编码与操作

|  |  |  |
| --- | --- | --- |
| 指令 | 编码形式 | 操作 |
| addu | 000000 rs rt rd 00000 100001 | GPR[rd] = GPR[rs] + GPR[rt] |
| add | 000000 rs rt rd 00000 100000 | GPR[rd] = GPR[rs] + GPR[rt] 检测溢出 |
| subu | 000000 rs rt rd 00000 100011 | GPR[rd] = GPR[rs] - GPR[rt] |
| sub | 000000 rs rt rd 00000 100010 | GPR[rd] = GPR[rs] - GPR[rt] 检测溢出 |
| ori | 100011 base rt offset | GPR[rt] = GPR[rs] OR zero\_extend(immediate) |
| lw | 100011 base rt offset | Addr = GPR[base] + sign\_ext(offset)  GPR[rt] = memory[Addr] |
| sw | 101011 base rt offset | Addr = GPR[base] + sign\_ext(offset)  memory[Addr] = GPR[rt] |
| beq | 000100 rs rt offset | if (GPR[rs] == GPR[rt])  PC = PC + 4 + sign\_extend(offset||00)  else  PC = PC + 4 |
| lui | 001111 00000 rt immediate | GPR[rt] = immediate||0^16 |
| j | 000010 instr\_index | PC = PC[31:28] || instr\_index || 00 |
| jr | 000000 rs 10{0} 5{0} 001000 | PC = GPR[rs] |
| jal | 000011 instr\_index | PC = PC[31:28] || instr\_index || 00  GPR[31] = PC + 4 |
| nop | 0x00000000 |  |
| mfc0 | 010000 00000 rt rd 10’b0 | GPR[rt] = CP0[rd] |
| mtc0 | 010000 00100 rt rd 10’b0 | CP0[rd] = GPR[rt] |
| eret | 01000010000000000000000000011000 | PC = CP0[epc] |

2.AT编码器

根据指令变量与，产生Tuse变量和Res编码值（Res是Tnew的变形，用于指定产生新值的功能部件）。

此处将指令分为Cal\_r类(addu, subu)、Cal\_i类(ori)、Beq类(beq)、Load类(lw)、Save类(sw)分类处理，J类的三条指令(j, jal, jr)分开处理。

表4.2.1 指令集的Tuse 表3.2.2 指令集的Tnew

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **指令** | **Tuse** | |  | **指令** | **功能部件** | **Tnew** | | |
| **rs** | **rt** |  | **E** | **M** | **W** |
| Cal\_r类 | 1 | 1 |  | Cal\_r类 | ALU | 1 | 0 | 0 |
| Cal\_i类 | 1 |  |  | Cal\_i类 | ALU | 1 | 0 | 0 |
| load类 | 1 |  |  | load类 | DM | 2 | 1 | 0 |
| store类 | 1 | 2 |  | store类 |  |  |  |  |
| beq | 0 | 0 |  | beq |  |  |  |  |
| lui |  |  |  | lui | ALU | 1 | 0 | 0 |
| nop |  |  |  | nop |  |  |  |  |
| j |  |  |  | j |  |  |  |  |
| jal |  |  |  | jal | PC | 0 | 0 | 0 |
| jr | 0 |  |  | jr |  |  |  |  |
| mfc0 |  |  |  | mfc0 | DM | 2 | 1 | 0 |
| mtc0 |  | 1 |  | mtc0 |  |  |  |  |
| eret |  |  |  | eret |  |  |  |  |

3.冲突检测器

根据策略矩阵计算分类指令的暂停条件，根据条件按照逻辑取与和或运算。

表3.3.1 rs策略矩阵

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tnew**  **Tuse** | **E** | | | **M** | | | **W** | | |
| **ALU** | **DM** | **PC** | **ALU** | **DM** | **PC** | **ALU** | **DM** | **PC** |
| 1 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | **S** | **S** | F | F | **S** | F | F | F | F |
| 1 | F | **S** | F | F | F | F | F | F | F |

表4.3.2 rt策略矩阵

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tnew**  **Tuse** | **E** | | | **M** | | | **W** | | |
| **ALU** | **DM** | **PC** | **ALU** | **DM** | **PC** | **ALU** | **DM** | **PC** |
| 1 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | **S** | **S** | F | F | **S** | F | F | F | F |
| 1 | F | **S** | F | F | F | F | F | F | F |
| 2 | F | F | F | F | F | F | F | F | F |

4.转发控制器

根据转发多选器需求和原则产生转发多选器的控制信号，控制信号的优先级由顺序决定，通过三目运算符的比较赋值顺序实现。

转发条件：转发接受级读寄存器编号与被转发级写寄存器编号相同不为0，转发级寄存器写使能信号为1。

转发来源：RES所标记的被转发级指令对应的新值产生部件储存在流水寄存器中的值。

表3.4.1 转发控制信号

|  |  |  |
| --- | --- | --- |
| **控制信号** | **取值** | **功能说明** |
| MF\_RD1\_D | MF\_PC8\_E | 向D级需要RD1值的端口转发PC8\_E |
| MF\_PC8\_M | 向D级需要RD1值的端口转发PC8\_M |
| MF\_AO\_M | 向D级需要RD1值的端口转发AO\_M |
| MF\_RD | D级需要RD1值的端口无需转发 |
| MF\_RD2\_D | MF\_PC8\_E | 向D级需要RD2值的端口转发PC8\_E |
| MF\_PC8\_M | 向D级需要RD2值的端口转发PC8\_M |
| MF\_AO\_M | 向D级需要RD2值的端口转发AO\_M |
| MF\_RD | D级需要RD2值的端口无需转发 |
| MF\_RD1\_E | MF\_PC8\_M | 向E级需要RD1值的端口转发PC8\_M |
| MF\_AO\_M | 向E级需要RD1值的端口转发AO\_M |
| MF\_WD\_W | 向E级需要RD1值的端口转发W级的WD值 |
| MF\_RD | E级需要RD1值的端口无需转发 |
| MF\_RD2\_E | MF\_PC8\_M | 向E级需要RD2值的端口转发PC8\_M |
| MF\_AO\_M | 向E级需要RD2值的端口转发AO\_M |
| MF\_WD\_W | 向E级需要RD2值的端口转发W级的WD值 |
| MF\_RD | E级需要RD2值的端口无需转发 |
| MF\_RD2\_M | MF\_WD\_W | 向M级需要RD2值的端口转发W级的WD值 |
| MF\_RD | M级需要RD2值的端口无需转发 |

在datapath中新增EPC的转发：当E或M级为MTC0且写EPC寄存器时，从E级和M级的流水寄存器向D级NPC的EPC端口转发EPC值，而其他转发不受影响。

|  |  |  |
| --- | --- | --- |
| MF\_EPC\_D | MF\_EPC\_E | 向D级NPC\_EPC端口转发MF\_RD2\_E |
| MF\_EPC\_M | 向D级NPC\_EPC端口转发MF\_RD2\_M |
| MF\_EPC | D级NPC\_EPC端口无需转发 |

四、测试程序

1.新增指令测试

表5.1 CPU测试程序与期望输出（1）

|  |  |
| --- | --- |
| **测试程序** | **期望输出或现象** |
| 1 ori $t0,$t0,0x3018 | 00003000: $ 8 <= 00003018 |
| 2 mtc0 $t0,$14 | CP0的EPC值变为0x00003018 |
| 3 mfc0 $t1,$14 | 00003008: $ 9 <= 00003018 |
| 4 eret | 跳转至第7行 |
| 5 ori $t0,$t0,0x2333 | 更改为nop |
| 6 ori $t1,$t1,0x2333 | 不执行 |
| 7 ori $t1,$t1,0xaaaa | 00003018: $ 9 <= 0000baba |

2.bridge与timer测试

表5.2 CPU测试程序与期望输出（2）

|  |  |
| --- | --- |
| **测试程序** | **期望输出或现象** |
| 1 ori $t0,$t0,0x3 #初值 | 00003000: $ 8 <= 00000003 |
| 2 ori $t1,$t1,0x9 #模式0 | 00003004: $ 9 <= 00000009 |
| 3 sw $t1,0x7f00($0) | timer0的PRESET变为3 |
| 4 sw $t0,0x7f04($0) | timer0的CTRL后四位变为1001 |
| 5-9 nop | timer0模式0计数 |
| 10 ori $t1,$t1,0xb #模式1 | 00003024: $ 9 <= 0000000b |
| 11 sw $t1,0x7f00($0) | timer0的CTRL后四位变为1011 |
| 12 lw $t2,0x7f08($0) | 00003030: $10 <= 00000000 |
| 13- nop | timer0模式1计数a |

图5.1 timer模式0工作方式

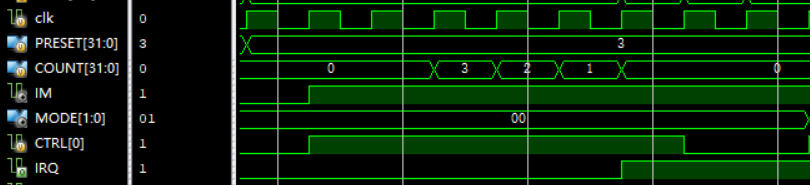
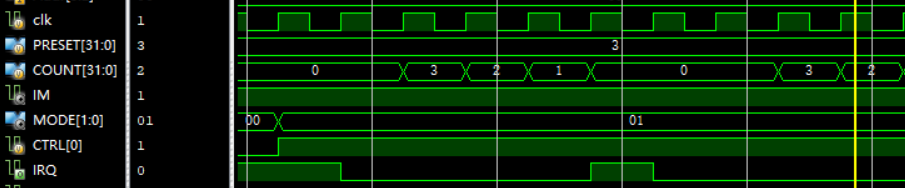


图5.2 timer模式1工作方式



3.中断异常进入测试

表5.3 CPU测试程序与期望输出（3）

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **测试目的** | **测试程序** | **期望输出或现象** |
| 1 | F级  RI异常 | ori $t0,0xffff  or $t1,$t0,$t0 | or进入F级后产生ExcCode=10，D级变为nop，进入M级后进入异常处理。 |
| 2 | F级  AdEL异常 | ori $t0,0x3009  jr $t0  ...  0x2000,0x5000 | jr跳转的PC值进入F级后产生ExcCode=4，指令变为nop，进入M级后进入异常处理。 |
| 3 | E级  Ov异常 | lui $t0,0x8000  add $t1,$t0,$t0 | add进入E级后产生ExcCode=12，进入M级后进入异常处理。 |
| 4 | E级  AdEL异常 | ori $t0,0x8000  lw $t1,0x8000($t0) |  |
| 5 | E级  AdES异常 | ori $t0,0x8000  sw $t1,0x8000($t0) |  |
| 6 | M级  AdEL异常 | lw $t0,3($0)  ...  0x3000,0x7f0e | lw进入M级后产生ExcCode=4,然后进入异常处理。 |
| 7 | M级AdES  延迟槽指令异常 | j hhh  sw $t0,3($0)  ...  0x3000,0x7f0e | sw进入M级后产生ExcCode=5,然后进入异常处理，不写DM。  EPC存入j的地址。 |
| 8 | 中断 | ori $t0,0x3  ori $t1,0x9  sw $t0,0x7f04($0)  sw $t1,0x7f00($0)  ori $t2,0x0401  mtc0 $t2,$12 | 计时器装载3开始模式0工作，SR的中断屏蔽0位与全局中断使能打开。  当计时器到0时，进入中断。 |

六、思考题

Q1：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”？

A1：沟通硬件和软件的东西喵？

Q2.1：在我们设计的流水线中，DM处于CPU内部，请你考虑现代计算机中它的位置应该在何处。

A2.1：CPU外部。

Q2.2：BE部件对所有的外设都是必要的吗？

A2.2:不是。不一定需要截取特定字节。

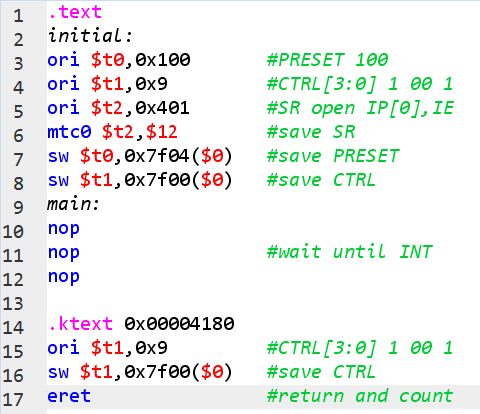
Q3.1: 请开发一个主程序以及定时器的exception handler。完成如下功能：

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。）

A3.1:

Q3.2：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？

A3.2：鼠标和键盘进行输入时向CPU发出中断信号，并产生对应操作的信号，在中断中传给CPU进行处理，处理结束后结束中断。