MIPS微系统

设计文档

贾博驿

2023年12月3日

设计实现指令

**共54条**

R型指令：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 | 5 | 5 | 5 | 5 | 6 |
| opcode | rs | rt | rd | shamt | funct |

I型指令：

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 5 | 5 | 16 |
| opcode | rs (base) | rt | imm16 (immediate, offset) |

J型指令：

|  |  |
| --- | --- |
| 6 | 26 |
| opcode | instr\_index |

注：

1.以下的RTL除分支与跳转类指令均省略PC←PC+4。

2.测试程序保证未知指令的测试用例中opcode和funct码的组合一定没有在MIPS基本指令集中出现，故没有对之前实现的指令进行删除。

**算数与逻辑类**

1. 加减法类
2. R型

|  |  |  |
| --- | --- | --- |
| 名称 | funct | RTL |
| add | 100000 | temp ← (GPR[rs]31||GPR[rs]31..0) + (GPR[rt]31||GPR[rt]31..0)  if temp32 ≠ temp31 then  SignalException(Ov)  else  GPR[rd] ← temp31..0  endif |
| addu | 100001 | GPR[rd] ← GPR[rs] + GPR[rt] |
| sub | 100010 | temp ← (GPR[rs]31||GPR[rs]31..0) − (GPR[rt]31||GPR[rt]31..0)  if temp32 ≠ temp31 then  SignalException(Ov)  else  GPR[rd] ← temp31..0  endif |
| subu | 100011 | GPR[rd] ← GPR[rs] - GPR[rt] |

1. I型

|  |  |  |
| --- | --- | --- |
| 名称 | opcode | RTL |
| addi | 001000 | temp ← (GPR[rs]31||GPR[rs]31..0) + sign\_extend(immediate)  if temp32 ≠ temp31 then  SignalException(Ov)  else  GPR[rt] ← temp31..0  endif |
| addiu | 001001 | GPR[rt] ← GPR[rs] + sign\_extend(immediate) |

1. 逻辑类
2. R型

|  |  |  |
| --- | --- | --- |
| 名称 | funct | RTL |
| and | 100100 | GPR[rd] ← GPR[rs] AND(bitwise) GPR[rt] |
| or | 100101 | GPR[rd] ← GPR[rs] OR(bitwise) GPR[rt] |
| xor | 100110 | GPR[rd] ← GPR[rs] XOR(bitwise) GPR[rt] |
| nor | 100111 | GPR[rd] ← GPR[rs] NOR(bitwise) GPR[rt] |
| sll | 000000 | GPR[rd] ← GPR[rt](31-shamt)..0 || 0shamt |
| srl | 000010 | GPR[rd] ← 0shamt || GPR[rt]31..shamt |
| sra | 000011 | GPR[rd] ← (GPR[rt]31)shamt || GPR[rt]31..shamt |
| sllv | 000100 | s ← GPR[rs]4..0 GPR[rd] ← GPR[rt](31-s)..0 || 0s |
| srlv | 000110 | s ← GPR[rs]4..0 GPR[rd] ← 0s || GPR[rt]31..s |
| srav | 000111 | s ← GPR[rs]4..0 GPR[rd] ← (GPR[rt]31)s || GPR[rt]31..s |

注：根据寄存器值移位指令只依照寄存器值的低五位移位，不受高位值影响。

1. I型

|  |  |  |
| --- | --- | --- |
| 名称 | opcode | RTL |
| andi | 001100 | GPR[rt] ← GPR[rs] AND(bitwise) zero\_extend(immediate) |
| ori | 001101 | GPR[rt] ← GPR[rs] OR(bitwise) zero\_extend(immediate) |
| xori | 001110 | GPR[rt] ← GPR[rs] XOR(bitwise) zero\_extend(immediate) |

1. 乘除法类（R型）

|  |  |  |
| --- | --- | --- |
| 名称 | funct | RTL |
| mult | 011000 | prod ← GPR[rs]31..0 × GPR[rt]31..0 LO ← prod31..0 HI ← prod63..32 |
| multu | 011001 | prod ← (0 || GPR[rs]31..0) × (0 || GPR[rt]31..0) LO ← prod31..0 HI ← prod63..32 |
| div | 011010 | q ← GPR[rs]31..0 ÷ GPR[rt]31..0  r ← GPR[rs]31..0 % GPR[rt]31..0  LO ← q  HI ← r |
| divu | 011011 | q ← (0 || GPR[rs]31..0) ÷ (0 || GPR[rt]31..0)  r ← (0 || GPR[rs]31..0) % (0 || GPR[rt]31..0)  LO ← q31..0  HI ← r31..0 |
| mfhi | 010000 | GPR[rd] ← HI |
| mthi | 010001 | HI ← GPR[rs] |
| mflo | 010010 | GPR[rd] ← LO |
| mtlo | 010011 | LO ← GPR[rs] |

**寄存器操作类**

1. R型

|  |  |  |
| --- | --- | --- |
| 名称 | funct | RTL |
| slt | 101010 | GPR[rd] ← 0GPRLEN-1 || GPR[rs] < GPR[rt] |
| sltu | 101011 | GPR[rd] ← 0GPRLEN-1 || (0 || GPR[rs]) < (0 || GPR[rt]) |

注：slt类指令成立时置1，**不成立时置0**。

1. I型

|  |  |  |
| --- | --- | --- |
| 名称 | opcode | RTL |
| slti | 001010 | GPR[rt] ← 0GPRLEN-1 || GPR[rs] < sign\_extend(immediate) |
| sltiu | 001011 | GPR[rt] ← 0GPRLEN-1 || (0 || GPR[rs]) < (0 || **sign\_extend**(immediate)) |
| lui | 001111 | GPR[rt] ← immediate || 016 |

注：sltiu对立即数的处理是先**有符号扩展**再进行**无符号比较**。

**分支与跳转类**

1. 分支类（I型）

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | opcode | rt | RTL |
| (all) |  |  | **I+1:** if condition then  PC ← PC + sign\_extend(offset || 02)  endif |
| bltz | 000001 | 00000 | **I:** condition ← GPR[rs] < 032 |
| bgez | 000001 | 00001 | **I:** condition ← GPR[rs] ≥ 032 |
| beq | 000100 | rt | **I:** condition ← (GPR[rs] = GPR[rt]) |
| bne | 000101 | rt | **I:** condition ← (GPR[rs] ≠ GPR[rt]) |
| blez | 000110 | 00000 | **I:** condition ← GPR[rs] ≤ 032 |
| bgtz | 000111 | 00000 | **I:** condition ← GPR[rs] > 032 |

注：流水线CPU分支指令不需要PC + 4后再加上offset，直接在延迟槽指令的地址上加。

1. 跳转类
2. R型

|  |  |  |
| --- | --- | --- |
| 名称 | funct | RTL |
| jr | 001000 | **I:**  **I+1:** PC ← GPR[rs] |
| jalr | 001001 | **I:** GPR[rd] ← PC + 8  **I+1:** PC ← GPR[rs] |

1. J型

|  |  |  |
| --- | --- | --- |
| 名称 | opcode | RTL |
| j | 000010 | **I:**  **I+1:** PC ← PC31..28 || instr\_index || 02 |
| jal | 000011 | **I:** GPR[31] ← PC + 8  **I+1:** PC ← PC31..28 || instr\_index || 02 |

注：流水线CPU跳转指令由于延迟槽存在应该加上PC + 8，而非单周期的PC + 4。

**系统桥操作类（I型）**

|  |  |  |
| --- | --- | --- |
| 名称 | opcode | RTL |
| (all) |  | addr ← sign\_extend(offset) + GPR[base] |
| lb | 100000 | byte ← addr1..0  GPR[rt] ← sign\_extend(memory[addr]7+8\*byte..8\*byte) |
| lh | 100001 | if addr0 ≠ 0 then  SignalException(AdEL)  endif  half ← addr1  GPR[rt] ← sign\_extend(memory[addr]15+8\*half..8\*half) |
| lw | 100011 | if addr1..0 ≠ 02 then  SignalException(AdEL)  endif  GPR[rt] ← memory[addr] |
| lbu | 100100 | byte ← addr1..0  GPR[rt] ← zero\_extend(memory[addr]7+8\*byte..8\*byte) |
| lhu | 100101 | if addr0 ≠ 0 then  SignalException(AdEL)  endif  half ← addr1  GPR[rt] ← zero\_extend(memory[addr]15+8\*half..8\*half) |
| sb | 101000 | memory[addr]7..0 ← GPR[rt]7..0 |
| sh | 101001 | if addr0 ≠ 0 then  SignalException(AdES)  endif  memory[addr]15..0 ← GPR[rt]15..0 |
| sw | 101011 | if addr1..0 ≠ 02 then  SignalException(AdES)  endif  memory[addr]31..0 ← GPR[rt] |

注：

1.针对字节和半字操作时只修改对应字节和半字的值，**同一字的其他部分不变**。

2.计算地址时加法溢出、地址不在存取合法地址区段等情况也会引发对应的AdE异常，具体见中断与异常表。

**异常处理类**

mfc0（COP0型）：GPR[rt] ← CPR[0,rd]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | rs | rt | rd | 0 | sel |
| 010000 | 00000 |  |  | 00000000 | 000 |

mtc0（COP0型）：CPR[0, rd] ← GPR[rt]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | rs | rt | rd | 0 | sel |
| 010000 | 00100 |  |  | 00000000 | 000 |

eret（COP0型）：SREXL ← 0 PC ← EPC

|  |  |  |  |
| --- | --- | --- | --- |
| opcode | rs | 0 | funct |
| 010000 | 10000 | 000000000000000 | 011000 |

syscall（R型）：SignalException(Syscall)

|  |  |  |
| --- | --- | --- |
| opcode | 0 | funct |
| 000000 | 00000000000000000000 | 001100 |

注：

1.因为P7实现的CP0不涉及使用sel域进行同编号寄存器的选择功能，认为其固定为000，若是非000值也不响应。

2.eret指令判断组合为opcode与funct，而不是opcode与rs，**且eret指令没有延迟槽。**

转发与暂停设计

**指令分类**

|  |  |  |  |
| --- | --- | --- | --- |
| 类型名 | 信号名 | 数量 | 指令 |
| 算数逻辑寄存器 | ALREG | 16 | \_arith（算数）：add, addu, sub, subu  \_logic（逻辑）：and, or, xor, nor  \_shift（移位）：sll, srl, sra, sllv, srlv, srav  \_comp（比较）：slt, sltu |
| 算数逻辑立即数 | ALIMM | 7 | \_sign（有符号拓展）：addi, addiu, slti, sltiu  \_zero（无符号拓展）：andi, ori, xori |
| 乘除法 | MLUTDIV | 4 | mult, multu, div, divu |
| 乘除寄存器读取 | MLFROM | 2 | mfhi, mflo |
| 乘除寄存器写入 | MLTO | 2 | mthi, mtlo |
| 高位加载 | EXTLUI | 1 | lui |
| 两数比较分支 | BRANCHE | 2 | beq, bne |
| 与零比较分支 | BRANCHZ | 4 | bltz, bgez, blez, bgtz |
| 仅读取跳转 | JUMPR | 1 | jr |
| 仅写入跳转 | JUMPW | 1 | jal |
| 读取写入跳转 | JUMPRW | 1 | jalr |
| 系统桥读取 | LOAD | 5 | lb, lh, lw, lbu, lhu |
| 系统桥写入 | STORE | 3 | sb, sh, sw |
| 读取协处理器寄存器 | MFCOP | 1 | mfc0 |
| 写入协处理器寄存器 | MTCOP | 1 | mtc0 |

注：不包括j、eret、syscall指令，因为它们不读取通用寄存器也不写入通用寄存器。

**Tuse与Tnew**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 信号名 | Tuse\_rs | Tuse\_rt |  | GRF\_WD\_W\_Sel | Tnew\_E | Tnew\_M | Tnew\_W |
| ALREG | 1 | 1 |  | 00 | 1 | 0 | 0 |
| ALIMM | 1 | (3) |  | 00 | 1 | 0 | 0 |
| MLUTDIV | 1 | 1 |  | (00) | (0) | (0) | (0) |
| MLFROM | (3) | (3) |  | 00 | 1 | 0 | 0 |
| MLTO | 1 | (3) |  | (00) | (0) | (0) | (0) |
| EXTLUI | (3) | (3) |  | 10 | 0 | 0 | 0 |
| BRANCHE | 0 | 0 |  | (00) | (0) | (0) | (0) |
| BRANCHZ | 0 | (3) |  | (00) | (0) | (0) | (0) |
| JUMPR | 0 | (3) |  | (00) | (0) | (0) | (0) |
| JUMPW | (3) | (3) |  | 11 | 0 | 0 | 0 |
| JUMPRW | 0 | (3) |  | 11 | 0 | 0 | 0 |
| LOAD | 1 | (3) |  | 01 | 2 | 1 | 0 |
| STORE | 1 | 2 |  | (00) | (0) | (0) | (0) |
| MFCOP | (3) | (3) |  | 01 | 2 | 1 | 0 |
| MTCOP | (3) | 2 |  | (00) | (0) | (0) | (0) |

注：

1.(3)表示该类指令不会读取rs或rt域翻译为寄存器编号后对应的寄存器，除GRF内部转发外不产生任何转发和暂停。

2.(00)和(0)表示表示该类指令GRFWE为0，虽然使用ALU数据但无意义。

**转发与暂停表**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SPL\_rs != 5’d0 && GRFWE\_X && SPL\_rs == GRF\_A3\_X | | | | | | | | | |
| 流水段(X) | E | | | | M (~ rs\_E\_premise) | | | | W |
| GRF\_WD\_W\_Sel\_X(Tnew\_X) | 00(1) | 01(2) | 10(0) | 11(0) | 00(0) | 01(1) | 10(0) | 11(0) | All |
| Tuse\_rs == 0 | S1 | S2 | F43 | F44 | F21 | S3 | F23 | F24 | F1 |
| Tuse\_rs == 1 | F51 | S4 | F32 |
| SPL\_rt != 5’d0 && GRFWE\_X && SPL\_rt == GRF\_A3\_X | | | | | | | | | |
| 流水段(X) | E | | | | M (~ rt\_E\_premise) | | | | W |
| GRF\_WD\_W\_Sel\_X(Tnew\_X) | 00(1) | 01(2) | 10(0) | 11(0) | 00(0) | 01(1) | 10(0) | 11(0) | All |
| Tuse\_rt == 0 | S1 | S2 | F43 | F44 | F21 | S3 | F23 | F24 | F1 |
| Tuse\_rt == 1 | F51 | S4 | F32 |
| Tuse\_rt == 2 | F62 |
| ISMULTDIV & (MULT\_Start | MULT\_Busy) | | | | | | | | | |
| SMULTDIV | | | | | | | | | |
| CP0WE\_X && rd\_X == 5’d14 (EPC) | | | | | | | | | |
| 流水段(X) | E | | | | M | | | | |
| / | FEPC\_E | | | | FEPC\_M | | | | |

注：

1.优先级**S > F4、F5、F6 > F2、F3。**

2.F2和F3之间、F4、F5和F6之间不会发生冲突，因为转发的是同一流水级的不同数据。

**转发信号表**

|  |  |  |  |
| --- | --- | --- | --- |
| SPL\_rs != 5’d0 && GRFWE\_X && SPL\_rs == GRF\_A3\_X | | | |
| 类型 | 描述 | FMUX\_V1\_D\_Sel | FMUX\_V1\_E\_Sel |
| S | 暂停 |  |  |
| F51 | V1\_E转发E\_RES\_M |  | 11 |
| F43 | V1\_D转发ext32\_E | 110 |  |
| F44 | V1\_D转发pc8\_E | 111 |  |
| F32 | V1\_E转发M\_RES\_W |  | 10 |
| F21 | V1\_D转发E\_RES\_M | 011 |  |
| F23 | V1\_D转发ext32\_M | 100 |  |
| F24 | V1\_D转发pc8\_M | 101 |  |
| F1 | GRF内部转发 |  |  |
| F0 | 不转发 | 000 | 00 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SPL\_rt != 5’d0 && GRFWE\_X && SPL\_rt == GRF\_A3\_X | | | | |
| 类型 | 描述 | FMUX\_V2\_D\_Sel | FMUX\_V2\_E\_Sel | FMUX\_V2\_M\_Sel |
| S | 暂停 |  |  |  |
| F62 | V2\_M转发M\_RES\_W |  |  | 1 |
| F51 | V2\_E转发E\_RES\_M |  | 11 |  |
| F43 | V2\_D转发ext32\_E | 110 |  |  |
| F44 | V2\_D转发pc8\_E | 111 |  |  |
| F32 | V2\_E转发M\_RES\_W |  | 10 |  |
| F21 | V2\_D转发E\_RES\_M | 011 |  |  |
| F23 | V2\_D转发ext32\_M | 100 |  |  |
| F24 | V2\_D转发pc8\_M | 101 |  |  |
| F1 | GRF内部转发 |  |  |  |
| 0 | 不转发 | 000 | 00 | 0 |

注：未标注的信号取值与不转发相同。

|  |  |  |
| --- | --- | --- |
| CP0WE\_X && rd\_X == 5’d14 (EPC) | | |
| 类型 | 描述 | FMUX\_EPC\_F |
| FEPC\_E | EPC\_E转发FMUX\_V2\_E | FMUX\_V2\_E |
| FEPC\_M | CP0内部转发 | CP0\_Q\_EPC |
| 0 | 不转发 | CP0\_Q\_EPC |

**暂停变化表**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| I | I+1 | I+2 |  | I | I+1 | I+2 |
| S1 | F21 |  |  | S3 | F1 |  |
| S2 | S3 | F1 |  | S4 | F32 |  |

**暂停操作**

1.数据寄存器

1)FR\_D暂停使能，保留取出指令的信息。

2)FR\_E清空数据，等价于将指令变成nop。

2.异常寄存器

1)ER\_D暂停使能，保留取指过程信息。

2)ER\_E流水ER\_D的数据，不清空数据，维护宏观PC，确保在暂停过程中出现中断时EPC正常写入数据。

注：如果发生取指异常或者未知指令异常，指令会被强制变成nop，不会引发转发或暂停，也不会在流水过程中引发其他的异常。

异常与中断设计

**中断与异常表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ExcCode | 助记符 | 描述 | 指令 | 流水级 | 发生情况 |
| 0 | Int | 外部中断 | (Any) | M | 设备中断请求 |
| 4 | AdEL | 取指异常 | (Any) | F | PC地址未字对齐或 |
| (Any) | F | PC地址不在0x3000-0x6ffc |
| 取数异常 | LOAD | E | 计算地址时加法溢出 |
| lw | M | 地址未字对齐 |
| lh | M | 地址未半字对齐 |
| LOAD | M | 地址不在存取合法地址区段 |
| lh, lb | M | 地址在TC地址区段 |
| 5 | AdES | 存数异常 | STORE | E | 计算地址时加法溢出 |
| sw | M | 地址未字对齐 |
| sh | M | 地址未半字对齐 |
| STORE | M | 地址不在存取合法地址区段 |
| STORE | M | 地址为TC\_Count地址 |
| sh, sb | M | 地址在TC地址区段 |
| 8 | Syscall | 系统调用 | syscall | D | 执行syscall指令 |
| 10 | RI | 未知指令 | (Other) | D | 执行未知指令 |
| 12 | Ov | 算术溢出 | add  addi  sub | E | 执行考虑溢出的算数指令时发生溢出 |

注：

1.流水级为判断是否发生该异常的流水级。

2.多种中断异常时，外部中断覆盖所有异常，流水级在前的异常覆盖流水级在后的异常。

3.发生取指异常时，受害指令是PC值不正确的指令，需要向EPC写入不正确的地址

4.发生取指异常时，视为nop直到提交到CP0。

注意官方Testbench给出的取指相关语句是：

reg [31:0] inst[0:5119];

assign i\_inst\_rdata = inst[((i\_inst\_addr - 32'h3000) >> 2) % 5120];

虽然在初始化时会把inst全部清零，但是在发生取值异常时，**取出的指令不一定是nop，需要手动置为nop**。

5.存取合法地址区段共4段：

0x0000-0x2fff、0x7f00-0x7f0b、0x7f10-0x7f1b、0x7f20-0x7f23

TC地址区段共2段：

0x7f00-0x7f0b、0x7f10-0x7f1b

TC\_Count地址区段共2段：

0x7f08-0x7f0b、0x7f18-0x7f1b

6.对于未知指令的判断仅需考虑opcode和R型指令的funct。测试程序保证当指令的opcode正确，且如为R型指令其funct也正确时，指令其他部分格式一定正确。保证未知指令的opcode和funct组合一定没有在MARS基本指令集中出现。

7.发生未知指令异常时，视为nop直到提交到CP0。

**CP0寄存器**

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 编号 | 读写性 | 功能 |
| SR（State Register） | 12 | R/W | 配置异常的功能 |
| Cause | 13 | R | 记录异常发生的原因和情况 |
| EPC | 14 | R/W | 记录异常处理结束后需要返回的PC |

注：

1.CP0寄存器的初始值均为0，**未实现位和未实现寄存器始终保持0**。

2.测试程序保证不会写入Cause。

SR：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31:16 | 15:10 | 9:2 | 1 | 0 |
| 0 | IM | 0 | EXL | IE |
| Always 0 | Interrupt Mask | Always 0 | Exception Level | Interrupt Enable |

注：当进入中断或异常状态时，由**硬件**将EXL置位，强制进入核心态并屏蔽中断信号；当退出中断或异常状态时，由**eret指令（软件）**将EXL清零。

Cause：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 | 30:16 | 15:10 | 9:7 | 6:2 | 1:0 |
| BD | 0 | IP | 0 | ExcCode | 0 |
| Branch Delay | Always 0 | Interrupt Pending | Always 0 | Exception Code | Always 0 |

注：

1.分支跳转指令无论跳转与否，延迟槽指令为受害指令时BD均需要置位。

2.无论是否允许中断IP域每周期都要写入外部硬件中断信号（HWInt）的值，是否进入中断或异常状态由CP0根据SR\_IM、SR\_EXL、SR\_IE判断。

EPC：

|  |
| --- |
| 31:0 |
| EPC |
| Exception Program Counter |

注：当该Cause\_BD为1时，即发生异常或中断时提交点的指令为分支跳转指令的延迟槽中的指令，EPC指向前一条指令（PC-4，即分支跳转指令），否则指向当前指令。

**CP0**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| RESET | I | 同步复位信号 |
| clk | I | 时钟信号 |
| WE | I | 写入使能信号 |
| A[4:0] | I | 操作的协处理器寄存器的编号 |
| D[31:0] | I | 写入协处理器寄存器的数据 |
| D\_BD | I | 当前M级指令是否为分支跳转指令的延迟槽指令 |
| D\_VPC[31:0] | I | 当前M级指令的地址，可能为错误的地址 |
| D\_ExcCode[4:0] | I | 当前M级指令产生的异常 |
| D\_HWInt[5:0] | I | 当前外部中断产生的情况 |
| EXL\_CLR | I | 下一周期清零SR\_EXL信号 |
| Req | O | 进入中断处理程序信号 |
| Q[31:0] | O | 读取的协处理器寄存器的值 |
| Q\_EPC[31:0] | O | 当前EPC寄存器的值 |
| Q\_EXL | O | 当前SR\_EXL的值 |

**ER\_D、ER\_E、ER\_M**

通用：

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| RESET | I | 同步复位信号 |
| clk | I | 时钟信号 |
| Req | O | 进入中断处理程序信号 |

以下端口均有I和O两个方向，I方向信号名以D\_开头，O方向信号名以Q\_开头。

ER\_D：

|  |  |
| --- | --- |
| 端口名 | 接入 |
| STALL\_EN\_N | CTRL\_S\_D\_EN\_N |
| ERET | CTRL\_D\_ERET |
| EPC[31:0] | CP0\_Q\_EPC |
| BD | CTRL\_D\_ISBRANCHJUMP |
| VPC[31:0] | IFU\_InstrAddr |
| ExcCode[4:0] | ECMUX\_F\_ExcCode\_F |

ER\_E：

|  |  |
| --- | --- |
| 端口名 | 接入 |
| BD | ER\_D\_BD |
| VPC[31:0] | ER\_D\_VPC |
| ExcCode[4:0] | ECMUX\_D\_ExcCode\_D |

ER\_M：

|  |  |
| --- | --- |
| 端口名 | 接入 |
| BD | ER\_E\_BD |
| VPC[31:0] | ER\_E\_VPC |
| ExcCode[4:0] | ECMUX\_E\_ExcCode\_E |

注：

1.各级ER中**VPC的复位值为32‘h0000\_3000**，维护宏观PC，确保在暂停过程中出现中断时EPC正常写入数据。

2.各级ER中ExcCode为5’d0时表示在该级之前没有发生异常。

**ECMUX\_F、ECMUX\_D、ECMUX\_E、ECMUX\_M、**

ECMUX\_F：

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| InstrAddr[31:0] | I | 即将进入流水线的指令地址 |
| ExcCode\_F[4:0] | O | F级产生的异常的ExcCode |
| AdEL\_F | O | F级是否产生取值异常 |

ECMUX\_D：

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| ExCode\_F[4:0] | I | F级产生的异常的ExcCode |
| SYSCALL | I | D级指令是否为syscall |
| UNKNOWN | I | D级指令是否为未知指令 |
| ExcCode\_D[4:0] | O | D级产生的异常的ExcCode |

ECMUX\_E：

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| ExCode\_D[4:0] | I | D级产生的异常的ExcCode |
| Overflow | I | E级指令是否发生算数溢出异常 |
| ISLOADSTORE | I | E级指令是否为系统桥操作类指令 |
| BridgeSel[3:0] | I | E级系统桥操作指令操作模式 |
| ExcCode\_E[4:0] | O | E级产生的异常的ExcCode |

ECMUX\_M：

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| ExCode\_E[4:0] | I | E级产生的异常的ExcCode |
| AdE[1:0] | I | M级指令是否发生取数异常(2’b10)或存数异常(2’b11) |
| ExcCode\_M[4:0] | O | M级产生的异常的ExcCode |

**宏观PC**

宏观PC为M级正在处理的指令。从ER\_M\_VPC引出。

**进入中断处理程序操作**

1.D\_ExcCode不为0或D\_HWInt和SR\_IM与运算后不为0，且SR\_EXL为0，SR\_IE为1，产生Req信号。

2.1根据多种中断异常时，外部中断覆盖所有异常，流水级在前的异常覆盖流水级在后的异常的原则确定ExcCode，下一周期写入Cause\_ExcCode。

2.2根据D\_BD和D\_VPC确定EPC，下一周期写入Cause\_BD和EPC。

2.3让SR\_EXL下一周期置位。

2.4禁止对于协处理器寄存器的写入。

3.让NPC输出0x4180，使PC下一周期跳转到中断处理程序开头位置。

4.让FR\_D、FR\_E、FR\_M、FR\_W下一周期清空数据，即清空流水线。

5.让ER\_D、ER\_E、ER\_M的VPC下一周期变为0x4180，清空BD和ExcCode，维护宏观PC。

6.禁止MULT启动乘除法计算和HI/LO的写入，禁止BRIDGE的写入操作执行。

**离开中断处理程序操作**

1.执行到中断处理程序的eret，产生ERET信号。

2.让NPC输出EPC，使PC下一周期跳转到EPC的位置。

3.让FR\_D下一周期清空数据，等价于将指令变成nop，实现eret指令没有延迟槽。

4.让ER\_D的VPC下一周期变为EPC，清空BD和ExcCode，维护宏观PC，确保在暂停过程中出现中断时EPC正常写入数据。

5.流水eret信号到M级，通过EXL\_CLR将SR\_EXL位清零。

注：

1.允许测试程序从0x417c直接前进到0x4180，即在正常状态下执行异常处理程序，此种情况下不应有中断响应等其他行为，故是否产生eret信号要结合SR寄存器中的EXL位判断。

2.测试程序保证包含异常的情况不会由0x417c直接前进到0x4180且eret只会出现在中断处理程序中，即不会出现以下情况：

.ktext

……

eret

eret

正常状态执行到第一个eret时产生了中断，执行中断处理程序到第一个eret返回。执行序列为第一个eret（异常状态）->第一个eret（正常状态）->第二个eret（正常状态）。当第二个eret（正常状态）进入到D级时，第一个eret（正常状态）才刚进入到M级，还未完成对EXL的修改，这样第二个eret（正常状态）还是会引发跳转，最终导致死循环。

CPU层功能模块设计

**暴露至MIPS体系结构层的端口**

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 外部信号（I方向） | 内部信号（O方向） |
| CPU\_RESET | I | reset |  |
| CPU\_clk | I | clk |  |
| CPU\_macroscopic\_pc[31:0] | O |  | ER\_M\_VPC |
| CPU\_IFU\_InstrAddr[31:0] | O |  | IFU\_InstrAddr |
| CPU\_IFU\_Instr[31:0] | I | i\_inst\_rdata |  |
| CPU\_GRFWE\_W | O |  | GRFWE\_W |
| CPU\_GRF\_A3\_W[4:0] | O |  | GRF\_A3\_W |
| CPU\_GRF\_WD\_W[1:0] | O |  | MUX\_GRF\_WD\_W |
| CPU\_Exam\_InstrAddr\_W[31:0] | O |  | Exam\_InstrAddr\_W |
| CPU\_Req | O |  | CP0\_Req |
| CPU\_BridgeSel\_M[3:0] | O |  | BridgeSel\_M |
| CPU\_ISLOADSTORE\_M | O |  | ISLOADSTORE\_M |
| CPU\_BRIDGE\_A[31:0] | O |  | E\_RES\_M |
| CPU\_BRIDGE\_D[31:0] | O |  | FMUX\_V2\_M |
| CPU\_BRIDGE\_AdE[1:0] | I | BRIDGE\_AdE |  |
| CPU\_BRIDGE\_HWInt[5:0] | I | BRIDGE\_HWInt |  |
| CPU\_BRIDGE\_Q[31:0] | I | BRIDGE\_Q |  |

**IFU**

1. 描述

取指令模块。内有PC子模块、NPC子模块。接收分支跳转指令控制信号、中断处理程序进入退出控制信号和相关数据，输出指令地址，通过与外部ROM通信获取指令。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| RESET | I | 同步复位信号 |
| clk | I | 时钟信号 |
| STALL\_EN\_N | I | 暂停使能信号 |
| Req | I | 进入中断处理程序信号 |
| ERET | I | 退出中断处理程序信号 |
| NPCSel[1:0] | I | NPC子模块控制信号 |
| BranchComp | I | NPC子模块分支指令控制信号 |
| offset[15:0] | I | 分支指令跳转偏移 |
| instr\_index[25:0] | I | j、jal指令跳转地址 |
| instr\_register[31:0] | I | jr、jalr指令跳转寄存器 |
| EPC[31:0] | I | 退出中断处理程序返回地址 |
| InstrAddr[31:0] | O | 即将进入流水线的指令地址 |

1. 控制信号

BranchComp：接COMP的BranchComp，表示分支指令的条件是否满足。

NPCSel：

|  |  |  |
| --- | --- | --- |
| 指令（信号）名 | 取值 | 描述 |
| 其他 | 00 | PC←PC+4 |
| BRANCHE、BRANCHZ | 01 | 使用offset，结合BranchComp判断 |
| j、jal | 10 | 使用instr\_index |
| jr、jalr | 11 | 使用instr\_register |

注：Req和ERET对NPC的影响的优先级高于NPCSel。

1. 关联MUX

EMUX\_Instr，控制信号ECMUX\_F\_AdEL\_F

|  |  |  |
| --- | --- | --- |
| 控制信号取值 | 数据 | 描述 |
| 0 | IFU\_Instr | 没有产生取指异常，正常输出指令 |
| 1 | 32’h00000000 (nop) | 产生取指异常，强制改成nop |

**IFU\_PC**

1. 描述

程序计数器子模块。在复位信号有效时复位为32'h00003000。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| RESET | I | 同步复位信号 |
| clk | I | 时钟信号 |
| STALL\_EN\_N | I | 暂停使能信号 |
| D[31:0] | O | 下一指令地址 |
| Q[31:0] | O | 即将进入流水线的指令地址 |

**IFU\_NPC**

1. 描述

下一条指令地址计算子模块。输入IFU接收分支跳转指令控制信号、中断处理程序进入退出控制信号和相关数据，计算出下一条指令的地址。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| Req | I | 进入中断处理程序信号 |
| ERET | I | 退出中断处理程序信号 |
| PC[31:0] | 1 | 即将进入流水线的指令地址 |
| NPCSel[1:0] | I | 同IFU |
| BranchComp | I | 同IFU |
| offset[15:0] | I | 同IFU |
| instr\_index[25:0] | I | 同IFU |
| instr\_register[31:0] | I | 同IFU |
| EPC[31:0] | I | 同IFU |
| NPC[31:0] | O | 下一指令地址 |

**SPL**

1. 描述

指令分线器模块。按照三种指令的格式将一条指令分为不同的部分，供其他模块使用。若M级发生未知指令异常，将指令强制改成nop。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| force\_nop | I | 发生未知指令异常，强制改成nop |
| Instr[31:0] | 1 | 当前指令 |
| O\_opcode[5:0] | O | 未经强制nop的指令的操作数 |
| O\_funct[5:0] | O | 未经强制nop的R型指令的funct |
| O\_rs[4:0] | O | 未经强制nop的R、I型指令的rs |
| O\_rt[4:0] | O | 未经强制nop的R、I型指令的rt |
| rs[4:0] | O | 经强制nop的R、I型指令的rs |
| rt[4:0] | O | 经强制nop的R、I型指令的rt |
| rd[4:0] | O | 经强制nop的R型指令的rd |
| shamt[4:0] | O | 经强制nop的R型指令的shamt |
| imm16[15:0] | O | 经强制nop的I型指令的立即数 |
| instr\_index[25:0] | O | 经强制nop的J型指令的跳转地址 |

**GRF**

**(GRF\_R、GRF\_W)**

1. 描述

寄存器堆模块，共有31个寄存器（0号寄存器直接接地）。通过A1、A2输入地址，可以取出指定寄存器的完成内部转发的数据。当WE使能时，通过A3输入地址，D输入数据，可以修改指定寄存器存储的数据并进行内部转发。在流水线设计时分为两个部分体现。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| RESET | I | 同步复位信号 |
| clk | I | 时钟信号 |
| WE | I | 写入使能信号 |
| A1[4:0] | I | 读取的第一个寄存器的编号 |
| A2[4:0] | I | 读取的第二个寄存器的编号 |
| A3[4:0] | I | 写入的寄存器的编号 |
| WD[31:0] | I | 写入寄存器的数据 |
| V1[31:0] | O | 读取的第一个寄存器的完成内部转发的值 |
| V2[31:0] | O | 读取的第二个寄存器的完成内部转发的值 |

1. 控制信号

GRFWE：

|  |  |  |
| --- | --- | --- |
| 指令信号名 | 取值 | 描述 |
| 其他 | 0 | 不涉及寄存器写入的指令 |
| ALREG、ALIMM、EXTLUI | 1 |  |
| MLFROM | 1 |  |
| LOAD | 1 |  |
| JUMPW、JUMPRW | 1 |  |
| MFCOP | 1 |  |

1. 关联MUX

MUX\_GRF\_A3\_D，控制信号CTRL\_D\_GRF\_A3\_D\_Sel

|  |  |  |
| --- | --- | --- |
| 指令信号名 | 取值 | 描述 |
| 其他 | 00 | 接入rd |
| ALIMM、EXTLUI | 10 | 接入rt |
| LOAD、MFCOP | 10 | 接入rt |
| JUMPW | 11 | 固定32’d31 |

MUX\_GRF\_WD\_W，控制信号GRF\_WD\_W\_Sel\_W

|  |  |  |
| --- | --- | --- |
| 指令信号名 | 取值 | 描述 |
| 其他 | 00 | 接入E\_RES\_W |
| LOAD、MFCOP | 01 | 接入M\_RES\_W |
| EXTLUI | 10 | 接入ext32\_W |
| JUMPW、JUMPRW | 11 | 接入pc8\_W |

**COMP**

1. 描述

比较模块，完两数相等、不等，一数有符号小于、小于等于、大于、大于等于零。比较结果用于判断分支指令是否执行。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| CompSel[2:0] | I | 比较方式控制信号 |
| A[31:0] | I | 第一个操作数 |
| B[31:0] | I | 第二个操作数 |
| BranchComp | O | 比较结果 |

1. 控制信号

CompSel：

|  |  |  |
| --- | --- | --- |
| 指令名 | 取值 | 描述 |
| bltz(00000) | 000 | 有符号小于0 |
| bgez(00001) | 001 | 有符号大于等于0 |
| beq(000100) | 100 | 两数相等 |
| bne(000101) | 101 | 两数不等 |
| blez(000110) | 110 | 有符号小于等于0 |
| bgtz(000111) | 111 | 有符号大于0 |

**EXT**

1. 描述

位扩展模块，将16位立即数无符号扩展、有符号扩展、右补16位0为32位。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| EXTSel[1:0] | I | 扩展方式控制信号 |
| imm16[15:0] | I | 16位立即数 |
| ext32[31:0] | O | 32位扩展结果 |

1. 控制信号

EXTSel：

|  |  |  |
| --- | --- | --- |
| 指令名 | 取值 | 描述 |
| 其他 | 00 | 有符号拓展 |
| andi、ori、xori | 01 | 无符号拓展 |
| lui | 10 | 右补16位0 |

**ALU**

1. 描述

算数逻辑模块，完成算数运算（不考虑算数溢出、考虑算数溢出的加、减）、逻辑运算（与、或、异或、或非）、移位运算（左移、逻辑右移、算数右移）、比较运算（两数有符号小于，无符号小于）。计算结果用于保存到寄存器。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| OPSel[1:0] | I | OP输出控制信号 |
| FuncSel[1:0] | I | 计算方式控制信号 |
| A[31:0] | I | 第一个操作数 |
| B[31:0] | I | 第二个操作数 |
| shamt[4:0] | I | 移位立即数 |
| OP[31:0] | O | 计算结果 |
| Overflow | O | 计算结果是否出现算数溢出 |

1. 控制信号

OPSel：

|  |  |  |
| --- | --- | --- |
| 指令（信号）名 | 取值 | 描述 |
| add、addu、sub、subu、addi、addiu | 00 | 算数运算 |
| LOAD、STORE | 00 | 算数运算 |
| and、or、xor、nor、andi、ori、xori | 01 | 逻辑运算 |
| sll、srl、sra、sllv、srlv、srav | 10 | 移位运算 |
| slt、sltu、slti、sltiu | 11 | 比较运算 |

FuncSel：接CTRL，根据OPSel选择。

OPSel == 00：

|  |  |  |
| --- | --- | --- |
| 指令名 | 取值 | 描述 |
| addu、addiu | 00 | 不考虑算数溢出加法 |
| subu | 01 | 不考虑算数溢出减法 |
| add、addi | 10 | 考虑算数溢出加法 |
| LOAD、STORE | 10 | 考虑算数溢出加法 |
| sub | 11 | 考虑算数溢出减法 |

注：默认值应为00，避免意外触发算数溢出异常。

OPSel == 01：ALREG接funct[1:0]、ALIMM时接opcode[1:0]

|  |  |  |
| --- | --- | --- |
| 指令名 | 取值 | 描述 |
| and(100100)、andi(001100) | 00 | 与 |
| or(100101)、ori(001101) | 01 | 或 |
| xor(100110)、xori(001110) | 10 | 异或 |
| nor(100111) | 11 | 或非 |

OPSel == 10：接funct[1:0]。

|  |  |  |
| --- | --- | --- |
| 指令名 | 取值 | 描述 |
| sll(000000)、sllv(000100) | 00 | 逻辑左移 |
| srl(000010)、srlv(000110) | 10 | 逻辑右移 |
| sra(000011)、srav(000111) | 11 | 算数右移 |

OPSel == 11：（只用设置01）

|  |  |  |
| --- | --- | --- |
| 指令名 | 取值 | 描述 |
| slt、slti | 00 | 有符号两数小于 |
| sltu、sltiu | 01 | 无符号两数小于 |

1. 关联MUX

MUX\_ALU\_B\_E，控制信号ALU\_B\_E\_Sel\_E

|  |  |  |
| --- | --- | --- |
| 指令信号名 | 取值 | 描述 |
| 其他 | 0 | 接入GRF |
| ALIMM、LOAD、STORE | 1 | 接入EXT |

**MULT**

1. 描述

乘除法模块，执行乘法的时间为5个时钟周期，执行除法的时间为10个时钟周期（包含写入内部的HI、LO寄存器）。自1个时钟周期Start信号有效后的第1个clock上升沿开始乘除法运算，同时Busy置位。在运算结果保存到HI寄存器和LO寄存器后，Busy清零。写入HI、LO均只需1个时钟周期，类似GRF。读取HI、LO直接读取，类似ALU。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| RESET | I | 同步复位信号 |
| clk | I | 时钟信号 |
| Req | I | 进入中断处理程序信号 |
| ISMULTDIV | I | 是否为乘除指令信号 |
| MULTSel[1:0] | I | 乘除模块控制信号 |
| A[31:0] | I | 第一个操作数 |
| B[31:0] | I | 第二个操作数 |
| Start | O | 乘除计算启动信号，输出ISMULTDIV & MULTSel[2] |
| Busy | O | 乘除计算进行信号 |
| HILO[31:0] | O | 读取的HI、LO的结果 |

1. 控制信号

ISMULTDIV：指令信号名为MULTDIV、MLTO、MLFROM时为1.

MULTSel：接{funct[3], funct[1:0]}。

|  |  |  |
| --- | --- | --- |
| 指令名 | 取值 | 描述 |
| mult(011000) | 100 | 准备执行**有符号**乘法操作 |
| multu(011001) | 101 | 准备执行**无符号**乘法操作 |
| div(011010) | 110 | 准备执行**有符号**除法操作 |
| divu(011011) | 111 | 准备执行**无符号**除法操作 |
| mfhi(010000) | 000 | HILO输出HI |
| mflo(010010) | 010 | HILO输出LO |
| mthi(010001) | 001 | 准备将A写入HI |
| mtlo(010011) | 011 | 准备将A写入LO |

1. 关联MUX

MUX\_E\_RES\_E，控制信号E\_RES\_E\_Sel\_E：

|  |  |  |
| --- | --- | --- |
| 指令信号名 | 取值 | 描述 |
| 其他 | 0 | 接入ALU\_OP |
| MLFROM | 1 | 接入MULT\_HILO |

1. 中断异常行为

1.MULTDIV在E级启动了乘法运算，流水到M级时产生了中断，此时无需停止乘法计算。

2.MLTO在E级修改了HI/LO，流水到M级时产生了中断，此时无需恢复HI/LO的值。

3.MULTDIV在E级（还未改变MDU状态），受害指令在M级，此时不应开始乘法计算。

4.MLTO在E级（还未改变MDU状态），受害指令在M级，此时不应修改HI/LO的值。

**（BRIDGE）**

1. 关联MUX

MUX\_M\_RES\_M，控制信号ISLOADSTORE\_M：

|  |  |  |
| --- | --- | --- |
| 指令信号名 | 取值 | 描述 |
| 其他 | 0 | 接入CP0\_Q |
| LOAD、STORE | 1 | 接入CPU\_BRIDGE\_Q |

**FR\_D、FR\_E、FR\_M、FR\_W**

1. 描述

D、E、M、W段流水线寄存器。

1. 接口

通用：

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| RESET | I | 同步复位信号 |
| clk | I | 时钟信号 |
| Req | I | 进入中断处理程序信号 |

以下端口均有I和O两个方向，I方向信号名以D\_开头，O方向信号名以Q\_开头。

FR\_D：

|  |  |
| --- | --- |
| 端口名 | 接入 |
| STALL\_EN\_N | CTRL\_S\_D\_EN\_N |
| ERET | CTRL\_D\_ERET |
| Instr[31:0] | EMUX\_Instr |
| InstrAddr[31:0] | IFU\_InstrAddr |

注：FR\_D的ERET是控制信号。

FR\_E：

|  |  |
| --- | --- |
| 端口名 | 接入 |
| STALL\_RESET | CTRL\_S\_FR\_E\_RESET |
| ISLOADSTORE | CTRL\_D\_ISLOADSTORE |
| GRFWE | CTRL\_D\_GRFWE |
| BridgeSel[3:0] | CTRL\_D\_BridgeSel |
| FuncSel[1:0] | CTRL\_D\_FuncSel |
| OPSel[1:0] | CTRL\_D\_OPSel |
| GRF\_WD\_W\_Sel[1:0] | CTRL\_D\_GRF\_WD\_W\_Sel |
| ALU\_B\_E\_Sel | CTRL\_D\_ALU\_B\_E\_Sel |
| E\_RES\_E\_Sel | CTRL\_D\_E\_RES\_E\_Sel |
| MULTSel[2:0] | CTRL\_D\_MULTSel |
| ISMULTDIV | CTRL\_D\_ISMUTLDIV |
| CP0WE | CTRL\_D\_CP0WE |
| ERET | CTRL\_D\_ERET |
| V1[31:0] | FMUX\_V1\_D |
| V2[31:0] | FMUX\_V2\_D |
| rd[4:0] | SPL\_rd |
| shamt[4:0] | SPL\_shamt |
| GRF\_A3[4:0] | MUX\_GRF\_A3\_D |
| ext32[31:0] | EXT\_ext32 |
| pc8[31:0] | pc8\_D |
| FMUX\_V1\_E\_Sel[1:0] | CTRL\_F\_FMUX\_V1\_E\_Sel |
| FMUX\_V2\_E\_Sel[1:0] | CTRL\_F\_FMUX\_V2\_E\_Sel |
| FMUX\_V2\_M\_Sel | CTRL\_F\_FMUX\_V2\_M\_Sel |

注：FR\_E的ERET是流水数据。

FR\_M：

|  |  |
| --- | --- |
| 端口名 | 接入 |
| ISLOADSTORE | ISLOADSTORE\_E |
| GRFWE | GRFWE\_E |
| BridgeSel[3:0] | BridgeSel\_E |
| GRF\_WD\_W\_Sel[1:0] | GRF\_WD\_W\_Sel\_E |
| CP0WE | CP0WE\_E |
| ERET | ERET\_E |
| V2[31:0] | FMUX\_V2\_E |
| rd[4:0] | rd\_E |
| E\_RES[31:0] | MUX\_E\_RES\_E |
| GRF\_A3[4:0] | GRF\_A3\_E |
| ext32[31:0] | ext32\_E |
| pc8[31:0] | pc8\_E |
| FMUX\_V2\_M\_Sel | FMUX\_V2\_M\_Sel\_E |

FR\_W：

|  |  |
| --- | --- |
| 端口名 | 接入 |
| GRFWE | GRFWE\_M |
| GRF\_WD\_W\_Sel[1:0] | GRF\_WD\_W\_Sel\_M |
| E\_RES[31:0] | E\_RES\_M |
| M\_RES[31:0] | MUX\_M\_RES\_M |
| GRF\_A3[4:0] | GRF\_A3\_M |
| ext32[31:0] | ext32\_M |
| pc8[31:0] | pc8\_M |
| Exam\_InstrAddr[31:0] | ER\_M\_VPC |

**CTRL\_Decoder**

1. 描述

译码器模块。通过opcode、funct、rs和rt判断指令的类型，输出各个模块的控制信号。

1. 接口

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 端口名 | 方向 |
| opcode[5:0] | I | MULTSel[2:0] | O |
| funct[5:0] | I | ISMULTDIV | O |
| rs[4:0] | I | ISLOADSTORE | O |
| rt[4:0] | I | BridgeSel[3:0] | O |
| CP0\_Q\_EXL | I | CP0WE | O |
| SYSCALL | O | GRFWE | O |
| UNKNOWN | O | GRF\_A3\_D\_Sel[1:0] | O |
| ISBRANCHJUMP | O | ALU\_B\_E\_Sel | O |
| ERET | O | E\_RES\_E\_Sel | O |
| CompSel[2:0] | O | GRF\_WD\_W\_Sel[1:0] | O |
| EXTSel[1:0] | O | Tuse\_rs[1:0] | O |
| NPCSel[1:0] | O | Tuse\_rt[1:0] | O |
| OpSel[1:0] | O |  |  |
| FuncSel[1:0] | O |  |  |

**CTRL\_Stall**

1. 描述

暂停控制器模块。判断是否需要暂停，若需要暂停则产生暂停信号。

1. 接口

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 端口名 | 方向 |
| Tuse\_rs[1:0] | I | GRF\_A3\_E[4:0] | I |
| Tuse\_rt[1:0] | I | GRF\_A3\_M[4:0] | I |
| SPL\_rs[4:0] | I | ISMULTDIV | I |
| SPL\_rt[4:0] | I | MULT\_Start | I |
| GRFWE\_E | I | MULT\_Busy | I |
| GRFWE\_M | I | IFU\_EN\_N | O |
| GRF\_WD\_W\_Sel\_E[1:0] | I | D\_EN\_N | O |
| GRF\_WD\_W\_Sel\_M[1:0] | I | FR\_E\_RESET | O |

**CTRL\_Forward**

1. 描述

转发控制器模块。判断是否需要转发，若需要转发转发什么数据。

1. 接口

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 端口名 | 方向 |
| Tuse\_rs[1:0] | I | GRF\_A3\_E[4:0] | I |
| Tuse\_rt[1:0] | I | GRF\_A3\_M[4:0] | I |
| SPL\_rs[4:0] | I | FMUX\_V1\_D\_Sel[2:0] | O |
| SPL\_rt[4:0] | I | FMUX\_V2\_D\_Sel[2:0] | O |
| GRFWE\_E | I | FMUX\_V1\_E\_Sel[1:0] | O |
| GRFWE\_M | I | FMUX\_V2\_E\_Sel[1:0] | O |
| GRF\_WD\_W\_Sel\_E[1:0] | I | FMUX\_V2\_M\_Sel | O |
| GRF\_WD\_W\_Sel\_M[1:0] | I |  |  |

**MIPS体系结构层功能模块设计**

**暴露至外部设备层的端口**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 内部信号 |
| clk | I | CPU\_clk、TC0\_clk、TC1\_clk |
| reset | I | CPU\_RESET、TC0\_RESET、TC1\_RESET |
| interrupt | I | BRIDGE\_HWInt\_2 |
| macroscopic\_pc[31:0] | O | CPU\_macroscopic\_pc |
| i\_inst\_addr[31:0] | O | CPU\_IFU\_InstrAddr |
| i\_inst\_rdata[31:0] | I | CPU\_IFU\_Instr |
| m\_data\_addr[31:0] | O | CPU\_BRIDGE\_A |
| m\_data\_rdata[31:0] | I | BRIDGE\_rdata\_DM |
| m\_data\_wdata[31:0] | O | BRIDGE\_wdata |
| m\_data\_byteen[3:0] | O | BRIDGE\_byteen\_DM |
| m\_int\_addr[31:0] | O | CPU\_BRIDGE\_A |
| m\_int\_byteen[3:0] | O | BRIDGE\_byteen\_2 |
| m\_inst\_addr[31:0] | O | CPU\_macroscopic\_pc |
| w\_grf\_we | O | CPU\_GRFWE\_W |
| w\_grf\_addr[4:0] | O | CPU\_GRF\_A3\_W |
| w\_grf\_wdata[31:0] | O | CPU\_GRF\_WD\_W |
| w\_inst\_addr[31:0] | O | CPU\_Exam\_InstrAddr\_W |

**BRIDGE**

1. 描述

系统桥模块。连接CPU与DM、Interrupt Generator、Timer0、Timer1等外部设备，将LOAD、STORE指令根据地址图分发到各个外部设备，判断是否产生AdE异常，收集外部设备的中断请求并集中发送给CPU。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| ISLOADSTORE | I | 是否为系统桥操作类指令 |
| BridgeSel[3:0] | I | 系统桥操作指令操作模式 |
| Req | I | 进入中断处理程序信号 |
| A[31:0] | I | 读取或写入的地址 |
| D[31:0] | I | 原始的写入数据 |
| rdata\_DM[31:0] | I | DM原始的读取数据 |
| rdata\_(0-5)[31:0] | I | 共6个，外部设备原始的读取数据 |
| HWInt\_(0-5) | I | 共6个，外部设备的中断信号 |
| wdata[31:0] | O | 处理后的写入数据 |
| byteen\_DM[3:0] | O | DM字节使能信号 |
| byteen\_(0-5)[3:0] | O | 共6个，外部设备的字节使能信号 |
| AdE[1:0] | O | 是否产生AdE异常以及产生种类 |
| HWInt\_Hub[5:0] | O | 收集的所有外部设备HWInt中断信号 |
| Q[31:0] | O | 处理后的读取数据 |

注：Req高电平时，即将进入中断处理程序时，禁止写入。

1. 控制信号

ISLOADSTORE：指令信号名为LOAD、STORE时为1。只有其为1时才可以执行判断异常和写入操作。

BridgeSel：接opcode[3:0]。

|  |  |  |
| --- | --- | --- |
| 指令名 | 取值 | 描述 |
| lb(100000) | 0000 | 以byte读取，结果有符号拓展 |
| lh(100001) | 0001 | 以halfword读取，结果有符号拓展 |
| lw(100011) | 0011 | 以word读取 |
| lbu(100100) | 0100 | 以byte读取，结果无符号拓展 |
| lhu(100101) | 0101 | 以halfword读取，结果无符号拓展 |
| sb(101000) | 1000 | 以byte写入 |
| sh(101001) | 1001 | 以halfword写入 |
| sw(101011) | 1011 | 以word写入 |

1. 输出信号

byteen\_DM、byteen\_(0-5)：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令名 | BridgeSel | A[1:0] | byteen | A[1:0] | byteen | A[1:0] | byteen | A[1:0] | byteen |
| 其他 |  | 00 | 0000 | 01 | 0000 | 10 | 0000 | 11 | 0000 |
| sb | 1000 | 0001 | 0010 | 0100 | 1000 |
| sh | 1001 | 0011 | / | 1100 | / |
| sw | 1011 | 1111 | / | / | / |

注：字节使能信号表示D的哪些字节要被写入，若要将低位的数据写入高位**需要进行移位**。

AdE：

|  |  |
| --- | --- |
| 异常类型 | AdE |
| 无异常 | 00 |
| AdEL | 10 |
| AdES | 11 |

1. 地址图

|  |  |  |
| --- | --- | --- |
| 设备 | 地址范围 | 说明 |
| 数据存储器（DM） | 0x0000\_0000∼0x0000\_2FFF |  |
| 指令存储器（IM） | 0x0000\_3000∼0x0000\_6FFF |  |
| 计时器0寄存器（Timer0） | 0x0000\_7F00∼0x0000\_7F0B | 计时器0共有3个寄存器 |
| 计时器1寄存器（Timer1） | 0x0000\_7F10∼0x0000\_7F1B | 计时器1共有3个寄存器 |
| 中断发生器响应地址 | 0x0000\_7F20∼0x0000\_7F23 |  |

注：测试程序只会通过指令sb $0, 0x7f20($0)访问中断发生器以模拟响应中断，且只会在中断处理程序中访问。

1. 外部设备连接

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 信号路 | 外部设备 | rdata | byteen | HWInt |
| DM | DM | m\_data\_rdata | m\_data\_byteen | / |
| 0 | Timer0 | TC0\_rdata | TC0\_byteen | TC0\_IRQ |
| 1 | Timer1 | TC1\_rdata | TC1\_byteen | TC1\_IRQ |
| 2 | Interrupt Generator | 接地 | m\_int\_byteen | interrupt |
| 3 | 无 | 接地 | 输出0，浮空 | 接地 |
| 4 | 无 | 接地 | 输出0，浮空 | 接地 |
| 5 | 无 | 接地 | 输出0，浮空 | 接地 |

注：Interrupt Generator并没有真正的存储单元，规定读出的数据始终保持0。

**Timer0、Timer1**

1. 描述

计时器模块。由课程组设计，有两种模式。模式0用于产生定时中断，模式1用于产生周期性脉冲，具体见课程组提供的PDF文件《COCO定时器设计规范》。

注：TC的寄存器的值只允许以字的形式读取写入，否则将引发AdE错误。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| RESET | I | 同步复位信号 |
| clk | I | 时钟信号 |
| byteen[3:0] | I | 字节使能信号 |
| addr[31:0] | I | 操作的计时器寄存器的地址 |
| wdata[31:0] | I | 写入计时器寄存器的数据 |
| rdata[31:0] | O | 读取的计时器寄存器的值 |
| IRQ | O | 中断请求信号 |

P7思考题

1. 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？

通过外部中断的机制。鼠标或键盘内部有一些寄存器，保存鼠标或键盘最近一段时间的操作信息。当产生了新的操作时，产生外部中断，CPU通过访问特定的地址可以访问到鼠标或键盘的寄存器，经过分析可以得到鼠标或键盘的操作。

1. 请思考为什么我们的CPU处理中断异常必须是已经指定好的地址？如果你的CPU支持用户自定义入口地址，即处理中断异常的程序由用户提供，其还能提供我们所希望的功能吗？如果可以，请说明这样可能会出现什么问题？否则举例说明。（假设用户提供的中断处理程序合法）

## 理论上可以实现希望的功能。但是若实现不知道中断处理程序的地址，可能在正常执行程序时会对存放中断处理程序的区域进行误操作读写，从而使中断处理程序被破坏。且支持自定义中断处理程序的入口地址也会增加硬件设计难度。

1. 为何与外设通信需要Bridge？

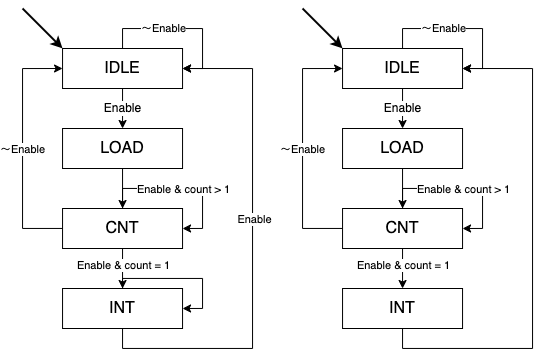
使CPU以统一的方式访问不同种类的外部设备。在增加新的外部设备时也更加方便，不需要影响CPU的既有结构，符合“高内聚低耦合”的原则。

1. 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移图。

模式0：当计数器倒计数为0后，计数器停止计数，持续产生中断信号，当使能被再次置位后，初值寄存器值再次被加载至计数器，计数器重新启动倒计数。通常用于产生定时中断。

模式1：当计数器倒计数为0后，只产生一周期中断信号，初值寄存器值被自动加载至计数器，计数器继续倒计数。通常用于产生周期性脉冲。

模式0状态转移图如下图左图所示，模式0状态转移图如下图右图所示。



1. 倘若中断信号流入的时候，在检测宏观PC的一级如果是一条空泡（你的CPU该级所有信息均为空）指令，此时会发生什么问题？在此例基础上请思考：在P7中，清空流水线产生的空泡指令应该保留原指令的哪些信息？

丢失是否为延迟槽指令和受害PC信息，将导致EPC写入错误的数值。实际上，在进入异常中断程序而清空流水线的情况下，无需保留任何信息，但是要注意将受害PC置位中断处理程序的开始的地址（在P7中即0x4180）。若是因为流水线暂停而导致在E级插入的nop，应该要注意让E级流水D级是否为延迟槽指令和受害PC信息。

1. 为什么jalr指令为什么不能写成jalr $31, $31？

若jalr读写同一寄存器，则会将PC + 8写入该寄存器。离开中断处理程序后再次执行这条指令的效果与之前不一致。

在MIPS指令集中也有相关约束：Register specifiers rs and rd must not be equal, because such an instruction does not have the same effect when reexecuted. The result of executing such an instruction is **UNPREDICTABLE**. This restriction permits an exception handler to resume execution by re-executing the branch when an exception occurs in the branch delay slot.