流水线CPU

设计文档

2.0

贾博驿

2023年11月19日

设计实现指令

**共50条**

**实际指令47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 |

以下除分支与跳转类指令，所有的RTL省略PC←PC+4。

**算数与逻辑类**

1. 加减法类
2. R型

|  |  |  |
| --- | --- | --- |
| 名称 | funct | RTL |
| add | 100000 | GPR[rd] ← GPR[rs] + GPR[rt] |
| addu | 100001 | GPR[rd] ← GPR[rs] + GPR[rt] |
| sub | 100010 | GPR[rd] ← GPR[rs] - GPR[rt] |
| subu | 100011 | GPR[rd] ← GPR[rs] - GPR[rt] |

注：实现的add实际上为addu、sub实际上为subu。二者均按照无符号指令处理。

1. I型

|  |  |  |
| --- | --- | --- |
| 名称 | opcode | RTL |
| addi | 001000 | GPR[rt] ← GPR[rs] + sign\_extend(immediate) |
| addiu | 001001 | GPR[rt] ← GPR[rs] + sign\_extend(immediate) |

注：实现的addi实际上为addiu。按照无符号指令处理。

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：根据寄存器移位指令只依照寄存器值的低五位移位，高位的值不影响。

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跳转指令由于分支指令存在应该加上PC + 8，而非单周期的PC + 4。

**内存操作类（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 | half ← addr1  GPR[rt] ← sign\_extend(memory[addr]15+8\*half..8\*half) |
| lw | 100011 | GPR[rt] ← memory[addr] |
| lbu | 100100 | byte ← addr1..0  GPR[rt] ← zero\_extend(memory[addr]7+8\*byte..8\*byte) |
| lhu | 100101 | 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 | memory[addr]15..0 ← GPR[rt]15..0 |
| sw | 101011 | memory[addr]31..0 ← GPR[rt] |

注：针对halfword操作需要addr为2的非负整数倍，针对word操作需要addr为4的非负整数倍。针对byte和halfword操作时只修改对应byte和halfword的值，同一word的其他部分不变。

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 |
| 乘除寄存器写入 | MLTO | 2 | mthi, mtlo |
| 乘除寄存器读取 | MLFROM | 2 | mfhi, mflo |
| 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 |
| 读取内存 | LOAD | 5 | lb, lh, lw, lbu, lhu |
| 写入内存 | STORE | 3 | sb, sh, sw |

注：不包括j指令，因为j指令不读取寄存器也不写入寄存器。

**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) |
| MLTO | 1 | (3) |  | (00) | (0) | (0) | (0) |
| MLFROM | (3) | (3) |  | 00 | 1 | 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) |

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 = 0，虽然使用ALU数据但无意义。

**转发与暂停**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SPL\_rs != 5’d0 && GRFWE\_X && SPL\_rs == GRF\_A3\_X | | | | | | | | | |
| 流水段(X) | E | | | | M | | | | 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 | | | | 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 | | | | | | | | | |

**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OP\_M |  | 11 |
| F43 | V1\_D转发ext32\_E | 110 |  |
| F44 | V1\_D转发pc8\_E | 111 |  |
| F32 | V1\_E转发DM\_Q\_W |  | 10 |
| F21 | V1\_D转发OP\_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\_OP\_M\_Sel |
| S | 暂停 |  |  |  |
| F62 | DM\_D\_M转发DM\_Q\_W |  |  | 1 |
| F51 | V2\_E转发OP\_M |  | 11 |  |
| F43 | V2\_D转发ext32\_E | 110 |  |  |
| F44 | V2\_D转发pc8\_E | 111 |  |  |
| F32 | V2\_E转发DM\_Q\_W |  | 10 |  |
| F21 | V2\_D转发OP\_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 |

**暂停变化表**

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

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。

3.当MULT\_Busy 信号或MULT\_Start\_E信号为1时，MLUTDIV、MLTO、MLFROM指令均被阻塞在D流水级，即SMULTDIV暂停。

4.未标注的信号取值与不转发相同。

模块设计

**IFU**

1. 描述

取指令模块。内有PC子模块、NPC子模块。接收分支和跳转指令的控制信号和数据并实现跳转。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 |
| 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 |
| InstrAddr[31:0] | O | 即将进入流水线的指令地址，接入顶层i\_inst\_addr |

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 |

**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接收的分支和跳转指令的控制信号和数据，使用MUX，计算出下一条指令的地址。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| 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 |
| 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。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| Instr[31:0] | 1 | 当前指令 |
| opcode[5:0] | O | 指令的操作数 |
| rs[4:0] | O | R、I型指令的rs |
| rt[4:0] | O | R、I型指令的rt |
| rd[4:0] | O | R型指令的rd |
| shamt[4:0] | O | R型指令的shamt |
| funct[5:0] | O | R型指令的funct |
| imm16[15:0] | O | I 型指令的立即数 |
| instr\_index[25:0] | O | 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，同时转发输出到顶层w\_grf\_we |
| 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，同时转发输出到顶层w\_grf\_addr |
| D[31:0] | I | 写入寄存器的数据，同时转发输出到顶层w\_grf\_wdata |
| 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 | 1 |  |
| ALIMM、EXTLUI | 1 |  |
| MLFROM | 1 |  |
| LOAD | 1 |  |
| JUMPW、JUMPRW | 1 |  |

1. 关联MUX控制信号

GRF\_A3\_D\_Sel：

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

GRF\_WD\_W\_Sel：

|  |  |  |
| --- | --- | --- |
| 指令信号名 | 取值 | 描述 |
| 其他 | 00 | 接入OP\_W |
| LOAD | 01 | 接入DM\_Q\_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。

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（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 |

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：（只用设置01）

|  |  |  |
| --- | --- | --- |
| 指令名 | 取值 | 描述 |
| add、addu、addi、addiu | 00 | 加法 |
| LOAD、STORE | 00 | 加法 |
| sub、subu | 01 | 减法 |

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控制信号

ALU\_B\_E\_Sel：

|  |  |  |
| --- | --- | --- |
| 指令信号名 | 取值 | 描述 |
| ALREG | 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置位为1。在运算结果保存到HI寄存器和LO寄存器后，Busy清为 0。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 |
| 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控制信号

OP\_E\_Sel：

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

**DM**

1. 描述

内存处理模块。将控制信号和数据处理为与外部RAM通信的合法信号。

1. 接口

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| WE | I | 写入使能信号 |
| DMSel[2:0] | I | 读取存储模式控制信号 |
| A[31:0] | I | 读取或写入的地址，同时转发输出到顶层m\_data\_addr |
| D[31:0] | I | 原始的写入数据 |
| rdata[31:0] | I | 原始的读取数据，接入顶层m\_data\_rdata |
| wdata[31:0] | O | 处理后的读取数据，接入顶层m\_data\_wdata |
| byteen[3:0] | O | 字节使能信号，接入顶层m\_data\_byteen |
| Q[31:0] | O | 处理后的读取数据 |

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。

1. 控制信号

DMWE：

|  |  |  |
| --- | --- | --- |
| 指令信号名 | 取值 | 描述 |
| 其他 | 0 | 不涉及写入寄存器的指令 |
| STORE | 1 |  |

DMSel：接opcode[2:0]。

|  |  |  |
| --- | --- | --- |
| 指令名 | 取值 | 描述 |
| lb(100000)、sb(101000) | 000 | 针对byte操作，**读取时**有符号拓展 |
| lh(100001)、sh(101001) | 001 | 针对halfword操作，**读取时**有符号拓展 |
| lw(100011)、sw(101011) | 011 | 针对word操作 |
| lbu(100100) | 100 | 针对byte操作，读取时无符号拓展 |
| lhu(100101) | 101 | 针对halfword操作，读取时无符号拓展 |

1. 输出控制信号

byteen：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令名 | DMWE | DMSel | A[1:0] | byteen |  |  |  |  |  |  |
| 其他 | 0 |  | 00 | 0000 | 01 | 0000 | 10 | 0000 | 11 | 0000 |
| sb | 1 | 000 | 0001 | 0010 | 0100 | 1000 |
| sh | 1 | 001 | 0011 | / | 1100 | / |
| sw | 1 | 011 | 1111 | / | / | / |

**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 |

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\_FR\_D\_EN |
| Instr[31:0] | 顶层i\_inst\_rdata |
| InstrAddr[31:0] | IFU\_InstrAddr |

FR\_E：

|  |  |
| --- | --- |
| 端口名 | 接入 |
| STALL\_RESET | CTRL\_S\_FR\_E\_RESET |
| DMWE | CTRL\_D\_DMWE |
| GRFWE | CTRL\_D\_GRFWE |
| DMSel[2:0] | CTRL\_D\_DMSel |
| 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 |
| OP\_E\_Sel | CTRL\_D\_OP\_E\_Sel |
| MULTSel[2:0] | CTRL\_D\_MULTSel |
| ISMULTDIV | CTRL\_D\_ISMUTLDIV |
| V1[31:0] | FMUX\_V1\_D |
| V2[31:0] | FMUX\_V2\_D |
| 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\_DM\_D\_M\_Sel | CTRL\_F\_ FMUX\_DM\_D\_M\_Sel |

FR\_M：

|  |  |
| --- | --- |
| 端口名 | 接入 |
| DMWE | DMWE\_E |
| GRFWE | GRFWE\_E |
| DMSel[2:0] | DMSel\_E |
| GRF\_WD\_W\_Sel[1:0] | GRF\_WD\_W\_Sel\_E |
| V2[31:0] | FMUX\_V2\_E |
| OP[31:0] | MUX\_OP\_E |
| GRF\_A3[4:0] | GRF\_A3\_E |
| ext32[31:0] | ext32\_E |
| pc8[31:0] | pc8\_E |
| FMUX\_DM\_D\_M\_Sel | FMUX\_DM\_D\_M\_Sel\_E |

FR\_W：

|  |  |
| --- | --- |
| 端口名 | 接入 |
| GRFWE | GRFWE\_M |
| GRF\_WD\_W\_Sel[1:0] | GRF\_WD\_W\_Sel\_M |
| OP[31:0] | OP\_M |
| DM\_Q[31:0] | DM\_Q |
| GRF\_A3[4:0] | GRF\_A3\_ M |
| ext32[31:0] | ext32\_ M |
| pc8[31:0] | pc8\_ M |

**CTRL\_Decoder**

1. 描述

译码器模块。通过opcode、funct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 | DMSel[2:0] | O |
| funct[5:0] | I | DMWE | O |
| rt[4:0] | I | GRFWE | O |
| CompSel[2:0] | O | GRF\_A3\_D\_Sel[1:0] | O |
| EXTSel[1:0] | O | GRF\_WD\_W\_Sel[1:0] | O |
| NPCSel[1:0] | O | ALU\_B\_E\_Sel | O |
| OpSel[1:0] | O | OP\_E\_Sel | O |
| FuncSel[1:0] | O | Tuse\_rs[1:0] | O |
| MULTSel[2:0] | O | Tuse\_rt[1:0] | O |
| ISMULTDIV | 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 | FR\_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\_DM\_D\_M\_Sel | O |
| GRF\_WD\_W\_Sel\_M[1:0] | I |  |  |

**P5思考题**

1. 在课上测试时，我们需要你现场实现新的指令，对于这些新的指令，你可能需要在原有的数据通路上做哪些扩展或修改？提示：你可以对指令进行分类，思考每一类指令可能修改或扩展哪些位置。

|  |  |
| --- | --- |
| 类型名 | 拓展部分 |
| 算数逻辑寄存器 | ALU拓展对应的运算。  CTRL\_Decoder加ALU的控制信号。 |
| 算数逻辑立即数 | ALU拓展对应的运算。  EXT拓展对应的扩展方式。  CTRL\_Decoder 加ALU和EXT的控制信号。 |
| 立即数加载 | EXT拓展对应的扩展方式。  CTRL\_Decoder 加ALU和EXT的控制信号。 |
| 比较与跳转 | COMP拓展对应的比较方式。  CTRL\_Decoder加COMP和NPC的控制信号。 |
| 读取内存 | DM拓展对应的读取的方式。  CTRL\_Decoder加DM的控制信号。 |
| 写入内存 | DM拓展对应的读取的方式。  CTRL\_Decoder加DM的控制信号。 |

此外还有两种：

1.条件写入寄存器（内存）。实现方法如下：拓展COMP。不执行该指令时，寄存器（内存）的使能信号为控制器输出的。执行该指令时，将控制器输出的信号与BranchCOMP进行与然后再进入E级。

2.不同条件写入寄存器（内存）不同种类的数据。实现方法如下：若正常情况下写入的数据为ext32或者pc8，执行该指令时将BranchCOMP的信号作为与ext32或者pc8的信号相连的MUX的控制信号。选择ext32/pc8与给定的其他值。若正常情况下写入的数据不是ext32或者pc8，还要将BranchCOMP的信号作为MUX\_GRF\_A3\_D后的一个MUX的控制信号，正常情况使用MUX\_GRF\_A3\_D，异常情况转到ext32或者pc8。

注意以上两种情况控制信号不能影响到正常使用EXT或者pc8信号的指令。

1. 确定你的译码方式，简要描述你的译码器架构，并思考该架构的优势以及不足。

采用集中式译码方式。所有的译码工作均在D级完成。译码器分为解码器\_Decoder，暂停器\_Stall和转发器\_Forward。解码器负责通过指令输出各个模块的控制信号。暂停器负责根据当前流水线的情况确定是否要在D级暂停。转发器负责根据当前流水线的情况确定是否要进行转发以及如何进行转发。

优势：译码器数量少，移植和进行拓展快，控制信号跟随流水线往下走非常清晰。

不足：单个译码器代码庞大，需要在流水线寄存器中增加用于传递控制信号的寄存器。暂停器和转发器的输入信号存在重复。

1. 我们使用提前分支判断的方法尽早产生结果来减少因不确定而带来的开销，但实际上这种方法并非总能提高效率，请从流水线冒险的角度思考其原因并给出一个指令序列的例子。

第三题若分支指令的判断在ALU处完成，则其Tuse=1。以下情况将不会发生阻塞。但是提前到D级执行，将导致阻塞。

lw $1, 0($0)

ori $2, $0, 0

beq $1, $2, label

1. 因为延迟槽的存在，对于 jal 等需要将指令地址写入寄存器的指令，要写回 PC + 8，请思考为什么这样设计？

因为PC + 4是延迟槽指令的地址，跳转返回时不是从延迟槽指令开始而是从延迟槽后的一条指令开始。所以需要写回PC+8。

1. 我们要求大家所有转发数据都来源于流水寄存器而不能是功能部件（如 DM、ALU），请思考为什么？

因为在一个周期中的很长一段时间，功能部件因为存在延迟，输出的都不是期望值，如果转发这些值进行后续指令的计算，将会导致后续指令的计算出现错误，等到功能部件输出期望值时，后续指令的使用的功能部件可能来不及得到新的正确值一个周期就结束了。而流水线寄存器的值是稳定输出的期望值。

1. 我们为什么要使用 GPR 内部转发？该如何实现？

通过GPR内部转发可以统一解决W级向D级的转发问题。因为W级所有指令的Tuse都为0，数据都可以转发。且无论在之后的哪一级使用W级要写入的寄存器的数据，都可以在通过GPR内部转发在D级完成转发。实现方式如下：

assign RD1 = (A1 == 5'd0) ? 32'h00000000 : RF[A1];

assign RD2 = (A2 == 5'd0) ? 32'h00000000 : RF[A2];

assign V1 = (A1 != 5'd0 && WE && A3 == A1) ? D : RD1;

assign V2 = (A2 != 5'd0 && WE && A3 == A2) ? D : RD2;

1. 我们转发时数据的需求者和供给者可能来源于哪些位置？共有哪些转发数据通路？

见上文转发与暂停设计中“转发与暂停”表。

**P6思考题**

1. 为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

因为CPU执行乘除法的时间较长，而且执行乘除法的次数不多。如果将乘除法整合进ALU，执行次数更多的其他算数逻辑指令将会被阻塞，设计单独的乘除法部件可以让较慢的乘除法在执行过程中，其他的仍然可以执行。因为乘除法的结果不能用一个寄存器表示，需要使用两个寄存器，这两个寄存器专用于乘除法。由高内聚低耦合的原则，需要设置独立的HI、LO寄存器而不是使用通用寄存器。

1. 真实的流水线 CPU 是如何使用实现乘除法的？请查阅相关资料进行简单说明。

## 使用独立于主流水线的特殊硬件功能单元来处理整数乘法和除法，可能使用器件包括乘累加乘法器、阵列乘法器、Baugh-Wooley乘法器、Radix-2 Booth乘法器、Radix-4 Booth乘法器、恢复余数法除法器、不恢复余数法除法器、基2 SRT算法除法器等。

1. 请结合自己的实现分析，你是如何处理 Busy 信号带来的周期阻塞的？

当Busy信号或Start信号为1时且进入D级的指令为MULTDIV、MLTO、MLFROM时暂停，其余指令因为不使用的乘除法模块按照正常的转发暂停逻辑执行。

1. 请问采用字节使能信号的方式处理写指令有什么好处？（提示：从清晰性、统一性等角度考虑）

清晰性上，通过字节使能信号能够直接读出需要修改的字节。

统一性上，统一了DM写使能信号和具体写入的字节的地址信号，无需传递两个信号。

1. 请思考，我们在按字节读和按字节写时，实际从 DM 获得的数据和向 DM 写入的数据是否是一字节？在什么情况下我们按字节读和按字节写的效率会高于按字读和按字写呢？

按字节读和按字节写的时候获得的数据并不是一字节，而是一字。只是将字的其他部分进行了处理，当对单个字节进行读写时，按字节读和按字节写的效率更高，例如在C语言中处理char类型数组时，如果按字读和按字写还要在一个字进行定位。

1. 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

主要采取的手段包括将指令进行分类（见转发与暂停设计一节），利用指令的分类进行译码（见各模块控制信号），在处理数据冲突的时候也是将分类之后的指令进行Tuse和Tnew的归类来进行暂停与转发的设计（见转发与暂停设计一节）

1. 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

发现并处理的指令类型组合产生的冲突见转发与暂停设计一节的转发与暂停表，根据数据的情况全力进行转发，无法通过转发处理的进行暂停处理。使用Python辅以序列设计进行测试样例的构建。

1. 如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。

使用Python辅以序列设计进行测试样例的构建，先确定要进行测试的指令的组合，然后通过lui和ori进行组合对初始数据进行赋值，然后将要测试的指令放置在相应的流水级，前面的流水级随机从可生成数据的指令中选取，后面的流水级均用与冲突寄存器无关的指令填充。生成时模拟指令的行为，如beq模拟指令的行为计算跳转时的值，在初始数据中赋值，bne模拟指令的行为计算不跳转时的值，并且在初始数据中赋值。

面前遇到的问题是jal和jr的测试比较困难。