MIPS 流水线处理器架构文档

何源 电子工程系 2018011182

一、整体情况

本次设计实现的 MIPS 流水线处理器设计图纸参阅如下文件：

	位图	矢量图（Adobe Illustrator 2020）
MIPS 流水线架构图	Architecture.jpg	Architecture.ai
MIPS 流水线模块图	Modules.jpg	Modules.ai

该 MIPS 流水线处理器的模块结构以及每个模块下的硬件单元为：

- IF
  - PC
  - Instruction Memory
- ID
  - IF/ID
  - Register
  - Sign Extend
  - Ctrl
  - Compare
  - Exception Unit
- EX
  - ID/EX
  - ALU
  - ALU Ctrl
- MEM
  - EX/MEM
  - Data Memory
- WB
  - MEM/WB
- Forward
  - Forward Unit

因此，整个硬件设计分为三个层次：顶层为 CPU，负责各个模块的连接以及引脚的约束；中层为各模块，负责模块下各个硬件单元的逻辑连接等操作；底层为硬件单元，负责基本的时序逻辑或组合逻辑功能。

二、指令集和指令格式

该处理器支持的指令集包括：

nop

lw, sw, lui

add, addu, sub, subu, addi, addiu

and, or, xor, nor, andi, ori, sll, srl, sra, slt, sltu, slti, sltiu

beq, bne, blez, bgtz, bltz

j, jal, jr, jalr

上述大部分指令的格式详见*数字逻辑与处理器基础大作业-2020.pdf*

ori, beq, bne, blez, bgtz, bltz 的指令格式如下：

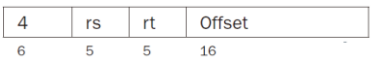
（出自 *Computer Organization Design 5th Edition, Appendix A*）

OR immediate

ori rt, rs, imm

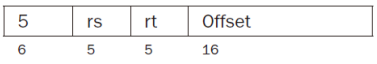
Branch on equal

beq rs, rt, label



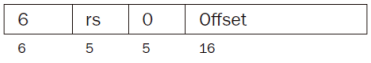
Branch on not equal

bne rs, rt, label



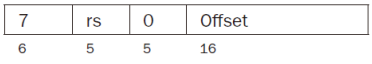
Branch on less than equal zero

blez rs, label



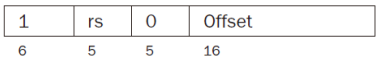
Branch on greater than zero

bgtz rs, label



Branch on less than zero

bltz rs, label



三、控制信号

控制单元 Ctrl 将产生当前指令对各元件的控制信号。除本模块控制信号直接进行控制之外，其余信号均向后传递。

控制信号共包括：

- IF/ID
  - PCSrc
  - Branch
  - CpCode

beq	000
bne	001
blez	010
bgtz	011
bltz	100

  - CtrlFlush
  - LuOp
  - ExtOp：1 补最高位，0 补 0
  - BadOp
- ID/EX
  - ALUSrc1
  - ALUSrc2
  - ALUOp
  - RtRd
- EX/MEM
  - MemRead

- MemWrite

- MEM/WB
  - MemtoReg
  - RegWrite
  - RegDst

各指令的控制信号一览表：

	PCSrc[1:0]	Branch	CpCode[2:0]	CtrlFlush	LuOp	ExtOp	BadOp	ALUSrc1	ALUSrc2	RtRd	MemRead	MemWrite	MemtoReg[1:0]	RegWrite	RegDst
lw	0	0	x	0	0	1	0	0	1	0	1	0	1	1	0
sw	0	0	x	0	0	1	0	0	1	x	0	1	x	0	x
lui	0	0	x	0	1	x	0	0	1	0	0	0	0	1	0
add	0	0	x	0	x	x	0	0	0	1	0	0	0	1	0
addu	0	0	x	0	x	x	0	0	0	1	0	0	0	1	0
sub	0	0	x	0	x	x	0	0	0	1	0	0	0	1	0
subu	0	0	x	0	x	x	0	0	0	1	0	0	0	1	0
addi	0	0	x	0	0	1	0	0	1	0	0	0	0	1	0
addiu	0	0	x	0	0	1	0	0	1	0	0	0	0	1	0
and	0	0	x	0	x	x	0	0	0	1	0	0	0	1	0
or	0	0	x	0	x	x	0	0	0	1	0	0	0	1	0
xor	0	0	x	0	x	x	0	0	0	1	0	0	0	1	0
nor	0	0	x	0	x	x	0	0	0	1	0	0	0	1	0
andi	0	0	x	0	0	1	0	0	1	0	0	0	0	1	0
ori	0	0	x	0	0	1	0	0	1	0	0	0	0	1	0
sll	0	0	x	0	x	x	0	1	0	1	0	0	0	1	0
srl	0	0	x	0	x	x	0	1	0	1	0	0	0	1	0
sra	0	0	x	0	x	x	0	1	0	1	0	0	0	1	0
slt	0	0	x	0	x	x	0	0	0	1	0	0	0	1	0
sltu	0	0	x	0	x	x	0	0	0	1	0	0	0	1	0
slti	0	0	x	0	0	1	0	0	1	0	0	0	0	1	0
sltiu	0	0	x	0	0	0	0	0	1	0	0	0	0	1	0
beq	0	1	0	0	0	1	0	x	x	x	0	0	x	0	x
bne	0	1	1	0	0	1	0	x	x	x	0	0	x	0	x
blez	0	1	2	0	0	1	0	x	x	x	0	0	x	0	x
bgtz	0	1	3	0	0	1	0	x	x	x	0	0	x	0	x
bltz	0	1	4	0	0	1	0	x	x	x	0	0	x	0	x
j	1	0	x	1	x	x	0	x	x	x	0	0	x	0	x
jal	1	0	x	1	x	x	0	x	x	x	0	0	2	1	1
jr	2	0	x	1	x	x	0	x	x	x	0	0	x	0	x
jalr	2	0	x	1	x	x	0	x	x	1	0	0	2	1	0
Other	0	0	x	0	x	x	1	x	x	x	0	0	x	0	x

注：nop 指令为 sll $0, $0, 0，因此无需单独实现。

四、Forward 信号

Forward 单元 Forward Unit 将产生当前周期处理器各部分的 Forward 信号。

Forward 信号共包括：

- RegData1
- RegData2
- DataSrc1
- DataSrc2

各 Forward 信号的判断逻辑：

```
if (EX/MEM.RegWrite &&
    EX/MEM.RegWriteDst != 0 &&
    EX/MEM.RegWriteDst == ID/EX.RegisterRs)
     DataSrc1 <= 2’b10;
else if (MEM/WB.RegWrite &&
          MEM/WB.RegWriteDst != 0 &&
          MEM/WB.RegWriteDst == ID/EX.RegisterRs)
     DataSrc1 <= 2’b01;
else
     DataSrc1 <= 2’b00;

if (EX/MEM.RegWrite &&
    EX/MEM.RegWriteDst != 0 &&
    EX/MEM.RegWriteDst == ID/EX.RegisterRt)
     DataSrc2 <= 2’b10;
else if (MEM/WB.RegWrite &&
          MEM/WB.RegWriteDst != 0 &&
          MEM/WB.RegWriteDst == ID/EX.RegisterRs)
     DataSrc2 <= 2’b01;
else
     DataSrc2 <= 2’b00;

if (EX/MEM.RegWrite &&
    EX/MEM.RegWriteDst != 0 &&
    EX/MEM.RegWriteDst == IF/ID.RegisterRs)
     RegData1 = 1’b1;
else
     RegData1 = 1’b0;

if (EX/MEM.RegWrite &&
    EX/MEM.RegWriteDst != 0 &&
    EX/MEM.RegWriteDst == IF/ID.RegisterRt)
     RegData2 = 1’b1;
else
```

```
    RegData2 = 1’b0;
```

五、Exception 信号

Exception 单元 Exception Unit 将产生 Exception 信号。

Exception 信号共包括：

- IrqPCSrc
- ExceptPCSrc
- PCBackup
- ExpFlush

各 Exception 信号的判断逻辑：

```
if Irq
    IrqPCSrc = 1’b1
else
    IrqPCSrc = 1’b0

if BadOp
    ExceptPCSrc = 1’b1
    PCBackup = 1’b1
    ExpFlush = 1’b1
else
    ExceptPCSrc = 1’b0
    PCBackup = 1’b0
    ExpFlush = 1’b0
```

六、冒险控制

采用在汇编代码中手动添加 nop 的方式进行处理。需要保证以下原则：

1) load-use hazard: EX 阶段用到 lw 结果的应当与 lw 相隔 1 条指令以上，ID 阶段需要用到 lw 结果的应当与 lw 相隔 2 条指令以上（beq, jr 和 jalr）。
2) j-hazard: j 型指令后应紧跟一个 nop

七、开发注意事项

- Register 和 Data Memory 为实现先写后读，均采用 Read 为组合电路，Write 为时序电路的设计方式。每个周期的逻辑中，时序逻辑应仅分布于一层，不应出现时序逻辑串接的情况，这需要两个周期才能完成运算。因此，Data_Memory 的与写相关的信号应当接到 ID 模块的输出，而不是 ID/EX 寄存器的输出；Register 的与写相关的信号应当接到 EX 阶段的输出，而不是 MEM/WB 寄存器的输出。因此，Data_Memory 的读地址输入端和写地址输入端所接之处不应当相同：读地址输入端连接组合逻辑，应接在 ID/EX 寄存器的输出；写地址输入端连接时序逻辑，应接在 ID 模块的输出端，与 ID/EX 同级。
- 每个 always 中，应保持每个信号仅被赋值一次。
- 应当首先建好文件架构，而后按照底层、中层、顶层的顺序逐层开发
- 应当做好版本控制，在完成关键代码后及时 commit 版本，以便重大修改失败后回退历史节点。
- 保持良好的代码规范，包括代码格式和命名规范。