RV32I指令集流水线CPU设计报告

张劲暾 PB16111485

待完成模块设计思路

ALU

算数逻辑运算单元，接受 `Operand1` 和 `Operand2` 两个操作数，按照控制信号 `AluContrl` 执行对应的算术逻辑运算，将结果从 `AluOut` 输出。各种算术逻辑运算对应控制信号如下：

```
//ALUContrl[3:0]
    `define SLL  4'd0   // 逻辑左移
    `define SRL  4'd1   // 逻辑右移
    `define SRA  4'd2   // 算术右移
    `define ADD  4'd3   // 加法
    `define SUB  4'd4   // 减法
    `define XOR  4'd5   // 异或
    `define OR   4'd6   // 或
    `define AND  4'd7   // 与
    `define SLT  4'd8   // 有符号数比较
    `define SLTU 4'd9   // 无符号数比较
    `define LUI  4'd10  // 立即数加载
```

BranchDecisionMaking

跳转判断单元，根据控制信号 `BranchTypeE` 指定的分支类型，对操作数 `Operand1` 和 `Operand2` 进行比较并决定是否跳转，将判断结果通过 `BranchE` 输出。各分支类型对应的控制信号如下：

```
//BranchType[2:0]
    `define NOBRANCH  3'd0  // 不跳转
    `define BEQ  3'd1       // 等于跳转
    `define BNE  3'd2       // 不等跳转
    `define BLT  3'd3       // 小于跳转
    `define BLTU  3'd4      // 无符号小于跳转
    `define BGE  3'd5       // 大于跳转
    `define BGEU  3'd6      // 无符号大于跳转
```

`ControlUnit`

控制模块（译码器），根据指令的操作码部分`Op`，func3部分`Fn3`和func7部分`Fn7`产生如下控制信号：

- `JalD`：标志`Jal`指令到达指令译码阶段
- `JalrD`：标志`Jalr`指令到达指令译码阶段
- `RegWriteD`：指令译码阶段的寄存器写入模式：

```
//RegWrite[2:0]  six kind of ways to save values to Register
    `define NOREGWRITE  3'b0    //  Do not write Register
    `define LB  3'd1            //  load 8bit from Mem then signed extended to 32bit
    `define LH  3'd2            //  load 16bit from Mem then signed extended to 32bit
    `define LW  3'd3            //  write 32bit to Register
    `define LBU  3'd4           //  load 8bit from Mem then unsigned extended to 32bit
    `define LHU  3'd5           //  load 16bit from Mem then unsigned extended to 32bit
```

- `MemToRegD`：标志指令需要从`data memory`读取数据到寄存器
- `MemWriteD`：指示`data memory`的四个字节中哪些需要写入
- `LoadNpcD`：标志将`NextPC`输出到`ResultM`
- `RegReadD`：标志两个源寄存器的使用情况，`RegReadD[1]` == 1，表示`A1`对应的寄存器值被使用到了，`RegReadD[0]` == 1，表示`A2`对应的寄存器值被使用到了
- `BranchTypeD`：分支类型（参见`BranchDecisionMaking`部分）
- `AluContrlD`：算数逻辑运算种类（参见`ALU`部分）
- `AluSrc2D`：`Operand2`的类型
- `AluSrc1D`：`Operand1`的类型
- `ImmType`：立即数编码类型：

```
//ImmType[2:0]
    `define RTYPE  3'd0
    `define ITYPE  3'd1
    `define STYPE  3'd2
    `define BTYPE  3'd3
    `define UTYPE  3'd4
    `define JTYPE  3'd5
```

`DataExt`

非字对齐load指令处理单元， 对从`data memory`读取的`IN`的四个字节中由`LoadedBytesSelect`指定的字节按照`RegWriteW`指定的寄存器写入模式（参见`ControlUnit`部分）得到最终要写入寄存器的值`OUT`

EXSegReg

支持同步清零的段寄存器

HarzardUnit

流水线冲突处理模块，基本手段：(1)插入气泡`stall`，(2)定向路径`forward`，(3)冲刷流水段`flush`，信号说明：

- `CpuRst`: CPU初始化与工作控制信号，当`CpuRst` == 1 时CPU全局复位清零（所有段寄存器flush），`Cpu_Rst` == 0 时CPU开始执行指令
- `BranchE`,`JalrE`,`JalD`: 控制相关处理信号
- `Rs1D`,`Rs2D`,`Rs1E`,`Rs2E`,`RdE`,`RdM`,`RdW`: 译码，执行，访存，写会阶段处理数据相关的信号，对应的源寄存器和目标寄存器号码。
- `RegReadE`: 标记`A1`和`A2`对应的寄存器值是否被用到。
- `MemToRegE`: 标志EX段从`data mamory`加载数据到寄存器
- `RegWriteM`,`RegWriteW`: 标记MEM段和WB段是否有目标寄存器写入操作。
- `StallF`,`FlushF`: IF段插入气泡/冲刷
- `StallD`,`FlushD`: ID段插入气泡/冲刷
- `StallE`,`FlushE`: EX段插入气泡/冲刷
- `StallM`,`FlushM`: MEM段插入气泡/冲刷
- `StallW`,`FlushW`: WB段插入气泡/冲刷
- `Forward1E`,`Forward2E`: 定向路径控制信号

IDSegReg

`IF-ID`段寄存器，“内置”（关联）`InstructionMemory`，在时钟，使能和清空信号的控制下存取指令并对下一条指令的`PC`进行准备，另外实现RD段寄存器stall和clear功能。

IFSegReg

`IF-ID`段寄存器

ImmOperandUnit

立即数扩展单元，根据`Type`信号指定的立即数编码类型（参见`ControlUnit`部分），将操作码以外25位`In`中的立即数扩展为32位立即数`Out`参与运算。

MEMSegReg

`EX-MEM`段寄存器

NPC_Generator

`NextPC`值生成模块，根据三种跳转指令的跳转目标和是否确定跳转的信号，决定最后的跳转目标（新PC值）`PC_In`的产生，这个单元存在的原因是`Jal`指令可能和 `Branch` 或 `Jalr` 同时确定跳转，这个时候就需要判断按照哪个目标跳转并冲刷流水段。

RegisterFile

上升沿写入，异步读的寄存器堆，0号寄存器值始终为32'b0

RV32Core

RV32I 指令集CPU的顶层模块

WBSegReg

`WB`段寄存器，类似于`IDSegReg`单元，但内置`DataMemory`，五组输入输出信号的解读：

```
module WBSegReg(
    input wire clk,                     // 时钟，使能，清空，逻辑控制信号组
    input wire en,
    input wire clear,
    //Data Memory Access                // 数据内存访存控制信号组
    input wire [31:0] A,                // 32位数据地址
    input wire [31:0] WD,               // 32位写入数据
    input wire [3:0] WE,                // 字节写使能（每次得到的是四个字节但不是都要写）非字对齐store
    output wire [31:0] RD,              // 32位数据读取
    output reg [1:0] LoadedBytesSelect, // 字节读取指示信号（一次读四字节但不是都需要）非字对齐字节load
    //Data Memory Debug                 // 调试信号组
    input wire [31:0] A2,
    input wire [31:0] WD2,
    input wire [3:0] WE2,
    output wire [31:0] RD2,
    // 流水线传递：
    //input control signals             // 输入控制信号组
    input wire [31:0] ResultM,          //
    output reg [31:0] ResultW,          // 这两个是ALU计算结果
    input wire [4:0] RdM,               //
    output reg [4:0] RdW,               // 目的寄存器编号
    //output constrol signals           // 输出控制信号组
    input wire [2:0] RegWriteM,         //
    output reg [2:0] RegWriteW,         // 是否写目的寄存器
    input wire MemToRegM,               //
    output reg MemToRegW                // 是否读取到寄存器
    );
```

问题回答

01. 为什么将`DataMemory`和`InstructionMemory`嵌入在段寄存器中?

因为同步读 memory 相当于异步读 memory 的输出外接D触发器，需要时钟上升沿才能读取数据，此时如果再通过段寄存器缓存，那么需要两个时钟上升沿才能将数据传递到Ex段，因此在段寄存器模块中调用该同步memory，直接将输出传递到ID段组合逻辑。

02. `DataMemory`和`InstructionMemory`输入地址是字（32bit）地址，如何将访存地址转化为字地址输入进去?

用访存地址的高30位输入，一次性读取四个字节。

03. 如何实现`DataMemory`的非字对齐的`Load`?

用`DataExt`模块处理，先一次性读取4个字节，再指定其中的某个字节按照指定的寄存器写入方式扩展，最终得到要写入寄存器的数据。

04. 如何实现`DataMemory`的非字对齐的`Store`?

使用`MemWriteD`信号指明`data memory`中写入的字节。

05. 为什么`RegFile`的时钟要取反?

这样可以做到上升沿写入，异步读取。

06. `NPC_Generator`中对于不同跳转`target`的选择有没有优先级?

有，因为只可能是后来的`Jal`的`ID`段冲突之前`Branch`或`Jalr`的`EX`段，那么这时`EX`在前具有较高优先级，按`EX`段结果跳转并冲刷掉后面的`ID`段。

07. `ALU`模块中，默认`wire`变量是有符号数还是无符号数?

可以是有符号数也可以是无符号数，取决于程序员解释。

08. `AluSrc1E`执行哪些指令时等于`1'b1`?

`AluSrc1E`表示`ALU`输入源`A1`的选择，0使用寄存器，1使用PC值：只有在执行`Jalr`指令和`AUIPC`指令时 等于`1'b1`。

---

09. `AluSrc2E`执行哪些指令时等于`2'b01`?

`AluSrc2E`表示`ALU`输入源`A2`的选择，`00`使用寄存器，`01`使用位移次数，`10`使用立即数：只有在执行 立即数位移指令`SLLI`、`SRLI`和`SRAI`时等于`2'b01`。

---

10. 哪条指令执行过程中会使得`LoadNpcD==1`?

只有无条件跳转指令`Jal`和`Jalr`才会使得`LoadNpcD` == 1。

---

11. `DataExt`模块中，`LoadedBytesSelect`的意义是什么?

选取`IN`的四个字节中用于扩展写入寄存器的字节。

---

12. `Harzard`模块中，有哪几类冲突需要插入气泡?

控制相关的冲突以及`Load`类指令和紧接着的`ALU`指令的数据相关需要插入气泡。

---

13. `Harzard`模块中采用默认不跳转的策略，遇到`branch`指令时，如何控制`flush`和`stall`信号?

当确定跳转时，控制模块会产生`BranchE`信号。此时，`Harzard`模块应当产生对`IF`、`ID`、`EX`的`flush`信号，不需要产生`stall`信号。

---

14. `Harzard`模块中，`RegReadE`信号有什么用?

`RegReadE`表示`EX`段指令对应的寄存器值是否被使用，用于判断数据相关。如果对应的寄存器值被使用，而这个寄存器又是之前`MEM`或`WB`段指令的目的寄存器，就有数据相关，需要`forward`或者`stall`。

---

15. 0号寄存器值始终为0，是否会对`forward`的处理产生影响?

会，需要对其特殊处理。在处理冲突时，如果冲突发生在0号寄存器上，就忽略该冲突。