流水线总体思想

第1个上升沿：

取指译码，对应信息送RegFile和CU，PC自增（CU的isFullOut为真时不自增），RegFile准备好dataOut1、2和labelOut1、2，CU给保留站或队列送使能信号并通过mux4to1选择空闲的保留站号或者队列号送给RegFile（尚未写入RegFile）；

第2个上升沿：

若指令对应的保留站或队列未满，则当前指令写入保留站或队列中，上一阶段中CU送来的空闲保留站号或队列号写入RegFile。当前指令中已就绪的操作数，直接从RegFile读进来，从而与寄存器文件解耦合。对于操作数未就绪的指令，在保留站或队列中存好该数据的的最新来源（保留站号或队列号，从寄存器状态表读取），等待数据写入。操作数已就绪的一种特殊情况是此时CDB广播的数据恰好是该操作数，此时直接从CDB都进来即可。如果指令已经就绪，则从保留站向ALU或从队列向memory发请求，如果ALU向保留站或memory向队列发回了requireAC，保留站置对应项目busy为0或者队列出队。否则继续保持请求，直到被响应。

第N+2个上升沿（假设执行需要N个周期）：

保留站完成运算或队列完成存取，向CDB发请求，如果CDB发回requireAC，将保留站或队列的available置为1（执行过程中保留站和队列的available为0），否则一直保持请求，直到被响应。CDB向保留站、队列、寄存器堆广播，在任何一个上升沿，如果广播信号有效，保留站、队列和寄存器堆会监听CDB，更新需要更新的值。

CDB and CDBHelper

接收来自mfALU，pmfALU，memory的data和label，接收来自mfState，pmfState，momory的require信号，以此选择一个有效的data和label，以及一个有效信号EN进行广播输出，输出到三个Queue和两个ReservationStation以及RegFile。同时也要把选择的是哪个这一信息返回给mfState，pmfState和memory。

CU

接收Decoder输出的func和op，以及来自两个ReservationStation和三个Queue（三个queue只需要一个信号就够了）的isFull信号。

输出包括

ALUSel，即选择哪个ALU，送给mux4to1\_4，mux4to1\_4根据该值选择空的队列号或者空的保留站号（它的其余输入即为两个保留站和队列的writeableLabel）；

ALUop，送给两个保留站；

QueueOp，即读或写，送给三个Queue；ResStationEN，送给两个保留站和三个Queue；

isFullOut，判断当前指令对应的ReservationStation或者Queue的isFull信号是不是有效，在top取反后记为labelEN信号，再送给regFile的RegWr（RegWr为什么要CU的isFull为0才有效，isFull的时候为什么不能写寄存器？见下文RegFile部分）和PC的pcWrite（结构冲突，PC不应再自增）；

VkSrc，代表rt是源操作数（R型）还是目的操作数（I型），在top中通过VkSrc，来判断给RegFile的WriteAddr和ReadAddr2送rt还是rd，给两个ReservationStation的dataIn2、label2送RegFile的DataOut2、LabelOut2，还是送扩展后的立即数、全零的label（**全零代表什么**）。

RegFile

寄存器堆不仅包括编号为1到31的（0号的值总为0）32位regData，还包括与之一一对应的编号为1到31的regLabel（保留站号或队列号）。

输入包括

BCEN、BCdata、BClabel；

ReadAddr1，即rs，ReadAddr2，见上文CU部分的VkSrc；

RegWr，见CU部分的isFullOut；

WriteAddr见CU部分的VkSrc，WriteLabel见CU部分的ALUSel；

Clk、nRet。

输出为DataOut1和DataOut2，即用ReadAddr1和ReadAddr2在寄存器堆中取出值，以及LabelOut1和LabelOut2，即对应的label。其中Out1会送到两个ReservationStation和RS\_queue，Out2会送到两个ReservationStation和RT\_queue。

在nRST下降沿（？），置寄存器堆和label堆为0；

在clk上升沿，如果RegWr有效（如果要用的保留站或队列满了，RegWr为0，此时PC不应该自增，RegLabel也不应修改，即结构冲突，流水线应该停下），把WriteAddr对应的寄存器的regLabel置为WriteLabel；如果BCEN有效，找到与BClabel相同的regLabel，将其值重置为0，将对应的regData置为BCdata。

Memory

输入包括

WEN，由三个队列的require信号与得到**（为什么要与一下）**；

dataIn1和dataIn2分别来自RS\_queue和Immd\_queue的dataOut；

labelIn来自RT\_queue的labelOut，含义为取数指令的目标队列号，writeData来自RT\_queue的dataOut（RegFile的Out2是送到RT\_queue的），op来自RT\_queue的opOut，控制读写（opOut来自哪个queue都可以，统一选择RT\_queue作为三个queue的代表，同理包括labelOut、isFull、queue\_writeable\_label、require，不过require信号三个queue都连了出来）；

requireAC来自CDB的反馈信号。

输出包括

available输出到三个队列的requireAC，作为对他们送来的require信号的反馈；

isLastState输出到三个队列的isLastState，控制队列是否可以进行出队；

require信号送给CDB；

labelOut和loadData分别送到CDB的label3和data3。

Memory内部包含一个RAM模块，通过维护内部的状态，实现了一个需要10个周期进行读或写的数据存储器。Memory在clk的上升沿开始工作，接收来自队列的存取指令，dataIn1和dataIn2分别来自RS\_queue和Immd\_queue，将这两个值相加得到正确的存取地址。读取操作完成后labelOut和loadData送至CDB并向CDB发出require信号，其中labelOut的值就是labelIn的值。

ReservationStation

输入包括

EXEable，该保留站对应的ALU是否可执行，来自对应的mfState或pmfState的available；

ResStationDst，该保留站的编号；

WEN，来自CU的选择信号ResStationEN，opCode，来自CU的ALUop；

来自RegFile的dataIn1、dataIn2、label1、label2；

来自CDB的BCEN、BClabel、BCdata。

输出包括

OutEN，送给对应的mfState或pmfState的WEN端口；

opOut，送给对应ALU（及对应的State）的op端口，dataOut1、dataOut2送给对应ALU的dataIn，ready\_labelOut就绪指令的保留站号，送给对应ALU的labelIn；

writeable\_labelOut，可写的保留站号，送给mux4to1。

保留站的一部分工作见流水线总体思想的第二个上升沿部分。除此之外，在上升沿到来时，保留站还会根据CDB的广播数据，把相应的保留站号的busy重置为0，按需把各保留站项的Vj、Vk、Qj、Qk更新。保留站还需要实时地（不需要在上升沿）更新cur\_addr（当前可用的空项中编号最小的那一个）和ready\_addr（哪个项的所有操作数都准备好了，也是小编号优先）。

Queue

队列实质上也是保留站，其输入输出和行为与保留站类似。但由于存取指令的特殊性，将存取指令的保留站特化为三个队列，即Immd\_queue、RS\_queue、RT\_queue。

对于存取指令，队列的require与memory的WEN相连，memory的available连回队列的requireAC，而memory的require和requireAC与CDB交互；

对于运算指令，ReservationStation的OutEn与对应的ALU的WEN相连，ALU的available连回ReservationStation的EXEable，而ALU的require和requireAC与CDB交互。

当队首项有效且队首项数据准备好时，require为有效。当memory送来的isLastState有效时，说明队首指令在下一个上升沿到来时便可以处理完毕，置poppable为1,队首指令出队，剩余工作交由memory与CDB交互。

在上升沿到来时，检测CDB，更新队列中可更新的值。如果WEN有效（即收到来自CU的选择信号，新流出指令为存取指令），if poppable，出队，新指令放在出队前的最后一个有效项的位置，else，新指令放在队列中第一个空的位置；否则，如果WEN无效，if poppable，出队，else，什么都不做。

ALU

输入包括来自ReservationStation的WEN、op、dataIn1、dataIn2、labelIn，来自CDB的requireAC。

输出包括labelOut和result，送到CDB。

ALU与ReservationStation的交互与前述Memory与Queue的交互逻辑基本相同。

在运算结束前的一个上升沿，require置为1，当require和requireAC均为1或内部的state为idle时（前者表明CDB虽然ALU虽忙但当前结果已被CDB相应，下一个clk同样available，后者表明就是空闲）。注意由于require在运算结束前的一个状态就已经置为1，因此在保留站刚送OutEn信号和数据过来时，加、与、或操作运算完成，同时require信号置1，向CDB发请求。也就是说，加、与、或操作的执行阶段只需要0个周期就可以完成，而减法分了两步，先取反加一，下一个clk再相加，因此需要1个周期。

增加指令时改了端口和top的连线的位数，忘记改保留站内部reg的opcode的位数。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 助记符 | 指令格式 | | | | | | 示例 | 示例含义 | 操作及其解释 |
| Bit # | 31..26 | 25..21 | 20..16 | 15..11 | 10..6 | 5..0 |  |  |  |
| R-type | op | rs | rt | rd | shamt | func |  |  |  |
| add | 000000 | rs | rt | rd | 00000 | 100000 | add $1,$2,$3 | $1=$2+$3 | rd <- rs + rt   ；其中rs＝$2，rt=$3, rd=$1 |
| addu | 000000 | rs | rt | rd | 00000 | 100001 | addu $1,$2,$3 | $1=$2+$3 | rd <- rs + rt   ；其中rs＝$2，rt=$3, rd=$1,无符号数 |
| sub | 000000 | rs | rt | rd | 00000 | 100010 | sub $1,$2,$3 | $1=$2-$3 | rd <- rs - rt   ；其中rs＝$2，rt=$3, rd=$1 |
| subu | 000000 | rs | rt | rd | 00000 | 100011 | subu $1,$2,$3 | $1=$2-$3 | rd <- rs - rt   ；其中rs＝$2，rt=$3, rd=$1,无符号数 |
| and | 000000 | rs | rt | rd | 00000 | 100100 | and $1,$2,$3 | $1=$2 & $3 | rd <- rs & rt   ；其中rs＝$2，rt=$3, rd=$1 |
| or | 000000 | rs | rt | rd | 00000 | 100101 | or $1,$2,$3 | $1=$2 | $3 | rd <- rs | rt   ；其中rs＝$2，rt=$3, rd=$1 |
| xor | 000000 | rs | rt | rd | 00000 | 100110 | xor $1,$2,$3 | $1=$2 ^ $3 | rd <- rs xor rt   ；其中rs＝$2，rt=$3, rd=$1(异或） |
| nor | 000000 | rs | rt | rd | 00000 | 100111 | nor $1,$2,$3 | $1=~($2 | $3) | rd <- not(rs | rt)   ；其中rs＝$2，rt=$3, rd=$1(或非） |
| slt | 000000 | rs | rt | rd | 00000 | 101010 | slt $1,$2,$3 | if($2<$3)    $1=1 else    $1=0 | if (rs < rt) rd=1 else rd=0 ；其中rs＝$2，rt=$3, rd=$1 |
| sltu | 000000 | rs | rt | rd | 00000 | 101011 | sltu $1,$2,$3 | if($2<$3)    $1=1 else    $1=0 | if (rs < rt) rd=1 else rd=0 ；其中rs＝$2，rt=$3, rd=$1   (无符号数） |
| sll | 000000 | 00000 | rt | rd | shamt | 000000 | sll $1,$2,10 | $1=$2<<10 | rd <- rt << shamt  ；shamt存放移位的位数，   也就是指令中的立即数，其中rt=$2, rd=$1 |
| srl | 000000 | 00000 | rt | rd | shamt | 000010 | srl $1,$2,10 | $1=$2>>10 | rd <- rt >> shamt ；(logical) ，其中rt=$2, rd=$1 |
| sra | 000000 | 00000 | rt | rd | shamt | 000011 | sra $1,$2,10 | $1=$2>>10 | rd <- rt >> shamt  ；(arithmetic) 注意符号位保留  其中rt=$2, rd=$1 |
| sllv | 000000 | rs | rt | rd | 00000 | 000100 | sllv $1,$2,$3 | $1=$2<<$3 | rd <- rt << rs  ；其中rs＝$3，rt=$2, rd=$1 |
| srlv | 000000 | rs | rt | rd | 00000 | 000110 | srlv $1,$2,$3 | $1=$2>>$3 | rd <- rt >> rs  ；(logical)其中rs＝$3，rt=$2, rd=$1 |
| srav | 000000 | rs | rt | rd | 00000 | 000111 | srav $1,$2,$3 | $1=$2>>$3 | rd <- rt >> rs  ；(arithmetic) 注意符号位保留  其中rs＝$3，rt=$2, rd=$1 |
| jr | 000000 | rs | 00000 | 00000 | 00000 | 001000 | jr $31 | goto $31 | PC <- rs |
| I-type | op | rs | rt | immediate | | |  |  |  |
| addi | 001000 | rs | rt | immediate | | | addi $1,$2,100 | $1=$2+100 | rt <- rs + (sign-extend)immediate ；其中rt=$1,rs=$2 |
| addiu | 001001 | rs | rt | immediate | | | addiu $1,$2,100 | $1=$2+100 | rt <- rs + (zero-extend)immediate ；其中rt=$1,rs=$2 |
| andi | 001100 | rs | rt | immediate | | | andi $1,$2,10 | $1=$2 & 10 | rt <- rs & (zero-extend)immediate ；其中rt=$1,rs=$2 |
| ori | 001101 | rs | rt | immediate | | | andi $1,$2,10 | $1=$2 | 10 | rt <- rs | (zero-extend)immediate ；其中rt=$1,rs=$2 |
| xori | 001110 | rs | rt | immediate | | | andi $1,$2,10 | $1=$2 ^ 10 | rt <- rs xor (zero-extend)immediate ；其中rt=$1,rs=$2 |
| lui | 001111 | 00000 | rt | immediate | | | lui $1,100 | $1=100\*65536 | rt <- immediate\*65536 ；将16位立即数放到目标寄存器高16          位，目标寄存器的低16位填0 |
| lw | 100011 | rs | rt | immediate | | | lw $1,10($2) | $1=memory[$2  +10] | rt <- memory[rs + (sign-extend)immediate] ；rt=$1,rs=$2 |
| sw | 101011 | rs | rt | immediate | | | sw $1,10($2) | memory[$2+10]  =$1 | memory[rs + (sign-extend)immediate] <- rt ；rt=$1,rs=$2 |
| beq | 000100 | rs | rt | immediate | | | beq $1,$2,10 | if($1==$2)    goto PC+4+40 | if (rs == rt) PC <- PC+4 + (sign-extend)immediate<<2 |
| bne | 000101 | rs | rt | immediate | | | bne $1,$2,10 | if($1!=$2)   goto PC+4+40 | if (rs != rt) PC <- PC+4 + (sign-extend)immediate<<2 |
| slti | 001010 | rs | rt | immediate | | | slti $1,$2,10 | if($2<10)    $1=1 else    $1=0 | if (rs <(sign-extend)immediate) rt=1 else rt=0 ；    其中rs＝$2，rt=$1 |
| sltiu | 001011 | rs | rt | immediate | | | sltiu $1,$2,10 | if($2<10)    $1=1 else    $1=0 | if (rs <(zero-extend)immediate) rt=1 else rt=0 ；   其中rs＝$2，rt=$1 |
| J-type | op | address | | | | |  |  |  |
| j | 000010 | address | | | | | j 10000 | goto 10000 | PC <- (PC+4)[31..28],address,0,0   ；address=10000/4 |
| jal | 000011 | address | | | | | jal 10000 | $31<-PC+4;  goto 10000 | $31<-PC+4；PC <- (PC+4)[31..28],address,0,0    ；address=10000/4 |