# **Decode**

译码阶段要完成的工作：识别指令的类型，指令所需操作数（寄存器号或立即数）以及指令的一些控制信号。

1.产生指令对应的FU信号以及运算控制信号control。

表1 控制信号转化表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 对应FU | 汇编助记符 | 操作码 | funct3 | funct7 | control | fu\_wen |
| ALUR |  |  |  |  |  |  |
| 15条 | ADD | 0110011 | 000 | 0000000 | 0000 | 1000 |
|  | SUB | 0110011 | 000 | 0100000 | 0001 | 1000 |
|  | SLL | 0110011 | 001 | 0000000 | 0010 | 1000 |
|  | SLT | 0110011 | 010 | 0000000 | 0011 | 1000 |
|  | SLTU | 0110011 | 011 | 0000000 | 0100 | 1000 |
|  | XOR | 0110011 | 100 | 0000000 | 0101 | 1000 |
|  | SRL | 0110011 | 101 | 0000000 | 0110 | 1000 |
|  | SRA | 0110011 | 101 | 0100000 | 0111 | 1000 |
|  | OR | 0110011 | 110 | 0000000 | 1000 | 1000 |
|  | AND | 0110011 | 111 | 0000000 | 1001 | 1000 |
|  | ADDW | 0111011 | 000 | 0000000 | 1010 | 1000 |
|  | SUBW | 0111011 | 000 | 0100000 | 1011 | 1000 |
|  | SLLW | 0111011 | 001 | 0000000 | 1100 | 1000 |
|  | SRLW | 0111011 | 101 | 0000000 | 1101 | 1000 |
|  | SRAW | 0111011 | 101 | 0100000 | 1110 | 1000 |
| ALUI |  |  |  |  |  |  |
| 14条 | ADDI | 0010011 | 000 |  | 0000 | 0100 |
|  | SLTI | 0010011 | 010 |  | 0001 | 0100 |
|  | SLTIU | 0010011 | 011 |  | 0010 | 0100 |
|  | XORI | 0010011 | 100 |  | 0011 | 0100 |
|  | ORI | 0010011 | 110 |  | 0100 | 0100 |
|  | ANDI | 0010011 | 111 |  | 0101 | 0100 |
|  | SLLI | 0010011 | 001 | 000000 | 0110 | 0100 |
|  | SRLI | 0010011 | 101 | 000000 | 0111 | 0100 |

表1 控制信号转化表（续）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 对应FU | 汇编助记符 | 操作码 | funct3 | funct7 | control | fu\_wen |
|  | SRAI | 0010011 | 101 | 010000 | 1000 | 0100 |
|  | LUI | 0110111 |  |  | 1001 | 0100 |
|  | ADDIW | 0011011 | 000 | 0000000 | 1010 | 0100 |
|  | SLLIW | 0011011 | 001 | 0000000 | 1011 | 0100 |
|  | SRLIW | 0011011 | 101 | 0100000 | 1100 | 0100 |
|  | SRAIW | 0011011 | 101 | 0000000 | 1101 | 0100 |
| BRU |  |  |  |  |  |  |
| 9条 | JAL | 1101111 |  |  | 0000 | 0010 |
|  | JALR | 1100111 | 000 |  | 0001 | 0010 |
|  | BEQ | 1100011 | 000 |  | 0010 | 0010 |
|  | BNE | 1100011 | 001 |  | 0011 | 0010 |
|  | BLT | 1100011 | 100 |  | 0100 | 0010 |
|  | BGE | 1100011 | 101 |  | 0101 | 0010 |
|  | BLTU | 1100011 | 110 |  | 0110 | 0010 |
|  | BGEU | 1100011 | 111 |  | 0111 | 0010 |
|  | AUIPC | 0010111 |  |  | 1000 | 0010 |
| AGU |  |  |  |  |  |  |
| 11条 | LB | 0000011 | 000 |  | 0000 | 0001 |
|  | LH | 0000011 | 001 |  | 0001 | 0001 |
|  | LW | 0000011 | 010 |  | 0010 | 0001 |
|  | LBU | 0000011 | 100 |  | 0011 | 0001 |
|  | LHU | 0000011 | 101 |  | 0100 | 0001 |
|  | LWU | 0000011 | 110 |  | 0101 | 0001 |
|  | LD | 0000011 | 011 |  | 0110 | 0001 |
|  | SB | 0100011 | 000 |  | 0111 | 0001 |
|  | SH | 0100011 | 001 |  | 1000 | 0001 |
|  | SW | 0100011 | 010 |  | 1001 | 0001 |
|  | SD | 0100011 | 011 |  | 1010 | 0001 |

注：共49条指令，黄色区域为RV64I中对32位数据进行运算的指令，蓝色区域移位次数字段为6位，所以末尾的funct字段为6位

1. 完成立即数的拼接：按照不同需求分为9类:

I\_unsign(SLTU,LBU,LHU,LWU)4

I\_sign(ADDI,SLTI,XORI,ORI,ANDI,ADDIW,JALR,LB,LH,LW,LD)11

I\_shamt(SLLI,SRLI,SRAI)3

I\_shamt\_w(SLLIW,SRLIW,SRAIW)3

S(SB,SH,SW,SD)4

B\_unsign(BEQ,BNE,BLT,BGE)4

B\_sign(BLTU,BGEU)2

U(LUI,AUIPC)2

J(JAL)1

表2 立即数拼接表

|  |  |  |
| --- | --- | --- |
| 指令格式 | | 立即数 |
| I | unsign | {52'b0,instr[31:20]} |
| sign | {{52{instr[31]}},instr[31:20]} |
| shamt | {58'b0,instr[25:20]} |
| shamt\_w | {59'b0,instr[24:20]} |
| S | | {{52{instr[31]}},instr[31:25],instr[7],instr[11:8]} |
| B | unsign | {51'b0,instr[31],instr[7],instr[30:25],instr[11:8],0} |
| sign | {{51{instr[31]}},instr[31],instr[7],instr[30:25],instr[11:8],0} |
| U | | {{44{instr[31]}},instr[31:12]} |
| J | | {{32{instr[31]}},instr[31:12],12'b0} |

3.译码阶段的输入输出：

input：instr0[31:0]，instr1[31:0]（来自于IF1/ID流水线寄存器）。

output：rs0[4:0]，rt0[4:0]，rd0[4:0]，rs1[4:0]，rt1[4:0]，rd1[4:0]，rs0\_en，rt0\_en，rd0\_wen，rs1\_en，rt1\_en，rd1\_wen，（寄存器重命名需要的信号）imm0[63:0]，imm1[63:0]，fu\_wen0[3:0]，fu\_wen1[3:0]，bru\_rs\_src0，bru\_rt\_src0，bru\_type\_src0，bru\_rs\_src1，bru\_rt\_src1，bru\_type\_src1，mem\_mask0[7:0]，mem\_mask[7:0]，memread0，memread1，memwrite0，memwrite1，trob\_wen0，trob\_wen1，rob\_wen0，rob\_wen1。

4.实现：通过instr[6:2](op)，instr[14:12](funct3)，instr30(为0时是加法或左移，为1时是减法或右移)可以完成所需信号的产生。mem\_mask[7:0]的产生：当指令为LB，LBU，SB时，mem\_mask = 01H；当指令为LH，LHU，SH时，mem\_mask = 03H；当指令为LW，LWU，SW时，mem\_mask = 0FH；当指令为LD，SD时，mem\_mask = FFH；

5.分支预测错误后需要进行恢复，刷新之后需要从正确的地址开始取指，因此需要一个保存分支指令正确跳转地址，考虑到分支指令在是少数，如果放在ROB中浪费资源，因此单独设计了tROB用于刷新后的正确取指。因此对于分支指令需要产生trob\_wen信号，可以通过检测instr[6:5]为11，做与运算即为trob\_wen信号。

# **Register Renaming**

1.寄存器重命名方式：基于统一的PRF进行重命名

2.选择原因：寄存器的值只需要写入一次，不需要再进行移动，功耗低；

源寄存器的值相比其他两种只存在于一个地方（PRF），在指令退休时，另外两种方法需通知使用该寄存器的指令，因此基于统一PRF进行重命名降低了设计复杂度。

3.sRAT表项设计：32个表项，4个读端口，2个写端口表项设计如表3所示：

表3 sRAT表项设计

|  |
| --- |
| PRF |
| 6 |

6位宽，表项数量等于ARF的数量，为32个；共32\*6=192bit，PRF为物理寄存器号。初始r0-r31映射p0-p31放入sRAT，p32-p63放入空闲列表。

4.FreeList设计：由于设计采用动态双发射，考虑最坏情况，空闲列表需要双读端口,双写端口；同时由于空闲列表为空时不能进行重命名，因此为空时需要产生空信号暂停取值，译码。

5.RAW相关性检测：解决RAW相关性采用组内相关性检查的办法，源寄存器对应的物理寄存器可能来自于sRAT,也可能来自于本周期的空闲列表。双发射时，第一条指令不会出现这种情况，只对第二条进行组内相关性检测（与访问sRAT并行）。

6.WAW相关性检测：当两条指令的目的寄存器相同时，只需要instr1写RAT就可以了，将instr0对应的RAT写端口使能信号置为无效。（5、6检测电路可由图1可知）

7.旧映射关系的释放：当指令退休时，会将新的映射关系写入aRAT中，加上我们比较特殊的恢复策略，可以利用aRAT写入aFreeList中的旧映射关系(详细内容见8)，同时写入FreeList中，即可完成旧映射关系的释放。

8.sRAT和FreeList恢复：本次设计不考虑中断和异常，因此只有在分支预测失败后需要进行恢复，团队的恢复策略是检测到错误暂停前面的流水线，后面的流水线继续执行，直到分支预测错误的指令成为ROB中最旧的一条，刷新流水线，等待恢复完成，从正确地址重新取指。综合考虑了恢复代价和复杂度，选择使用aRAT和aFreeList对sRAT和FreeList进行恢复（可一拍完成），具体设计如下：

1.aRAT和aFreeList的初始化与sRAT和FreeList的保持一致（aRAT和sRAT都顺序映射前32个物理寄存器号，aFreeList和FreeList中都存放后32个物理寄存器号）注;因为初始化一致，且指令重命名和提交都是顺序的，因此aFreeList弹出的寄存器号一定是对应的要写入的新映射的物理寄存器号。

2.在提交阶段，ROB先给aFreeList发送读信号，也就是本周期退休指令中regwrite信号为1的个数。共三中情况（0，1，2），读指针对应的变化为（不变，加1，加2），然后将目的寄存器号发送到aRAT,读出旧映射关系写入aFreeList的队尾，同时将新映射关系写入aRAT中。

由此可以保证aRAT和aFreeList总是正确的，刷新时覆盖sRAT和FreeList就可完成恢复。

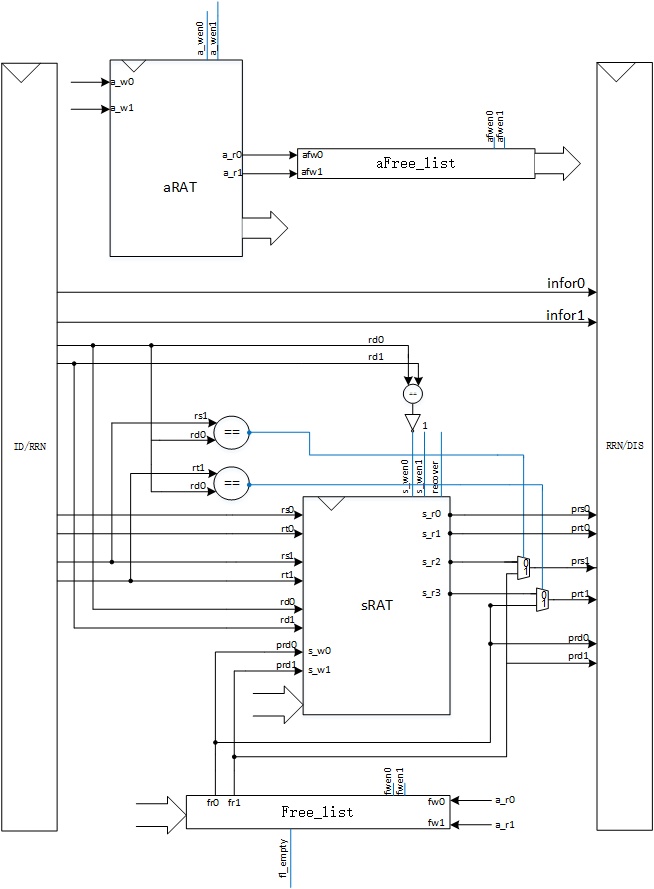


图1 重命名阶段总图

# **Dispatch**

本次设计将就绪状态检测放到了发射阶段，因此本阶段需要完成:

1. 对ROB写入，同时反馈写指针指向的地址rob\_id0,rob\_id1。

图1 写rob拼接信息表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Br\_type | areg | preg | memwrite | regwrite | complete | Bru\_precorrect | br\_taken |
| trob\_wen | ard | prd | memwrite | rd\_wen | 0 | 0 | 0/1 |

注：JAL，JALR指令bru\_taken = 1，其他指令均为0。

1. 对tROB写入，同时反馈写指针指向的地址trob\_id0,trob\_id1。

图2 写trob拼接信息表

|  |  |
| --- | --- |
| modify | pc |
| 0 | pc |

1. 根据译码阶段产生的fu\_wen信号，将指令信息以及rob\_id,trob\_id按照对应的发射队列表项进行拼接，带着fu\_wen进入流水寄存器。

fu\_wen = 1000:{1000,alur\_data[31:0],89’b0,232’b0,102’b0}

fu\_wen = 0100:{0100,32’b0,alui\_data[88:0],232’b0,102’b0}

fu\_wen = 0010:{0010,32’b0,89’b0,bru\_data[231:0],102’b0}

fu\_wen = 0001:{0001,32’b0,89’b0,232’b0,agu\_data[101:0]}

因为将rdy检测放到了发射阶段，因此rdy全部置0。没有源寄存器但发射队列表项有的指令拼接时对应内容置000000，既不会影响计算结果，也不会阻碍发射。

# Issue

发射阶段要完成的工作有：

1.完成就绪位检测；

存储物理寄存器（StateRegister）就绪位寄存器的设计：

64个1bit寄存器，0为未就绪，1为就绪。

初始化：x0为0寄存器，x1保存子程序调用返回地址，x2用作栈指针并保存栈基址，x3用作全局指针并保存全局变量所在位置的基址；由于这四个寄存器的特殊用途，因此初始化为1（就绪），其他28个寄存器（x4-x31）初始化为0（未就绪）。

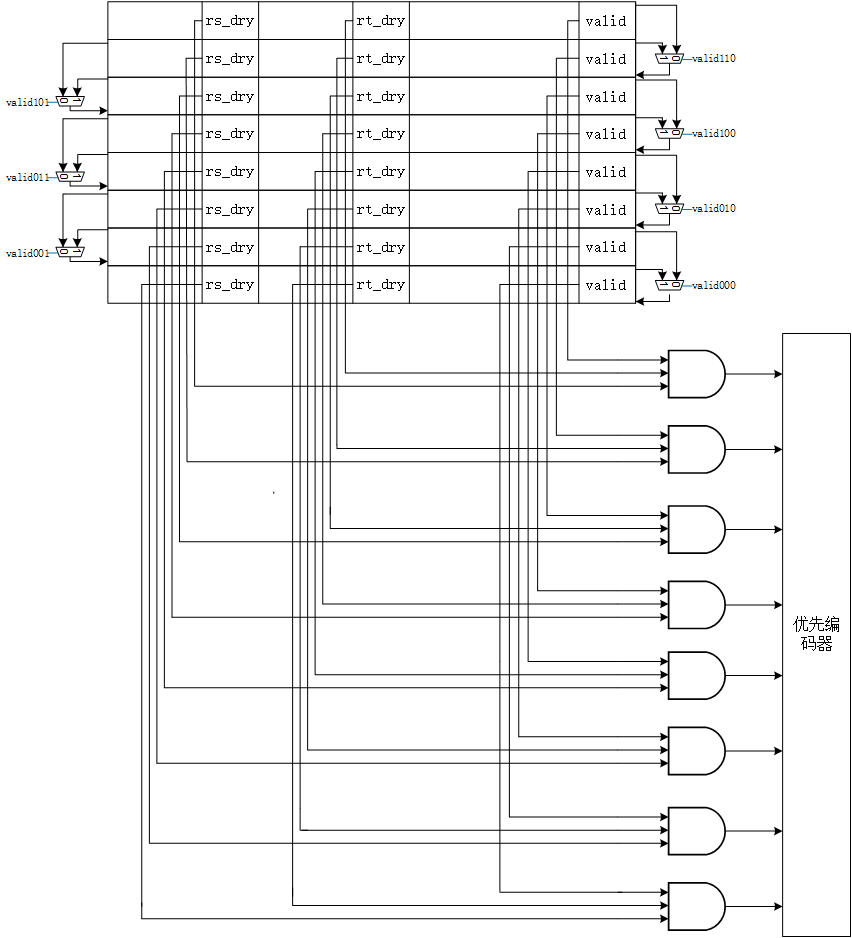
就绪置位：ALUR，ALUI，BRU在发射后将对应的目的寄存器进行置1，AGU在取出数据也就是取数据完成进行置1。

未就绪置位：当指令退休时，将对应的旧的映射关系对应的物理寄存器置0（本设计的来源为退休阶段从aRAT读出的旧的物理寄存器）;当分支预测失败时，增加一个永远正确的aStateRegister（指令退休才会修改状态）,利用aStateRegister恢复StateRegister。这么设计主要还是考虑本设计分支预测错误时的恢复是在该分支成为ROB中最旧的指令进行刷新，希望能在一拍完成，因此没有采用回退。

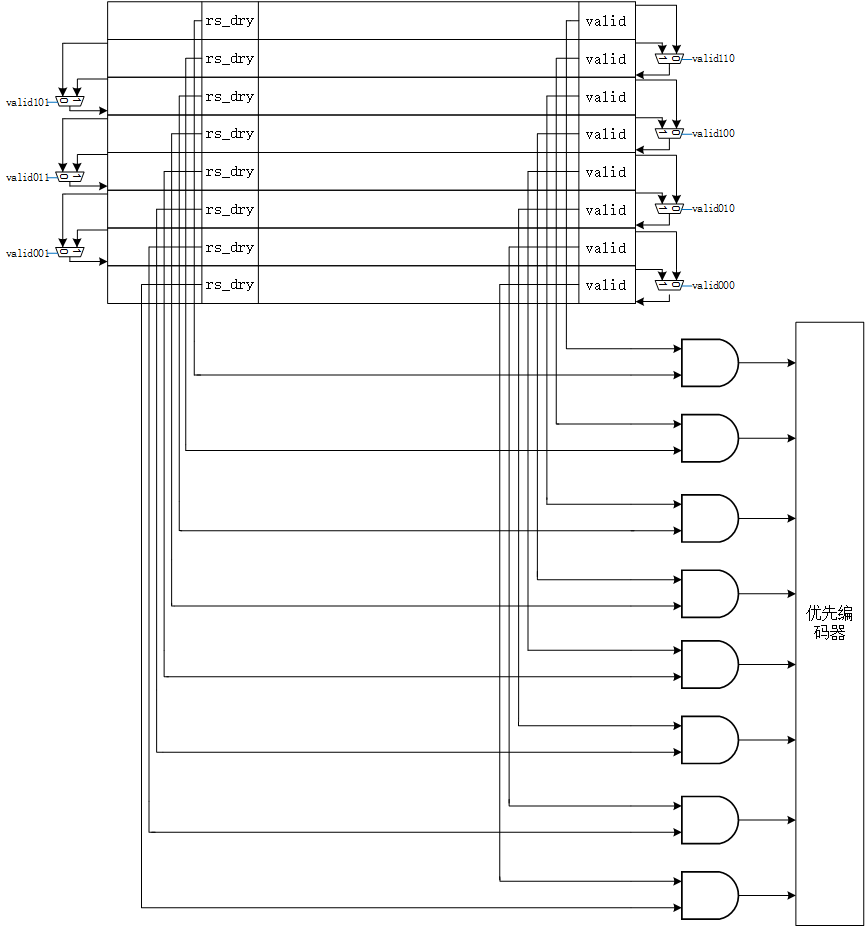
2.就绪位检测放在发射阶段的原因：如果在分派阶段进行检测，发射阶段准备发射的指令0与分发阶段指令1存在RAW相关，这时还没有发射，也就没有修改StateRegister，到了下一周期指令0已经离开了发射队列，也就无法通过唤醒电路对指令1进行唤醒，就会导致指令0永远无法被唤醒的错误。

3.仲裁电路设计：由于AGU采用部分乱序，所以Store指令只有是发射队列中最旧的指令且就绪才可发射，发射队列中第一条Store指令之前的Load指令可乱序发射（也就是可被仲裁电路选中）其他三个队列。

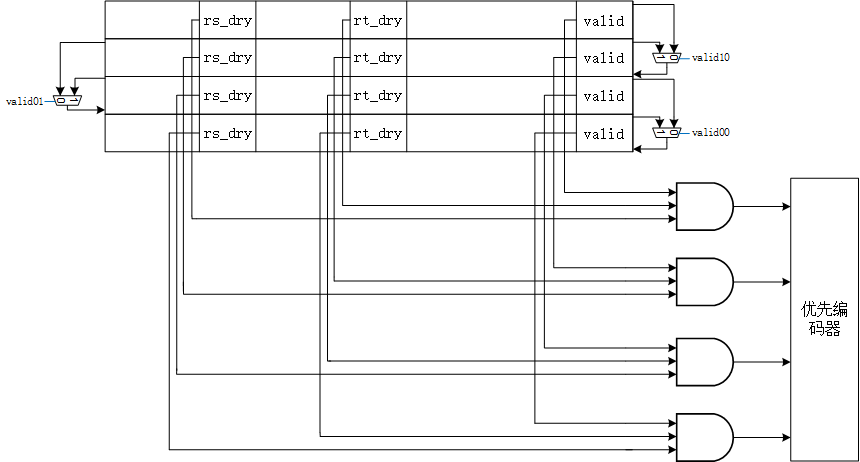
ALUR\_RS：



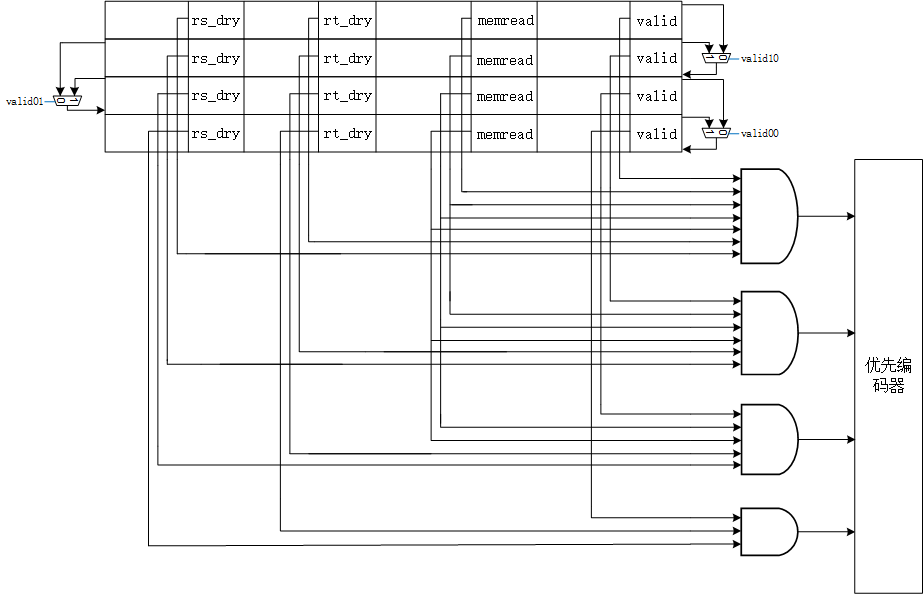
ALUI\_RS：



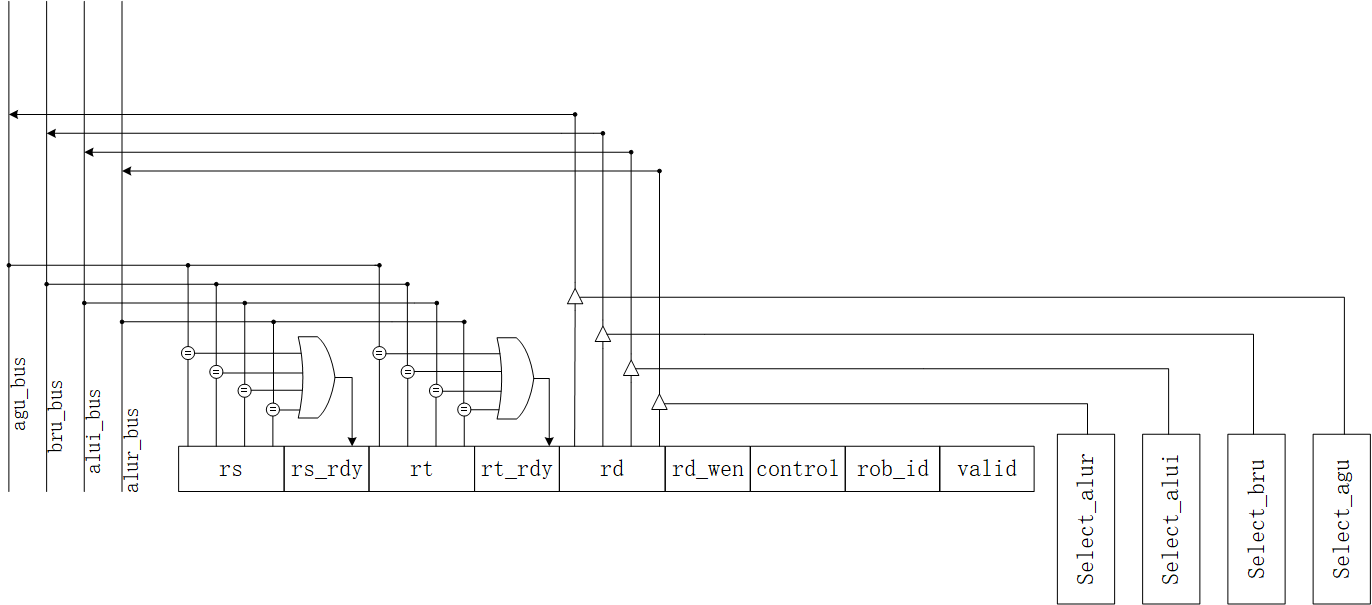
BRU\_RS：



AGU\_RS:



4. 唤醒电路设计：由于Load和Store指令执行时间不确定，因此在完成存储器访问进行唤醒，其他指令在发射后就可唤醒。



5.发射队列表项设计：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| rs | rs\_rdy | rt | rt\_rdy | rd | rd\_wen | control | rob\_id | valid |
| 6 | 1 | 6 | 1 | 6 | 1 | 4 | 6 | 1 |

ALUR:

rs:源寄存器1，参与唤醒，转发，读寄存器

rs\_rdy:源寄存器1就绪位，访问就绪状态寄存器得到，后通过唤醒电路修正

rt:源寄存器2，参与唤醒，转发，读寄存器

rt\_rdy:源寄存器2就绪位，访问就绪状态寄存器得到，后通过唤醒电路修正

rd:目的寄存器，参与唤醒，转发，写寄存器

rd\_wen:目的寄存器写使能信号

control:ALUR运算的控制信号

rob\_id:指令在ROB中的对应位置，用于执行阶段对完成位置位

valid:表示该表项是否有效，作为发射条件以及压缩多路选择器信号

ALUI:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| rs | rs\_rdy | imm | rd | rd\_wen | control | rob\_id | valid |
| 6 | 1 | 64 | 6 | 1 | 4 | 6 | 1 |

rs:源寄存器1，参与唤醒，转发，读寄存器

rs\_rdy:源寄存器1就绪位，访问就绪状态寄存器得到，后通过唤醒电路修正

imm:译码阶段按照指令所需拼接好的64位立即数

rd:目的寄存器，参与唤醒，转发，写寄存器

rd\_wen:目的寄存器写使能信号

control:ALUI运算的控制信号

rob\_id:指令在ROB中的对应位置，用于执行阶段对完成位置位

valid:表示该表项是否有效，作为发射条件以及压缩多路选择器信号

BRU:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| rs | rs\_rdy | rt | rt\_rdy | imm | rd | rd\_wen | rs\_src | rt\_src | type\_src |
| 6 | 1 | 6 | 1 | 64 | 6 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| pc | pre\_taken | pre\_target | control | rob\_id | trob\_id | valid |
| 64 | 1 | 64 | 4 | 6 | 4 | 1 |

rs:源寄存器1，参与唤醒，转发，读寄存器

rs\_rdy:源寄存器1就绪位，访问就绪状态寄存器得到，后通过唤醒电路修正

rt:源寄存器2，参与唤醒，转发，读寄存器

rt\_rdy:源寄存器2就绪位，访问就绪状态寄存器得到，后通过唤醒电路修正

imm:译码阶段按照指令所需拼接好的64位立即数

rd:目的寄存器，参与唤醒，转发，写寄存器

rd\_wen:目的寄存器写使能信号

rs\_src:源操作数1选择信号

rt\_src:源操作数2选择信号

type\_src:有条件分支指令或无条件跳转选择信号

pc:指令在存储器中的地址

pre\_taken:分支预测方向

pre\_target:分支预测地址

control:ALUI运算的控制信号

rob\_id:指令在ROB中的对应位置，用于执行阶段对完成位置位

trob\_id:用于执行阶段访问pc，预测地址以及对预测错误的指令进行修正

valid:表示该表项是否有效，作为发射条件以及压缩多路选择器信号

AGU:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| rs | rs\_rdy | rt | rt\_rdy | imm | rd | rd\_wen |
| 6 | 1 | 6 | 1 | 64 | 6 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| memwrite | memread | mem\_mask | control | rob\_id | valid |
| 1 | 1 | 8 | 4 | 6 | 1 |

rs:源寄存器1，参与唤醒，转发，读寄存器

rs\_rdy:源寄存器1就绪位，访问就绪状态寄存器得到，后通过唤醒电路修正

rt:源寄存器2，参与唤醒，转发，读寄存器

rt\_rdy:源寄存器2就绪位，访问就绪状态寄存器得到，后通过唤醒电路修正

imm:译码阶段按照指令所需拼接好的64位立即数

rd:目的寄存器，参与唤醒，转发，写寄存器

rd\_wen:目的寄存器写使能信号

memwrite:存储器写信号（store）

memread:存储器读信号（load）

mem\_mask：01H表示字节（8bits），03H表示半字(16bits)，0FH表示字(32bits)，FFH表示双字(64bits)。

control:ALUI运算的控制信号

rob\_id:指令在ROB中的对应位置，用于执行阶段对完成位置位

valid:表示该表项是否有效，作为发射条件以及压缩多路选择器信号