指令编码

- opcode=0x0b 00010[0x02]+11(其余的都是压缩指令)

ins	cvec[31:27]	funct2[26:25]	rs2[24:20]	rs1[19:15]	funct3[14:12]	rd[11:7]
ld_tilerow	8:ofmap 0-3:ifmap 4-7:kernel	0	/	地址	010	/
aamul	1:02[0] 2:12[1] 3:21[2] 4:31[3] 5:1221[tt]	1	/	/	000	/
triadd	1:0tt[0] 2:3tt[3]	2	/	/	000	/
oacc	3	2				
wb_tile	0	3	/	地址	010	/

注意事项

- 未解决（目前是将ld8放到序列最前暂时解决）
  - ld8 fault，导致后续指令错乱
    - 可能是因为错误路径上的cal对cvec进行了写，异常处理之后同一指令又接着写
  - 目前处理方法是增加archCtrlTable，EMMM

        - 退休时写-cmtinsts
        - squashed 退休【按序】时，若它cusdone（意味着改变了cusCtrlTable）则replay----全设置策略（这样实现最简单）

- squash
    - 是这么个情况：某些被squash的custom_ins，拉高了busy，但无法将其拉低造成阻塞
    - squash也会在rob里退休（没进rob的除外，但这些指令肯定也不会拉高busy），在相应代码调一下函数
        - 判断一些条件，然后拉低busy即可

- 访存违例问题指的是：load同地址指令先于store去做，在咱们这不会
    - 例如store1 load1 store2 load2，store1->load1->load2

- assert：sn大的指令不会唤醒sn小的指令

- aamul1221目前是写9和10，triadd也用9和10
    - 但aamul1221完成后不能将12置为empty，因为aamul31可能要用；还是由oacc来全置零

- 目前更新kernel和其他custom保持互定序
    - ldk:必须等其他cus_ins不再busy
    - 其他：等ldk不busy，且无notrdyldk
        - notrdyldk不同于notrdylist，前者是暂时不可发射的ldk(包括因rs1未就绪导致)，后者仅包含cusCtrl导致的未发射指令
        - 目前notrdyldk基本能维护只有至多一个inst在list里，所以pop_front和push_back用得有些不对也无所谓，但有机会还是得改改吧

策略

- 循环外ld_tilerow将 kernel写入 4 5 6 7 号寄存器
- 循环内ld_tilerow将 ifmap 写入 0 1 2 3 号寄存器
    - ld_tilerow 将 ofmap 写入 8 号寄存器
- 只有一个流水部件：4浮点加 4浮点加 4浮点乘
    - map就绪的aamul执行，delay=7，修改map03 map:t1t2
        - 1221是delay+1=8

- map就绪的triadd执行，delay=4，修改map03，
- map038就绪，即可执行oacc，【delay=(4)+1，其实是两组，但流水】map[0123910]=0,map[8]=2
- map8 ready,exe vstore,map[8]=0

指令相关性

迭代相关用busyVec处理：

- 同一条指令只有先来的那个指令会被执行
- 指令在插入iq和插入notrdyList时都是按序的

设置一个map，记录每个vec的进行状态【其实就是一个整型数】，数据和控制相关用map处理：

ins	写后读前相关	读后写前相关	写后读后相关	读后写后相关
ld_tilerow	rs1-地址	map[cvec]=0	map[cvec]=1	
aamul	map[idx12]=1 *map[idx1+4]=1		map[idx1]=2	
aamul1221	map[1256]=1		map[tt]=2	
triadd	map[0tt3]=2		map[03]=3	
oacc	map[03]=3 map[8]=1		map[8]=2	map[0123]=0
wb_tile	map[8]=2 rs1-地址			map[8]=0

部件

ins	stage1	stage2	stage3
aamul	vec_a + vec_b = vec_temp1	vec_temp1 内部元素 fadd = vec_temp2	vec_temp2 * vec_c = vec_res
triadd	vec_a + vec_b = vec_temp1	vec_c + vec_temp1 = vec_res	

ins	stage1	stage2	stage3
oacc	vo_elem + va_elem = v_t1 va_elem + va_elem = v_t2	v_t1+v_t2=v_res	

QA

- 为什么不一下子读16个
  - 因为部件有限，不想等前面16个数全部清空才唤醒指令
  - 对于后面指令的唤醒也有同样的道理
  - 此外，这里load部件刚好也是4个，做4*4也算合理
- 那为什么不能1个1个读
  - 首先是感觉可能一起读效率高
  - 其次是后面我需要进行向量操作
    - 因为发现扩充部件后，刚好有合适的策略
      - 可以流水啥的

Code

- 之前没咋记录，可能只能看看github提交记录了
  - iq-insert，writebackinst_wakeup是两个关键点，看看注释
- busyVec循环迭代 ctrlvec数据相关
- notrdylist特指cus_ctrl带来的依赖；notrdyldk包含all未发射ldk，用于ldk与other_cus互定序
  - 指令被any列表引用，他是不能被销毁的，会造成count>1500；
  - 所以目前是在check_notrdy时销毁squashed，而notrdyldk应该最多一条，所以无需

实验

- 编译器改动：识别扩展指令
- 源程序改动：使用扩展指令

- Gem5源码改动：实现扩展指令
  - isa相关
    - 修改decoder.isa等、扩充寄存器，完成扩展指令定义
    - 这一部分定义了指令的需要执行的操作
  - cpu相关
    - 主要修改发射、写回部分（FU仅需简单配置）
      - 将扩展指令适配在流水线中