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:output 0-3:input 4-7:kernel	0	/	地址	010	/
aamul	1:02[0] 2:12[1] 3:21[2] 4:31[3] 5:1221[12]	1	/	/	000	/
triadd/oacc	1:012[0] 2:321[3] 3:oacc[8]	2	/	/	000	/
wb_tile	0	3	/	地址	010	/

注意事项

- aamul1221是担心aamul12和aamul21读写冲突，因为是流水且操作数相同，所以延迟也就多一拍
- triadd 那23不知道可不可以提前置0，若置零可能后面的ld会把寄存器填上,所以暂时放在oacc全置零
- 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
- 目前更新kernel和其他custom保持互定序
- ***branch mispredict ????? ????? ????? ????? ????? ????? ????? ????? ?????***
- 瓶颈
  - output只有一个，其实可以双buffer，这样不用等写回，就可以load

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=6/7，修改map0123
  - map就绪的triadd执行，delay=3/4，修改map03，map12待定
  - map038就绪，即可执行oacc，【delay=(3or4)+1，其实是两组，但流水】map[03]=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	数据前相关	控制前相关	数据后相关	控制后相关
ld_tilerow	rs1-地址	map[cvec]=0	map[cvec]=1	
aamul	map[idx12]=1 *map[idx1+4]=1		map[idx1]=2	
triadd	map[idx123]=2		map[idx1]=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

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

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
    - cpu相关
        - 主要修改发射、唤醒部分