Basic Knowledge

乱序的一些常见的Trade-off

分支预测-性能&硬件复杂度

load&store-相关性检查&访存miss硬件复杂度

CheckPoint个数设置&面积

IQ个数-性能&复杂度（仲裁）-关键路径

...

处理方式

Centralized&Distributed

CIQ：共用一个IQ

DIQ：每个FU分配一个IQ

Tradeoff:

对CIQ而言，IQ塞得比较满，利用率比较高，IssueWidth选择导致的Select和Wakeup的延迟比较高

对DIQ而言，IQ容量小一点，Select复杂度下降但是Wakeup复杂度还是较高，不是每个IQ都能充分利用，存在存在指令不平衡时候的stall。

CIQ+DIQ，可以进行耦合的同类指令内部采用CIQ方式，之间采用DIQ方式（eg：浮点、整型、地址等）

Compressing&Non-compressing

压缩：类比传统链表，FIFO，满足oldest-first

非压缩：类比静态链表，需要rdy位，不满足oldest-first

Tradeoff:

对压缩而言，在allocate和select时比较简单，但是复杂度比较高(mux)，指令移动功耗比较大

对非压缩而言，指令无须移动，但是产生更复杂的oldest-first选择电路，分配时产生“找到合适的IQ项”的开销

Data-capture&Non-data-capture

数据捕捉：在issue前读取ref，将ref的值或编号带入IQ，可以直接通过bypass从payload RAM中进行读写ROB rename+Data-capture -> (retire)ROB->ARF

非数据捕捉：寄存器被从IQ中选中后再去读取物理寄存器送到FU，不需要payload RAM

Tradeoff:

对数据捕捉而言，IQ相对较大，时序可能比较差，而且由于payload RAM的存在，面积更大

对非数据捕捉而言，在面积和功耗上有一定优势，但是设计复杂度比较高(update PRF->bypass)

Allocate

事实上，分配属于Dispatch的一部分，但是又关系到issue的select和wakeup，所以放在这里也是合适的

分配时主要针对压缩和非压缩的队列进行讨论，

针对非压缩IQ，由于不满足oldedt-first，所以在选择entry的时候，需要获取到表项的free状态，组织为free_list进行维护。另外，如果IQ过大，这里产生的latency也是不可接受的，所以可以将IQ拆分为不同的segment进行并行的查找。这里同样存在着**tradeoff**，即当某个segment full时，即使其余segment not full，依然不能进行分配。只能等该segment中存在空闲位置时，在in-order地进行写入。

针对压缩IQ，只需要从队列顶不断选择即可。

Select

1 of M

考虑指令的相关性，在乱序执行时，应该满足oldest-first。但是针对IQ（双端FIFO），仅通过读写指针位置，在队列顶弹出后，无法记录年龄。所以引入一个position bit去进行记录。

position bit相同时，ROB地址值越小，越old

position bit不同时，ROB地址值越大，越old

所以在实现1 of M的过程，需要进行logM次比较，在rdy的entry中选择age最小的指令(oldest)送入FU。

当然这里的比较同样存在简单的**tradeoff**，即当M过大时，mux实现比较电路过大，面积和延迟很差。这时可以考虑将地址信息和年龄比较并行送入比较，来优化掉部分延迟

N of M

同样，这里存在仲裁选择的**tradeoff**。由于我们有简单的1 of M Arbiter，当N比较小时，直接复用上面的仲裁器就可以简单得到结果。但是随着N增大，面积和延迟会拖垮性能，这时可以考虑分布式处理，即将FU和IQ拆成不同的type，通过demultiplexer来实现伪多路仲裁的效果。这里其实涉及到的就是CIQ和DIQ的**tradeoff**问题，如何拆分组合IQ是实现多路仲裁的关键。

Wake-up

唤醒在我看来就是“按图索骥”的过程，“按图”即把选中的指令的寄存器编号送上总线(stage1)，“索骥”即在IQ中寻找编号相同的过程(stage2)。

delay tag broadcast

(stage1)指令执行周期>1时，延迟N-1个周期再将其送上总线。这样可能会产生冲突，即被delay的指令和当周期的指令可能会同一周期被唤醒(tag bus冲突)。这时就需要一个list对各条指令进行监控，并通过Arbiter进行选择。这里同样存在**tradeoff**，即如果采用“如果没产生冲突则选择当周期执行，产生冲突则交给仲裁电路”的串行策略，会增大周期时间，但是如果采用“同时访问list和arbiter，但是可能产生指令未被仲裁其选中（更新&不产生冲突）“，会导致周期的浪费

delay wake-up

(stage2)在送上总线进行比较之后，将比较结果delay N-1个周期再置为ready。这样对tag bus的负载是均衡的，但是需要额外电路来实现对rdy位的延迟操作，即每个entry需要一个DELAY值来进行延迟，并将该值送上delay bus进行移位(one-hot编码)，直到延迟结束时才将rdy置1

rdy位和DELAY BUS的解耦

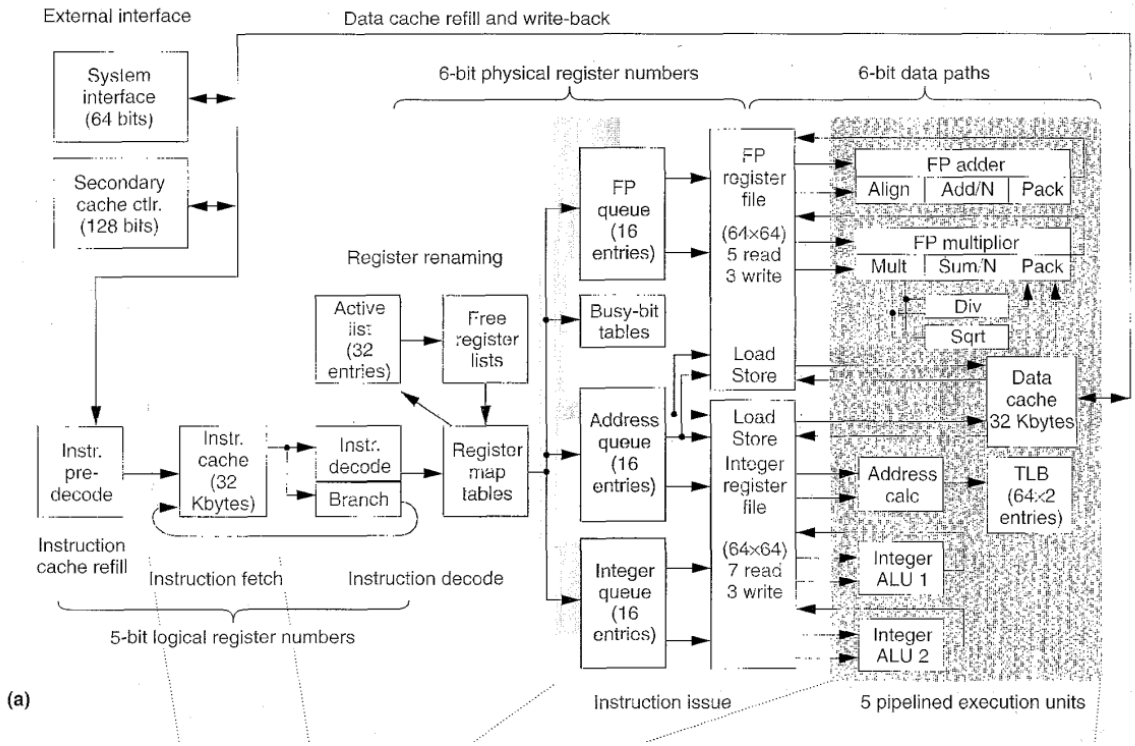
speculative wake-up

在上述提到的处理方法中，都要基于指令的周期数N来进行延迟操作。但是对于load之类无法确定指令周期的指令，N是未知的，对这类指令我们为了保证性能，通常期望其尽可能背靠背执行。即在访存时如果能提前获知其执行周期数（Dcache hit），则可以通过上述方法执行，否则可以通过Issue Queue Based Replay(即等指令确认正确执行后再离开IQ，自然需要一个isssued位去识别)或者Replay Queue Based Replay（即设置RQ，先执行指令后恢复）的方法进行解决。这部分内容展开有关Dcache时序和IQ的中指令的相关性，内容比较复杂，暂且不放在这里详细讨论，只需要记住把握住两个原则即：

1. 在访问内存时，并行执行的其他指令尽可能和访存指令不相关。否则尽量通过bypass实现”背靠背“执行
2. 因为可能需要Replay，故需要一组vector去记录执行的状态，以便清除和复原

论文相关

MIPS R10K



MIPS 10000K芯片在issue级将跳转指令和NOP外的指令放在三个指令队列中，即整型队列、浮点队列、地址队列，其队列采用非数据捕捉，非压缩的结构

1. 对整型队列而言：共16个entry，发射到两个ALU。其中：ALU1对分支和移位指令的优先级更高，ALU2对整型运算和div类指令优先级更高。由于采用非压缩结构，所以它并没有实现传统的oldest-first结构，而是选择通过FIFO的位置来决定issue priority。但是每次访问查询FIFO的请求都会提高older指令在ALU2中的优先级

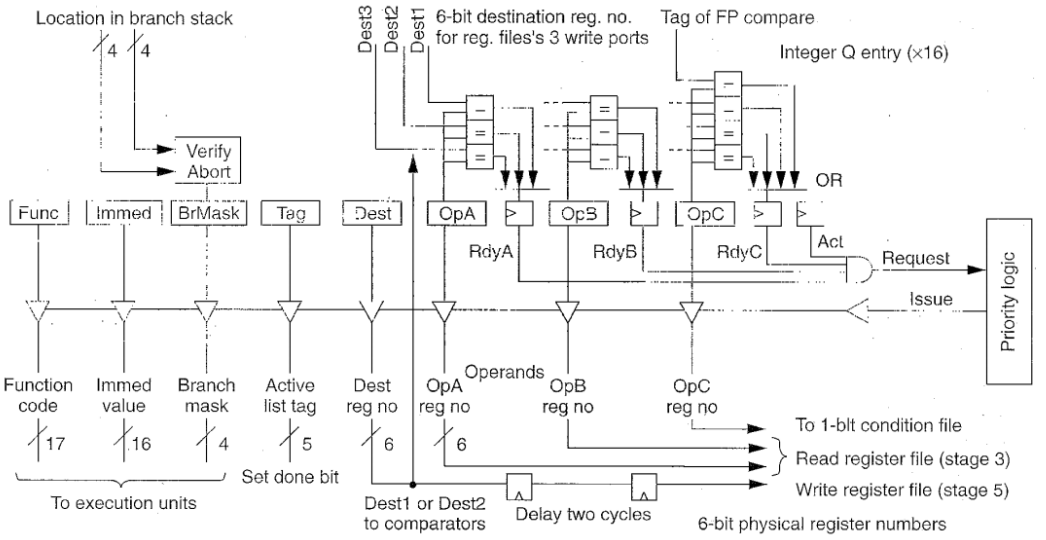


Figure 6. Integer instruction queue, showing only one issue port. The queue issues two instructions in parallel.

该图为整型指令队列的一个发射端口的示意图，该队列并行地发射两条指令。

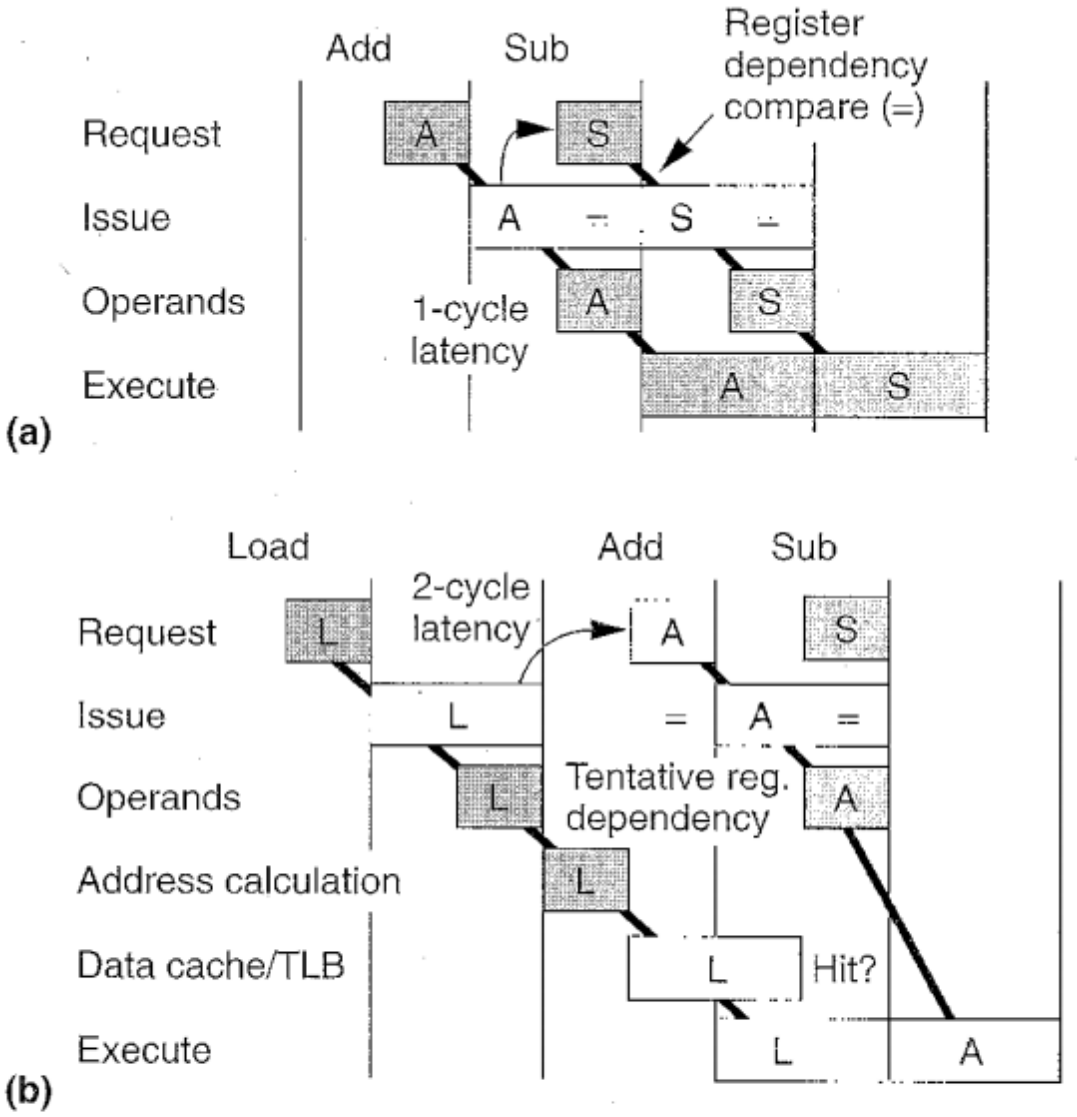


Figure 7. Releasing register dependency in the integer queue (a) and tentative issue of an instruction dependent on an earlier load instruction (b).

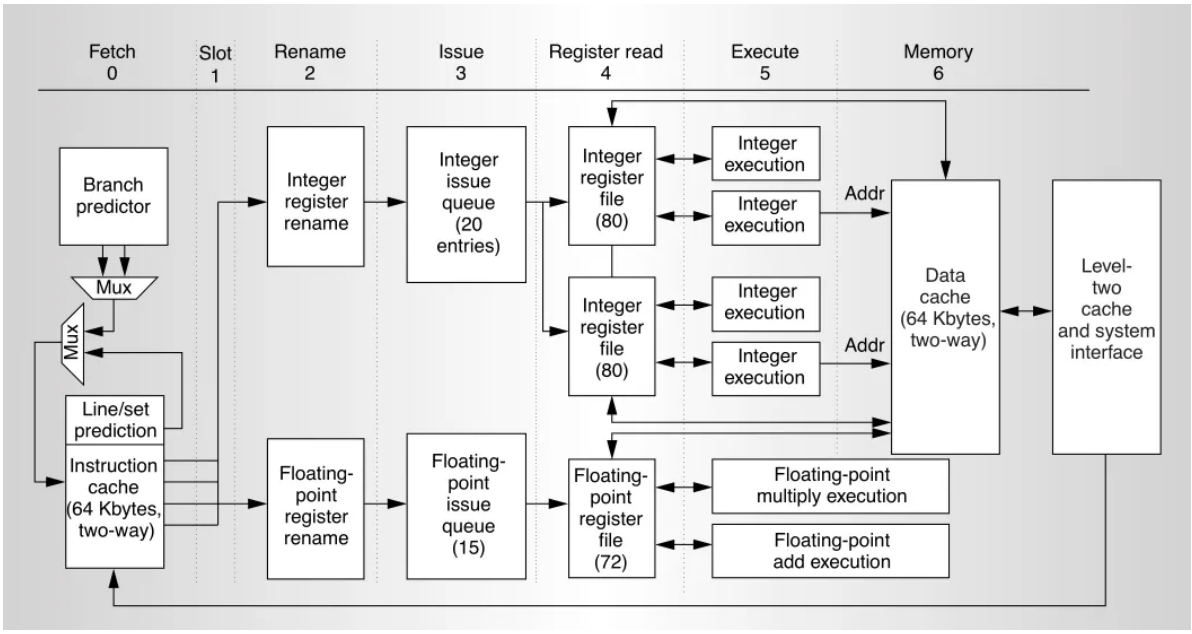
对Load类指令，为了实现2个周期的加载时延，会对对load指令有依赖的指令进行试探发射，如果load加载成功则该依赖指令因为在前一周期被发射，可以直接读数据。否则如果因为Dcache miss或者load的依赖失败，则暂时放弃对这条指令的发射

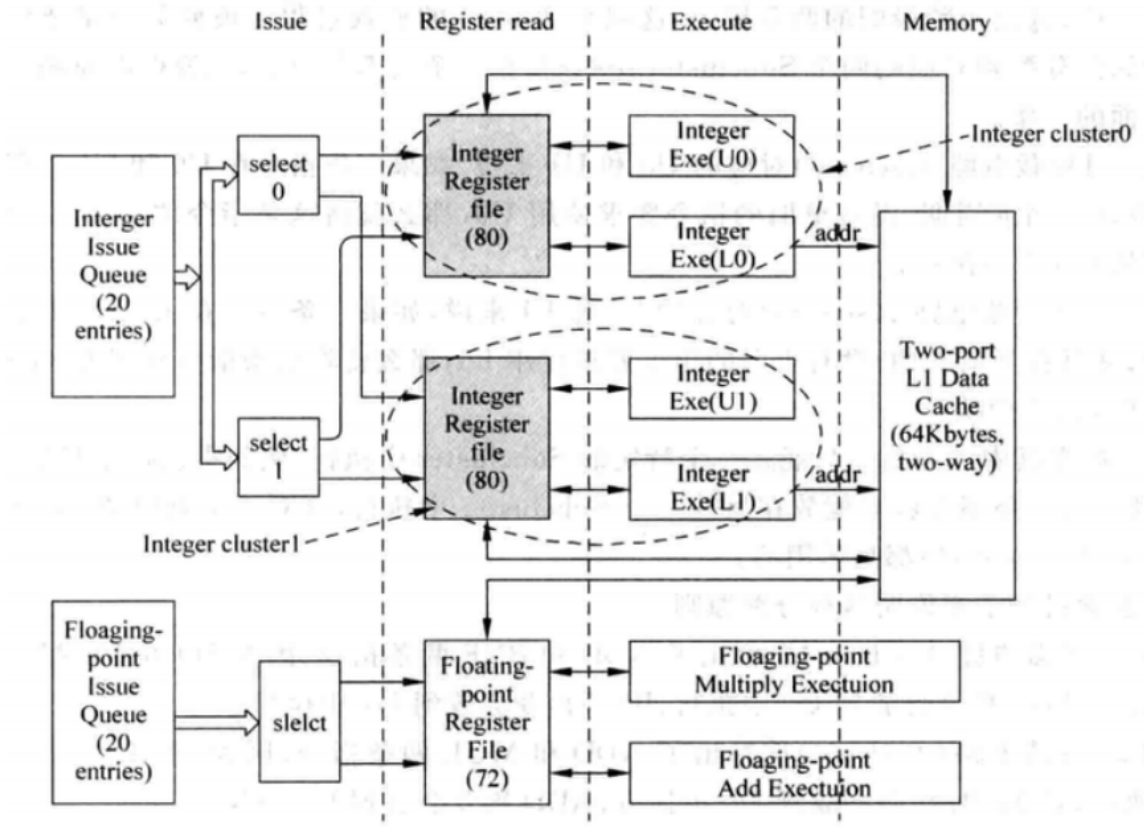
2. 浮点队列：和整型队列类似，但是不包括立即数部分。其Float Load有3个周期时延（存在writing delay的额外的写线路时延）
3. 地址队列：共16个entry，才用First in first out的循环链表来保持原始指令顺序，并在执行Load/Store指令时分配一个entry，在retire后释放，对内存的访问会根据其age来分配访存的优先级。在恢复一个错误的分支预测时，需要通过移动写指针来从队尾将此IQ进行释放
    1. 在检测访存指令的独立性时，该处理器采用2个16b*16b的矩阵进行处理。其中，第一个矩阵通过跟踪entry对于cache set的访问(Vaddr[13:5])来避免cache thrashing。该矩阵的任何一行或列都能直接被乱序指令访问，但是如果有多个队列同时访问该cache set address时，选择该line中oldest的项执行。第二个矩阵用于trace等待访问相同地址的指令，通过比较双字的地址和8位mask来判断地址的访问。当外部接口访问数据缓存时，处理器将其index与所有挂起队列的entry进行比较，如果访存地址匹配，则直接bypass到目标寄存器，如果匹配失败则clear该entry的状态
    2. 虽然IQ能按原始顺序进行访存，但是当外部接口提出访问时，可能会打破原始顺序。为了维护内存一致性，在指令retire之前，会创建一个exception并且flush整条流水线，然后重新开始执行。（简单粗

暴的异常+冲刷）

3. Store指令在其retire时才写入缓存
4. MIPS架构使用加载连线（Load-Link,LL）和条件存储（Store-Conditional,SC）指令对来模拟内存原子操作。由于它们并不需要对内存进行加锁访问，这些指令并不增加系统设计复杂性。eg：处理器用LL指令加载一个值，进行测试并修改，之后用SC指令条件写入储存。SC指令仅在此值没有冲突并且连线字（Link word）仍在缓存中时写入内存。处理器在结果寄存器中加载1或0来表明内存写是否完成。

Alpha 21264





该处理器采用了整型和浮点两个IQ，选择非数据捕捉，压缩结构。

其中：整型IQ可以存储20条指令，发射到4个FU，而浮点IQ存储15条指令，发射给2个FU。

对整型IQ而言：分成了两个cluster用以减少每个ref的读端口数，每个cluster又分出两个subcluster\:upper(U)和lower(L)。特殊的是该IQ没有实现4 of 20的仲裁，而是U0和U1公用select0，L0和L1共用select2，也就是将其简化为2 of 20的结构。其中，访存指令只能在L0和L1中执行，移位指令只在U0和U1中执行，对于加减运算和逻辑操作可以在任何cluster执行。由于采用压缩结构，故满足oldest-first，所以在选择仲裁指令时，只需要从IQ底部开始，查找两个rdy的指令issue即可。Alpha 21264除load外的其他指令的周期时间均是确定的（Store指令采取retire后才写入Dcache，故也是确定的），对于load指令，采取speculative wake-up的策略，遇到Dcache miss时，也会采用简单的clearSW并且重新仲裁的方式来执行。