北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

Computer Organization and Architecture

Chapter 14

Instruction Level Parallelism and Superscalar Processors

School of Computer Science（National Pilot Software Engineering School）
AO XIONG (熊翱)
xiongao@bupt.edu.cn

Preface

We have learned:

- Overview：
  - Basic Concepts and Computer Evolution 基本概念和计算机发展历史
  - Performance Issues 性能问题
- The computer system
  - Top level view of computer function and interconnection 计算机功能和互联结构顶层视图
  - Cache Memory cache存储器
  - Internal Memory 内部存储器
  - External Memory 外部存储器
  - Input& Output 输入输出
  - Operating System Support 操作系统支持

Preface

We have learned:

- The central processing unit：中央处理器
  - Computer arithmetic 计算机算术
  - Instruction sets: characteristics and function 指令集的特征和功能
  - Instruction Sets: Addressing Modes and Formats 指令集的寻址模式和格式
  - Processor Structure and Function 处理器结构和功能
  - Reduced Instruction Set Computers 精简指令集计算机
    - ✓ Instruction Execution Characteristics 指令执行特征
    - ✓ The Use of a Large Register File 大寄存器组方案
    - ✓ Compiler-Based Register Optimization 基于编译的寄存器优化
    - ✓ Reduced Instruction Set Architecture 精简指令集架构
    - ✓ RISC Pipelining RISC流水线
    - ✓ RISC Versus CISC Controversy RISC和CISC的争论

Preface

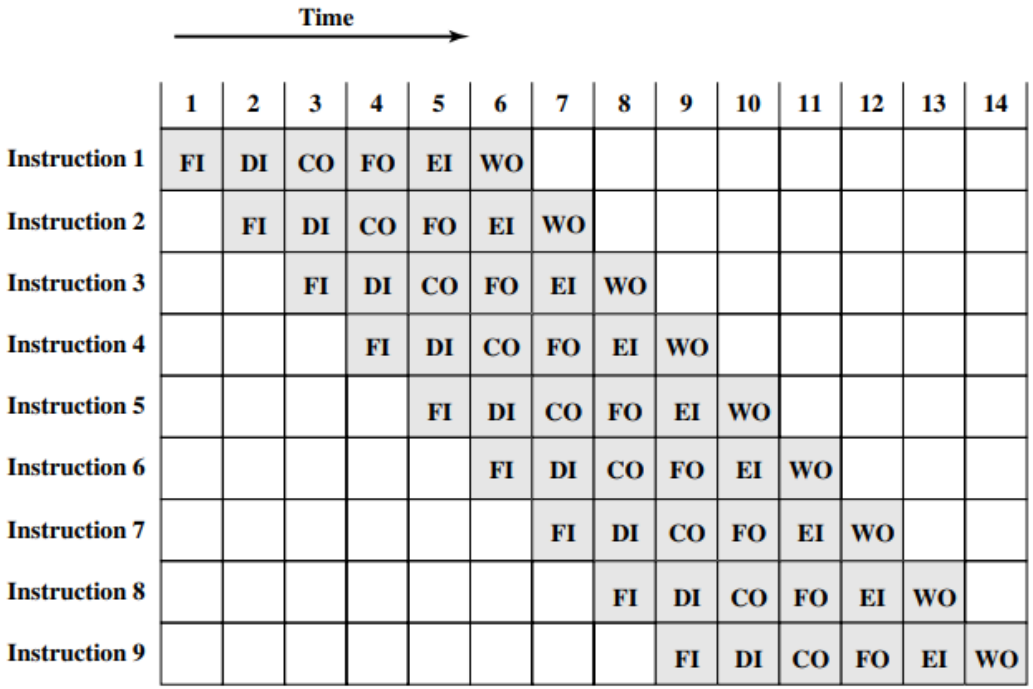
We will focus the following contents today:

- Instruction Level Parallelism and Superscalar Processors
  指令级并行和超标量处理器
  - Pipeline technology can improve the efficiency of instruction execution, so how to further improve the efficiency of instruction execution? 流水线技术可以提高指令执行的效率，那么如何进一步提高指令执行的效率呢?
  - How to solve instruction correlation in instruction level parallelism? 如何解决指令级并行中的指令相关性问题?
  - Working mode of typical superscalar processor 典型超标量处理器的工作模式

Outline

- Overview of Superscalar 超标量综述
- Design Issues of Superscalar 超标量设计问题
- Superscalar in Pentium 4 Pentium4中的超标量
- Superscalar in ARM CORTEX-A8 ARM中的超标量

Ideal pipeline 理想流水线



- 理想的指令流水线的执行过程
- 指令执行分为6个阶段，且不共享资源
- 每个时间单位都会有1个指令完成执行
- 指令数量足够多时，执行效率为原来的6倍

取指，译码，操作数地址计算，

取操作数，执行，写结果

Actual pipeline 实际的流水线

- Not all instructions require the same steps 不是所有的指令都需要相同的步骤
  - Some pipeline stages are idle 有些流水线阶段空闲
- Running time of different pipeline stages is different 不同的流水线阶段的运行时间是不一样的
  - running time of some pipeline stages is wasted 某些流水线阶段的运行时间存在浪费
- Instructions are not independent of each other 指令并不独立
  - Poor operation of the pipeline 流水线运行不畅

Problem about pipeline 流水线的问题

- The pipeline pauses due to dependencies between instructions, which is called pipeline risk 由于指令间存在依赖关系，导致流水线出现暂停的现象，称为流水线冒险
- There are three types of dependencies 有三种类型的依赖关系
  - Data dependence 数据相关性
  - Control dependence 控制相关性
  - resource dependence 资源相关性

Question

Assume that the pipeline runs in the ideal way

假定流水线按照理想的方式运行

- Is instruction pipelining truly parallel? 指令流水线算不算真正的并行执行?
- On the basis of pipeline, how to further improve the execution efficiency of instructions? 在流水线的基础上，如何进一步提高指令的执行效率?

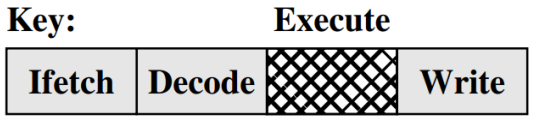
Question

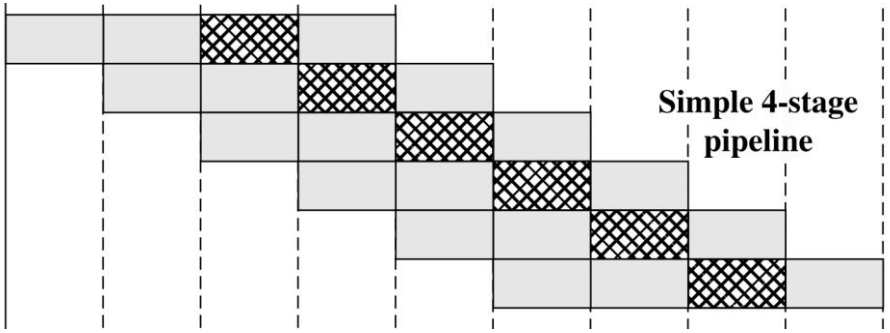
- Is instruction pipelining truly parallel? 指令流水线到底是不是指令的并行执行?
  - Yes: There are indeed multiple instructions in the pipeline being processed at the same time 是：流水线中确实有多个指令在同时处理
  - No: multiple instructions do not enter the pipeline at the same time 不是：多个指令不是同时进入流水线
- How to further improve the execution efficiency of instructions? 如何进一步提高指令的执行效率?
  - Optimize pipeline: super pipeline 优化流水线：超级流水线
  - True instruction level parallelism: superscalar pipelining 真正的指令级并行：超标量

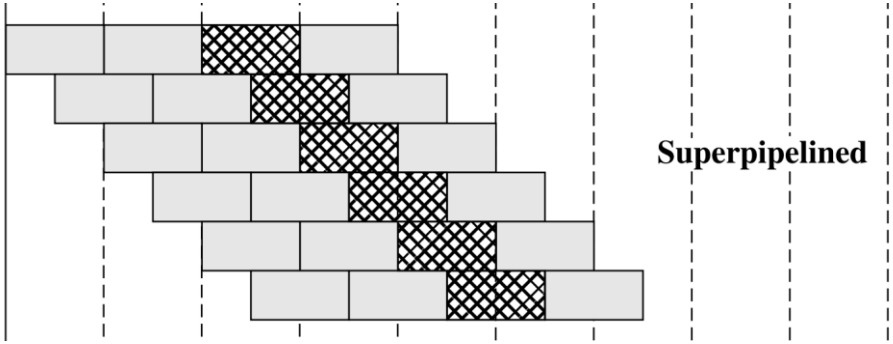
Superpipeline 超级流水线

- In an ordinary pipeline, each clock cycle can complete processing of one pipeline stage 普通流水线中，每个时钟周期可以完成1个流水阶段的处理
- Many pipeline stages need less than half a clock cycle 很多流水线阶段不需要1/2个时钟周期
- Superpipeline 超级流水线
  - Double internal clock rate is adopted for instruction scheduling 采用了双倍内部时钟速率来进行指令调度
  - Double internal clock speed gets two tasks per external clock cycle 双倍内部时钟速率可以在一个外部时钟周期完成两个流水阶段
- Get twice the instruction throughput 可以获得两倍的指令吞吐量

Superpipeline 超级流水线







- 四阶段流水线，指令划分为4个阶段
- 普通流水线中，每个阶段需要1个时钟周期来完成。最高能达到4倍的指令执行效率，每个时钟周期可以输出1个指令的执行结果
- 超级流水线。通过采用双倍内部时钟的方式，每0.5个外部时钟周期，就能完成1个指令阶段的执行。最高能达到8倍的执行效率

Limit of Superpipeline

- The effect is similar to that of increasing the main frequency 效果跟提高主频类似
- It is still not really instruction level parallelism 仍然还不是真正意义上的指令级并行
- The overall performance is limited by the clock cycle and the length of time the instruction phase executes 整体性能受限于时钟周期和指令阶段执行的时间长短
  - Long execution phases affect overall performance 执行时间长的阶段影响了整体的性能
- Another technology-superscalar 另一种技术-超标量

Vector and Scalar 向量和标量

- Scalar 标量
  - Also called “vector free”. Some physical quantities have only numerical value, but no direction. Some of them are positive or negative 也称“无向量”。有些物理量，只具有数值大小，而没有方向，部分有正负之分
  - A single number used to represent a single attribute of a thing 一个单独的数，用来表示一个事物某一个单独的属性
  - For example: temperature, length 例如：温度，长度
- Vector 向量
  - Originally refers to a quantity with size and direction 原指具有大小和方向的量
  - A group of orderly arranged numbers used to demarcate the quantitative characteristics of things 用来标定事物数量特征的一组有序排列的数
  - For example: the position of a point in the plane coordinate system (x, y) 例如：平面坐标系中的一个点的位置（x，y）

Scalar instruction 标量指令

- Instructions are divided into two categories according to the operating object：Vector instruction，Scalar instruction 指令根据操作对象分为两类：向量指令，标量指令
- Scalar instruction 标量指令
  - The instructions that do not have vector processing functions and only operate on a single quantity, namely a scalar quantity, are called scalar instructions 不具备向量处理功能，只对单个量即标量进行操作的指令称为标量指令
  - Most instructions are scalar 大部分指令为标量指令

Vector instruction 向量指令

- Vector instruction 向量指令
  - The basic operating object is a vector, that is, a group of numbers arranged in order 基本操作对象是向量，即有序排列的一组数
  - The instruction determines the address of the vector operand and directly or implicitly specifies vector parameters such as increment, vector length, etc 指令确定向量操作数的地址，并直接或隐含地指定如增量、向量长度等其他向量参数
  - The vector instruction specifies that the processor processes vector according to the same operation, which can effectively improve the operation speed 向量指令规定处理机按同一操作处理向量中的所有分量，可有效地提高运算速度
  - Some mainframes are equipped with vector operation instruction systems with complete functions 有些大型机设置功能齐全的向量运算指令系统

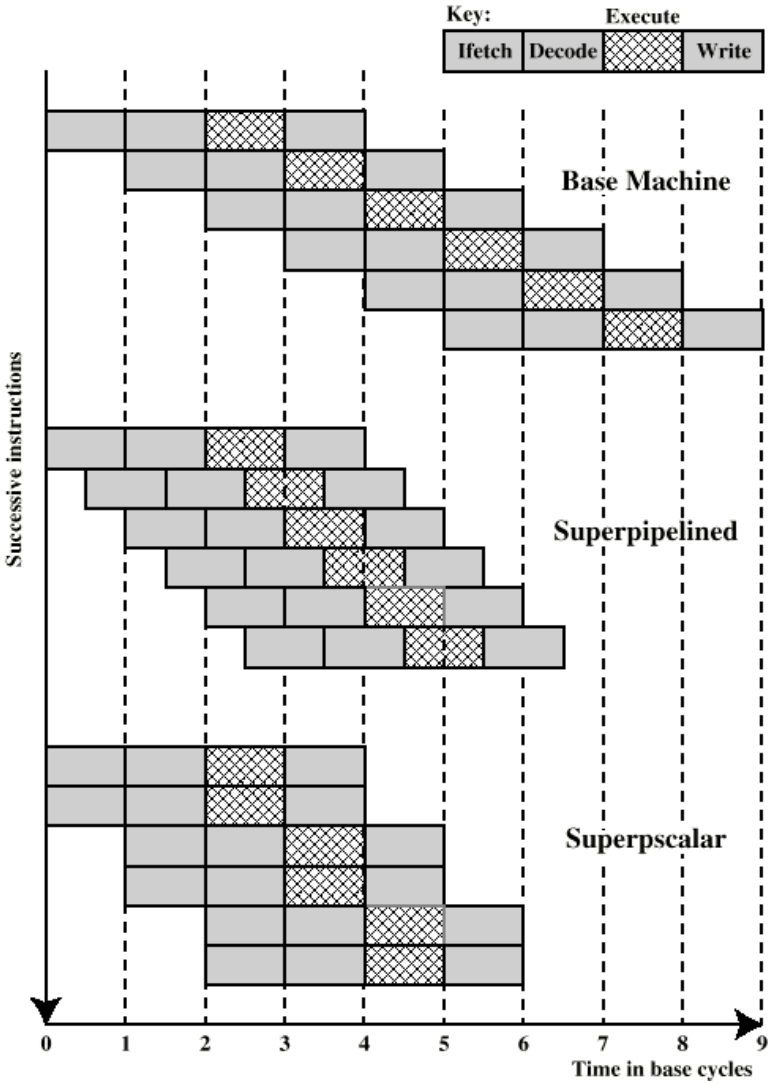
Why superscalar? 为什么用超标量?

- Most operations are on scalar quantities 大多数操作都是针对标量的操作
  - Common instructions includes arithmetic, load/store, conditional branch, etc 通用指令包括算术、加载/存储、条件分支等
  - These can be initiated and executed independently 这些指令可以独立启动和执行
  - Independent instructions do not affect each other and can be processed at the same time 不相关的指令之间相互不影响，可以完全同时处理
  - Improve these operations to get an overall improvement 改进这些操作以获得整体改进
- Completely parallel operation on irrelevant instructions is a method to improve instruction throughput 对不相关的指令进行完全并行操作，是提高指令吞吐量的一种方法

What is superscalar? 什么是超标量?

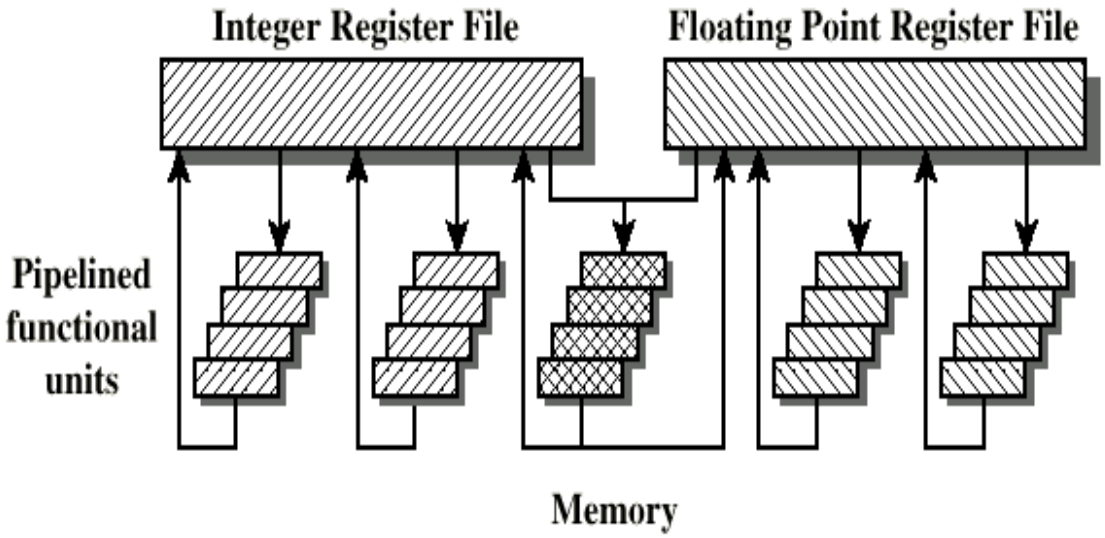
- Superscalar processor is a processor that uses multiple independent instruction pipelines 超标量处理器是一种使用多条相互独立的指令流水线的处理器
- Each pipeline runs according to the previous general pipeline mechanism, and each pipeline can process multiple instructions at the same time 每条流水线按照之前的一般流水线机制来运行，每条流水线能同时处理多个指令
- Superscalar processor fetches multiple instructions at one time, finds irrelevant instructions from them, and executes these instructions in parallel 超标量处理器一次取多个指令，从中找到彼此不相关的指令，并行执行这些指令
- Essentially, superscalar pipeline is the ability to execute instructions independently on different pipelines, a true instruction parallel processing technology 从本质上来说，超标量流水线是在不同的流水线上独立执行指令的能力，一种真正的指令并行处理技术

Superscalar 超标量流水线



- 超标量采用了2个独立的流水线
- 每个流水线都可以再进行指令的并行运行
- 能够并行执行每个阶段的2个指令
- 在稳定运行的状态下，可以达到8倍的执行效率

General superscalar organization 通用超标量组织



- 包含了2个整数运算单元，2个浮点数运算单元，一个存储单元
- 整数运算单元中，允许有2个指令并行执行，浮点数运算单元也允许2个浮点指令同时运行。与此同时，1个存储器操作也可以并行来执行
- 这个结构中，同时允许5个指令并行执行

Key problem of superscalar 超标量中的核心问题

- Superscalar implementations raise a number of complex design issues related to the instruction pipeline 超标量实现带来了许多与指令流水线相关的复杂设计问题
  - First, the relevance of the pipeline itself still exists 首先，流水线本身的相关性问题依然存在
  - Multiple pipelines bring more complex correlation problems 多流水线带来更复杂的相关性问题
- The compiler is required to have more complex optimization techniques to achieve greater instruction level parallelism 需要编译器具有更复杂的优化技术，以达到更大程度的指令级并行性

Application of superscalar 超标量的应用

- Superscalar technology itself is proposed and developed with the development of RISC technology 超标量技术本身随着RISC技术的发展而提出和发展
- RISC processors also tend to use superscalar technology RISC架构的处理器也倾向于使用超标量技术
- Although RISC machine lends itself readily to superscalar techniques, the superscalar approach can be used on either a RISC or CISC architecture. 尽管RISC机器易于采用超标量技术，但超标量方法可用于RISC或CISC体系结构
- Superscalar approach has now become the standard method for implementing high-performance microprocessors. 超标量方法现已成为实现高性能微处理器的标准方法

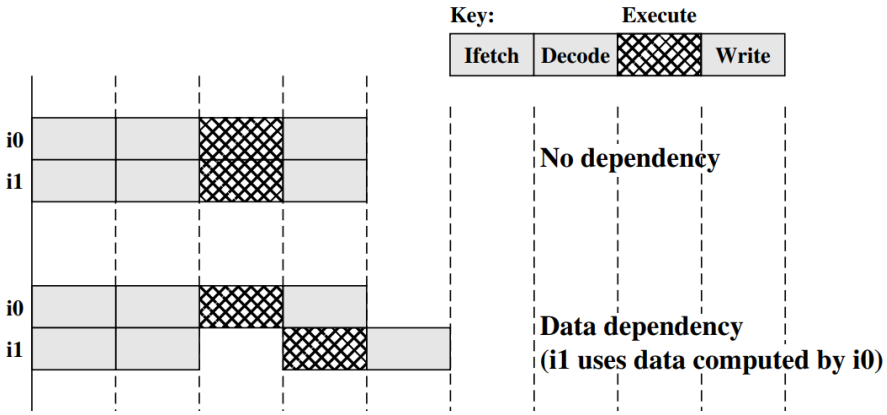
Factors limiting parallelism 限制并行的因素

- Instruction level parallelism 指令级并行性
  - The degree to which program instructions can be executed in parallel 程序指令能够并行执行的程度
- Compiler capabilities 编译器的能力
  - The compiler can maximize instruction level parallelism of programs 编译器能够最大程度地提高程序的指令级并行性
- Hardware techniques 硬件技术
  - Hardware capability supports parallel operation of instructions 支持指令的并行运行的硬件能力

Limitations 指令中的相关性

- The most important reason for limiting instruction level parallelism is the correlation between instructions in the program 限制指令级并行的最重要的原因就是程序中指令之间的相关性
- The dependencies between instructions include 指令之间的相关性包括:
  - True data dependency 真实数据相关性
  - Output dependency 输出相关性（后面讨论）
  - Anti-dependency 反相关性（后面讨论）
  - Procedural dependency 过程相关性
  - Resource conflicts 资源冲突

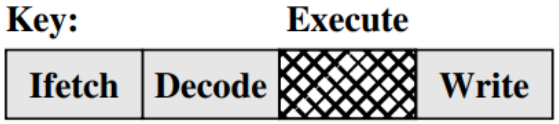
True Data Dependency 真实数据相关性

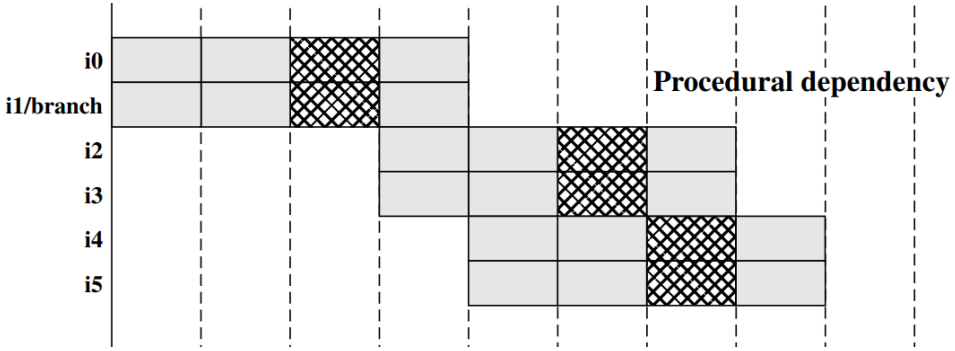


- ADD r1, r2 (r1 := r1+r2;)
- MOVE r3,r1 (r3 := r1;)

- I0和i1能够同时进行取指和解码
- 但是由于i1取的操作数是i0的结果，所以必须要等到i0执行完之后，i1才能进行取指
- 第二条指令存在一个时钟周期的延迟
- 先写后读，也称为“写后读相关性”
- 这种相关性和指令的执行顺序严格相关，是真实的相关性

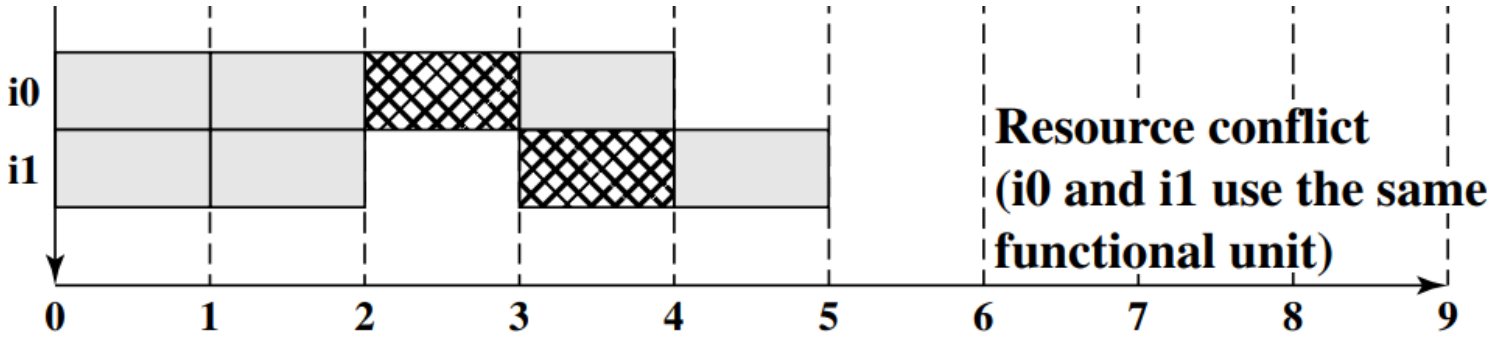
Procedural dependency 过程相关性





- 分支前和分支后的指令不能并行执行
- 如果指令非定长，则必须对指令进行解码，才能确定取多长的指令
- 如果使用的是变长指令，在取后续指令之前，前一个指令必须要部分译码，否则下一个指令不知道从内存的哪个位置去取，这阻止了同时取指的操作
- 超标量更适合RISC架构的理由之一，因为RISC的指令都是定长的，不会有这种相关性

Resource conflict 资源冲突



- 资源冲突，也称为资源相关性
- 指令i0和i1在执行过程中，都需要用到同一个功能单元，所以他们不能并行执行，只能串行处理。这里浪费了一个时钟周期
- 资源冲突和数据相关性的表现差不多，但是资源冲突可以通过复制资源来解决，例如在前面讲到的增加干衣机

Summary

- 超标量是真正意义上的指令级并行
- 超标量带来了许多与指令流水线相关的复杂设计问题
- 限制并行的关键因素
  - 指令并行性
  - 编译器能力
  - 硬件能力

Outline

- Overview of Superscalar 超标量综述
- Design Issues of Superscalar 超标量设计问题
- Superscalar in Pentium 4 Pentium4中的超标量
- Superscalar in ARM CORTEX-A8 ARM中的超标量

Parallelism 并行性

- Factors limiting parallelism 影响并行性的因素
  - Instruction level parallelism 指令级并行性
  - Compiler capabilities 编译器的能力
  - Hardware techniques 硬件技术
- Instruction level parallelism 指令级并行
  - Instructions have the characteristics of parallel execution 指令具有并行执行的特性
  - Instructions in a sequence are independent 指令序列独立
  - Execution can be overlapped 执行可以重叠
  - Governed by data and procedural dependency 受限于数据和过程相关性

Machine Parallelism 机器并行性

- Machine Parallelism 机器并行性
  - Ability to take advantage of instruction level parallelism 能够利用指令级并行性
  - Governed by number of parallel pipelines 受限于流水线的数量
  - The ability to find independent instructions and obtain instruction level parallelism 寻找独立指令，获得指令级并行性的能力

Example

- Instructions that can be executed in parallel 可以并行执行的指令
  - Load R1 <- R2(23)
  - Add R3 <- R3, “1”
  - Add R4 <- R4, R2
- Instructions that cannot be executed in parallel 不能并行执行的指令
  - Add R3 <- R3, “1”
  - Add R4 <- R3, R2
  - Store[R4] <- R0

Instruction issue 指令发射

- Instruction issue： the process of starting instructions to be executed by the functional unit of the processor 指令发射：启动指令去处理器的功能单元进行执行的过程
  - Instruction issue occurs when instruction moves from the decode stage of the pipeline to the first execute stage of the pipeline 指令发射发生在指令从流水线的解码阶段向第一个执行阶段移动的时候
- In order to improve the parallelism, it is necessary to use a reasonable issue order, instead of the original order 为了提高并行性，必须要使用合理的发射顺序来进行指令的发射，而不能按照原始的指令顺序去发射指令
- In essence, instruction emission is a strategy to find instructions that can enter the pipeline and be executed 本质上来说，指令发射就是寻找能够进入流水线并执行的指令的策略

Order about instruction issue 指令发射中的顺序

- Three sequences are involved in the command sending process 指令发射过程中涉及到三个顺序
  - Order in which instructions are fetched 获取指令的顺序
  - Order in which instructions are executed 指令执行的顺序
  - Order in which instructions change registers and memory 指令修改寄存器和内存的顺序
- The one constraint on the processor is that the result must be correct 处理器上的唯一约束是结果必须是正确的
- Instruction issue policy refers to the protocol used to start the execution of the command 指令发射策略指的是启动指令执行时所采用的协议

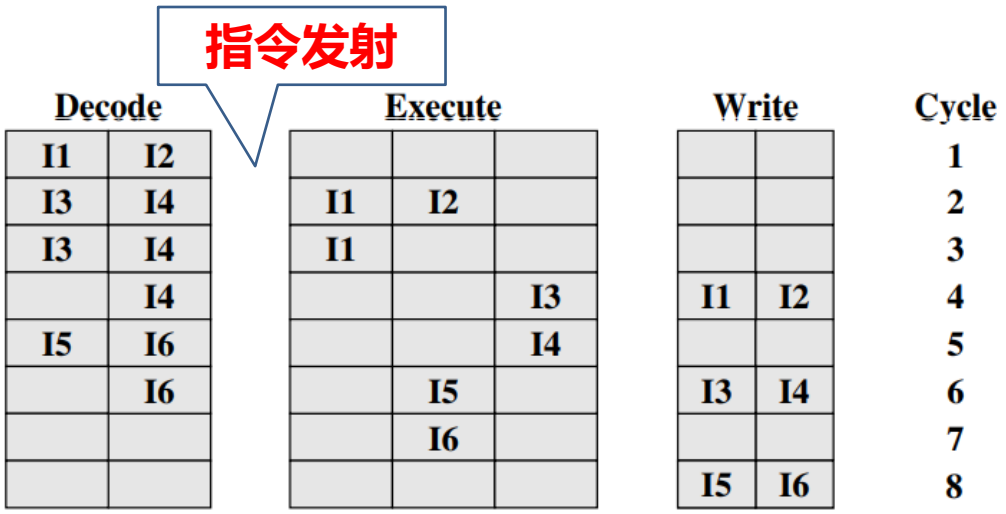
Instruction issue policy 指令发射策略

- The original instruction stream itself has dependencies 原始指令流本身存在相关性
- To improve the parallelism of execution, the processor may change the order in which instructions are executed 为了提高执行的并行性，处理器可能会更改指令的执行顺序
- The more sophisticated the processor, the less it is bound by a strict relationship between these orderings 处理器设计的越精巧，对这些顺序之间的限制就越少
- There are three issue policy 有三种发射策略
  - In-order issue with in-order completion 按序发射按序完成
  - In-order issue with out-of-order completion 按序发射乱序完成
  - Out-of order issue with out-of –order completion 乱序发射乱序完成

In-order issue/in-order completion 按序发射按序完成

- Issue instructions in the order they occur 按照指令发生的顺序发射指令
- Write the results in the same order to complete the execution of instructions 以同样的顺序写结果，完成指令执行
- Very inefficient，even scalar pipeline will not use policy 很低效，标量流水线都不会采用
- In Superscalar pipeline
  - May fetch more than one instruction 可能取多个指令
  - To ensure orderly completion, when the functional units conflict, or the execution of the functional units requires multiple cycles, instruction issue must wait 为了保证按序完成，当功能单元冲突，或者功能单元的执行需要多个周期时，指令发射必须要等待

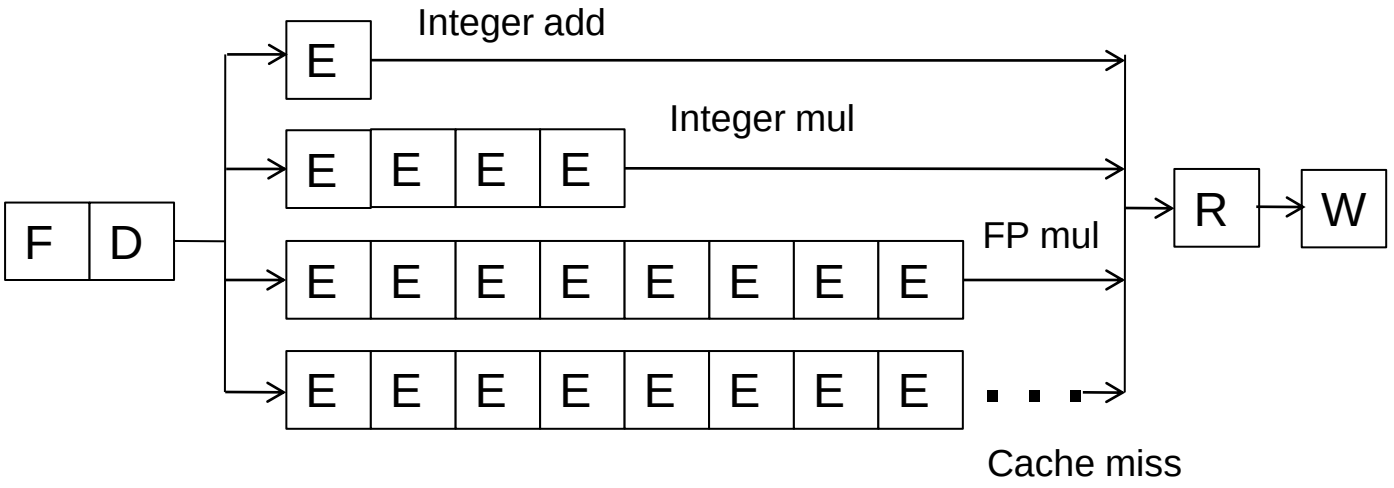
In-order issue/in-order completion 按序发射按序完成



- 超标量处理器有2个独立的流水线，能够同时取2个指令
- 有3个执行单元，以及2个写回的单元

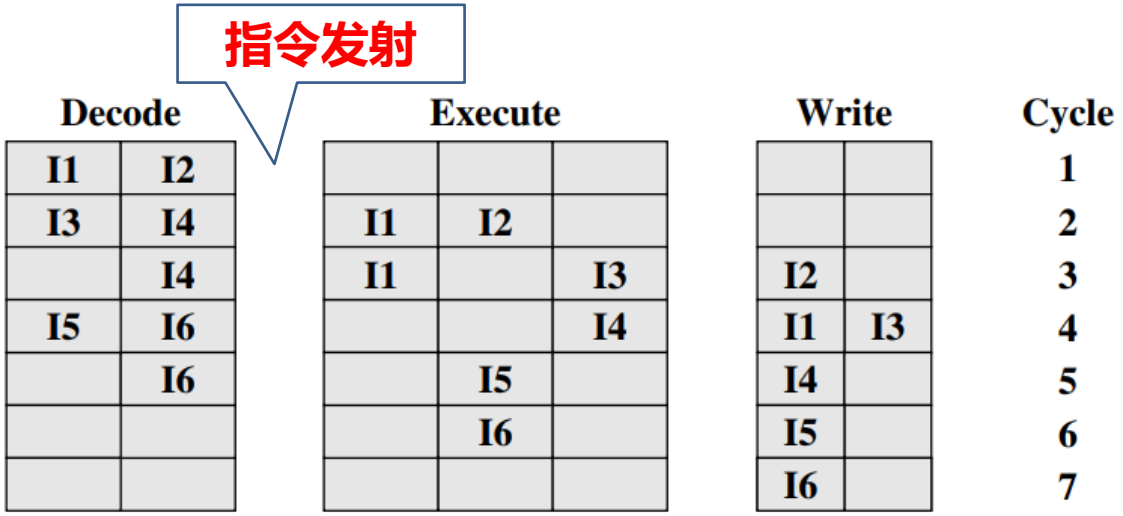
- I1需要2个周期完成执行；I3和I4需要同时使用某个功能单元，导致出现冲突；I5依赖于I4的结果；I5和I6需要同时使用某个功能单元，导致出现冲突。
- 成对取指并送到译码单元进行译码。I1需要花费2个时钟周期执行。所以I3和I4需要在第四个周期开始执行。由于I3和I4资源冲突，所以I3和I4需要顺序执行。I5需要依赖I4的结果，并且I5和I6存在资源冲突，所以I5和I6需要串行执行
- 8个指令，总共需要8个时钟周期才能完成

An in-order pipeline 按序的流水线



- 由于指令执行的时间不一样，所以如果同时发射的指令给执行单元，需要等全部执行完成之后，才能进行下一次发射
- 如果指令间存在数据依赖关系，需要停止调度行为，等具备条件之后，才能进行指令发射

In-order issue/out-of-order completion 按序发射乱序完成



- I1和I2同时发射到执行单元，由于I2只需要1个周期完成，所以I2可以先完成。I3可以和I1同时执行，并进入写的阶段
- I4由于和I3资源冲突，所以需要等I3完成之后才能执行。I5依赖于I4的结果，所以也需要等待I4，同理I6需要等待I5
- 整体上需要7个周期完成指令的执行

Output (write-write) dependency 输出相关性

- In the process of out of sequence completion, the execution order is different from the original order, which may lead to output dependency problems 乱序完成中，由于执行顺序和原始顺序不一样，可能会导致输出相关性问题
- Example
  - R3:= R3 + R5; (I1)
  - R4:= R3 + 1; (I2)
  - R3:= R5 + 1; (I3)
- Analyze
  - I2 depends on result of I1 - data dependency I2依赖于I1的结果-数据相关性
  - If I3 completes before I1, the result from I1 will be wrong - output (write-write) dependency 如果I3先于I1执行完，那么R3就是从I1得到的结果，这个是错的

How?

- Adopt dynamic scheduling strategy 采用动态调度策略
- Idea: Move the dependent instructions out of the way of independent ones (s.t. independent ones can execute) 将有依赖关系的指令和独立指令分离，独立指令可以执行
  - Rest areas for dependent instructions: Reservation stations 对有依赖关系的指令放在一个休息区，称为保留站
- Monitor the source "values" of each instruction in the resting area 对每个在休息区的指令监控它的关联值

How?

- When all source “values” of an instruction are available, “fire” (i.e. dispatch) the instruction 如果指令的所有关联值都可用，那么就发射这个指令
  - Instructions dispatched in dataflow order，not control-flow 指令发射依赖于数据流，而不是控制流
- Benefit 好处
  - Latency tolerance: Allows independent instructions to execute and complete in the presence of a long latency operation 允许在长延迟操作的情况下执行和完成独立指令
  - Reasonably schedule instructions with dependencies 对具有依赖关系的指令进行合理调度

Problem about In-order issue 按序发射的问题

- When decoding instructions, if there are related points or conflicting points, the decoding needs to stop 在对指令进行译码的时候，如果遇到相关点或冲突点，译码就需要停止
- In this way, subsequent instructions cannot be decoded 这样后续的指令也不能译码
- At this time, the processor cannot check whether any instruction is independent and can be executed on the pipeline 处理器此时也不能向前查看是否有指令是独立指令，可以在流水线上执行
- The third command issue policy is introduced: disordered transmission disordered completion 引入了第三种指令发射策略：乱序发射乱序完成

Out-of-order issue/out-of-order completion 乱序发射乱序完成

- Solution：decouple decode from execution 将解码和执行解耦
- Decode
  - Decode stage can continuously fetch and decode 解码阶段可以持续地进行取指和解码
  - Decoded instruction is put into the buffer 解码的指令放到缓冲区中
  - As long as the buffer is not full, fetching and decoding can continue 只要缓冲区没有充满，就可以持续进行取指和解码
- Execution
  - When the functional unit is available, transmit the executable instructions to execute 当功能单元可用时，发射可以执行的指令去执行
  - Since the instruction has been decoded, the processor can first identify whether the instruction can be executed 由于指令已被解码，处理器可以先行识别指令是否可以执行

Out-of-order issue/out-of-order completion 乱序发射乱序完成

Decode		Window	Execute			Write		Cycle
I1	I2							1
I3	I4	*I1, I2*	I1	I2				2
I5	I6	*I3, I4*	I1		I3	I2		3
		I4, I5, I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

- 第一个周期，I1和I2进行解码，完成解码后进入发射缓冲区。执行单元为空，I1和I2被发射出去执行
- 第二个周期，I3和I4进行解码，完成解码后进入发射缓冲区。由于I3和I4共用执行单元，I3发射出去进行执行
- 第三个周期，I5和I6进行解码，完成解码后进入发射缓冲区。此时，I3执行完了，所以I4可以发射了。同时由于I5和I4有数据相关性，所以I5不能发射，于是把I6发射出去执行
- 第四个周期，没有指令需要解码
- 第五个周期，没有指令需要解码。此时发射缓冲区只有I5。I6执行完成，可以发射I5指令。I5在第五个时钟周期完成执行，并在第六个时钟周期完成写入操作
- 整个过程需要6个时钟周期。比之前的又缩短了1个周期

Anti-dependency（Write-after-read） 反相关性

- Out-of-order issue/out-of-order completion also need to comply with restrictions 乱序发射乱序完成也需要遵守限制条件
- Anti correlation occurs 出现了反相关性的情况
- Example
  - R3:=R3 + R5; (I1)
  - R4:=R3 + 1; (I2)
  - R3:=R5 + 1; (I3)
  - R7:=R3 + R4; (I4)

I2要读R3，I3要写R3。读之后写

- Analyze
  - I3 can not complete before I2 starts as I2 needs a value in R3 and I3 changes R3 I3在I2开始读之前不能完成，因为I2需要R3的值，而R3会被I3修改

Dependency Analyzing 相关性分析

- True data dependency reflects the real dependency between data 真实数据相关性反映的是数据之间的真实依赖关系
- In essence, anti dependency and output dependency are caused by register conflict 反相关性和输出相关性从本质上来说，是寄存器冲突引起的
  - Register contents may not reflect the correct ordering from the program 寄存器内容可能无法反映程序的正确顺序
- Instruction issue stops, pipeline stall 指令发射停止，流水线延迟
  - Processor pauses for one cycle 处理器停顿一个周期
- This situation is more serious when register optimization technology is used 采用寄存器优化技术时，这种情况会更加严重
  - Register optimization technology maximizes the use of registers to improve performance 寄存器优化技术会最大限度利用寄存器以提高性能
  - Register conflicts will be more significant 寄存器的冲突会更加显著

Register renaming 寄存器重命名

- Register renaming 寄存器重命名
  - Registers are dynamically allocated by hardware 寄存器由硬件进行动态分配
  - When an instruction with a register as the destination operand is executed, a new register is allocated 以寄存器为目标操作数的指令执行的时候，会分配一个新的寄存器
  - The instruction that accesses the original register after this instruction must be modified to the newly allocated register to maintain consistency 这个指令之后的访问原来寄存器的指令必须修改成新分配的寄存器，以保持一致
- Avoid dependencies caused by register conflicts 避免因为寄存器的冲突而导致的相关性

Example

- **Original**
  - R3:=R3 + R5; (I1)
  - R4:=R3 + 1; (I2)
  - R3:=R5 + 1; (I3)
  - R7:=R3 + R4; (I4)

- I1和I2存在真实数据相关性
- I3和I4存在真实数据相关性
- I3和I2存在反相关性，读后写
- I3和I1存在输出相关性，写后写

- **Register renaming**
  - R3b:=R3 a+ R5a; (I1)
  - R4b:=R3b + 1; (I2)
  - R3c:=R5a + 1; (I3)
  - R7:=R3c + R4b; (I4)

- 采用寄存器重命名的规则，I1的R3修改成R3b，I3中的R3，修改成R3c。
- I3和I2之间的反相关性没有了，I3和I1之间的输出相关性也没有了，I3可以立即发射。
- 真实数据相关性无法通过寄存器重命名来解决

Analysis of three technologies 三种技术分析

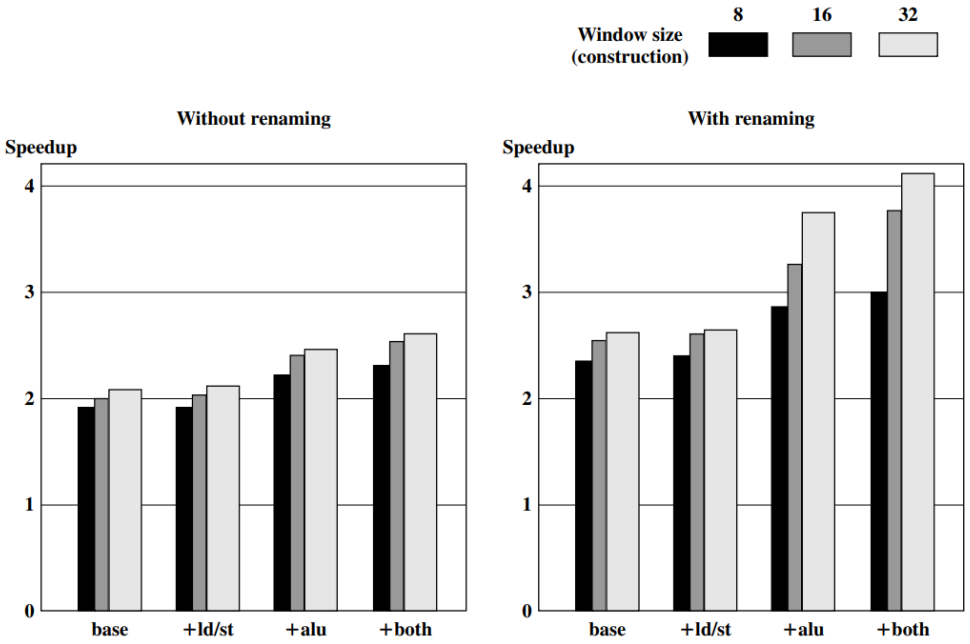
- Techniques for improving performance in superscalar processors 超标量处理器中提高性能的技术
  - Duplication of Resources 资源复制
  - Out of order issue 乱序发射
  - Renaming 重命名
- Resources are the foundation 资源是基础
  - Sufficient resources to execute multiple pipelines 有足够的资源来执行多个流水线
- Out of order issue is the method 乱序发射是方法
  - Provide executable instructions through disordered transmissionRenaming is a guarantee 通过乱序发射来提供可以执行的指令
- Renaming is a guarantee 重命名是保障
  - Rename mechanism reduces the correlation between instructions 重命名机制减少了指令间的相关性

About instruction window 关于指令窗口

- Out of order issue：register window is used to cache instructions after decoding 乱序发射中，使用寄存器窗口来缓存译码之后的指令
- Through the register window, the processor can identify independent instructions that can be placed in the execution segment 通过寄存器窗口，处理器可以识别能放到执行段的独立指令
- If the instruction window is very small, the probability of successful recognition is very low 如果指令窗口很小，那么识别成功的概率会非常低
- The instruction window needs to be large enough to find independent instructions and use the hardware more effectively 指令窗口需要足够大，能够找到独立的指令，更有效地利用硬件
- Need instruction window large enough (more than 8) 需要指令窗口足够大，超过8个

Effect of technology 技术的效果

Without Procedural Dependencies



- Base：不复制任何功能单元
- +Id/st：增加了装入/存储单元
- +alu：增加了ALU单元
- +both：增加了ALU和Id/st

- 不考虑过程相关性
- 没有采用寄存器重命名，增加硬件执行效果并不明显。而采用寄存器重命名后，增加了ALU会明显提高加速比
- 从发射窗口的角度来看，窗口数量从8个增加到16个，效果就很明显。但从16个到32个，效果稍差一些
- 资源复制、乱序发射、寄存器重命名三者相互影响

Consideration of control dependence 控制相关性的考虑

- Also called branch hazard 控制相关性也称为分支冒险
- When branching instructions, it is not possible to determine which instruction to execute after the branch 在分支指令时，无法确定分支之后执行哪一个指令
- In the pipeline, after prefetching the wrong instruction, it is necessary to discard and re fetch the instruction, which causes the pipeline to fail to run with full load 在流水线中，预取错误的指令后，需要丢弃并重新取指，导致流水线无法满载运行

Methods 方法

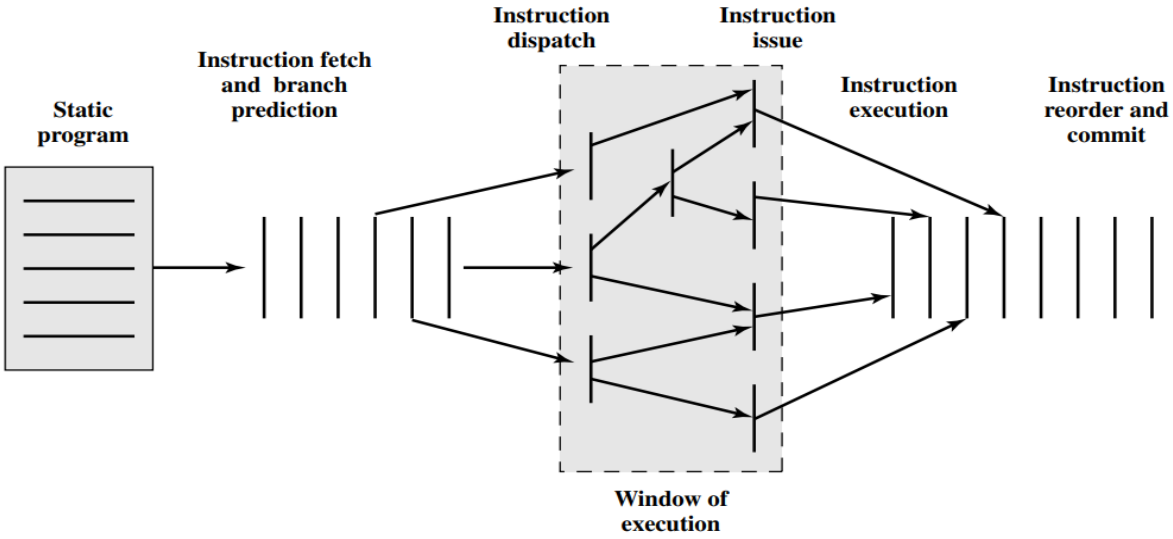
- Processing method of control dependence 控制相关性的处理方法
  - Multiple Streams 多指令流
  - Prefetch Branch Target 预取分支目标
  - Loop buffer 环形缓存
  - Branch prediction 分支预测
  - Delayed branching 延迟分支
- Goal: Keep the pipeline running full 目标：保持流水线充满运行

About delayed branch 关于延迟分支

- Delayed branching is often used in RIS RISC中常采用延迟分支的方法
- Calculate result of branch before unusable instructions pre-fetched

  在不需要的指令预取之前计算分支的影响
  - Instructions that are not affected by branches are immediately followed by branch 在分支指令之后紧跟不受分支影响的指令
  - Keeps pipeline full while fetching new instruction stream 获取新指令时保持流水线充满状态
- Not as good for superscalar 对超标量没有效果
  - Multiple instructions need to execute in delay slot 多个延迟槽
  - Instruction dependence problems 指令相关性问题
  - Often use branch prediction 经常使用分支预测

Superscalar execution 超标量的执行



- 静态程序通过取指和分支预测，形成动态的指令流
- 指令流经过处理器的相关性检查，会去掉不必要的相关性，比如反相关和输出相关。然后将指令放到执行窗口中，等待执行
- 在执行窗口中的指令，根据真实数据相关性来排序。处理器根据真实数据相关性和资源可用性，来发射指令到执行单元进行执行
- 最后的执行结果需要有一个提交的步骤。因为指令不是按照原有的顺序来执行的，同时分支预测和推测执行使得有些执行的结果需要丢弃。

Superscalar implementation 超标量实现1

- Simultaneously fetch multiple instructions 同时获取多条指令
  - Multiple fetching and decoding 多个取指和译码
  - Branch prediction logic 分支预测逻辑
- Logic to determine true dependencies involving register values 确定涉及寄存器值真相关的逻辑
  - Determine instruction position of true correlation 确定真相关的指令位置
- Dealing with unnecessary dependencies 处理不必要的相关性
  - Anti-dependency and output dependency 反相关和输出相关

Superscalar implementation 超标量实现2

- Mechanisms to initiate multiple instructions in parallel 并行启动多条指令的机制
  - Instruction window 指令窗口
  - Out of order issue logic 乱序发射逻辑
- Resources for parallel execution of multiple instructions 用于并行执行多条指令的资源
  - The system has sufficient resources 系统具有足够的资源
- Mechanisms for committing process state in correct order 按正确顺序提交处理状态的机制
  - Submit results according to the order of instructions 根据指令顺序提交结果

Summary

- Resources are the foundation 资源是基础
  - Machine parallelism 机器并行性
- Out of order issue is the method 乱序发射是方法
  - Instruction level parallelism 指令级并行性
- Renaming is a guarantee 重命名是保障
  - Methods of improving instruction level parallelism 提高指令级并行性的方法
- Through superscalar pipeline, multiple pipelines can run at the same time to achieve truly parallel operation at the instruction level 通过超标量流水线，实现多个流水线的同时运行，达到指令级的真正并行运行

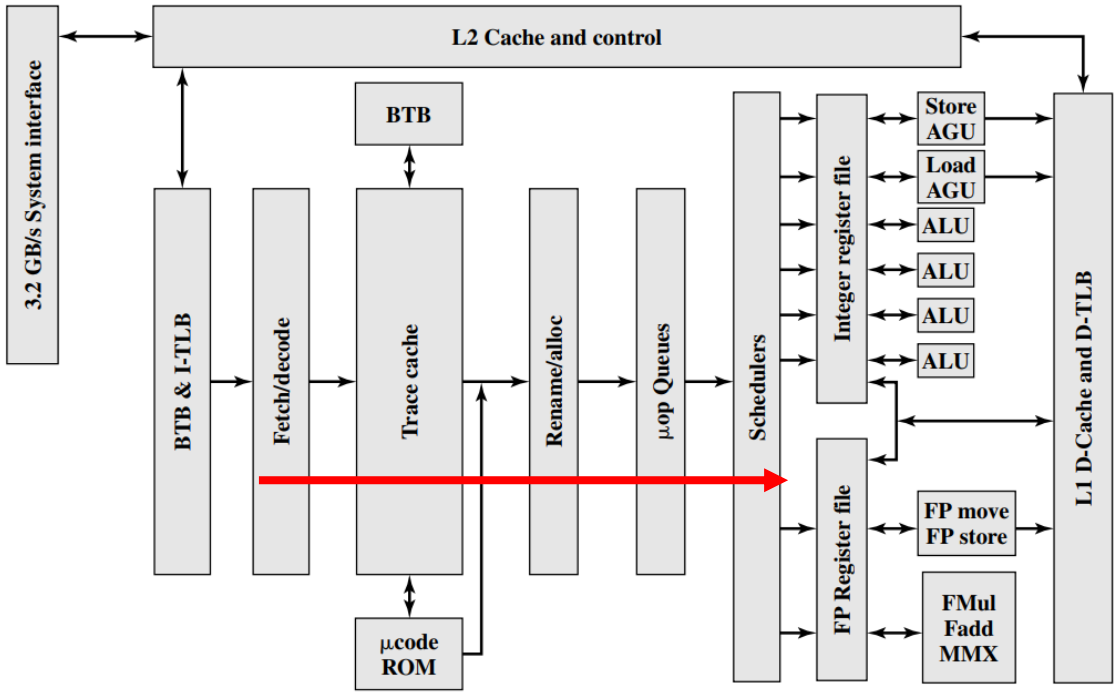
Outline

- Overview of Superscalar 超标量综述
- Design Issues of Superscalar 超标量设计问题
- Superscalar in Pentium 4 Pentium4中的超标量
- Superscalar in ARM CORTEX-A8 ARM中的超标量

History of x86 superscalar x86的超标量历史

- 80386- no pipeline 没有流水线
- 80486 – CISC with pipeline 具有流水线的CISC
- Pentium – The sprout of superscalar thought 奔腾处理器-超标量思想萌芽
  - Two separate integer execution units 2个独立的整数执行单元
- Pentium Pro – Full blown superscalar 奔腾pro全面超标量设计
- Subsequent models refine & enhance superscalar design 后续的型号具有更加精巧的、功能更加强大的超标量设计

Pentium 4 block diagram P4模块图



- 取指和译码后，生成微操作
- 进入踪迹cache，通过驱动将指令由踪迹cache给寄存器重命名/分配模块
- 微操作进入了排队阶段，由调度器取出微操作并派发执行
- 包含若干个整数执行单元和浮点执行单元
- 执行完成之后，需要进行分支检查，确定预测结果是否正确

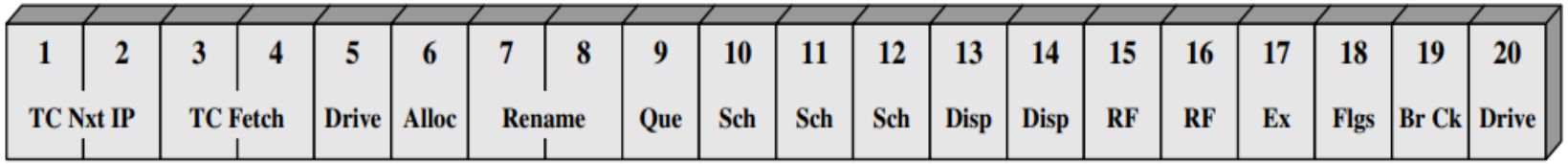
Pentium 4 operation P4操作

- Fetch instructions from memory in order of static program 按静态程序的顺序从内存中提取指令
- Translate instruction into one or more fixed length RISC instructions (micro-operations) 将指令转换为一个或多个固定长度RISC指令（微操作）
- Execute micro-ops on superscalar pipeline 在超标量流水线上执行微操作
  - micro-ops may be executed out of order 微操作可能会乱序执行
- Commit results of micro-ops to register set in original program flow order 按原始程序流顺序将微操作结果提交到寄存器集

Pentium 4 operation P4操作

- Outer CISC shell with inner RISC core CISC外壳，RISC内核
- Inner RISC core pipeline at least 20 stages 里面的RISC内核流水线包括至少20个阶段
  - Some micro-ops require multiple execution stages 一些微操作需要多个执行阶段
  - Longer pipeline 更长的流水线
  - c.f. five stage pipeline on x86 up to Pentium Pentium只有5段流水线
- Essentially, Pentium is a RISC processor in the guise of CISC 从本质上来说，Pentium是披着CISC外衣的RISC处理器

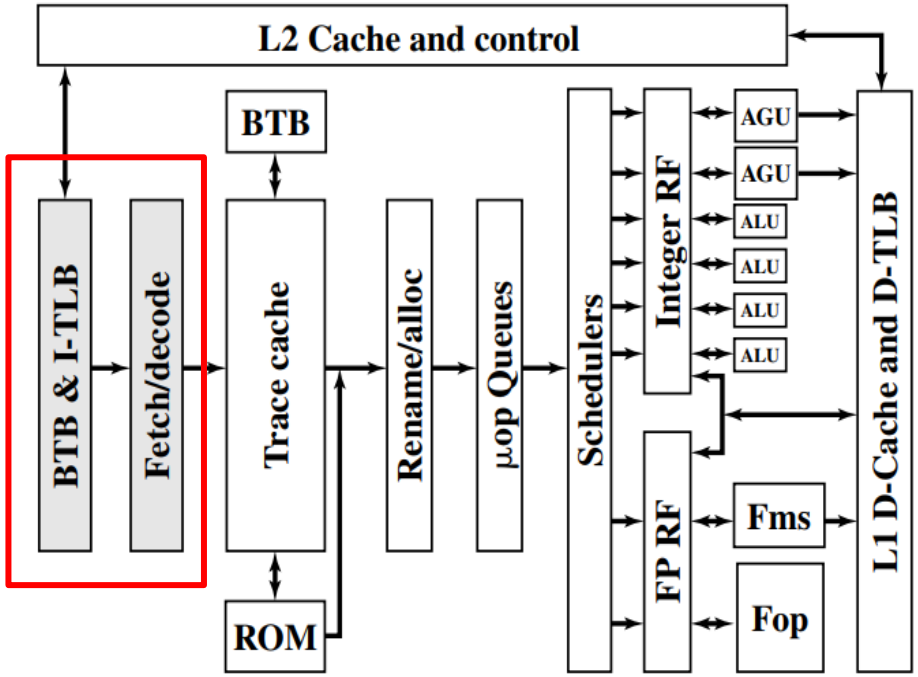
Pentium 4 pipeline Pentium4的流水线



- TC Next IP TC下一指令指针
- TC Fetch TC取指
- Drive 驱动
- Alloc 分配
- Rename 寄存器重命名
- Que 微操作队列
- Sch 微操作调度
- Disp 派发
- RF 寄存器组
- Ex 执行
- Flgs 标志
- Br Ck 分支检查

- 流水线从trace cache下一指令指针开始，负责在踪迹cache中选择指令。
- 进入到TC取指阶段。之后是驱动阶段，Alloc阶段负责进行资源分配
- Rename阶段负责进行寄存器的重命名
- 进入队列阶段。通过调度和分发阶段，将指令分发给相应的寄存器组，进行执行
- 运行过程中，需要设置标志位
- 执行完成之后，需要进行分支检查，并通过驱动来实现分支检查结果

Pentium 4 pipeline operation -1 P4流水线操作1



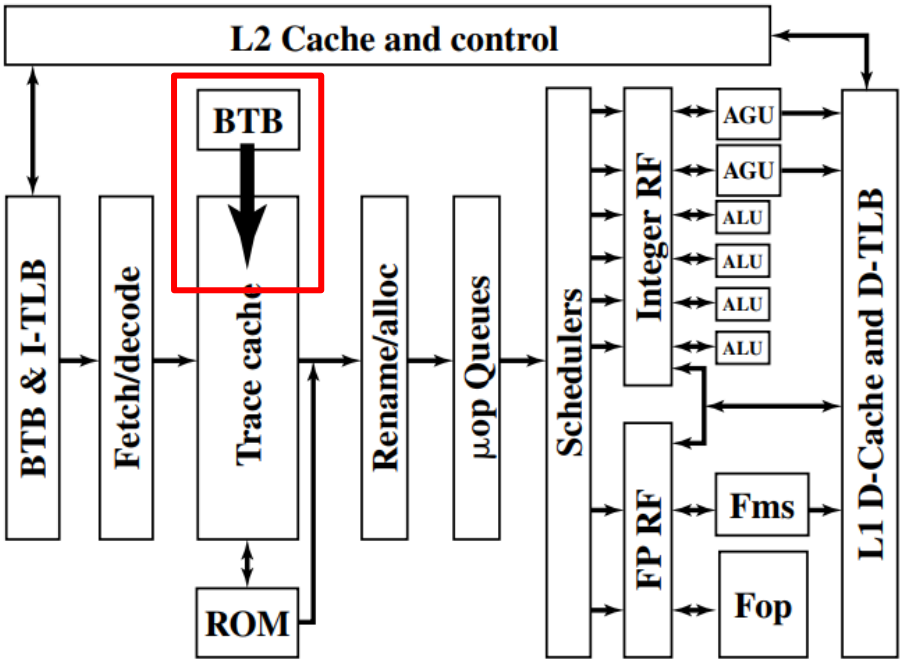
(a) Generation of micro-ops

BTB，branch target buffer，转移目标缓冲器

I-TLB，instruction translation lookaside buffer，指令地址转换后援缓冲器

- 通过BTB和I-TLB，取指译码单元从L2 cache中读取机器指令，并确定指令边界
- 将机器指令翻译成1~4个微操作，每个微操作是118位的RISC指令
- 生成的微操作保存在踪迹cache中
- 取指默认是顺序的，但BTB可以根据分支预测逻辑修改取指的顺序
- P4 的流水线是微操作开始的，因此微操作生成严格来说，不能算是流水线的阶段，可以成它为“按序操作的前端”

Pentium 4 pipeline operation -2 P4流水线操作2



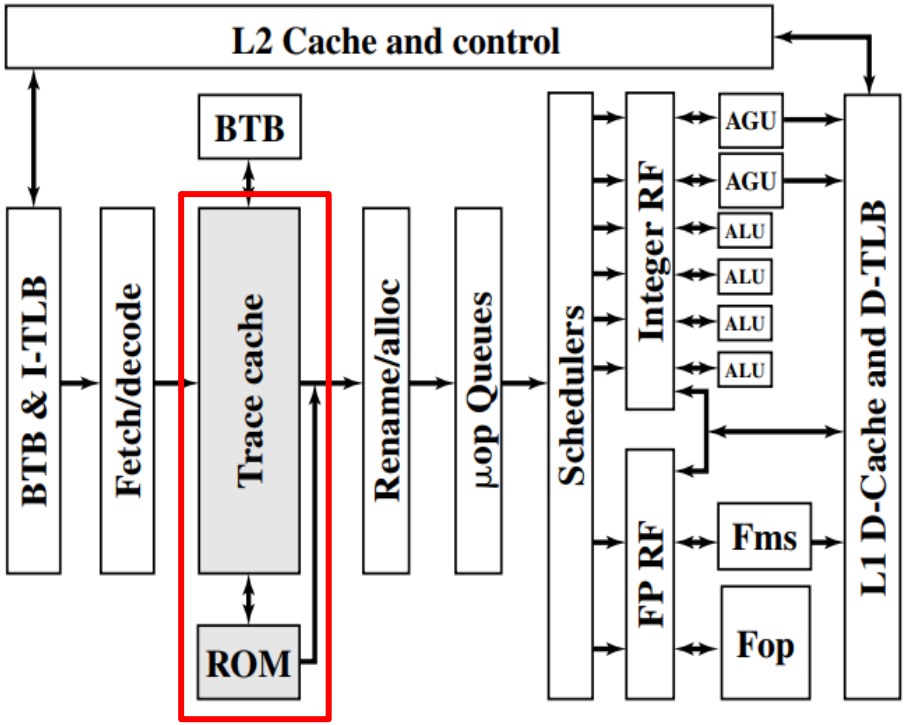
(b) Trace cache next instruction pointer

BTB，branch target buffer，转移目标缓冲器

踪迹cache下一指令指针，trace cache next instruction pointer

- 第一、二阶段完成在踪迹cache中选择指令。BTB保存了最近遇到的分支指令执行情况，用来预测分支指令
- 转移指令如果执行，需要同步维护BTB，以保证BTB的有效性。用4位历史位来记录分支指令的历史执行情况，可以降低预测的失误率
- BTB使用4路组关联cache的结构，共有512行
- 如果在BTB中没有转移指令记录，采用静态预测法，根据不同的情况来预测转移是否发生

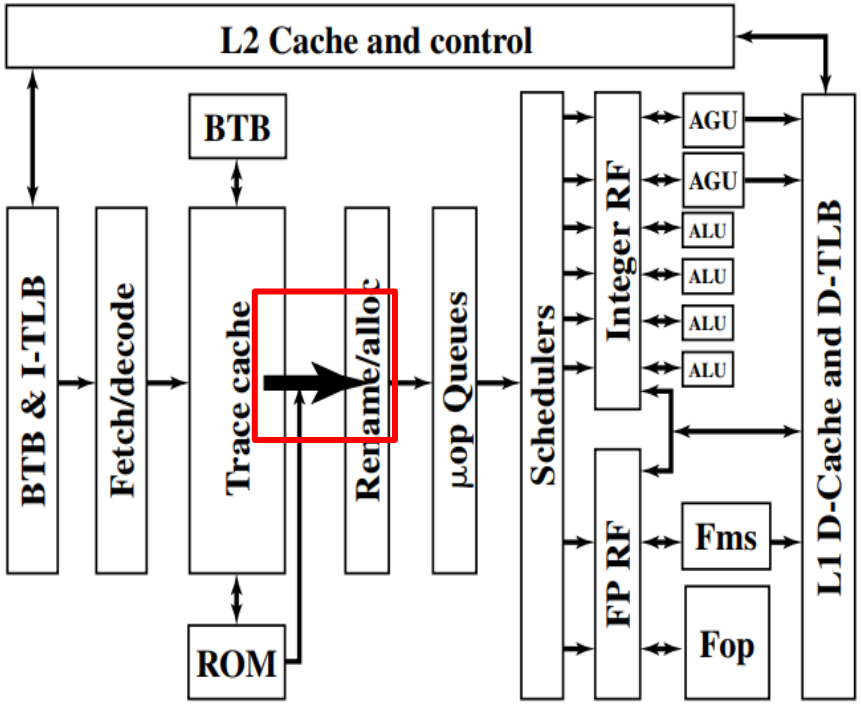
Pentium 4 pipeline operation -3 P4流水线操作3



(c) Trace cache fetch

- 下一个流水阶段是TC Fetch，，根据上一个阶段的原则，进行微操作的取指
- 存储器指令通过前端操作进行取指和译码，转换成1~4个微操作，保存在踪迹cache中。但是有的指令很复杂，要多于4个微操作才能完成，这些指令就需要送到微代码ROM中，由微代码ROM来将这个复杂指令转化成微操作

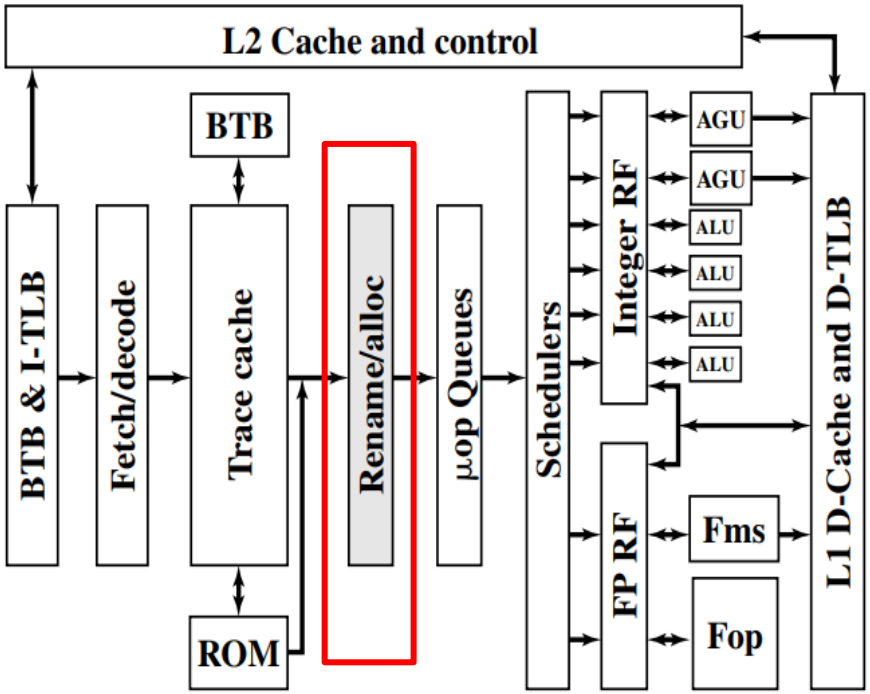
Pentium 4 pipeline operation -4 P4流水线操作4



(d) Drive

- TC取指完成之后，进入到流水线的下一个阶段
- 这个阶段是驱动阶段，负责将指令由踪迹cache提交给重命名/分配模块

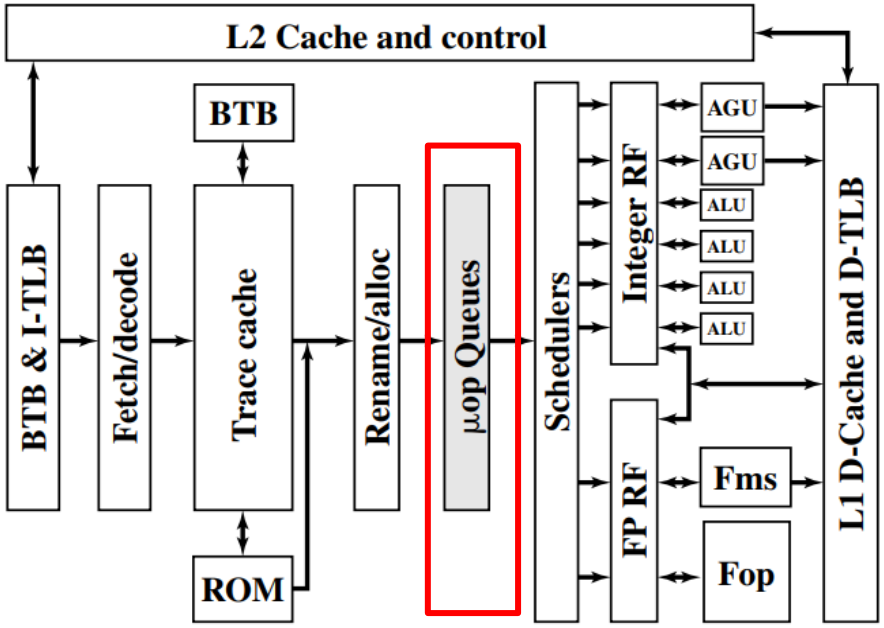
Pentium 4 pipeline operation -5 P4流水线操作5



(e) Allocate; register renaming

- 进入乱序执行逻辑，对微操作进行重新排序，只要指令的输入操作数就绪，就可以快速地进行发射和执行
- 包括两个阶段：分配和寄存器重命名。
- 分配是为微操作分配资源，包括寄存器资源，重排序缓冲器等。微操作进入重排序缓冲器，也就是ROB，reorder buffer后，由它去进行排队，调度、分发和执行，这些都是乱序的。最后，ROB的微操作登记项要按序回收
- 分配资源后，需要对寄存器进行重命名，解除数据的虚假相关性

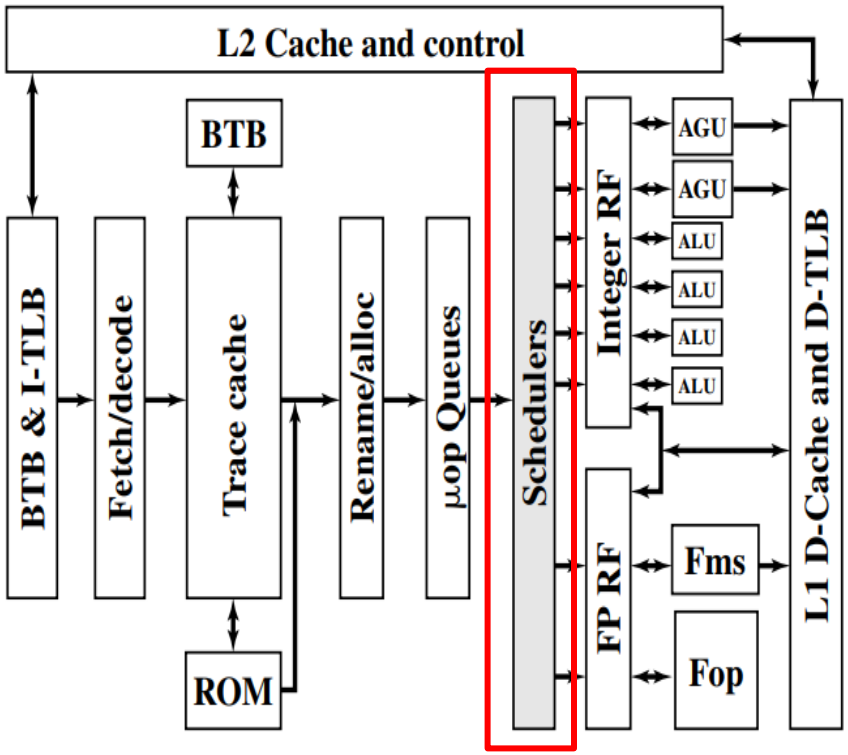
Pentium 4 pipeline operation -6 P4流水线操作6



(f) Micro-op queuing

- 分配资源后和寄存器重命名后，微操作进入流水线的下一个阶段，微操作排队
- 微操作排队中包括2个队列：一个是用于存储器操作的队列，包括装载和保存；另一个是不涉及存储器访问的微操作
- 微操作根据操作的性质，放入这2个队列中。队列采用FIFO的原则进行进出，但队列间没有次序关系

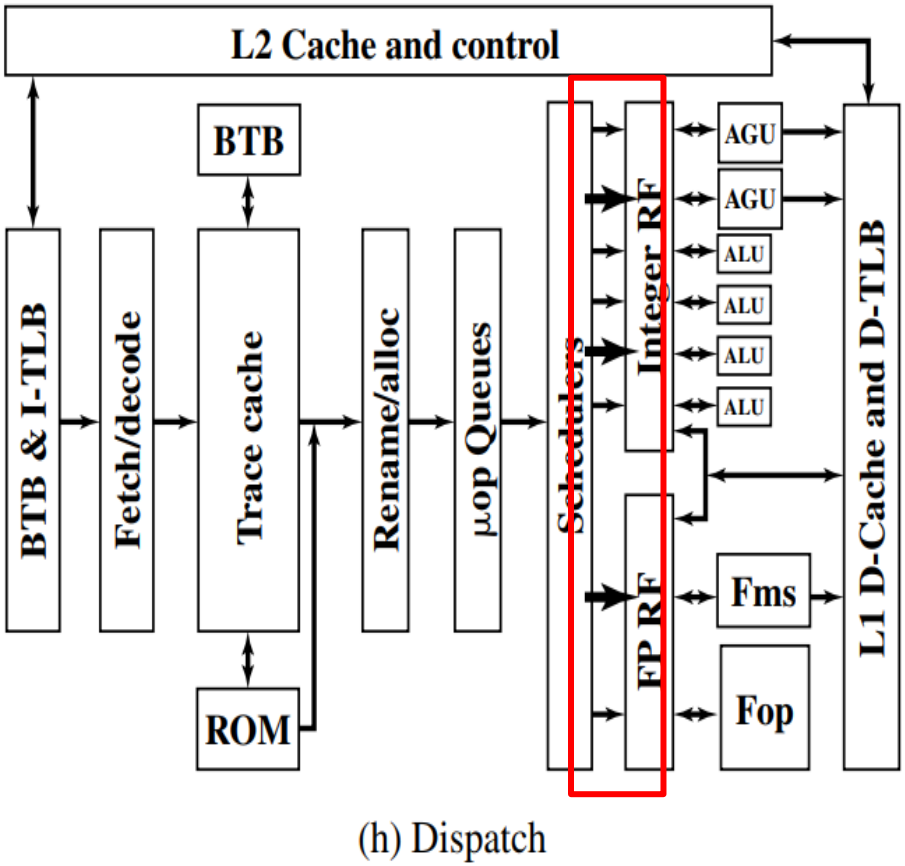
Pentium 4 pipeline operation -7 P4流水线操作7



(g) Micro-op scheduling

- 调度器负责从微操作排队的队列中取出微操作并派发他们去执行
- 调度器查找状态指示已具备全部操作数的微操作，并且如果它需要的执行单元处于可用状态，这时调度器就可以取出这个微操作，将它派发的相应的执行单元进行执行
- 每个周期最多可以派发6个微操作
- 如果存在资源冲突，则按照FIFO的顺序进行派发

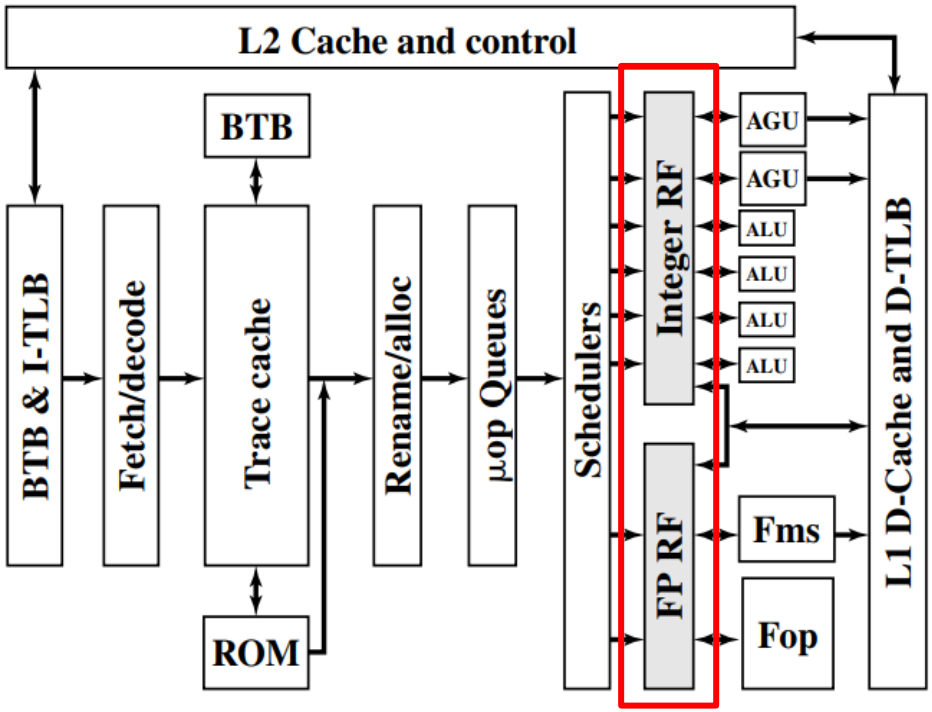
Pentium 4 pipeline operation -8 P4流水线操作8



(h) Dispatch

- 派发就是将调度器取出的微操作
- 派发给相应的执行单元去执行

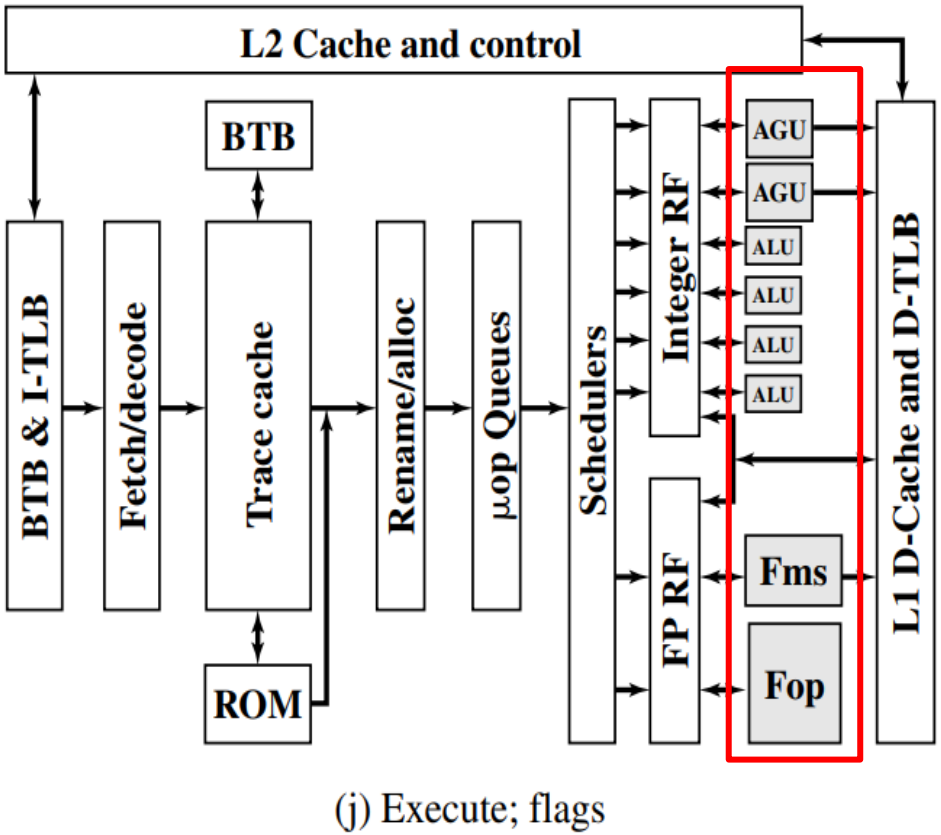
Pentium 4 pipeline operation -9 P4流水线操作9



(i) Register file

- 整数和浮点数寄存器组是需要完成的指令的数据源之一，还可以包括L1数据cache中取出的数
- 这些操作数送到执行单元，进行指令的执行

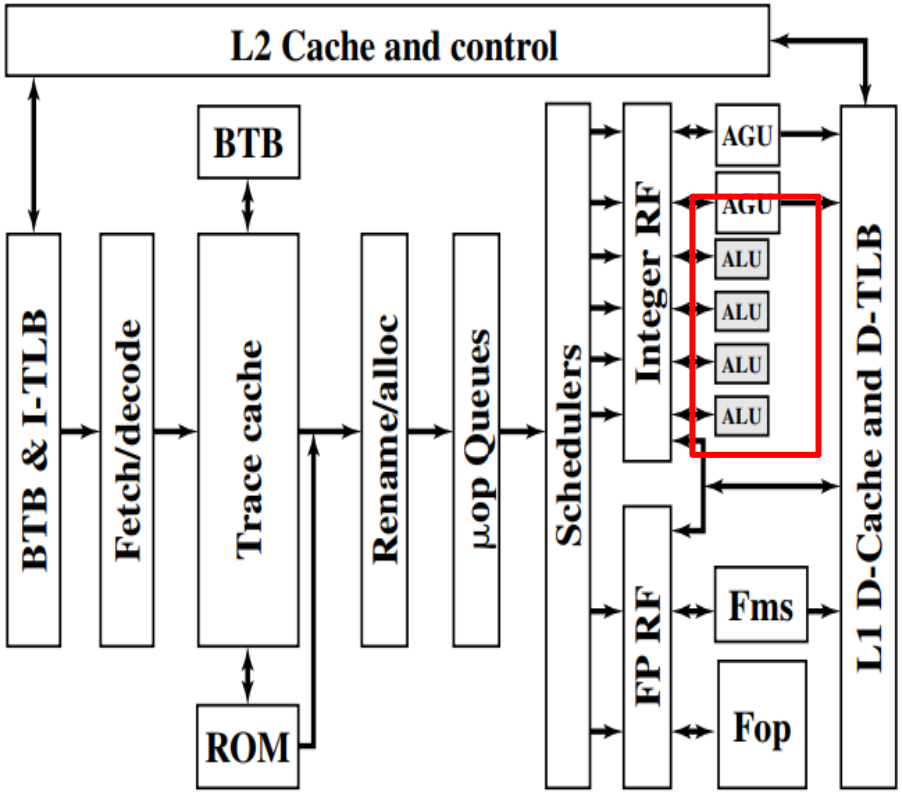
Pentium 4 pipeline operation -10 P4流水线操作10



(j) Execute; flags

- 流水线执行阶段
- Pentium还单独设置了一个用于计算标志的流水阶段

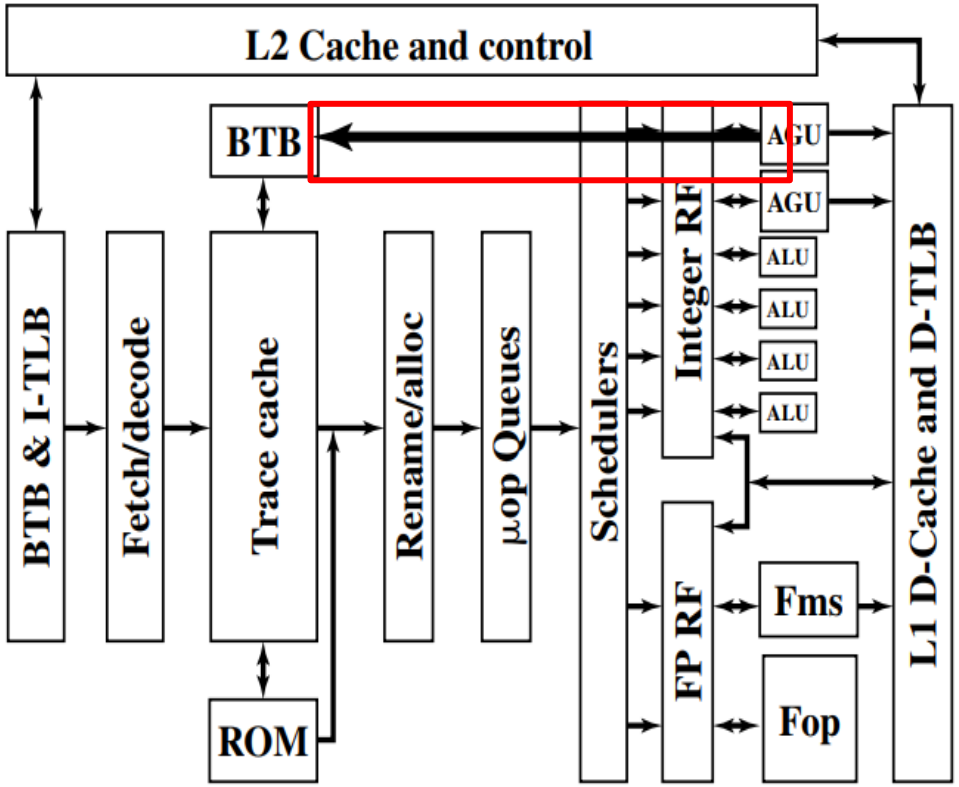
Pentium 4 pipeline operation -11 P4流水线操作11



(k) Branch check

- 执行完成之后，需要进行分支检测
- 将分支的实际结果和预测进行比较
- 确定是否需要将部分微操作从流水线中清除

Pentium 4 pipeline operation -12 P4流水线操作12



(l) Branch check result

- 最后一个驱动阶段，实现分支检查结果。如果预测是错的，那么需要将一些微操作从流水线中清除
- 如果预测是正确的，那么还需要更新BTB
- 整个指令的执行全部完成

Summary

- Pentium is a typical CISC processor Pentium是典型的CISC处理器
- Absorbed the characteristics of RISC architecture 吸取了RISC架构的特点
  - Although memory instructions are variable in length, each instruction is translated into a fixed length RISC instruction 存储器指令虽然是不定长的，但是每个指令被译成定长的RISC指令
  - Adopt well-designed pipeline mechanism 采用精心设计的流水线机制
- Perform micro ops on superscalar pipelines 在超标量流水线上执行微操作
- Micro operations are executed in the way of disordered issue and completion 微操作以乱序发射乱序完成的方式执行
- Reduce the data dependency of instructions through register renaming 通过寄存器重命名，减少了指令的数据相关性
- Submit the results in the order of the source program flow to ensure the consistency of the result 以源程序流的顺序提交结果，保证结果的一致性

Outline

- Overview of Superscalar 超标量综述
- Design Issues of Superscalar 超标量设计问题
- Superscalar in Pentium 4 Pentium4中的超标量
- Superscalar in ARM CORTEX-A8 ARM中的超标量

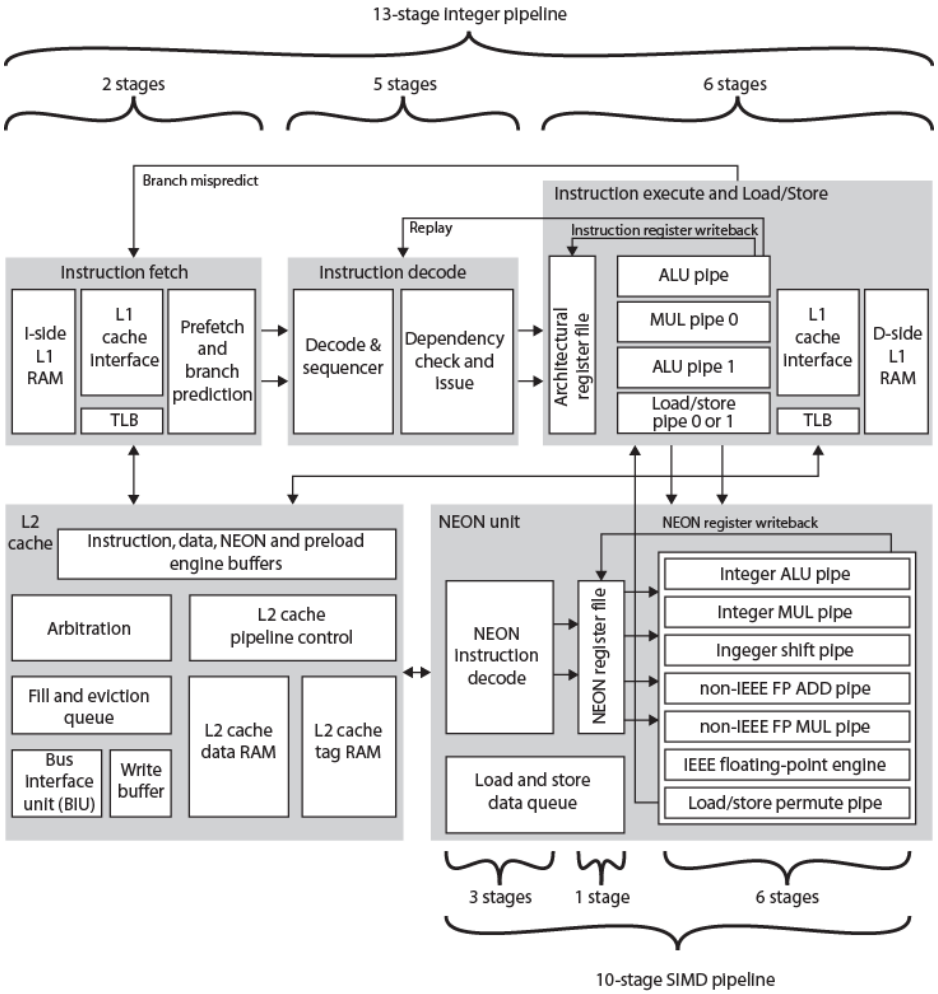
ARM Cortex-A8 -1

- ARM refers to Cortex-A8 as application processors Cortex-A8在ARM系列的处理器中被称为是应用处理器
- Embedded processor running complex operating system 运行复杂操作系统的嵌入式处理器
  - Wireless, consumer and imaging applications 无线通信，消费电子，以及图像应用
  - Mobile phones, set-top boxes, gaming consoles，automotive navigation/entertainment systems 手机，机顶盒， 游戏机，汽车导航，汽车娱乐系统

ARM Cortex-A8 -2

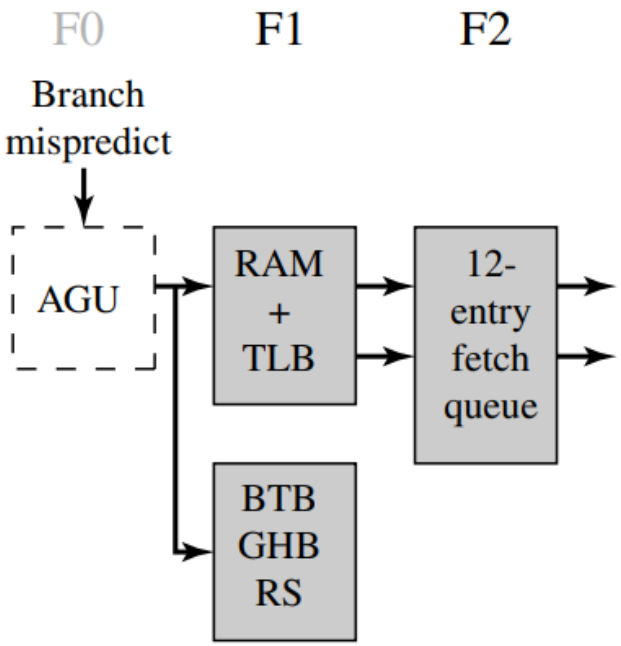
- Three functional units 三个功能单元
  - Fetch，decode，execute 取指单元，译码单元，执行单元
- Dual, in-order-issue, 13-stage pipeline 2个按序发射的13阶段流水线
  - Keep power required to a minimum 功耗保持最低
  - Out-of-order issue needs extra logic consuming extra power 乱序发射需要额外的耗费能量的逻辑
- Separate SIMD (single-instruction-multiple-data) unit 独立的单指令多数据单元
  - Realize 10-stage pipeline 实现10阶段流水线

ARM Cortex-A8 block diagram 模块图



- 整数主流水线分为13个阶段，其中取指2个阶段，译码3个阶段，执行和加载/保存6个阶段
- SIMD流水线，分为10个阶段

Instruction fetch pipeline 取指流水线



- AGU：Address Generation Unit 地址生成单元
- BTB：Branch Target Buffer分支目标缓冲器
- GHB：Global History Buffer全局历史缓冲器
- TLB：Translation Lookaside Buffer 转换后援缓冲器

- F0，地址生成单元生成一个虚拟地址
- F1，计算地址从L1高速缓存中取指
- F2，将指令放到指令队列中

Instruction fetch unit -1 取指单元1

- Predicts instruction stream 预测指令流
- Fetches instructions from the L1 instruction cache 从一级指令cache获取指令
  - Fetch unit includes L1 instruction cache 取指单元包括一级指令cache
  - Up to four instructions per cycle 每个周期最多四条指令
  - Instruction into buffer for decode pipeline 进入解码流水线的缓冲区
- Speculate instruction and fetches 推测指令并获取
- Branch or exceptional instruction cause pipeline flush 分支或异常指令导致流水线清空

Instruction fetch unit -2 取指单元2

- F0：address generation unit generates virtual address F0：地址生成单元生成虚拟地址
  - Normally next sequentially 通常下一个顺序
  - Can also be branch target address 也可以是分支目标地址
- F1：fetch instructions from L1 instruction cache F1：从一级指令缓存获取指令
  - In parallel fetch address used to access branch prediction arrays 访问分支预测阵列来并行取指
- F2：instruction are placed in queue F2：指令放在队列中
  - If branch prediction, new target address sent to address generation unit 若分支预测，则将新的目标地址发送给地址生成单元

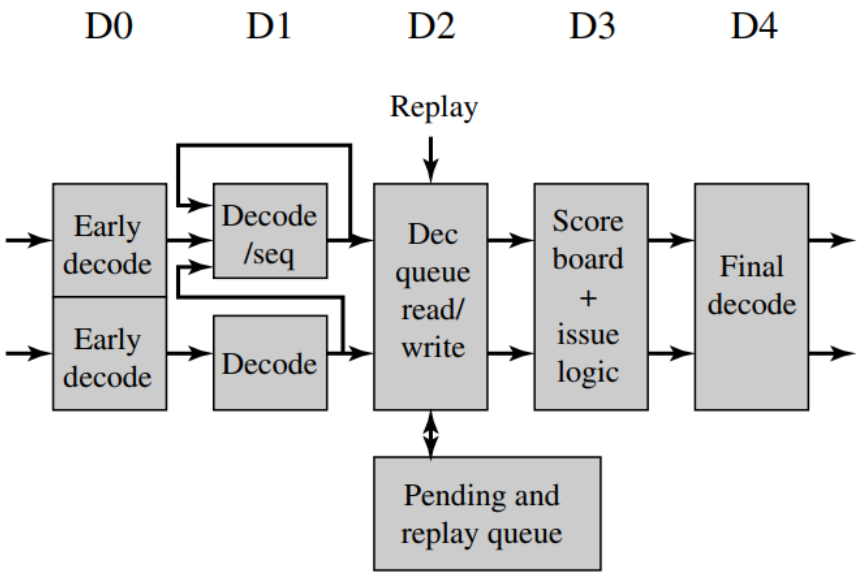
Instruction fetch unit -3 取指单元3

- Two-level global history branch predictor 两级全局历史分支预测器
  - Branch Target Buffer (BTB) and Global History Buffer (GHB) 分支目标缓冲器（BTB）和全局历史缓冲器（GHB）
  - BTB determines whether it is a branch instruction BTB确定是否为分支指令
  - GHB determines whether to transfer GHB确定是否进行转移
- Can fetch and queue up to 12 instructions 最多可以获取和排队12条指令
- Issues instructions two at one time 一次发出两条指令

Instruction fetch unit -4 取指单元4

- RS：return stack
  - Used to predict subroutine return addresses 用于预测子过程返回地址的返回栈
  - If an instruction of return type is predicted to be transferred 如果返回类型的指令被预测为要发生转移时
  - The return stack provides the address and status of the last pushed stack 返回栈提供最后被压入栈的地址和状态
  - Realize the acquisition of process return information 实现了过程返回信息获取

Instruction decode pipeline 译码流水线



- 译码流水线分为五个阶段
- D0，早期译码
- D1，继续完成译码
- D2，将译码指令写入到等待/重放队列中
- D3，指令调度逻辑
- D4，完成最终译码

Instruction decode unit -1 译码单元1

- Decodes and sequences all instructions 对所有指令进行解码和排序
- Dual pipeline structure, *pipe0* and *pipe1* 双流水线结构，pipe0和pipe1
  - Two instructions can progress at a time 一次可以执行两条指令
  - Pipe0 contains older instruction in program order pipe0中包含的是按程序顺序中靠前的指令
  - If instruction in pipe0 cannot issue, pipe1 will not issue 如果pipe0中的指令无法发出，pipe1将不会发出

Instruction decode unit -2 译码单元2

- Instructions issue and execute in order 指令按序发射按序完成进行
- Executing results written back to register file at end of execution pipeline 流水线结束时将执行结果写回寄存器组
  - Prevents WAR hazards 避免了写后读的冒险
  - Keeps tracking of WAW hazards and recovery from flush conditions straightforward 保持对写后写冒险的跟踪，并直接从流水线清空条件中恢复
  - Main concern of decode pipeline is prevention of RAW hazards 解码流水线的主要关注点是读后写冒险的预防

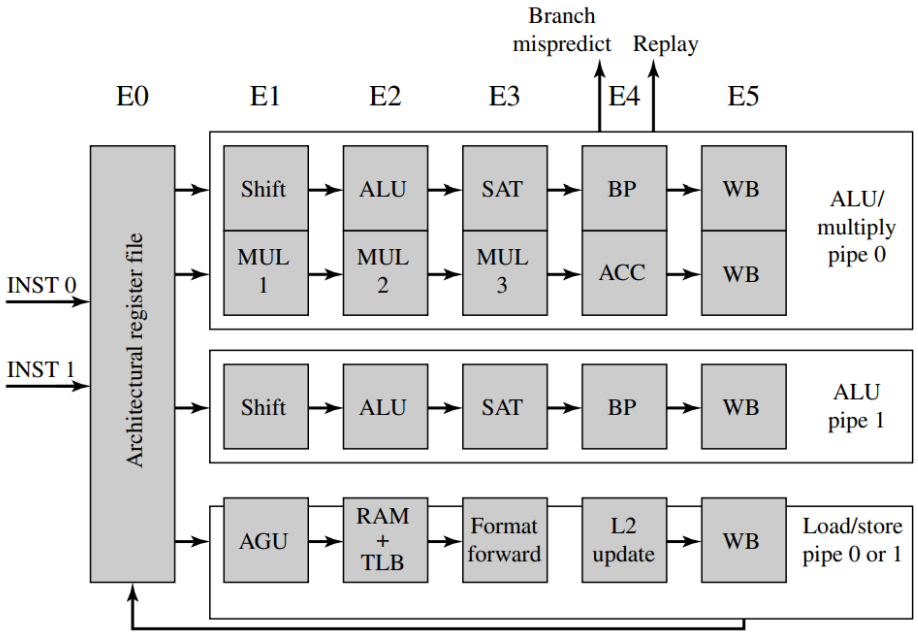
Decode processing stages -1 解码步骤1

- D0：Thumb instructions decompressed and preliminary decode is performed D0： 压缩指令解压缩并执行初步解码
- D1：Instruction decode is completed D1：指令解码完成
- D2：Write instruction to and read instructions from pending/replay queue D2：向挂起/重放队列写入指令和从中读取指令

Decode processing stages -2 解码步骤2

- D3：Contains the instruction scheduling logic D3：包含指令调度逻辑
  - Scoreboard predicts register availability using static scheduling
    记分板使用静态调度预测寄存器可用性
  - Hazard checking 冒险检查
- D4：Final decode for control signals for integer execute load/store units D4：完成最终译码，产生整数执行及装载单元需要的所有控制信号
- Through these five stages, the decoding of instructions is completed and the instructions are sent to the execution unit for execution 通过这五个阶段，完成了指令的译码，并将指令送到执行单元进行执行

Instruction execute pipeline 执行流水线



- 包括两个对称的算术逻辑单元ALU流水线pipe0和pipe1，以及装载/保存流水线
- ALU指令可以使用这2个流水线中的任意一个
- 乘法指令，只能用pipe0
- 装载/保存流水线和整数流水线可以并行运行

- 执行所有的整数ALU和乘法操作，包括标志位的生成
- 为装载和保存指令生成虚拟地址
- 为保存指令提供格式化后的数据，并转发数据及标志
- 处理分支和其他对指令流的改变，并计算指令条件码

Integer execution unit 执行单元

- Two symmetric ALU pipelines, an address generator for load and store instructions 两个对称的ALU流水线，一个加载和/保存流水线
  - ALU instruction can use any of the two pipelines ALU指令可以使用这2个流水线中的任意一个
  - Multiply instruction can only use pipe0 乘法指令只能用pipe0
  - Load/save pipeline and integer pipeline can run in parallel 装载/保存流水线和整数流水线可以并行运行

ALU pipeline ALU流水线

- E0：Access register file E0：访问寄存器组
  - Up to six registers for two instructions 2个指令最多6个寄存器
- E1：Barrel shifter if needed. E1：如果需要完成移位
- E2：ALU function E2 ALU功能
- E3：If needed, completes saturation arithmetic E3：如果需要完成饱和运算
- E4：Change in control flow prioritized and processed E4：改变控制流优先处理
- E5：Results written back to register file E5：结果写回寄存器组

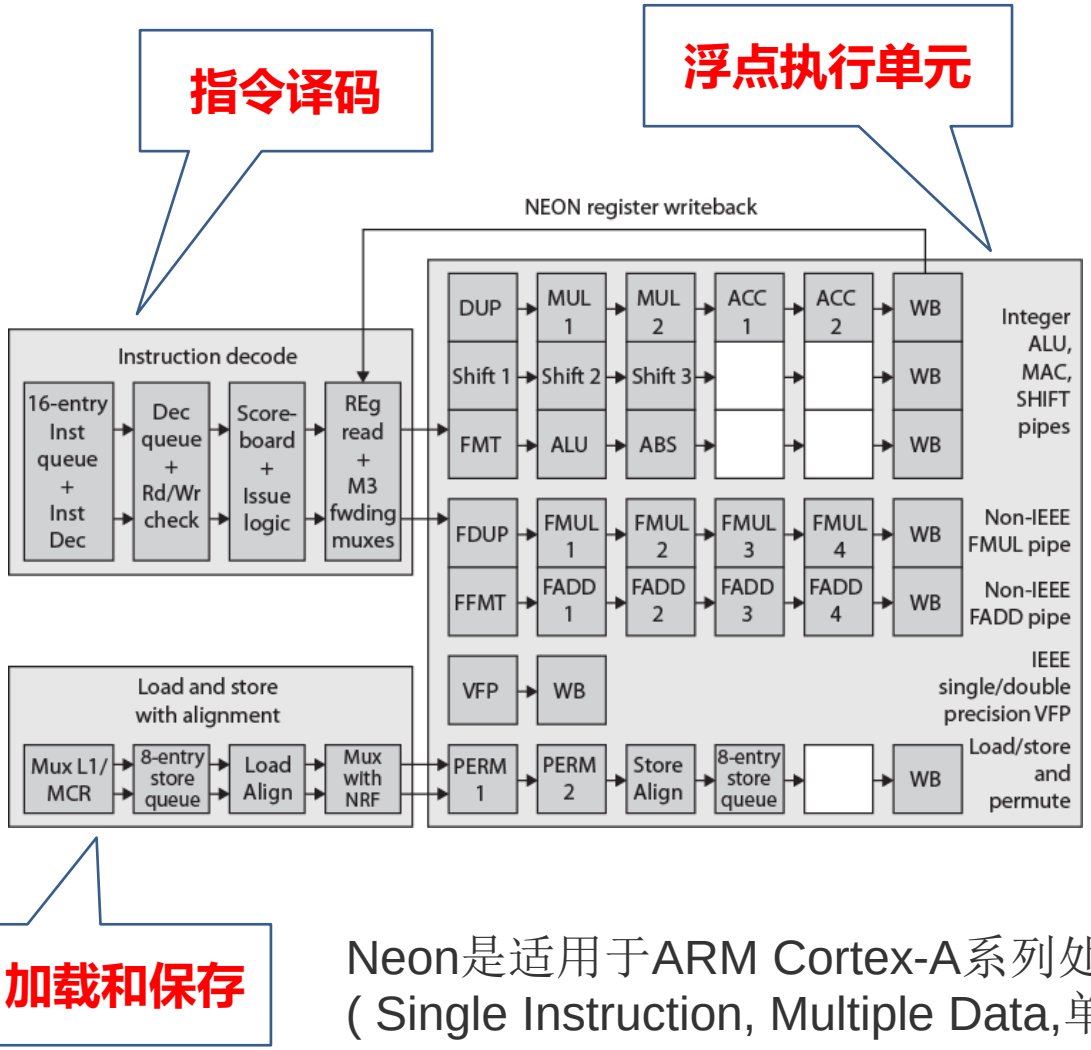
Multiply unit instruction 乘法单元指令

- Multiply unit instructions routed to pipe0 乘法单元指令路由到pipe0
- Performed in stages E1 through E3 在E1至E3阶段执行
- Multiply accumulate operation in E4 E4中进行乘法累加运算
- Write back in E5 E5阶段写回

Load/Store pipeline 加载/保存流水线

- E1：memory address generated from base and index register
  E1：从基址和变址寄存器生成存储器地址
- E2：address applied to cache arrays E2：地址用于cache阵列
- E3：
  - For load, data returned and formatted 对于加载，返回数据并格式化
  - For store, data are formatted and ready to be written to cache 对于保存，数据格式化并准备好写入cache
- E4：Updates L2 cache, if required E4：如果需要更新L2cache
- E5：Results are written to register file E5：结果写到寄存器组

NEON & floating point pipeline NEON和浮点流水线



- NEON包括三个部分：NEON指令译码模块，加载和保存队列，浮点执行单元
- 浮点流水线分为10个阶段，其中译码3个阶段，寄存器读+M3转发选通1个阶段，执行6个阶段

Neon是适用于ARM Cortex-A系列处理器的一种128位SIMD（Single Instruction, Multiple Data,单指令、多数据）扩展结构

SIMD and floating-point pipeline -1 SIMD和浮点流水线1

- SIMD and floating-point instructions pass through integer pipeline SIMD和浮点指令通过整数流水线
- Processed in separate 10-stage pipeline 由一个10阶段独立流水线处理
  - called NEON unit 称为NEON单元
  - Handles packed SIMD instructions 处理压缩的SIMD指令
  - Provides two types of floating-point （Non IEEE Standard Floating Point Numbers and IEEE Standard Floating Point Numbers ） support 提供两种类型的浮点数（非IEEE标准的浮点数，IEEE标准的浮点数）支持

SIMD and floating-point pipeline -2 SIMD和浮点流水线2

- If implemented, vector floating-point (VFP) coprocessor performs IEEE 754 floating-point operations 根据具体实现，可能会有一个向量浮点（VFP）协处理器，执行IEEE 754标准的浮点运算
- If not, separate multiply and add pipelines implement floating-point operations 如果没有，则使用单独的乘法和加法流水线来实现浮点操作

Key Terms

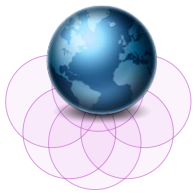
Antidependency	Instruction-level parallelism	Output dependency	Superscalar
Branch prediction	Instruction window	Procedural dependency	True data dependency
In-order issue	Machine parallelism	Register renaming	
in—order completion	Out-of-order completion	resource conflict	
Instruction issue	Out-of-order issue	superpipelined	

Summary and Question

- 小结
  - 对流水线存在的问题进行了分析，提出了超标量的概念
  - 对超标量中的设计问题进行了分析
  - 对Pentium4和ARM Cortex-A8的流水线进行了介绍
- 问题
  - 超标量流水线的本质是什么
  - 超标量会带来哪些问题？
  - 简述机器并行性、指令发射策略和重命名之间的关系

Assignments

- Review Questions
  - 16.1~16.9
- Problems:
  - 16.3, 16.4



谢谢大家！