计算机组织与体系结构

Computer Architectures

陆俊林

2024年春季学期

第八讲 流水线优化技术

本讲要点

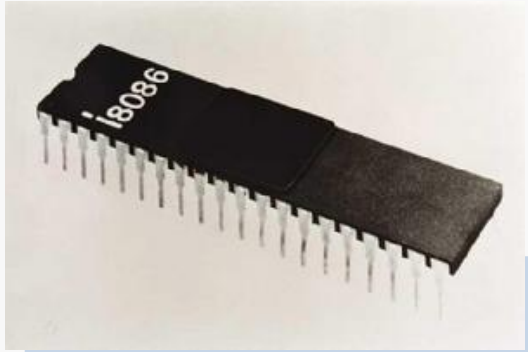
首先简要回顾流水线结构的发展，其次分析转移指令对流水线的影响，然后讲解转移预测技术，最后介绍过程返回指令的处理。

阅读教材《微型计算机……》：7.1

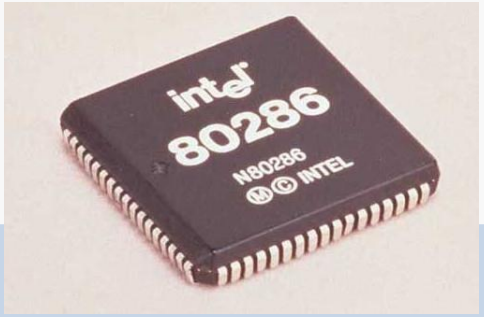
主要内容

通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

早期的x86 CPU



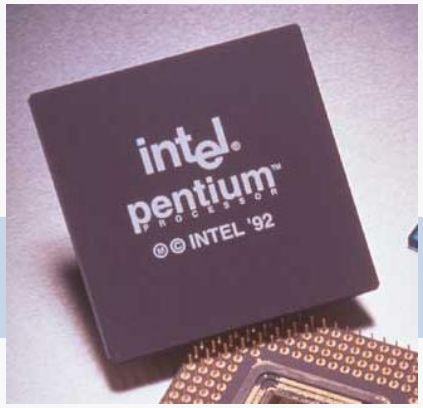
1978年
8086
主频4.77~10MHz
4万个晶体管



1982年
80286
主频6~20MHz
13.4万个晶体管

1985年
80386
主频12.5~33MHz
27.5万个晶体管

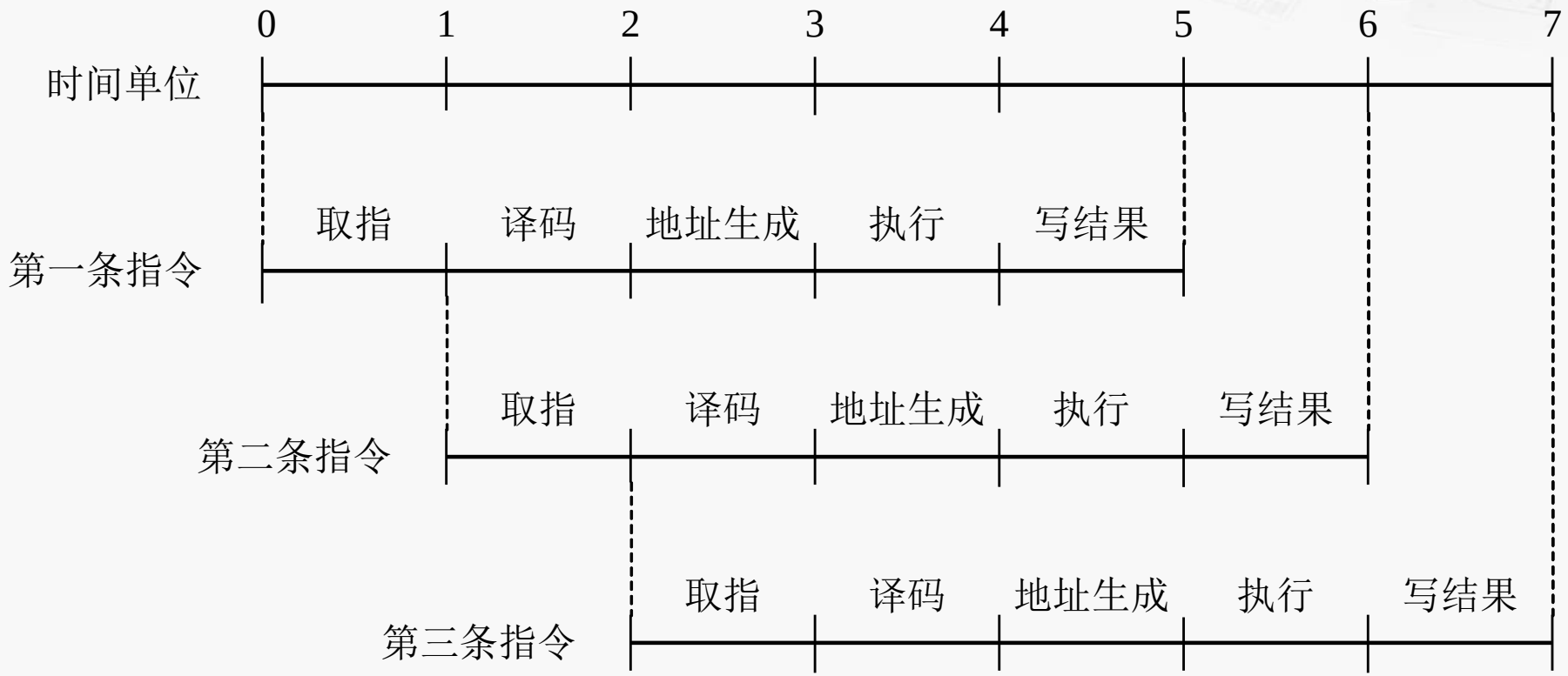
1989年
80486
主频25~100MHz
125万个晶体管



1993年
Pentium
主频60~200MHz
310万个晶体管

1995年
Pentium Pro
主频133~200MHz
550万个晶体管

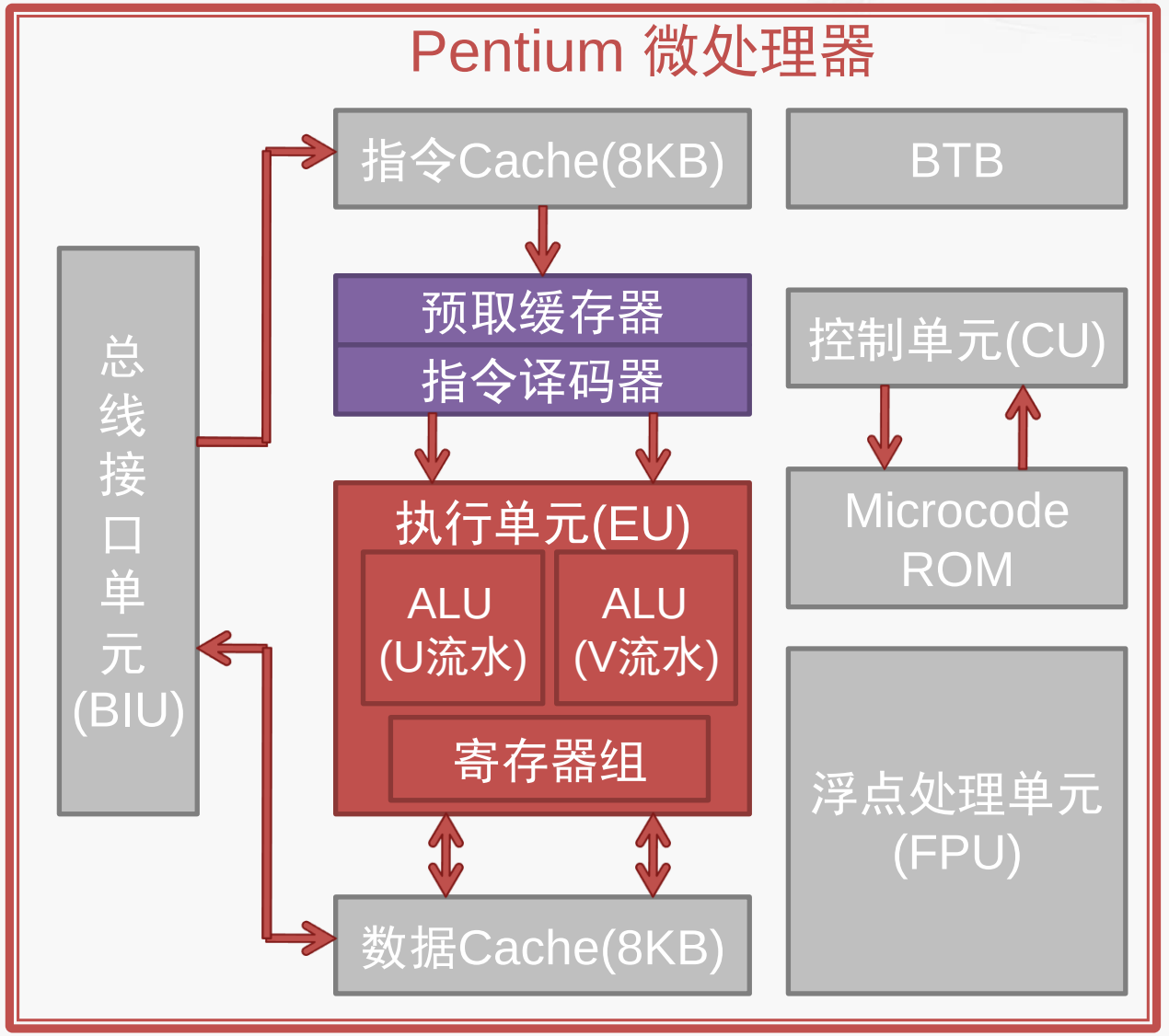
486的五级流水线设计（第一款流水线x86 CPU）



Pentium的超标量流水线（第一款超标量流水线x86 CPU）

双发射，5级流水线

- 两条流水线：“U流水”和“V流水”
- 每条流水线都拥有自己的地址生成逻辑、ALU及数据Cache接口
- 在一个时钟周期内，可以同时发送两条指令



处理器流水线深度的变化

1986年，MIPS R2000: 5级

1988年，MIPS R3000: 5级

1989年，80486: 5级

1991年，MIPS R4000（64位）: 8级

1993年，Pentium: 5级

取指	译码	地址生成	执行	回写

1995年，Pentium Pro: 12级

IF1	IF2	IF3	ID1	ID2	RAT	ROB	DIS	EX	WB	RR	RET

1997年，ARM9: 5级

取指	译码	执行	访存	写回

2002年，ARM11: 8级

取指1	取指2	译码	发射	执行1	执行2	执行3	写回

2004年，Pemtium 4(Prescott)：**31级**

2006年，Core 2 Duo(Merom)：14级

2008年，Core i7(Nehalem)：16级

2009年，Cortex-A8：13级

2010年，Cortex-A9：11级

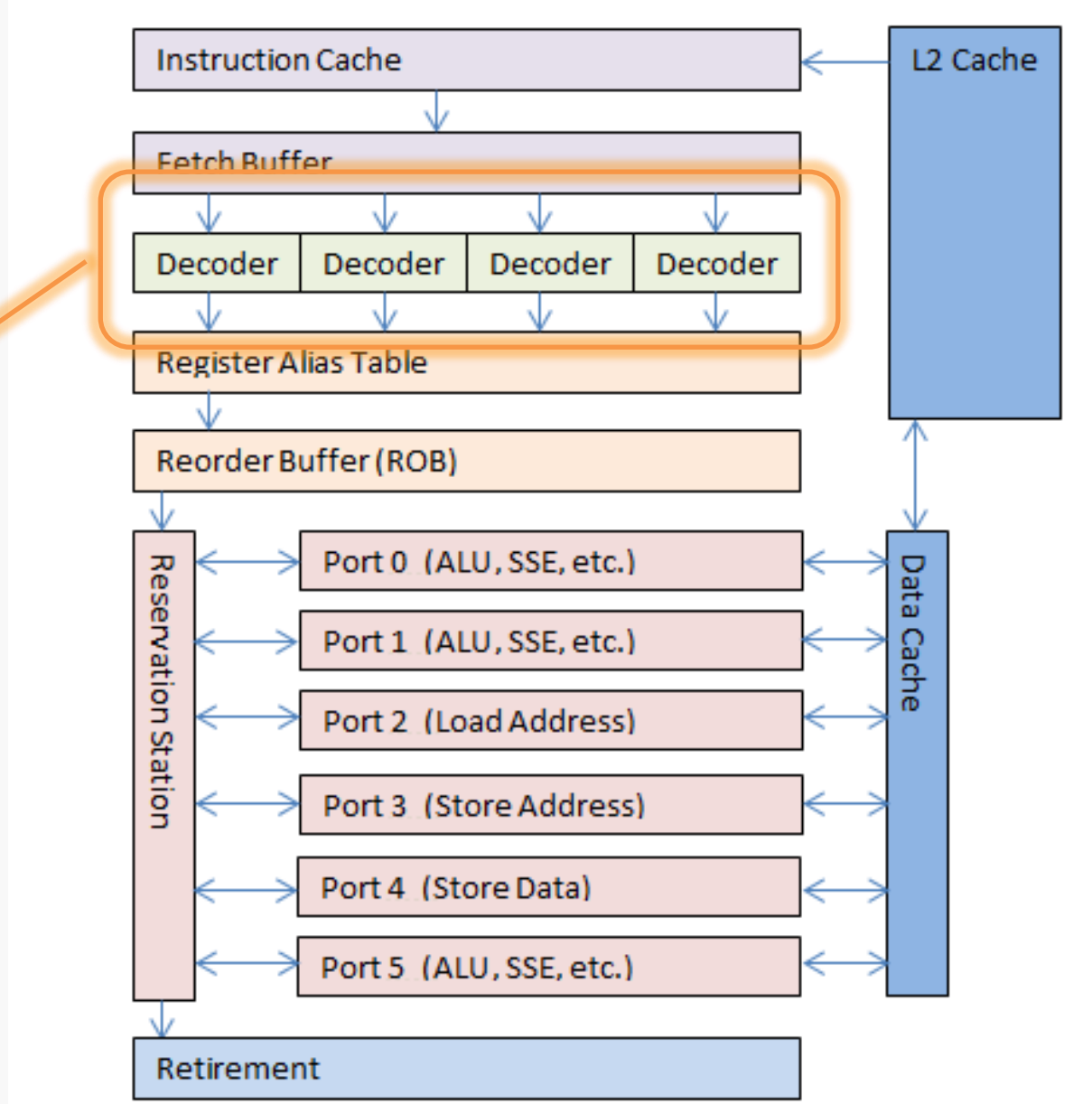
2011年，Cortex-A15：15级

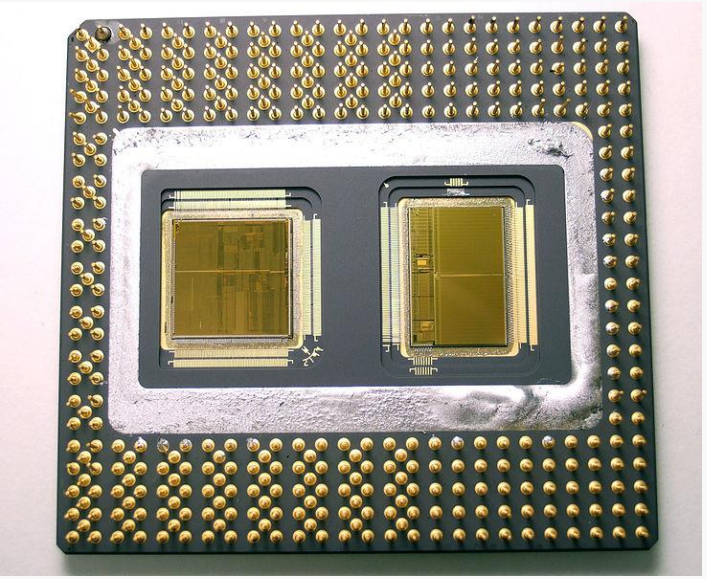
2013年，Core i7(Haswell)：14级

2013年，Cortex-A57：15级

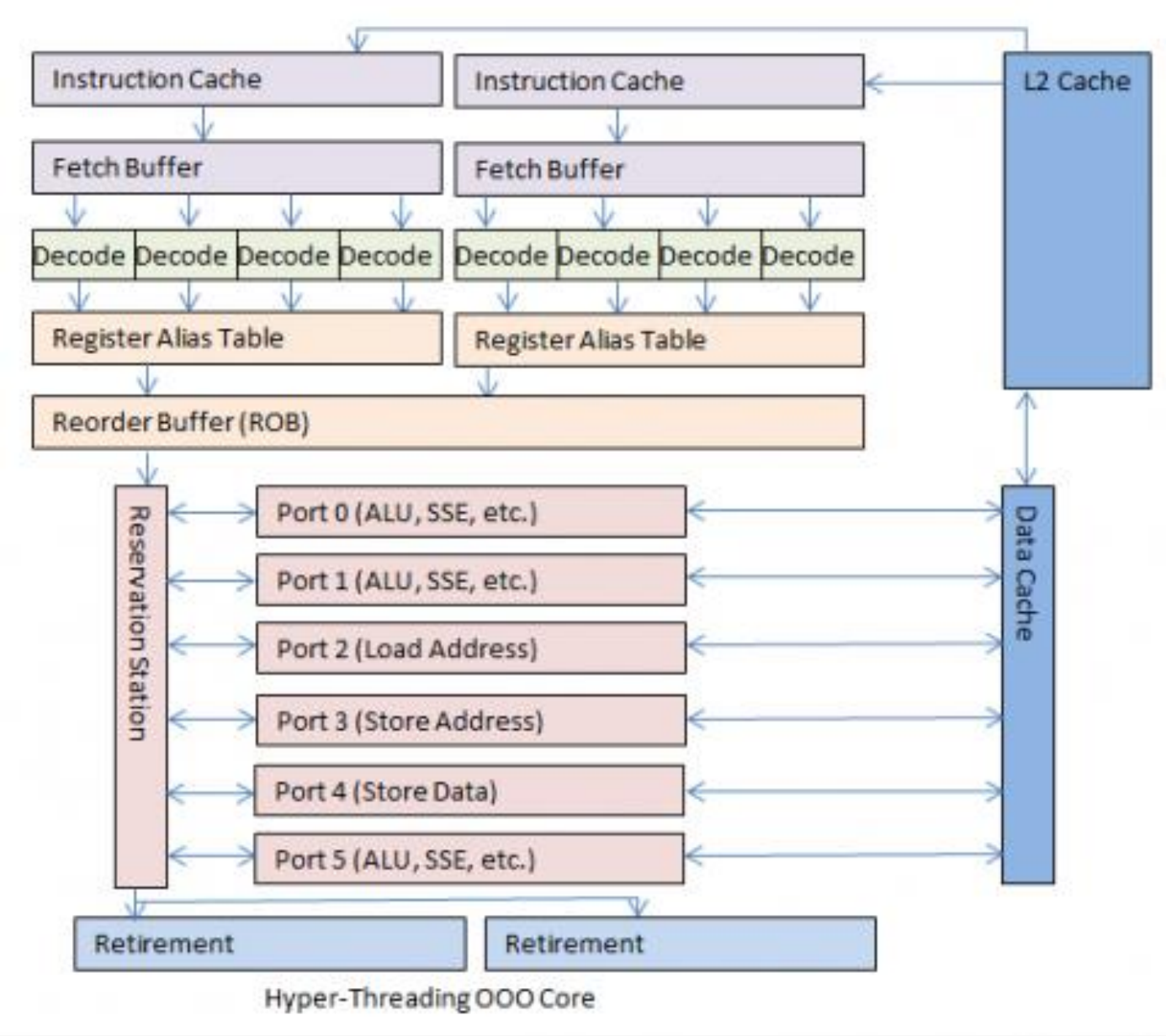
Pentium Pro的超标量流水线

将复杂的x86指令拆分成简单的微指令（微操作，μop）





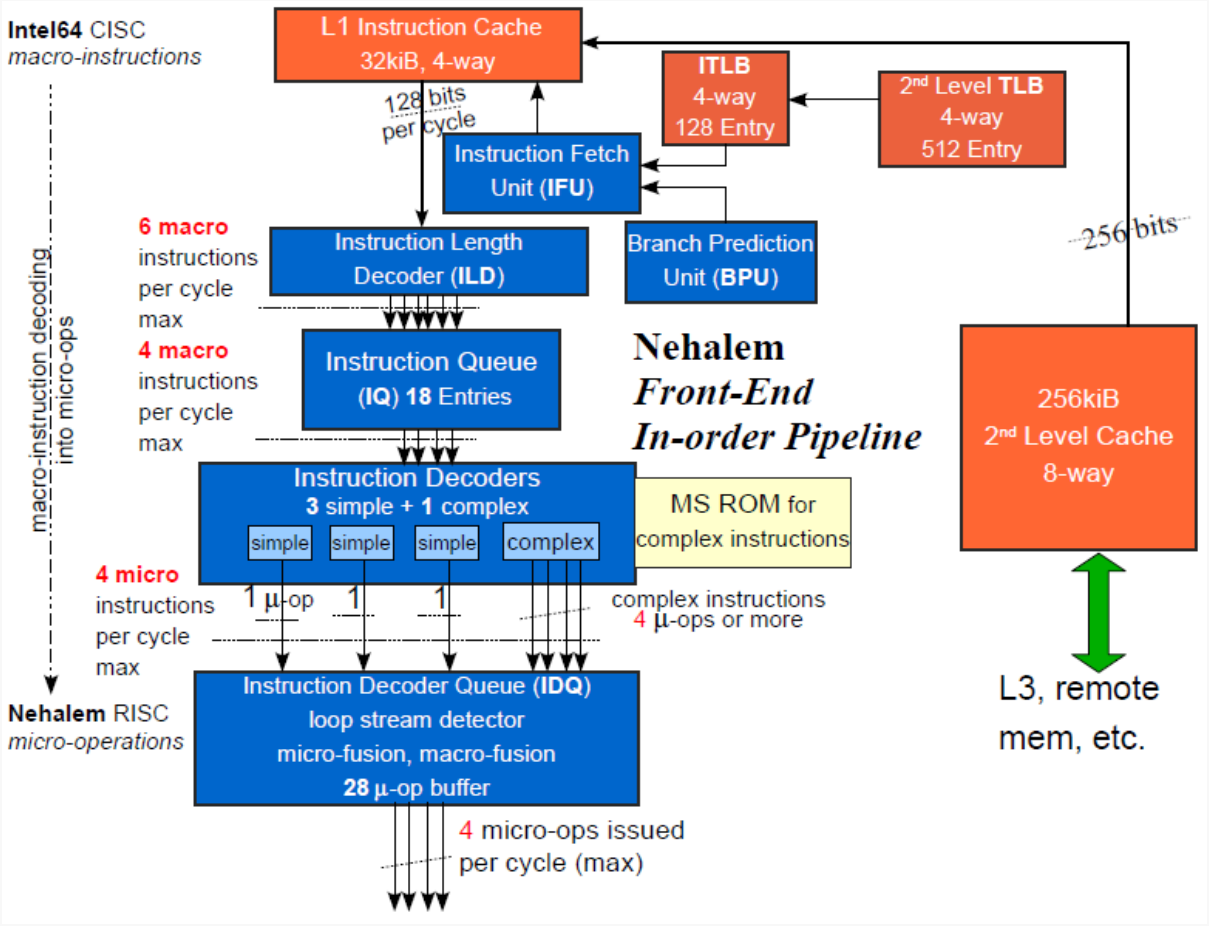
Hyper-Threading的流水线结构

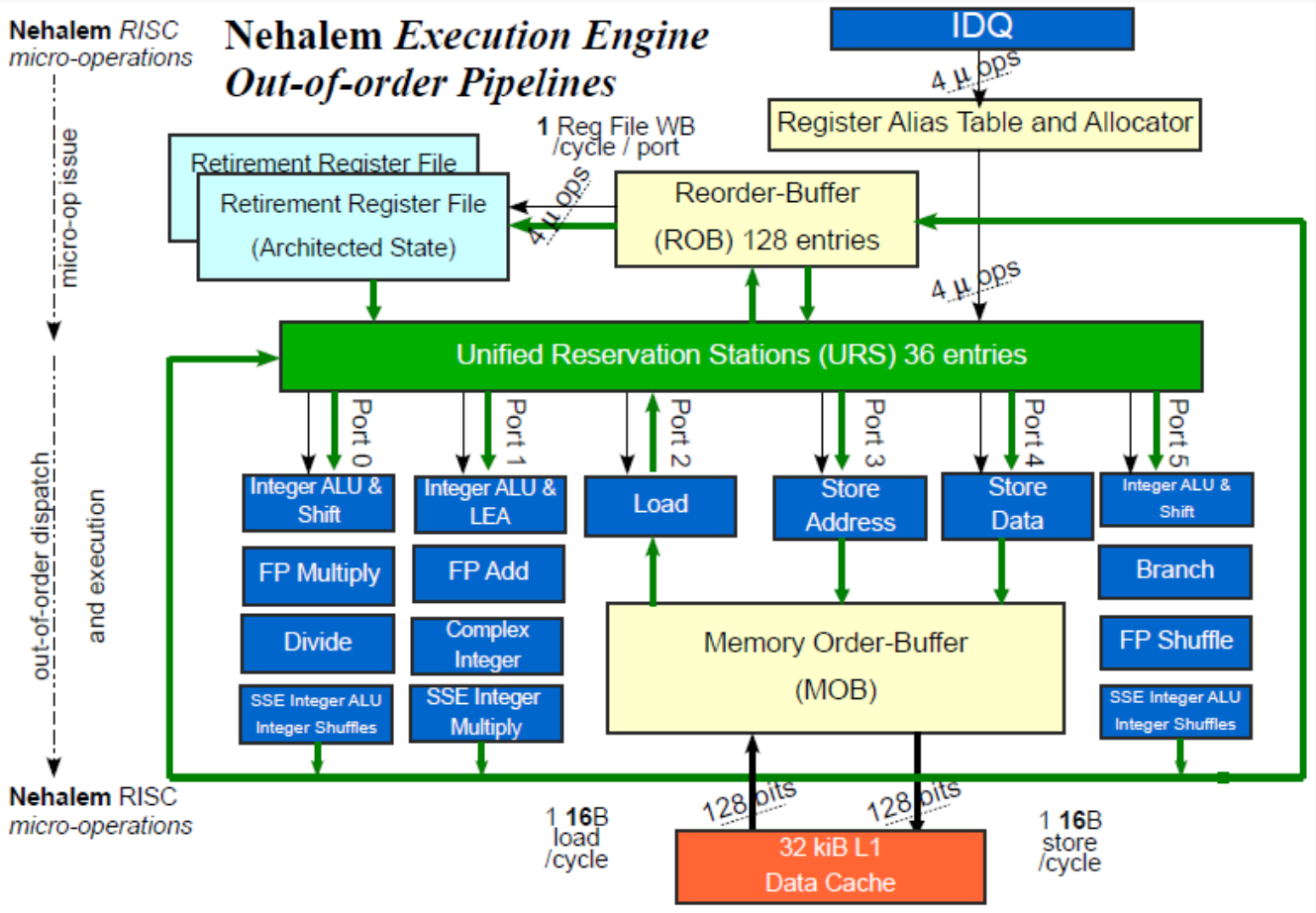


Pemtium 4
31级流水
主频3.2~3.8GHz

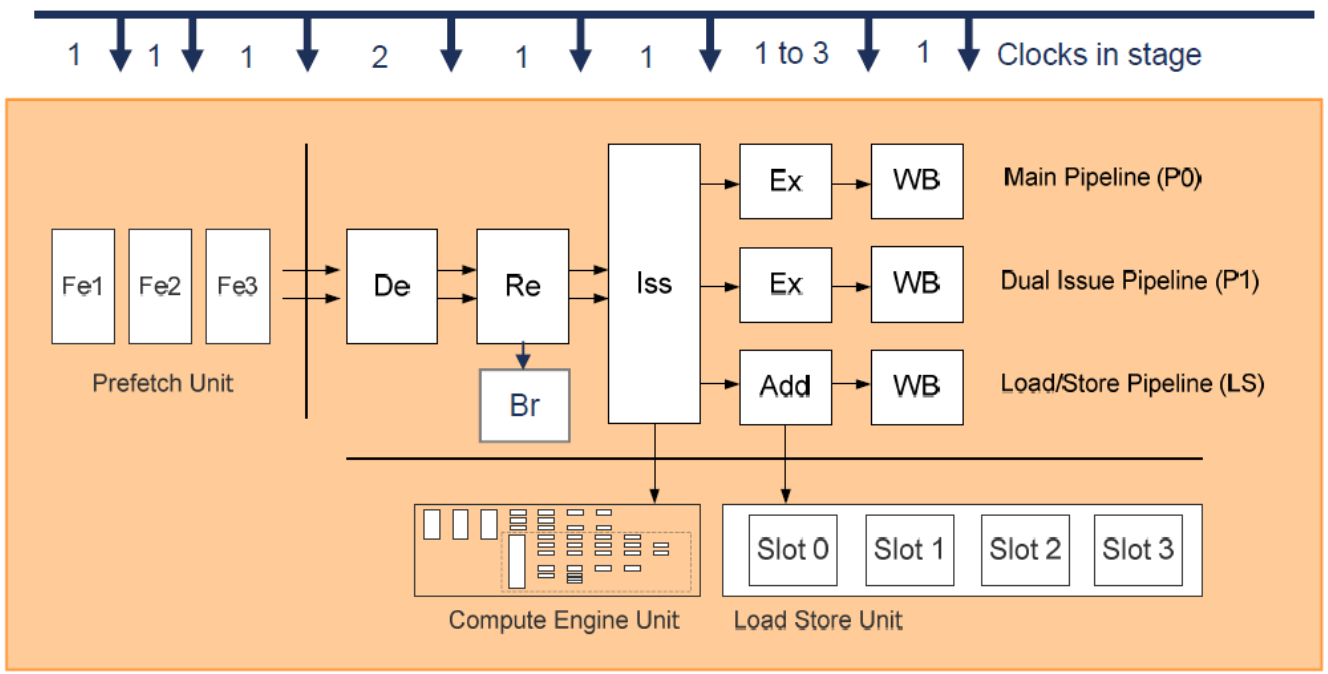
Intel Core i7

4发射、16级流水

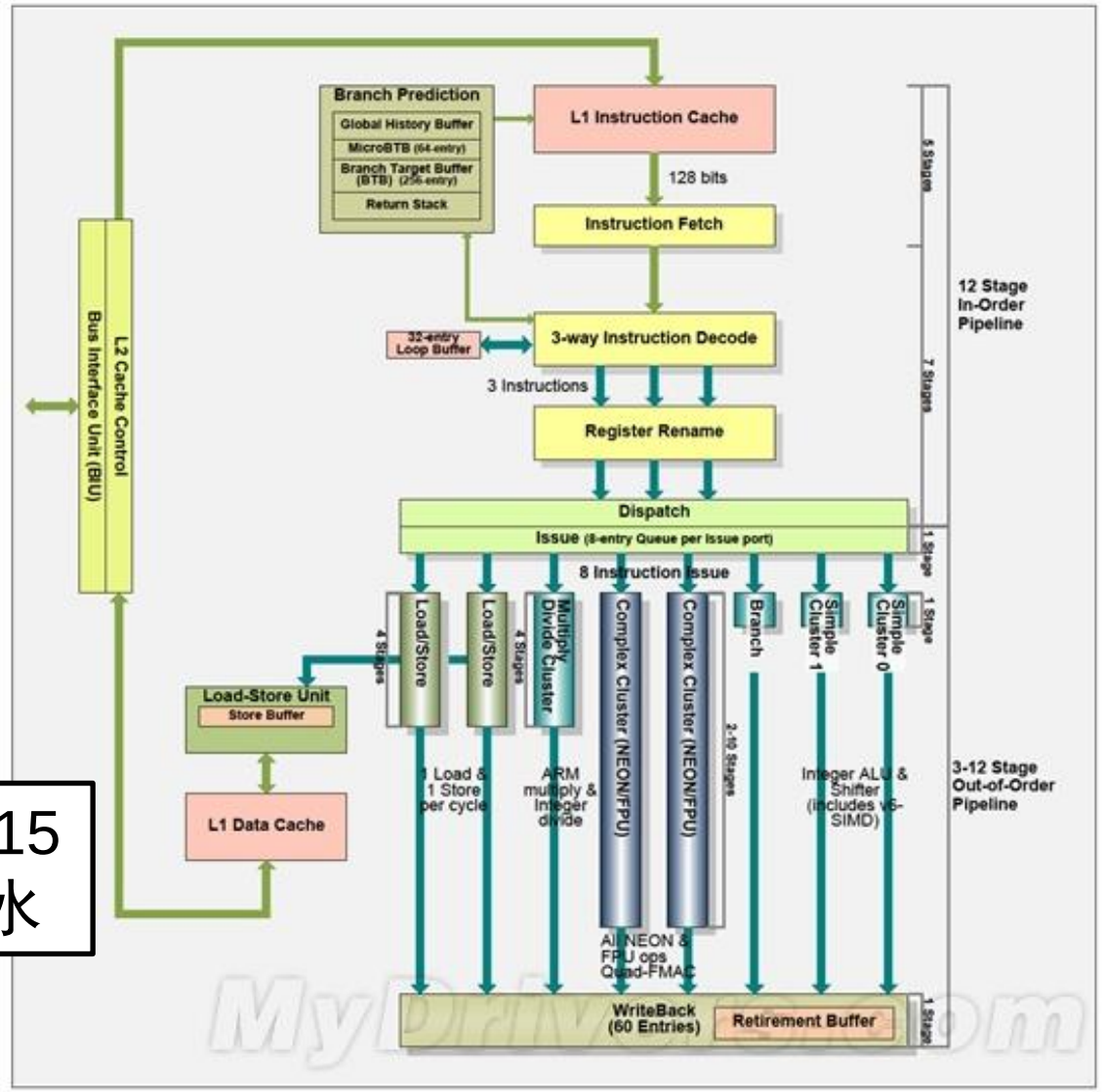




ARM Cortex-A系列



ARM Cortex-A9
4发射 8~11级流水



ARM Cortex-A15
3发射 15级流水

主要内容

通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

转移指令

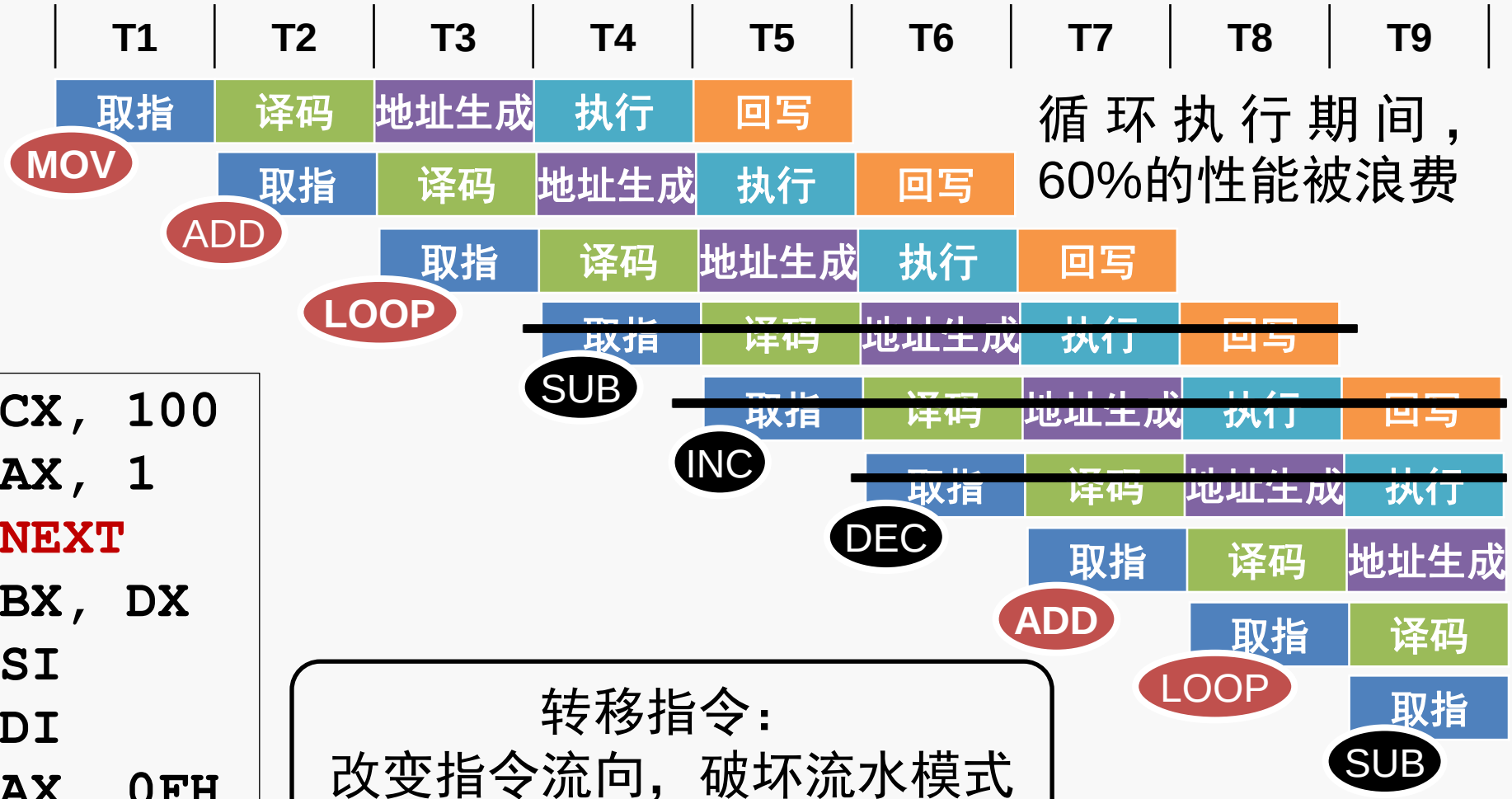
	无条件转移	条件转移

直接转移	**x86示例：**
	`JMP  Target`
	`CALL Target`
	
	MIPS示例：
	`j    Label_2`
	`jal  Lable_4`
		x86示例：
		`JZ   Target`
		`LOOP Target`
		
		MIPS示例：
		`beq  $t0, $t1, Target`
		`bgez $t0, Target`

间接转移	**x86示例：**
	`JMP  DWORD PTR [30H]`
	`JMP  [EAX]`
	`CALL EAX`
	MIPS示例：
	`jr   $t0`	

转移指令对流水线的影响

流水线的最终性能目标：最大的指令吞吐率




```
        MOV   CX, 100
NEXT:   ADD   AX, 1
        LOOP  NEXT
        SUB   BX, DX
        INC   SI
        DEC   DI
        OR    AX, 0FH
```

转移指令：
改变指令流向，破坏流水模式

转移指令对性能的影响

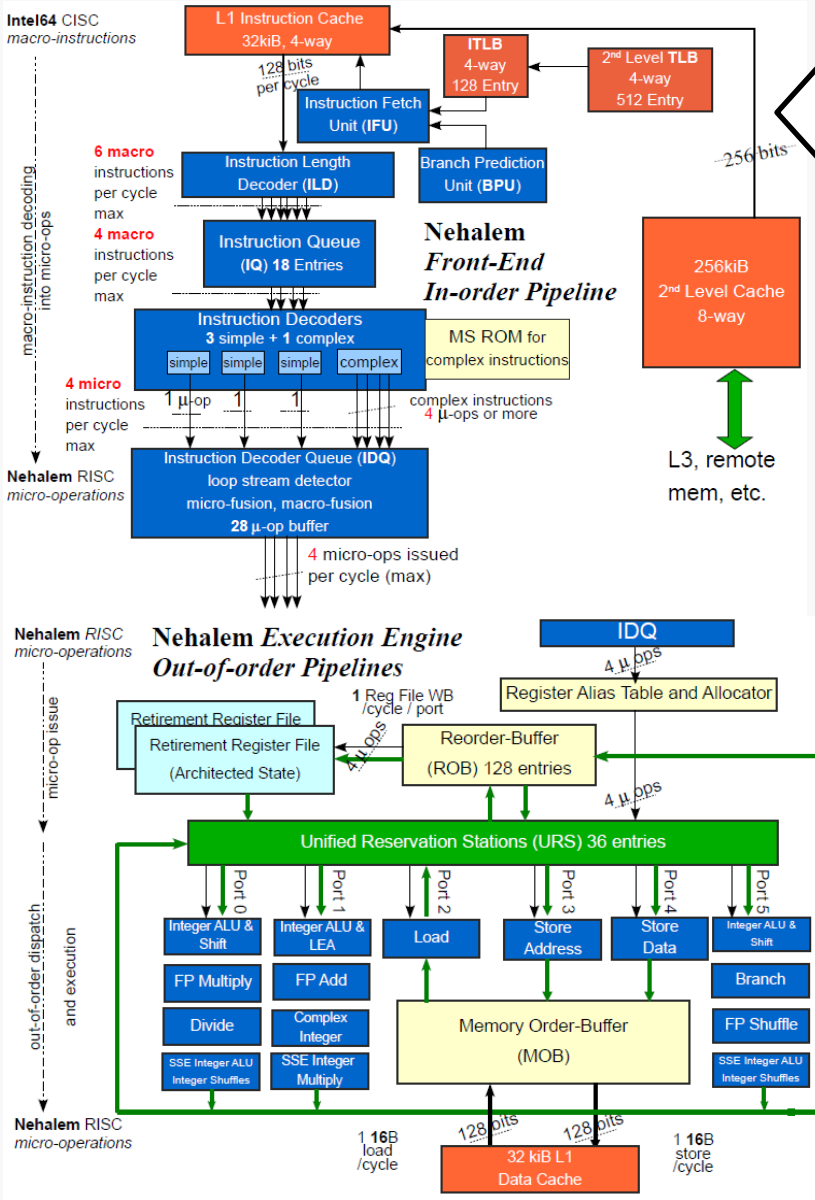
- 转移指令所占比例（Branch Frequency）
  - 每隔4到7条指令就会有一条转移指令
  - 转移指令所占比例大约为15%~25%

- 转移指令损失（Branch Penalty）
  - Alpha 21264：转移损失平均为7个周期
  - Pentium III：转移损失平均为10~15个周期
  - AMD Athlon K7：转移损失10个周期以上

Pipeline stall cycles from branches
= Branch Frequency × Branch Penalty

*Computer Architecture: A Quantitative Approach

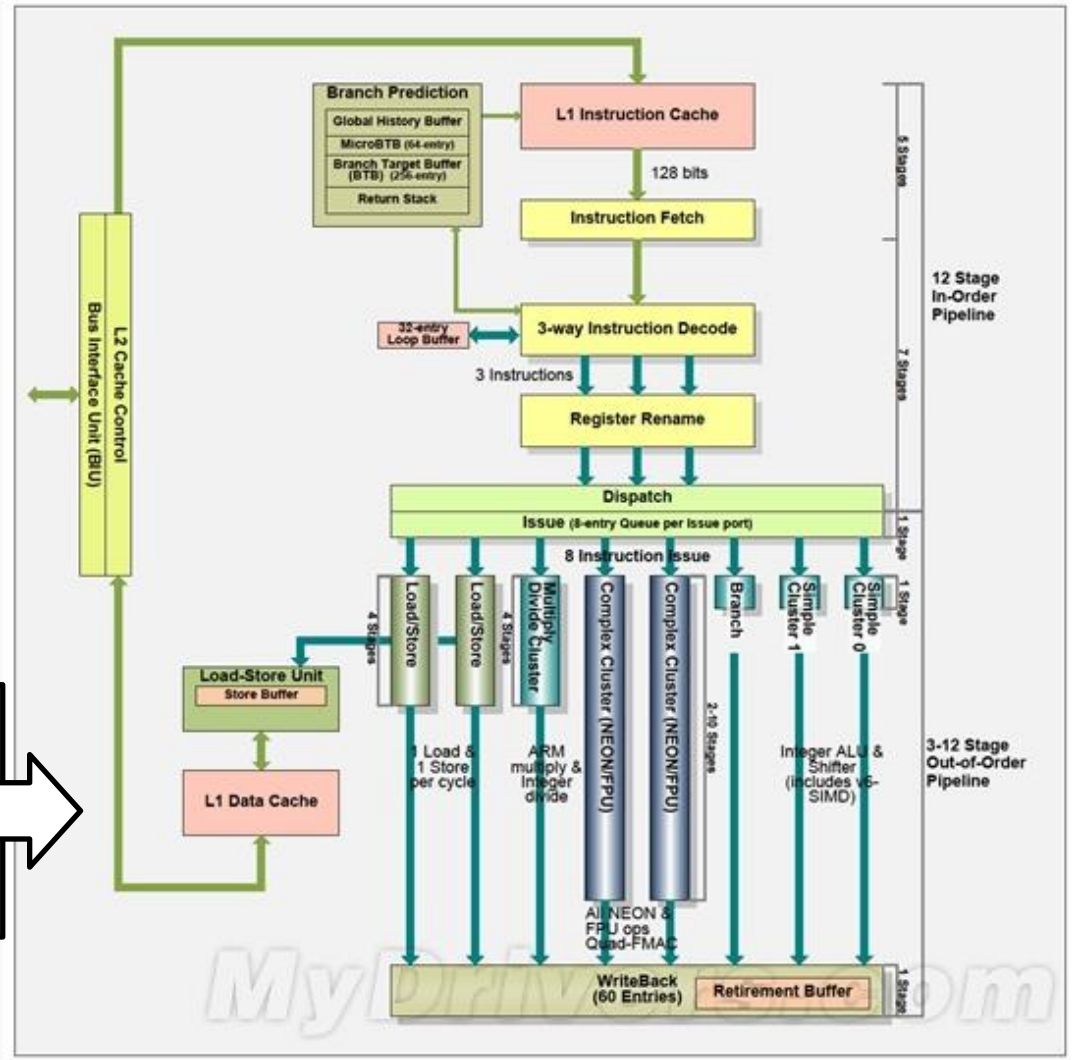
超标量处理器实例



Intel Core i7
4发射
16级流水

流水线越深
超标量数越多
转移指令影响越大

ARM Cortex-A15
3发射
15级流水



主要内容

通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

转移开销

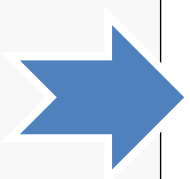
- 当转移指令被执行，并确实发生转移时，产生如下的开销，称为“转移开销”
  ① 将按顺序预取的指令废除（即“排空流水线”）
  ② 从转移目标地址重新取指令

- 转移开销的构成
  ① “要不要转移？”：转移条件判定引起的开销
  ② “转移到哪里？”：生成目标地址引起的开销

减小转移指令影响的技术举例（1）

- 延迟转移（Delayed Branching）技术
  - 在编译过程中，通过编译器调度，在转移指令之后插入一条或几条适当的指令
  - 当被调度的指令执行完成后，转移指令的目标地址和判断条件都已计算完成

```
        INC    AX
        INC    BX
        DEC    CX
        JNZ    NEXT
        …
NEXT:ADD    DX, 2
```



```
        INC    AX
        INC    BX
        DEC    CX
        JNZ    NEXT
        NOP
        NOP
        …
NEXT:ADD    DX, 2
```

延迟转移示意

```
        DEC    CX
        JNZ    NEXT
        INC    AX
        INC    BX
        …
NEXT:ADD    DX, 2
```

减小转移指令影响的技术举例（2）

转移预测（Branch Prediction）技术

- 依据指令过去的行为来预测将来的行为

① 转移条件判定的预测：预测“要不要转移”

② 转移目标地址的预测：预测“转移到哪里”

“转移预测”的实现要求

1、必须能够验证转移预测的结果

2、在预测失败时，能够恢复正确的执行方式

转移预测技术的实现要点

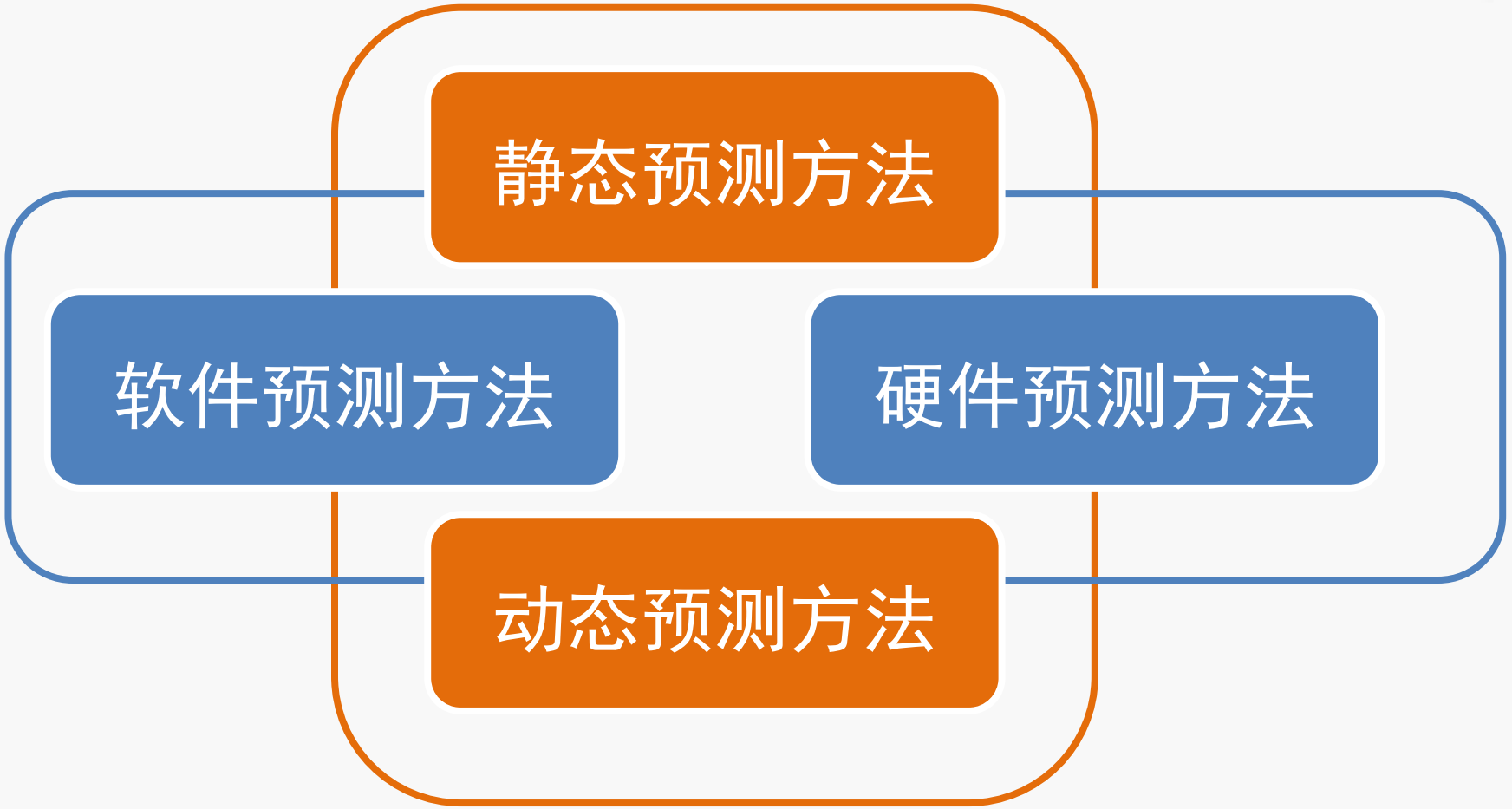
转移条件判定的预测

- 预测“要不要转移”

转移目标地址的预测

- 预测“转移到哪里”

转移条件判定的预测



转移条件判定的预测方法（1）

硬件固定预测不转移

- 在转移条件判定之前总是顺序地取下一条指令
- 优点：实现简单；缺点：预测效果不佳

编译制导的预测（Motorola M88110采用）

- 在转移指令的编码中增加1位，编译器设置该位来通知硬件是预测跳转还是预测不跳转
- 优点：软件可根据指令类型和历史信息，对不同指令进行不同的预测
- 缺点：需要软件支持；需要修改ISA；不适应多变的执行环境

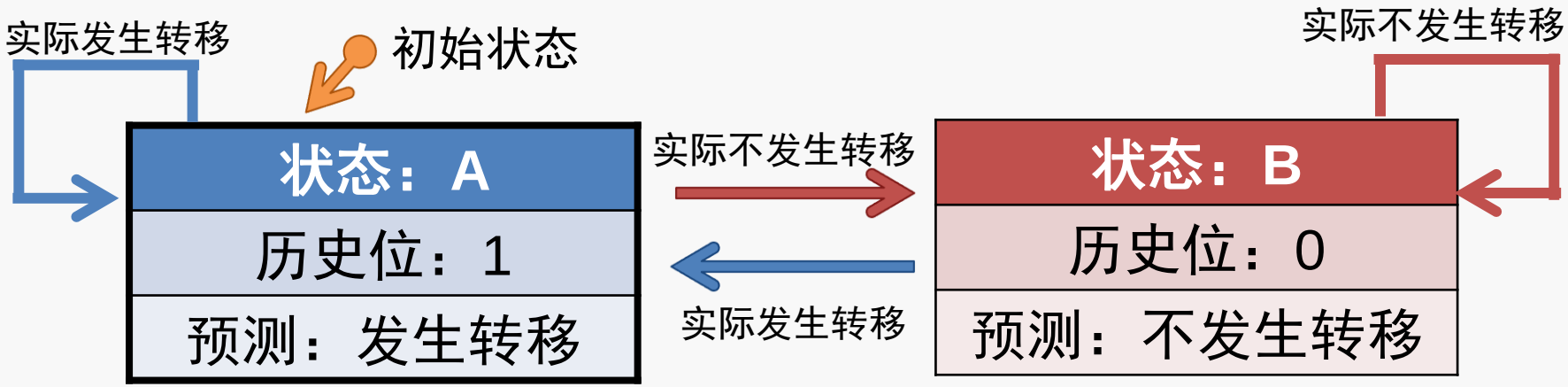
转移条件判定的预测方法（2）

- 基于偏移的预测（IBM RS/6000首先使用）
  - 如果转移指令地址和转移目标地址的相对偏移为负值（有可能是循环结束处的转移指令），则预测转移；否则，预测不转移
  - BTFN: Back Taken, Forward Not taken

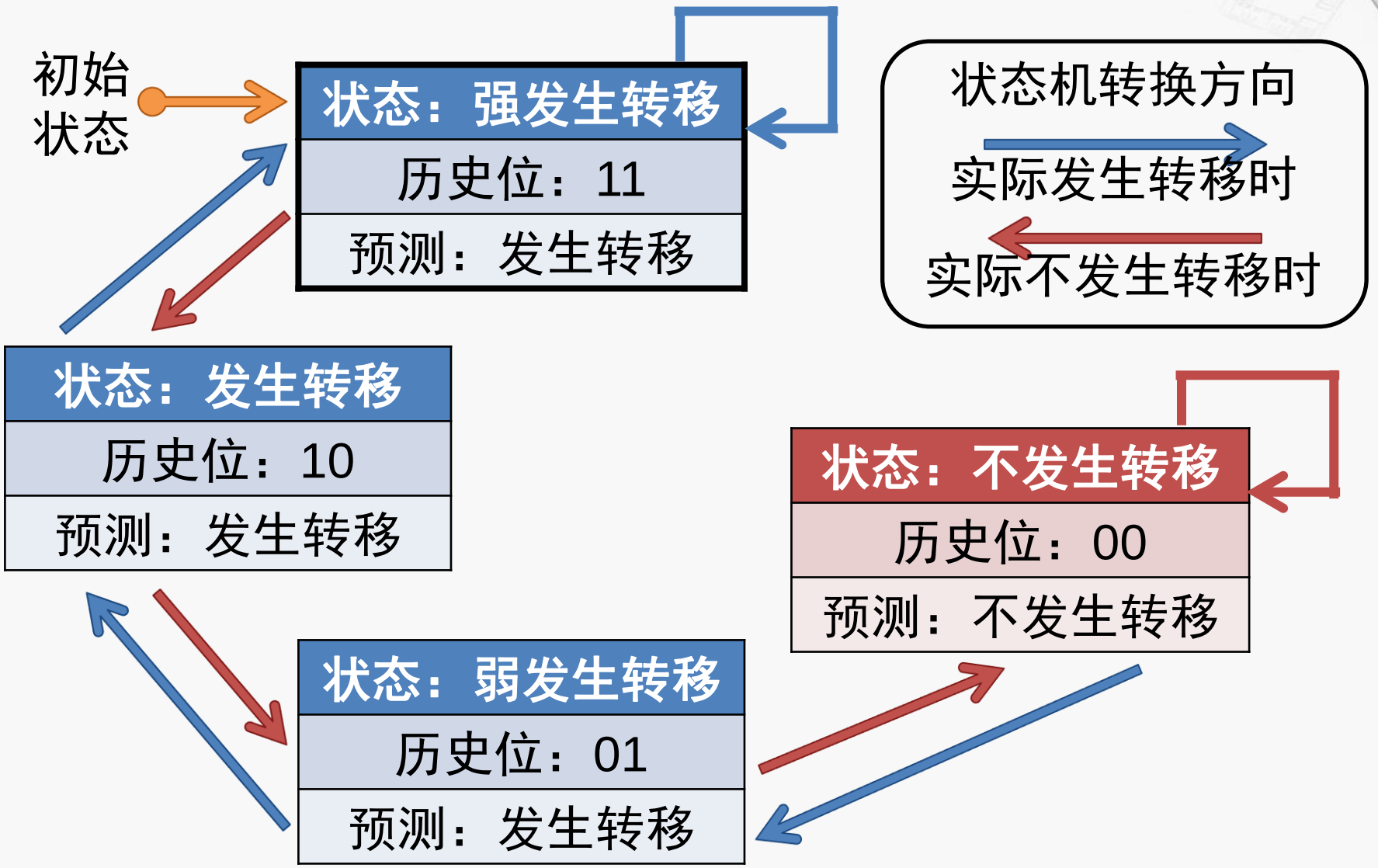
- 基于历史信息的预测（当前普遍采用）
  - 转移与否取决于先前的转移指令的执行情况
  - 设计考虑：跟踪多少历史信息；对各种历史信息进行何种预测

历史信息转移预测器的简化示例

- 在取指部件中，使用1位历史信息进行预测
  - 用1位记录最近一次转移指令是否实际发生转移
    - “1”代表实际发生转移，“0”代表实际不发生转移
  - 功能：当遇到转移指令时，根据历史信息预测是否发生转移；当转移指令执行完成时，根据实际是否发生转移，更新历史信息



Pentium的两位历史信息转移预测器



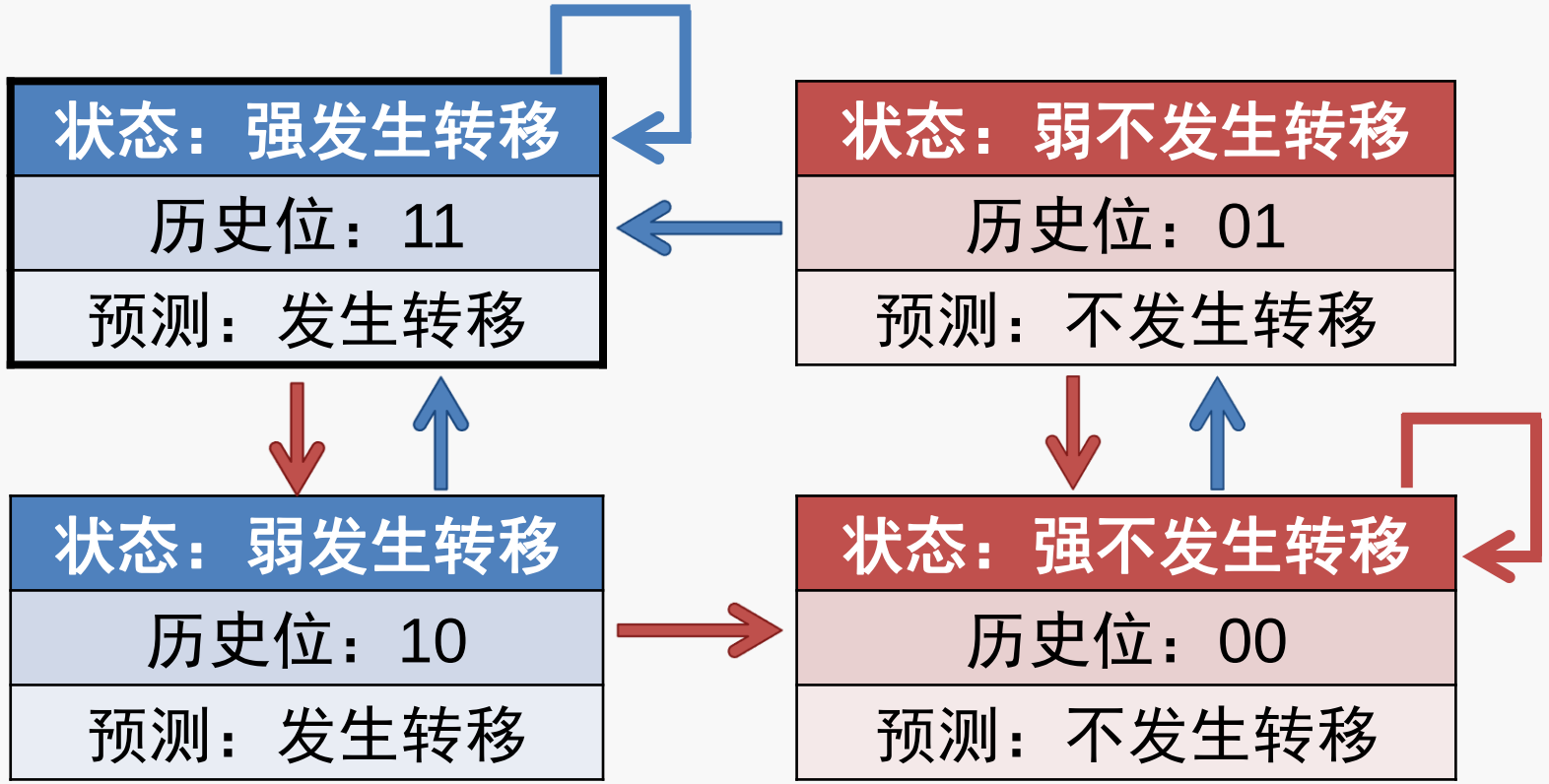
历史位设定分析（1）

- Pentium对历史位的设定更倾向于预测转移发生，三种历史位状态都预测转移发生

① 11：称为“强发生” 状态（strongly taken）
② 10：称为“发生” 状态（taken）
③ 01：称为“弱发生” 状态（weakly taken）

历史位设定分析（2）

- 历史位的设定，可以有不同于Pentium的多种方式。例如，下图所示的是一种相对无倾向性的设定



历史位设定分析（3）

- 对于循环程序，两种情况下可能需要重新计算转移地址，并造成流水线的停顿和等待
  - ① 首次进入循环时，预测错误：预测不发生转移，而实际发生转移
  - ② 退出循环时，预测错误：预测发生转移，而实际不发生转移

- 示例：假设有一段循环10次的代码被反复调用
  - 若使用1位历史信息，每轮调用时，2次预测错误8次正确
  - 若使用2位历史信息，每轮调用时，1次预测错误9次正确

转移预测器的效率

- N位历史信息的转移预测器

历史位数	转移预测正确率
N=1	82.5% ~ 96.2%
N=2	86.8% ~ 97.0%
N=3	88.3% ~ 97.0%
N≥4	继续提升，但很微弱

*IBM的Ravi Nair对转移预测器的效率分析

转移预测技术的实现要点

转移条件判定的预测

- 预测“要不要转移”

转移目标地址的预测

- 预测“转移到哪里”

转移目标地址的预测

- 转移目标缓冲器
  - BTB: Branch Target Buffer
  - 保存了此前若干次转移指令执行时的目标地址

BTB

转移目标地址	转移条件判定	转移指令地址
BTA_1	T/NT	BIA_1
BTA_2	T/NT	BIA_2
BTA_3	T/NT	BIA_3
……	……	……

*BIA: Branch Instruction Address
*BTA: Branch Target Address
*T: Taken
*NT: Not Taken

BTB的基本操作（1）

- 分配BTB表项
  - 当一条转移指令第一次执行时，在BTB中为其分配一个表项
  - 该指令自身的地址保存在“转移指令地址”域
  - 转移目标地址保存在“转移目标地址”域

- BTB表项比较
  - 将需要预测的指令地址与BTB的“转移指令地址”域进行比较。若有一项匹配，则称为BTB命中
  - BTB命中表明：该指令此前被执行过，而且为转移指令

BTB的基本操作（2）

- 产生转移目标地址
  - 若BTB命中，且该指令转移条件判定的预测结果为“发生转移”，则将该表项的“转移目标地址”域读出作为下一条指令地址

- 更新BTB
  - 转移指令最终执行得到的目标地址和条件判定结果要和预测结果进行比较，如果不一致，则需要根据最终执行结果更新BTB

BTB运行示例

```
                                        ;指令地址
TA1:   MOV    BX, OFFSET TA4    ;31000H
TA2:   JMP    BX                ;31003H
       NOP                      ;31005H

TA3    INC    AX                ;31006H
TA4:   MOV    DX, OFFSET TA2    ;31007H
       MOV    BX, OFFSET TA3    ;3100AH
       JMP    DX                ;3100DH
```

1.未命中，添加表项1

3.命中，从表项1取出目标地址“31007H”；预测错误，更新表项1

5.命中，从表项1取出目标地址“31006H”

2.未命中，添加表项2

4.命中，从表项取出目标地址“31003H”

BTB

转移目标地址	转移指令地址	
31006H ~~31007H~~	31003H	表项1
31003H	3100DH	表项2
		表项3
……	……	

进行BTB表项比较的时机

取指的同时（Xscale）

优点：在流水线较早阶段获得转移目标地址

缺点：每条指令均需访问BTB，功耗开销较大

译码完成后（Pentium）

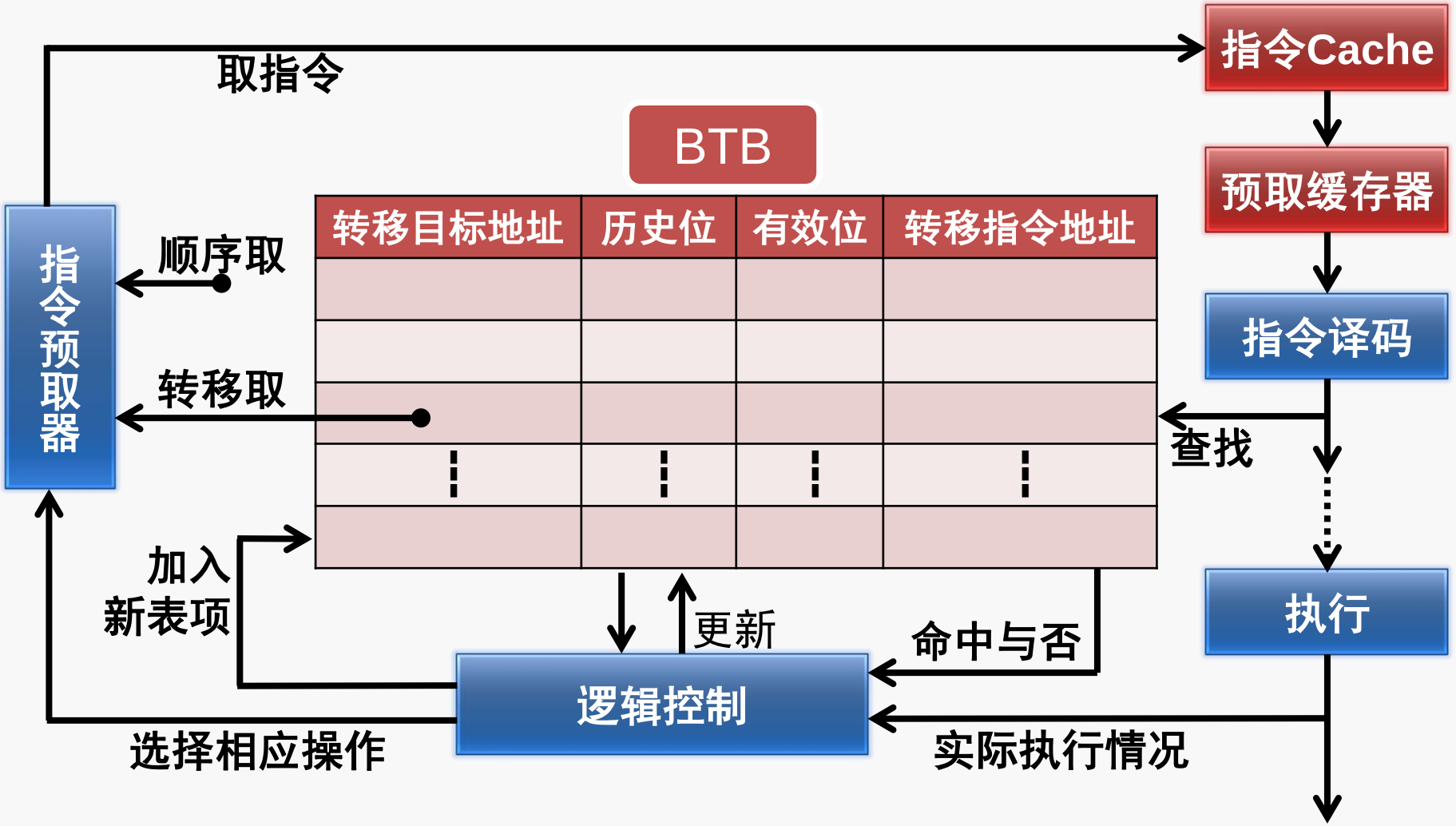
优点：转移指令才需访问BTB，功耗开销较小

缺点：在流水线较晚阶段获得转移目标地址

预译码完成后，取指的同时（UltraSPARC III）

将上述两种方法的优点进行结合

Pentium BTB的工作机制示意图



Pentium BTB工作机制说明（1）

- “指令译码”阶段检查指令
  - 检查从预取缓存器中取出的指令是否转移指令
  - 若是，则将此指令的地址送往BTB进行查找

- 若 “BTB命中”
  - 根据该表项的“历史位”状态，预测此指令是否发生转移

Pentium BTB工作机制说明（2）

- 若预测发生转移
  - 将该表项中的“转移目标地址”提交给指令预取器
  - 指令预取器从“转移目标地址”处取指令装入预取缓存器，即进行所谓“转移取”
- 若预测不发生转移
  - 从该转移指令之后的地址开始取指令，即进行所谓“顺序取”

Pentium BTB工作机制说明（3）

- 若“BTB未命中”
  - 说明BTB中没有该指令的历史记录，此时固定预测为不发生转移，即固定进行“顺序取”

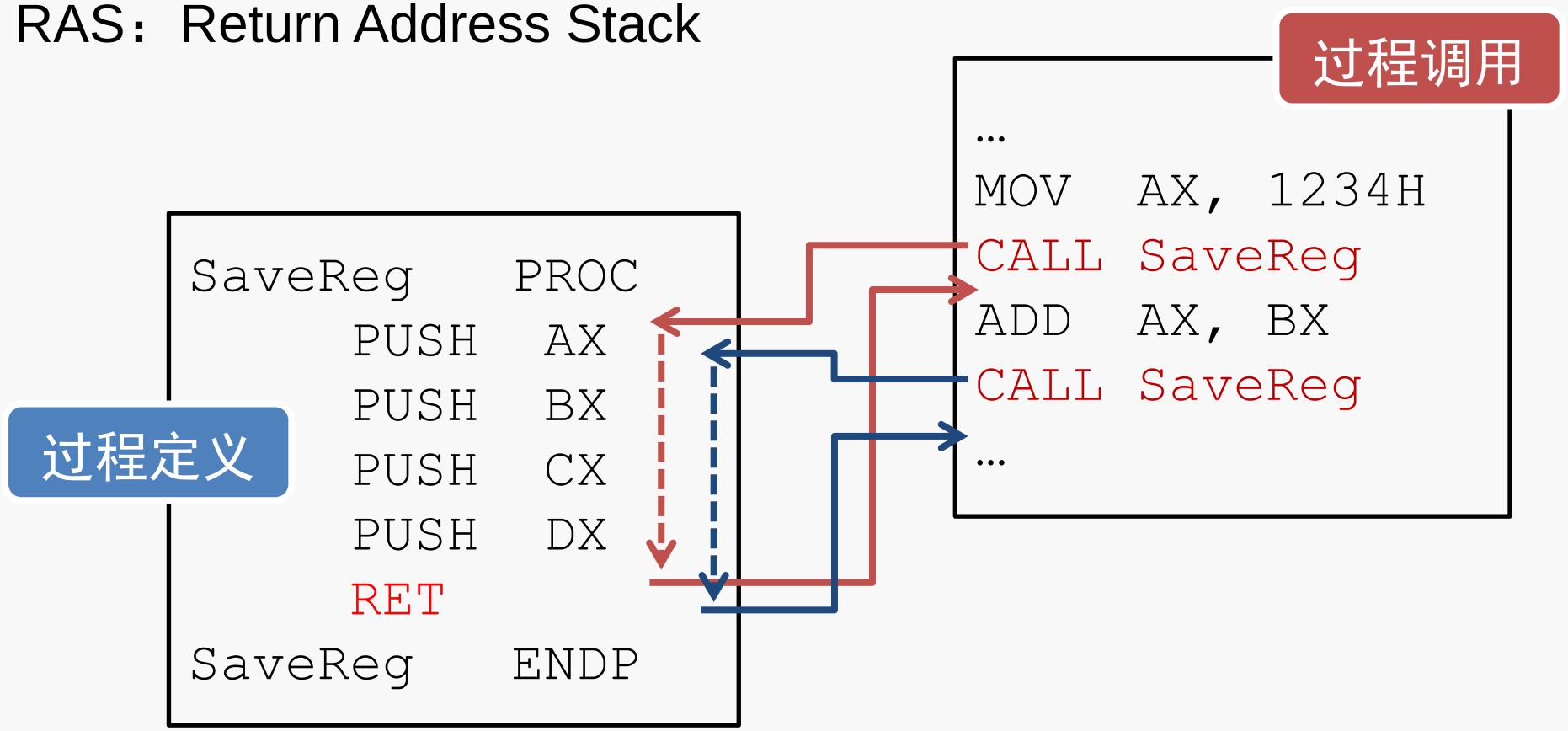
- 关于“BTB未命中”的指令
  - 若此指令实际没有发生转移，则无其它操作
  - 若此指令实际发生转移，则按“预测错误”处理，在BTB中建立一个新表项，设定“历史位”为11

主要内容

通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

返回地址栈RAS

- 特殊的转移指令：“过程返回”指令 RET
- 专用的预测部件：返回地址栈
  - RAS：Return Address Stack



“过程调用”/“过程返回”指令的功能

类型	示例	等效操作
段内直接调用	CALL Proc_A	PUSH IP JMP Proc_A
段内返回	RET	POP IP
段间直接调用	CALL FAR PTR Proc_B	PUSH CS PUSH IP JMP FAR PTR Proc_B
段间返回	RET	POP IP POP CS

“过程返回”指令的特点

优点

① 无需判定转移条件（均为无条件转移）

② 执行“过程返回”指令时，转移目标地址已经生成（早在执行“过程调用”指令的时候生成）

③ “过程返回”指令的出现是可预期的（“过程调用”指令和“过程返回”指令必须成对出现）

缺点

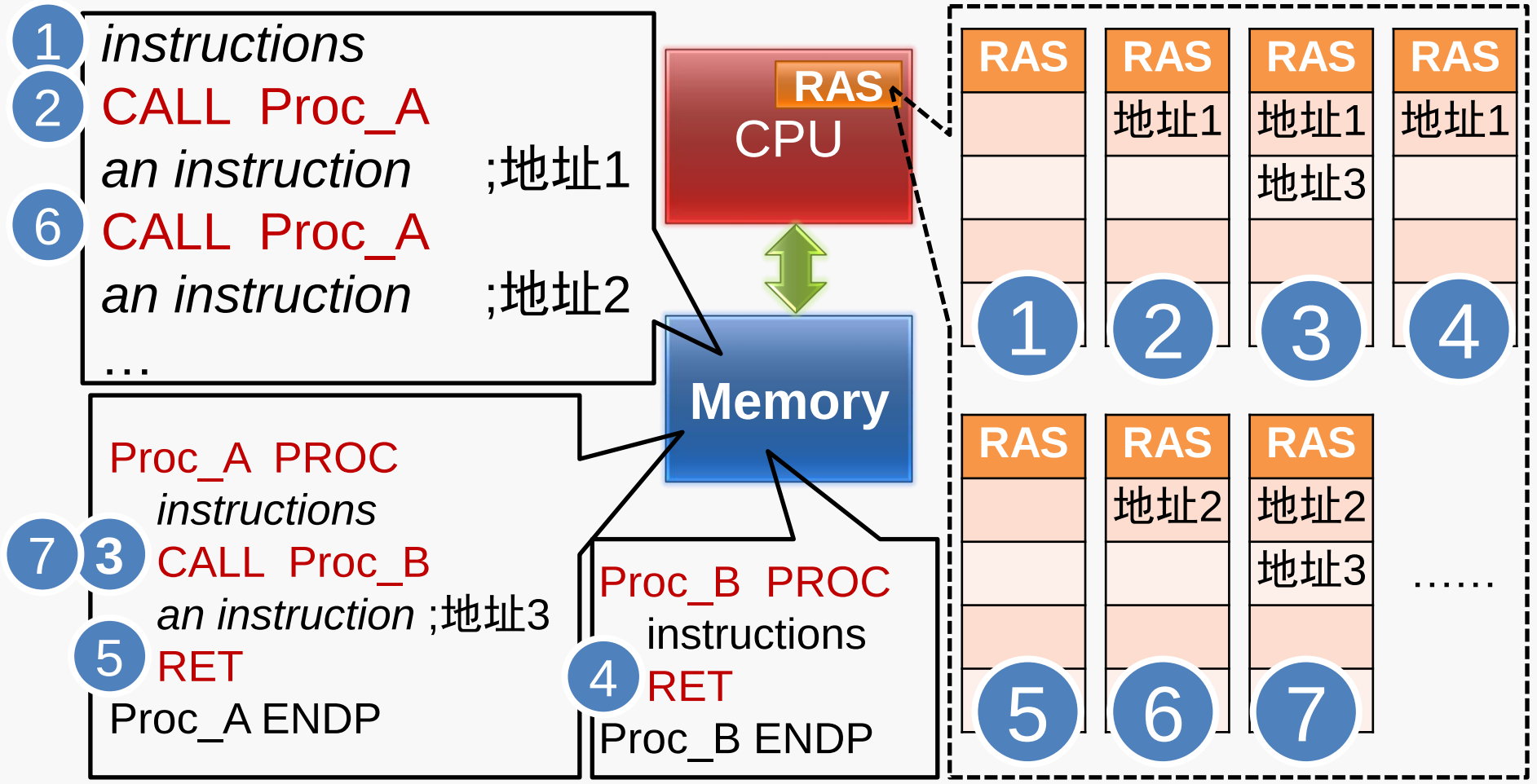
① 每次执行同一条“过程返回”指令时，转移目标地址往往不同

② 转移目标地址在存储器中，访问时间较长

③ 在流水线晚期才访问存储器获得转移目标地址

返回地址栈运行示例

- RAS：Return Address Stack



本讲到此结束，谢谢

欢迎继续学习本课程

计算机组织与体系结构 Computer Architectures

主讲：陆俊林

北京大学 Peking University